TÉCNICO LISBOA

Community Finding with Applications on Phylogenetic Networks

Luís Artur Domingues Rita

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisors: Prof. João André Nogueira Custódio Carriço

Prof. Alexandre Paulo Lourenço Francisco

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva

Supervisor: Prof. João André Nogueira Custódio Carriço

Members of the Committee: Prof. Sara Alexandra Cordeiro Madeira

May 2019

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Luís Rita

Preface

The work presented in this thesis was performed at the Ramirez, Mário Lab (Molecular Microbiology and Infection), Institute for Molecular Medicine, University of Lisbon (Lisbon, Portugal), during the period February 2018 – July 2019, under the supervision of Prof. João Carriço. And at the company National Health Institute Dr. Ricardo Jorge (Lisbon, Portugal), during the period May – July 2019, under the supervision of Dr. Vítor Borges. The thesis was co-supervised at Instituto Superior Técnico by Prof. Alexandre Francisco.

Abstract

With the advent of high-throughput sequencing methods, new ways for visualizing and analyzing increasingly amounts of data are needed. Although some software already exist, they do not scale well or require advanced technical knowledge to be useful in phylogenetics.

The aim of this thesis was to develop a web application – *Phyl* – to perform community finding and visualize the results in a minimalist way, in any browser or device.

Louvain, Infomap and Layered Label Propagation algorithms were implemented, tested and benchmarked against real (Amazon, Zachary's Karate Club and *Staphylococcus aureus*) and synthetic networks (Girvan-Newman and Lancichinetti-Fortunato-Radicchi). Interfaces were implemented and tested using Cytoscape.js and D3.js (SVG and Canvas).

For the first time, these algorithms were implemented in JavaScript. Unless otherwise stated, next conclusions are valid for both synthetic networks. In terms of speed, Louvain outperformed all others. In terms of accuracy, in networks with well-defined communities, Louvain was the most accurate. For higher mixing, Layered Label Propagation performed better. It is advantageous to use higher gamma parameters only for highly mixed Girvan-Newman networks, oppositely, to what was verified in well separated ones. The increase in the nodes' average degree improved community detection in weakly mixed networks for every algorithm. Detection worsens for higher mixing. Regarding benchmark algorithms, it is computationally more intensive to generate networks with higher mixing or average node degree. In the Lancichinetti-Fortunato-Radicchi implementation, the execution time remains linear. In the thesis implementation, it is exponential.

Along with PHYLOViZ and INSaFLU, these tools allow us to trace the source of an outbreak, infer about the pathogenicity of a recently discovered strain or to determine the most suitable strains to be targeted for vaccination. They are intended to assist distinct health professionals and are designed to enable medical and research purposes.

Community Finding | Data Visualization | Phylogenetic Trees | Web Application

Resumo

With the advent of high-throughput sequencing methods, new ways for visualizing and analysing increasingly amounts of data are needed. Although some software already exist, they do not scale well for big data or require advanced technical knowledge to be useful in phylogenetics.

The aim of this thesis was to develop a web application – *Phyl* – to perform community finding and visualize the results in a minimalist way, in any browser or device.

Louvain, Infomap, Label Propagation and Layered Label Propagation were implemented, tested and benchmarked against multiple databases with ground-truth communities containing millions of nodes and edges. New cutting-edge interfaces were built and tested using Cytoscape.js, D3.js (SVG) and D3.js (canvas) in order to display the results obtained after running these tools.

For the first time, these algorithms were implemented in JavaScript (next generation and scalable web programming language). Along with cutting edge visualization tools, intensive data analysis is now possible to be performed in any browser and device.

PHYLOViZ and INSaFLU, along with the developed tools, provide a brand-new insight into the field of microbiology. Based on microbial typing, they now allow us to determine how similar multiple strains are and to infer a possible evolutionary path. To trace the source of an outbreak, infer about the pathogenicity/virulence of a recently discovered strain or, based on serotype data (Fig. 1) or other phenotypic data, to determine the most suitable strains to be targeted for vaccination. These tools are intended to assist distinct health professionals, including doctors and bioinformaticians, and are designed to enable medical and research purposes.

Community Finding | Visualização Dados | Árvores Filogenéticas| Aplicação Web

Acknowledgments

Truly grateful to my supervisors: Alexandre Francisco, from INESC-ID; João Carriço, from Institute of Molecular Medicine, and Vítor Borges, from National Health Institute Dr. Ricardo Jorge, that crucially contributed to the development of my master thesis. To EIT Health which allowed me to look for different insights internationally: Madrid, Spain (1st Workshop Master in Technological Innovation in Health); Mannheim, Germany (International Master in Innovative Medicine Industry & Innovation Day) and Heidelberg, Germany (Antimicrobial Resistance Hackathon 2018). From students and professors from some of the best European medical/engineering universities. To the Student Association from Abel Salazar Biomedical Sciences Institute for giving me the opportunity to present a poster of the thesis in their Biomedical Congress in Oporto, Portugal. And to Santo Fortunato who helped me along the way of the benchmark testing.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AGM	Community-Affiliation Graph Model
AMR	Antimicrobial Resistance
CC	Clonal Complex
CF	Community Finding
CI	Confidence Interval
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
EIT	European Institute of Innovation & Technology
GN	Girvan-Newman
HTML	Hypertext Markup Language
INSaFLU	Inside the Flu
JS	JavaScript
JSON	JavaScript Object Notation
LFR	Lancichinetti-Fortunato-Radicchi
LLP	Layered Label Propagation
MD	Markdown
MDL	Minimum Description Length
MLVA	Multilocus Variable-Number Tandem Repeat
MTiH	Master in Technological Innovation in Health
NMI	Normalized Mutual Information
NPM	Node.js Package Manager
SNP	Single Nucleotide Polymorphism
SVG	Scalable Vector Graphics

1. Introduction

In an era where data is the most important resource in the world, new ways to storage, analyze and retrieve the results are becoming a pressing need.

One way to model and analyze complex systems is using networks. Although they are ubiquitous, their mathematical representation – graphs, it is still not a standard. Sometimes due to the lack of reliable data or for privacy issues.

NGS methods, developed in the last years, have been providing us massive quantities of microbiological data. This has been crucial, for instance, in all stages of infection prevention. From targeted vaccination, assessing the pathogenicity of a sequenced strain, to outbreak tackling. As AMR is becoming a central issue, efficient ways to handle these problems are needed (Figure 1).

AMR started with the use of drugs intended to treat diseases caused by pathogenic organisms (e.g., viruses, bacteria, fungi and other parasites). Although this is a natural evolutionary process, in which the most competent strains get selected over time, the frequency people are using antimicrobial drugs is accelerating the resistance. This is particularly relevant not only among the population (people are being mistreated and taking antimicrobials for prevention, inappropriately), but also in farms where cattle are being given preventive doses of antibiotics incorrectly. In addition, the frequency people are travelling around the world is facilitating the spread of contagious diseases.

All of this makes surveillance and, consequently, outbreak tackling, a progressively fundamental issue. As this requires the action of central health authorities in very short time scales, it is fundamental to give them scalable and easy-to-use tools to successfully handle these situations.

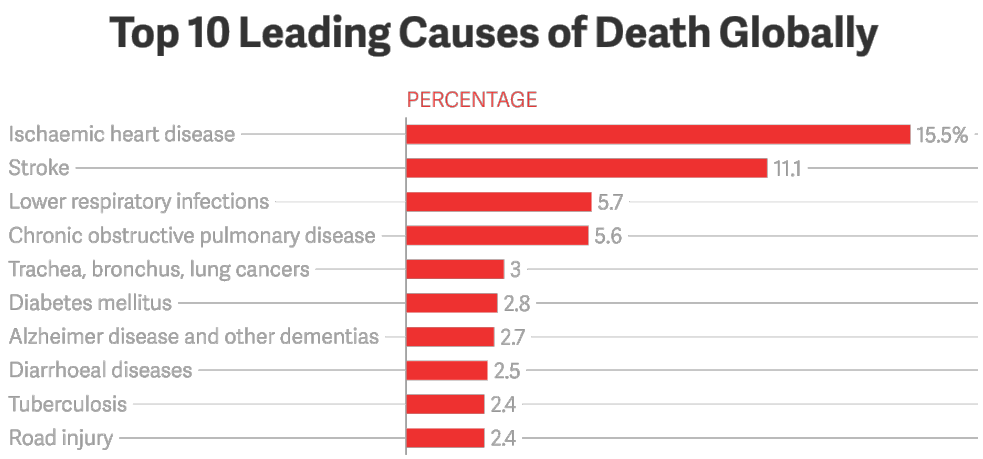

Figure 1 Infections are still one of the major causes of death globally. Data collected by WHO between 2000-2015 from a population of 90 000 people *[1]*.

The main goal of this thesis was to extract functional knowledge from the topological structure of phylogenetic networks. In other words, contrarily to Erdős-Rényi random networks, real ones present an asymmetrical

distribution of edges. From this asymmetry, clusters of nodes can be identified. By definition, communities are the regions in the network with higher density of edges. Nonetheless, this definition is not restrictive enough to undoubtedly identify them in graphs. This raises the following question: how can we detect these regions if it is not clear what we are looking for?

To answer this question, community finding theory has been developed in the past years. In spite there has been a considerable advance in terms of accuracy and speed of detection, some algorithms perform better than others in different networks. Accounting to the network being analyzed, a wide variety of algorithms with varying parameters should be tested, to find the one that performs the best.

In order to guide the reader, the structure of the thesis is outlined. In *Introduction*, important concepts of graph theory are presented. Next, still in the same section, community finding and phylogenetic topics are covered. In *Related Work,* the state of the art of all the key concepts included in *Methodology* are reviewed. In *Methodology*, community finding algorithms, benchmarking techniques and data visualization options are explored. In *Results*, community finding and benchmarking algorithms' performance is analyzed in terms of their accuracy and speed. Different visualization frameworks are also compared. In *Conclusion*, a retrospective analysis of the thesis work is done, as well as, a set of prospective improvements are discussed.

1.1. Graph Theory

Networks are a particular type of graphs that represent real systems and the interaction among their components [2]. Graphs are said to be constituted by vertices and edges (Figure 2), on the other hand, in networks, they are identified as nodes and links. In spite of the different terminology, they fundamentally pretend to simplify and facilitate the analysis of real networks.

Graphs are the object of study of Graph Theory, which joins a set of formalities indispensable for their study in a standardized and meaningful way.

Networks are ubiquitous in the present world. In an increasing order of magnitude and anthropocentric point of view: protein interactions, neurons organization/communication, human interactions and societal behavior are often explained by current Graph Theory.

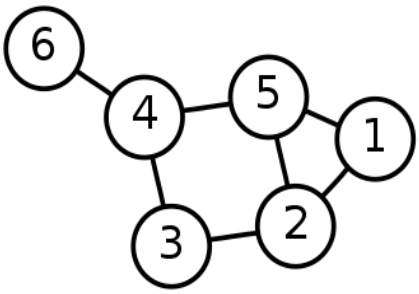

Figure 2 An example of a graph. 6 vertices connected by 7 edges *[3]*.

1.1.1. Networks

1.1.1.1. Properties

A key property of nodes is their degree. It is defined as the sum of the weights of the links to other nodes. Links are the structures connecting nodes. These can be directed, undirected, weighted or unweighted depending on the purpose of the analysis or the information available. In directed links, node degree property splits in incoming and outgoing degrees. Considering, respectively, the sum of the links pointing towards the node or outwards. Both nodes and links local properties have effects in the whole network.

The sum of the weights of all links in the network is identified as the total number of links:

$$L = \frac{1}{2}\sum_{i=1}^{N} k_i \tag{1}$$

$\frac{1}{2}$ coefficient corrects for the fact each link is counted twice in undirected networks. In directed networks, the total number of links is given by:

$$L = \sum_{i=1}^{N} k_i^{in} = \sum_{i=1}^{N} k_i^{out} \tag{2}$$

The average degree is given by:

$$\langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i \tag{3}$$

In the case of directed networks, where:

$$k_i = k_i^{in} + k_i^{out} \tag{4}$$

The average incoming and outgoing degrees are:

$$\langle k^{in} \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i^{in} = \langle k^{out} \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i^{out} = \frac{L}{N} \tag{5}$$

One measure that arises from the pattern of connections between nodes is the degree distribution. It gained a particular relevance when random and scale-free networks were discovered.

All previous measures can be derived from a single mathematical representation of the network: the adjacency matrix A_{ij}. It includes not only the connections between nodes in the network, but also the respective weights (in case the network is weighted).

Whenever nodes i and j are connected in an unweighted network, the adjacency matrix entry is:

$$A_{ij} = 1 \tag{6}$$

In a weighted network:

$$A_{ij} = w_{ij} \tag{7}$$

w_{ij} is the weight of the link.

If the same pair of nodes is not connected:

$$A_{ij} = 0 \tag{8}$$

The degree of each node can be derived based on this matrix:

$$k_i = \frac{1}{N}\sum_{j=1}^{N} A_{ji} = \frac{1}{N}\sum_{i=1}^{N} A_{ji} \tag{9}$$

In directed networks, the incoming and outgoing degrees are:

$$k_i^{in} = \frac{1}{N}\sum_{j=1}^{N} A_{ij} \tag{10}$$

$$k_i^{out} = \frac{1}{N}\sum_{j=1}^{N} A_{ji} \tag{11}$$

$$2L = \sum_{ij}^{N} A_{ij} \tag{12}$$

Still regarding the overall topology of the network, a set of additional concepts will help us understanding the theory behind. Paths in the network and distances between nodes are common definitions used in network theory. Generally, a path is a sequence of nodes connected by the respective links joining 2 nodes. Shortest paths are constituted by the sequence of nodes that minimize the distance between the pair. Multiple shortest paths can coexist in the same network. An average path length is defined as the average distance among the multiple available paths connecting the pair of nodes. Whenever a path has the same start and end node, it is called a cycle. Eulerian paths are the ones that do not contain repeated links. The diameter of a network can be defined as the longest distance among all the shortest paths between every node pair.

Connectedness is another important property of networks. A network is considered connected if it is possible to reach any node starting in any other. When this is not possible, multiple components coexist in the network. This property has consequences in the respective adjacency matrix. In the presence of a disconnected network, it is possible to write it in the block diagonal form (Figure 3).

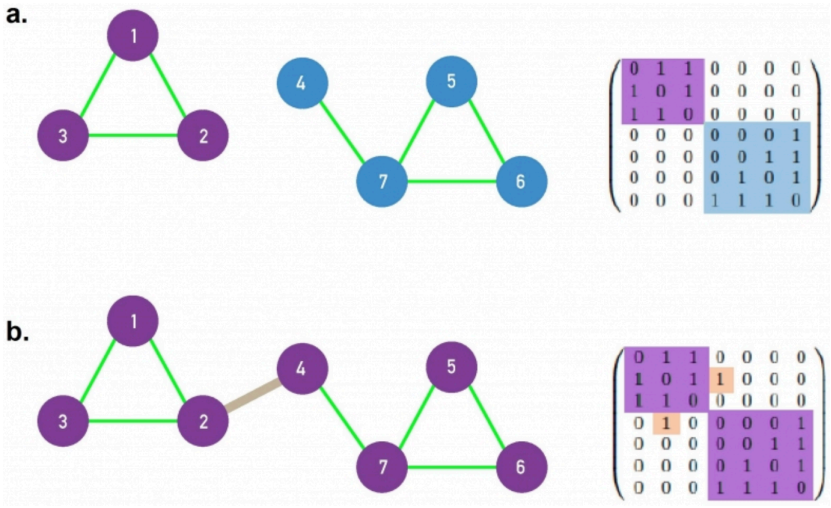

Figure 3 Adjacency matrix of a connected and a non-connected network *[2]* (Section 2.9).

Most real networks are sparse. This means the number of links can be approximated by the number of nodes. Assuming the maximum number of links in an undirected network is given by:

$$L_{max} = \frac{N(N-1)}{2} \tag{13}$$

The fraction $\frac{L}{L_{max}}$ is close to 0. This means most entries in the adjacency matrix are 0. Thus, it is often more efficient to store the links instead of the nodes of a network.

Local clustering coefficient examines the degree the neighbors of a given node are connected to each other:

$$C_i = \frac{2L_i}{k_i(k_i-1)} \tag{14}$$

L_i is the total number of edges connecting neighbors of node i and k_i its degree. $C_i$is 1 when all neighbors are connected and 0 when there are no links connecting them.

Generalizing this concept to the whole network:

$$\langle C \rangle = \frac{1}{N}\sum_{i=1}^{N} C_i \tag{15}$$

This is the probability of finding 2 neighbors connected in a random network.

One way to determine the value of a real network is given by the Metcalfe's Law. It states the value is proportional to the number of links between nodes. This means, it is proportional to the square number of nodes. In the case of technological networks, the cost of a network is proportional to the number of nodes (Figure 4).

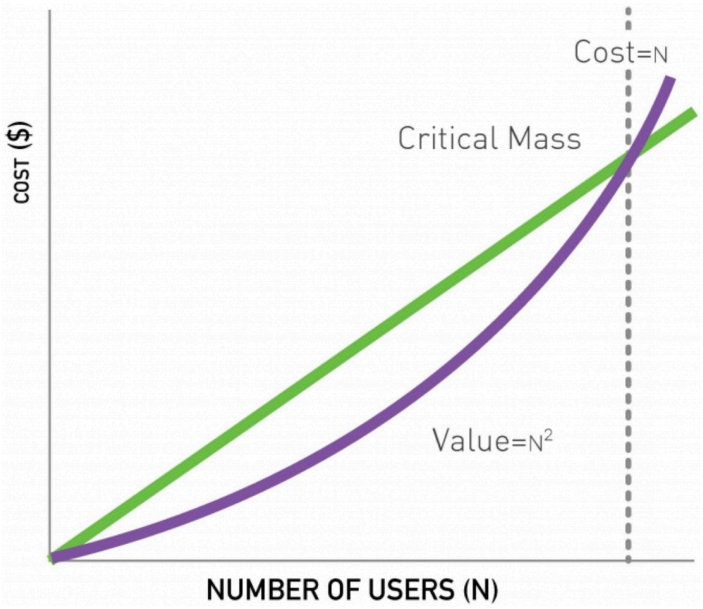

Figure 4 Variation of the value and costs of a common technological network in terms of the number of users *[2]* (Section 2.6).

1.1.1.2. Seven Bridges of Konigsberg

This problem was in the foundation of graph theory [2]. A paper written in 1736 by Leonhard Euler is often regarded as the first in its history (Figure 5) [4].

The city of Konigsberg in Prussia (now Kaliningrad, Russia), lay in both sides of Pregel River and includes 2 islands – Kneiphof and Lomse. These were connected to the 2 mainland portions by 7 bridges. The problem was to devise a walk such that one could cross every bridge exactly once. In fact, it shows how graph theory can be used to solve real-world problems described using networks.

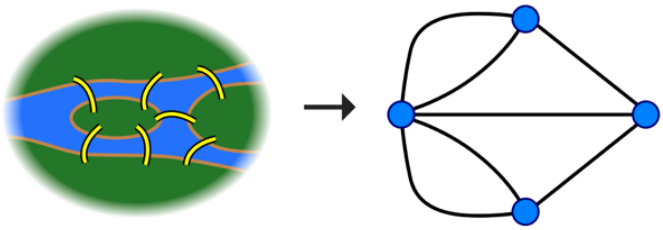

Figure 5 The Seven Bridges of Konigsberg problem is a mark in graph theory *[5]*.

It is possible to find a solution for this and similar small problems using a simple brute-force approach. By testing every possibility, it is verifiable there is no solution. Having said this, the power of Graph Theory arises when many potential solutions are present, such that even the most powerful computers are not able to find the best in a short time scale. This leads us to a class of problems called NP.

1.1.2. Network Models

In order to model real networks, a random model was the first to be proposed.

1.1.2.1. Random Model

There are 2 random model definitions that differ in the fixed variables. The one proposed by Erdős and Rényi [6] considers the number of links and the total number of nodes in the network fixed. In the model introduced by Gilbert [7], the probability of connection between 2 random nodes is fixed, along with the total number of nodes.

Once the first assumes an unrealistic property, fixed number of links, the second is considered to be the random model that better approximates a real network. By omission, it is considered until the end of this section.

Next, the features that arise from the random model are going to be tested against what is verified in real networks. In order to calculate the average number of links and the nodes' average degree of the model, the probability of finding L links in the network is calculated:

$$p_L = \binom{\frac{N(N-1)}{2}}{L} p^L (1-p)^{\frac{N(N-1)}{2}-L} \tag{16}$$

Once p_L is a binomial distribution, the average number of links is:

$$\langle L \rangle = \sum_{L=0}^{\frac{N(N-1)}{2}} L p_L = p\frac{N(N-1)}{2} \tag{17}$$

The average degree of the nodes in the network is:

$$\langle k \rangle = \frac{2\langle L \rangle}{N} = p(N-1) \tag{18}$$

Similarly to (16), (19) describes network's degree distribution:

$$p_k = \binom{N-1}{k} p^k (1-p)^{N-1-k} \tag{19}$$

Although (19) is the exact formula of the degree distribution in a random network, a Poisson alternative is well approximated for $N >> \langle k \rangle$:

$$p_k = e^{-\langle k \rangle} \frac{\langle k \rangle^k}{k!} \tag{20}$$

Once most real networks are sparse, this is a good approximation. Poisson distribution facilitates the calculation of network's average degree $\langle k \rangle$, second moment $\langle k^2 \rangle$ (useful for estimating variance), third moment $\langle k^3 \rangle$ (skewness measure) and standard deviation σ. For this reason, unless otherwise stated, this distribution will be further considered (Figure 6).

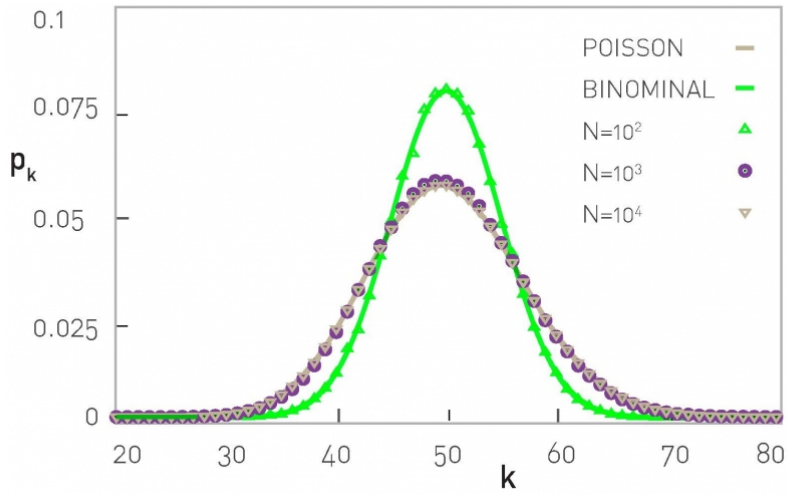

Figure 6 Binomial and Poisson distributions of networks with different number of nodes *[2]* (Section 3.4).

There are 3 properties of real networks that are not in agreement with the random model. Those are the presence of hubs, connectedness and the clustering coefficient of the networks.

Replacing the factorial of the number of links in the degree distribution, it becomes clear the reason why hubs are not present:

$$k! \approx \sqrt{2\pi k}\left(\frac{k}{e}\right)^k \tag{21}$$

Rewriting (20),

$$p_k = e^{-\langle k\rangle}\frac{\langle k\rangle^k}{\sqrt{2\pi k}\left(\frac{k}{e}\right)^k} = \frac{e^{-\langle k\rangle}}{\sqrt{2\pi k}}\left(\frac{e\langle k\rangle}{k}\right)^k \tag{22}$$

Whenever $e\langle k\rangle < k$, p_k decreases faster than exponentially due to the denominator terms.

In real networks, most nodes are connected. In order to observe this in random networks, it is necessary to increase the nodes' average degree until a given threshold. When this happens, a giant component appears connecting most of the nodes. In order to estimate this value, the probability a random node is not connected to this component is calculated:

$$(1-p)^{N_G} \approx (1-p)^N \tag{23}$$

Considering the giant component spreads approximately over the whole network, $N_G \approx N$. Consequently, the expected number of isolated nodes is:

$$I_N = N(1-p)^N = N\left(1-\frac{Np}{N}\right)^N \approx Ne^{-Np} \tag{24}$$

Assuming all nodes are connected:

$$I_N = 1 \Leftrightarrow Ne^{-Np} = 1 \Leftrightarrow p = \frac{\ln(N)}{N} \tag{25}$$

Once $Np = \langle k\rangle$,

$$\langle k \rangle = \ln(N) \tag{26}$$

The inconsistency between the random model and real networks is that the latter are connected and present an average node degree significantly lower than (26).

Local clustering coefficient is given by the proportion of links there are between the neighbors of a given node and all the possibilities:

$$C_i = \frac{2p\frac{k_i(k_i-1)}{2}}{k_i(k_i-1)} = p = \frac{\langle k \rangle}{N} \tag{27}$$

In random networks, clustering coefficient is independent of k. Although, in real networks, this is not the case.

Due to these 3 incompatibilities, it was necessary to build a model that could explain them.

1.1.2.2. Barabási-Albert Model

This model was proposed by Albert-László Barabási and Réka Albert. It focuses not only on the structural aspect of the networks, but also in their formation process. Thus, it is expected to fill the gaps left by the random models.

Hubs present in real networks and network's degree distribution are explained by 2 fundamental phenomena of their formation: growth and preferential attachment.

The model starts by considering a set of m_0 connected nodes. At each time step, a node with m links is added to the network ($m \leq m_0$). The links are connected to the existent nodes with a probability proportional to their degree:

$$\prod k_i = \frac{k_i}{\sum_j k_j} \tag{28}$$

This way, the rate node i increases its degree is proportional to k_i and m:

$$\frac{dk_i}{dt} = m\prod(k_i) = m\frac{k_i}{\sum_{j=1}^{N-1} k_j} \tag{29}$$

Once,

$$\sum_{j=1}^{N-1} k_j = 2mt - m \tag{30}$$

The rate of change of k_i is given by:

$$\frac{dk_i}{dt} = \frac{k_i}{2t-1} \tag{31}$$

Rearranging and integrating (31), $k_i(t)$ is obtained:

$$k_i(t) = m\left(\frac{t}{t_i}\right)^{\frac{1}{2}} \tag{32}$$

(32) suggests every node increases its degree at the same rate, independently of the moment it was added to the network. Not only this assures a universal power-law increase, but also suggests that older nodes tend to have more connections. The latter is usually called, in many domains, the first-mover advantage. Particularly important in the context of phylogenetics in the detection of ancestrality. This property justifies the presence of hubs and, consequently, the connectedness of the whole network.

In order to derive the degree distribution of the Barabási-Albert model, cumulative degree distribution is calculated.

Supposing,

$$k_i(t) < k \Leftrightarrow m\left(\frac{t}{t_i}\right)^{\frac{1}{2}} < k \Leftrightarrow t_i < t\left(\frac{m}{k}\right)^2 \tag{33}$$

The number of nodes with degree lower than k is $t\left(\frac{m}{k}\right)^2$.

Approximating the total number of nodes,

$$N = m_0 + t \approx t \tag{34}$$

The cumulative distribution is given by:

$$P(k) = 1 - \left(\frac{m}{k}\right)^2 \tag{35}$$

The degree distribution is the derivative of (35):

$$p_k = \frac{dP(k)}{dk} = \frac{2m^2}{k^3} \tag{36}$$

(36) evidences the power-law degree dependence, contrarily to the exponential verified in the Poisson/Binomial distributions. Whenever a network follows a power law distribution:

$$p_k \propto k^{-\gamma} \tag{37}$$

It favors the existence of hubs due to its long tail, which includes several nodes with higher degrees. Networks with this characteristic are called scale-free [8]. This term refers to the nature of the second and third moments of the degree distribution that diverge to infinity when the degree exponent is $2 \leq \gamma \leq 3$ (verified in many real networks) and the total number of nodes in the network goes to infinity [2].

While in the random model the average clustering coefficient is proportional to $\frac{1}{N}$, in Barabási-Albert networks, it behaves differently:

$$\langle C \rangle \propto \frac{(lnN)^2}{N} \tag{38}$$

The term in the numerator increases the clustering coefficient in large networks. This prediction is in line with what is verified in real networks, higher clustering [2].

There are a couple of open questions Barabási-Albert model cannot explain. First, it considers the degree exponent $\gamma = 3$ [2]. While, in real networks, $2 \leq \gamma \leq 5$ [2] (Chapter 5, Section 5.11). Only undirected networks were covered. Other dynamics for node and link generation were not considered. Finally, differences among nodes were resumed to their degrees, not considering other intrinsic characteristics.

1.1.3. Complexity

Due to the size of real networks, it is sometimes unfeasible to use brute-force algorithms to untangle communities. Algorithms used to handle these problems, in the best-case scenario, run in polynomial time. Although, most of the times, it is necessary to explore an exponential number of possibilities which grows with the network size. First problems are called *P problems* and, the second, *NP problems*.

P vs NP problem was introduced in 1971 by Stephen Cook [9] and, in 2000, was considered one of the most important open questions in computer science [10]. It is part of a set of six problems that were defined by the Clay Mathematics Institute as the 7 Millennium Prize Problems [11]. And the person who can solve one of them, will be awarded with a prize of 1 million dollars. *P vs NP problem* can be stated as: *Can every problem whose solution can be quickly verified by a computer also be quickly solved by a computer?* [12].

P problems are characterized for being solvable in polynomial time by a Deterministic Turing Machine (DTM). On the other hand, *NP* category includes those only solvable in polynomial time using a Non-deterministic Turing Machine (NTM). They are both part of a set of decision problems whose answer is binary. DTM is a non-branching machine that can only proceed to one next step at each time. This is how an ordinary computer works. NTM is a conceptual machine that is able to analyze, simultaneously, a polynomial number of different options at each step. Thus, a branching device. A *quickly verified* solution means it can be found in polynomial time. The word computer refers to a Deterministic Turing Machine (DTM).

Although the question whether *P = NP* remains unanswered, by definition, there is a relation between the 2 groups. *NP problems* include those whose solution may not be obtainable but can be verified in polynomial time in a DTM. This means, every *P problem* belongs to *NP* once it is always easier to verify a solution than to find it. And they can find it in polynomial time as discussed. Trivially, $P \subset NP$.

A different class of problems are the NP-complete. Not only they are NP, but also NP-hard (Figure 7). A problem is NP-hard when an algorithm belonging to NP can be reduced to another in the first class. This means, any problem in NP-hard class is at least as hard to solve than any in NP. The reduced term means that the inputs of a problem A can be converted in the inputs of a problem B, using a polynomial time algorithm, and provide the exact same output. In case A can be reduced to B, this has a couple of immediate consequences:

1. If $B \in P$, then $A \in P$;
2. If $B \in NP$, then $A \in NP$;
3. If A is NP-hard, then B is NP-hard.

Consequently, if a solution in polynomial time is found to a NP-complete problem, then *P vs NP problem* is solved and $P = NP$. Thus, it is not only fundamental to find a polynomial solution but also to prove a problem X is NP-complete. Based on the definition above, this can be split in 2 steps:

1. Showing X belongs to NP. It can be done by verifying a given solution in polynomial time or finding a non-deterministic algorithm for X;
2. Showing X is NP-hard. By reducing a known NP-complete problem to X. In this circumstance, the third consequence derived above implies X is NP-hard.

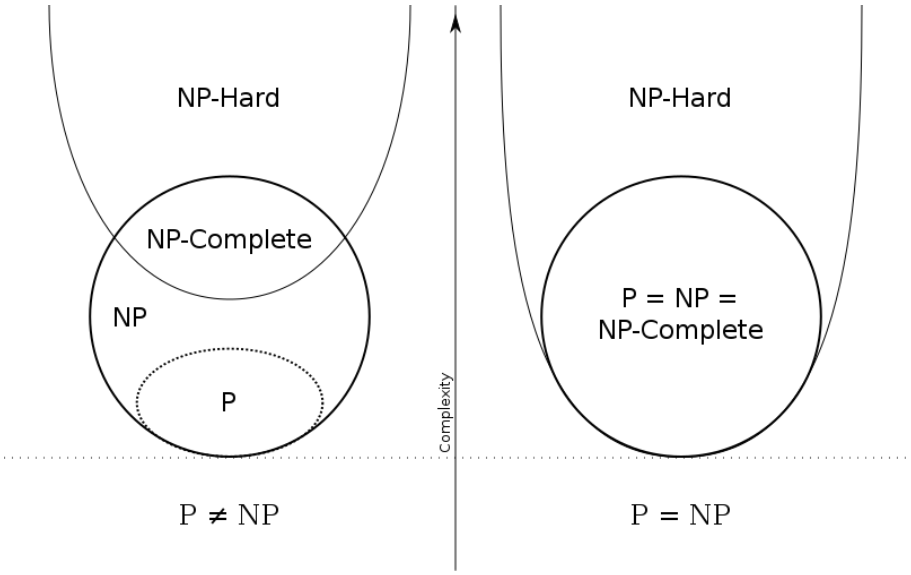

Figure 7 *P vs NP problem* diagram representation *[13]*.

Although it is not proved, it is believed by the majority of computer scientists $P \neq NP$ [14]. Proving $P = NP$ would have important consequences in the real world. Not only biological modelling would be enhanced, but also the efficiency in transportation systems, multimedia processing, economic simulations, among others, would be dramatically improved. On the other side, there are real systems based on the fact $P \neq NP$. One of them are the modern cryptographic systems which rely in prime factorization to secure data. This problem is considered *NP*.

A significant proportion of problems in network science are either NP, NP-hard or NP-complete. Thus, it is not surprising the importance of the solution in, for example, finding communities in phylogenetic networks.

A mathematical representation of time complexity of an algorithm is presented next.

1.1.3.1. Big-O Notation

While analyzing algorithmic performance, time and space complexity are usually considered. They are particularly important when there is limited computational power and the analysis is made in big data.

In some cases, the solution for the same problem can be found in completely different time scales. For this reason, it is important to have a way to upper bond the algorithms' execution time, but also to estimate its average value. This led to the creation of Big-O notation.

Depending on the number of instructions the algorithm has to execute and the size of the input, the runtime differs. Big-O notation offers a way to quantify the execution needs of an algorithm, independently of the machine that is running it. The identified complexity can be linear, logarithmic, quadratic, cubic, exponential… An ordered list, in terms of the time of execution of the most common algorithms (with some examples), is presented below:

N – when it is required the analysis and manipulation of an array;

log – divide-and-conquer algorithms are common when a main problem needs to be decomposed in several smaller ones;

N^2 – adjacency matrices often imply the use of quadratic algorithms. Cubic or higher order degrees are valid for similar data structures of higher dimensionality;

2^N – few algorithms are useful with this complexity. In situations where brute-force approaches are the only option, they may be considered.

After determining the complexity of an algorithm, it is possible to predict the time and memory requirements of the machine to execute it.

Definition

A function $f(N)$ is said to be $O(g(N))$ if constants k and N_0 exist, such that $0 \leq f(N) \leq k \times g(N)$ for all $N \geq N_0$ [15].

1.2. Community Finding

1.2.1. Historical Context

The first community finding experiment remotes 1927, when Stuart Rice tried to identify clusters in political bodies, using voting patterns [16]. Although, it was only in 2002, when Girvan and Newman used the Zachary's Karate Club network as a benchmark test, that community detection gained an increased interest among the scientific community [17]. As soon the paper was published, the number of citations in the theme steeply rose. Later on, Erdős and Rényi study [6] of random networks and the discovery of the scale-free property in real ones [8], turn community finding a central problem in network theory.

Along the thesis, lacking a clear definition of community, 4 hypotheses were formulated to provide us an intuition about the properties of these structures.

1.2.2. Community Hypothesis

Fundamental Hypothesis

It states that network's community structure is uniquely encoded in its wiring diagram [2]. Thus, they cannot be identified based exclusively in the degree distribution of a network. This can be verified upon degree-preserved randomization of networks.

Connectedness and Density Hypothesis

A community can be described as a dense and connected subgraph [2]. Depending on other criteria, stricter or looser rules can be used to refine this hypothesis. Identifying communities as maximum cliques, strong or weak communities are explored in the next section.

Random Hypothesis

Random networks do not present community structure [2]. This hypothesis allows us to find communities by comparing the studied network with a degree-preserved randomization of itself. This comparison measure is called modularity.

Maximal Modularity Hypothesis

A partition with higher modularity offers a better representation of the network's community structure [2]. This quality measure is detailed in section 1.2.5.

1.2.3. Communities

Based on the hypotheses above, additional criteria to clearly identify communities is proposed. Although, it is important to state that, for now, there is no golden-standard working the best in every network.

Maximum Clique

In the first references to communities, it was required all nodes to be connected to all others inside the same group [2]. Thus, each community is identified as a clique – a complete subgraph.

This definition was considered too rigid. In spite most real networks present triangles of connected nodes, larger cliques are rare. Moreover, this is not a robust definition of a community. One missing link would be enough to disqualify it as a clique. None of the networks present in the thesis would be correctly partitioned using it.

Strong Community

To overcome the rigidity of cliques, two definitions were proposed: strong and weak communities.

The definition of a strong community states that the inner degree of every node i should be greater than its external degree [2]:

$$k_i^{int}(C) > k_i^{ext}(C) \tag{39}$$

Weak Community

A weak community relaxes the previous definition in the way it only requires the sum of the internal degrees of every node from a community to be greater than the sum of the external degrees [2].

$$\sum_{i \in C} k_i^{int}(C) > \sum_{i \in C} k_i^{ext}(C) \tag{40}$$

k-cores

A community is formed by a set of nodes which are connected to at least k others belonging to the same community. In the same network, k-cores of different degrees may coexist [18].

p-cliques

The network is partitioned in groups in which all nodes have, at least, a fraction $0 \leq p \leq 1$ of links connecting them to nodes in the same cluster [19].

n-cliques

Whenever all nodes are separated by a distance not higher than n, they are part of the same community [20].

1.2.4. Complexity

Before presenting community finding algorithms, it is important to distinguish graph partitioning from community finding. In the first case, it is known the number of communities and the size of each one in the network. In community finding, the problem resides in partitioning the original graph without knowing, *a priori*, the number of communities or their sizes. This condition significantly increases the computational power needed to unleash them [2].

Graph Partitioning

The problem of reducing the number of edges connecting different groups is highly important in, for example, circuitry design and data processing. Whenever printing transistor (fundamental building block of modern electronic devices) circuitry in modules, it is fundamental to reduce wire overlap. Moreover, multicore property of actual CPUs is turning parallel computation possible. The goal here is to reduce communication between cores which significantly increases execution time [21].

The simplest problem of graph partitioning is called graph bisection. By analyzing the number of possible partitions in a network with size N and 2 communities with N_1 and N_2 nodes, it becomes rapidly unfeasible to analyze every single one. The number of possible partitions is given by:

$$\frac{N!}{N_1!\,N_2!} \tag{41}$$

Using Stirling approximation, a more meaningful formula is deduced:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{42}$$

$$\frac{N!}{N_1!\,N_2!} \approx \frac{\sqrt{2\pi N}\left(\frac{N}{e}\right)^N}{\sqrt{2\pi N_1}\left(\frac{N_1}{e}\right)^{N_1}\sqrt{2\pi N_2}\left(\frac{N_2}{e}\right)^{N_2}} = \frac{\sqrt{2\pi}N^{\frac{1}{2}}\,\frac{N^N}{e^N}}{\sqrt{2\pi}N_1^{\frac{1}{2}}\frac{{N_1}^{N_1}}{e^{N_1}}\sqrt{2\pi}N_2^{\frac{1}{2}}\frac{N_2^{N_2}}{e^{N_2}}} = \frac{1}{\sqrt{2\pi}}\frac{N^{\frac{1}{2}}\,\frac{N^N}{e^N}}{\frac{{N_1}^{N_1+\frac{1}{2}}N_2^{N_2+\frac{1}{2}}}{e^{N_1+N_2}}}$$
$$\propto \frac{N^{N+\frac{1}{2}}}{{N_1}^{N_1+\frac{1}{2}}N_2^{N_2+\frac{1}{2}}} \tag{43}$$

Assuming communities 1 and 2 have the same size, (43) turns:

$$\frac{N^{N+\frac{1}{2}}}{\left(\frac{N}{2}\right)^{\frac{N+1}{2}}\left(\frac{N}{2}\right)^{\frac{N+1}{2}}} = \frac{N^{N+\frac{1}{2}}}{\left(\frac{N}{2}\right)^{N+1}} = \frac{1}{\frac{\left(\frac{N}{2}\right)^{\frac{1}{2}}}{2^{N+\frac{1}{2}}}} = \frac{2^{N+1}}{\sqrt{N}} = e^{ln\left(\frac{2^{N+1}}{\sqrt{N}}\right)} = e^{(N+1)\,ln(2)-\frac{1}{2}ln(N)} \tag{44}$$

(44) shows the number of partitions grows exponentially with N.

Community Finding

Before presenting community finding algorithms, it is reasonable to ask how feasible it would be to use a brute force approach to inspect all partitions of a network. The total number of partitions in a network with N nodes is given by the Bell Number [2]:

$$B_N = \frac{1}{e}\sum_{j=0}^{\infty}\frac{j^N}{j!} \tag{45}$$

B_N increases faster than exponentially with N (Figure 8).

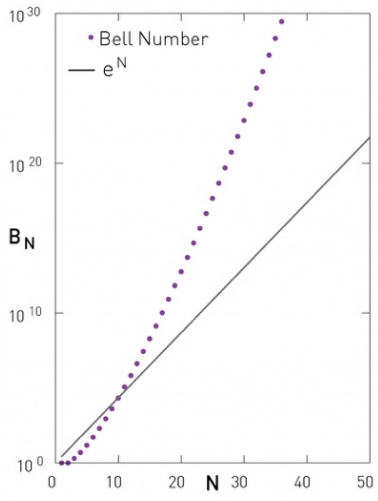

Figure 8 Evolution of the number of partitions with the size of the network *[2]* (Section 9.1).

Community finding algorithms executing in polynomial time are crucial to analyze a common size network. Nevertheless, for large networks, linear time is the ideal. Algorithms identifying separate and overlapped community structures are presented in the next section.

1.2.5. Algorithms

This section provides an overview of all kinds of community finding algorithms to be used in weighted, unweighted, directed or undirected networks.

1.2.5.1. Hierarchical Clustering

This category is subdivided in agglomerative and divisive procedures: Ravasz [22] and Girvan-Newman [17, 23] algorithms, respectively.

Ravasz Algorithm

It is divided in 4 sequential steps:

1. Definition of a Similarity Matrix

Matrix entries can represent evolutionary distances between nodes or the number of neighbors a pair of nodes has in common. In the second case, each entry is defined as:

$$x_{ij}^{o} = \frac{J(i,j)}{min\left(k_i, k_j\right) + 1 - \Theta(A_{ij})} \tag{46}$$

This implies $x_{ij}^{o} = 0$ when nodes i and j do not have neighbors in common:

$$J(i,j) = 0 \Rightarrow x_{ij}^{o} = 0 \tag{47}$$

The maximum value is obtained when both nodes are connected and have the same neighbors:

$$J(i,j) = min\left(k_i, k_j\right) \Rightarrow x_{ij}^{o} = 1 \tag{48}$$

2. Group Similarity Criteria

After joining the most similar nodes, clusters need to be compared with the remaining elements of the network (nodes/clusters). Three clustering approaches can be used:

- Single Linkage: similarity between two groups is equal to the similarity between the most similar elements;
- Complete Linkage: analogous to the previous measure but using as reference the most dissimilar nodes from each cluster;
- Average Linkage: considers the average distance of every possible pair combination in the 2 clusters.

3. Hierarchical Clustering Procedure

Having defined a similarity matrix and a similarity criterion to compare clusters, the following steps are executed:

- Assign a similarity value to every pair of nodes in the network;
- Identify the most similar community/node pair and join both. Similarity matrix is updated based on group similarity criteria;
- Second step is repeated until all nodes are in the same community.

4. Dendrogram Cut

At the end of the execution, a single tree joining all nodes is obtained – dendrogram. Although it is possible to identify the most similar nodes, it does not return the best partition of the network. In fact, the dendrogram can be cut in one out of several levels. To solve this, modularity is calculated for each partition and the one with the highest value is chosen.

Combining the four steps, the complexity of the algorithm is estimated:

- Step 1: similarity between every pair of nodes is calculated. Complexity should be $O(N^2)$, being N the number of elements in the network;
- Step 2: each community is compared against the others. This requires $O(N^2)$ calculations;
- Steps 3 and 4: using a convenient structure to represent data, in the worst-case scenario, the dendrogram can be built in $O(NlogN)$ steps.

$$O(N^2) + O(N^2) + O(NlogN) = O(N^2) \quad (49)$$

Despite this algorithm is slower than some presented in the next sections, it is significantly faster than the brute force approach which requires $O(e^N)$ operations.

Girvan-Newman Algorithm

Instead of connecting nodes based on similarity criteria, the algorithm developed by Michelle Girvan and Mark Newman removes edges based on centrality criteria. This is repeated until none remains.

1. Defining Centrality

GN algorithm identifies the pair of nodes that most likely belong to different communities and removes the link connecting them. Centrality matrix needs to be recalculated after each removal. Each entry is calculated using 2 alternative approaches:

- Link Betweenness: it is proportional to the number of shortest-paths connecting all pairs of nodes that cross the respective link. Complexity is $O(LN)$ or $O(N^2)$, in sparse networks.
- Random-Walk Betweenness: after picking random nodes m and n, a random path between those is traced. Doing it for every combination of nodes, the average number of times the link (i, j) is crossed is recorded. x_{ij} is proportional to this value. The first step of this calculation requires the inversion of a $N \times N$ matrix, thus computational complexity is $O(N^3)$. The average flowing over all pairs of nodes requires $O(LN^2)$ steps. In the case of a sparse network, the overall complexity is $O(N^3)$.

2. Hierarchical Clustering Procedure

After choosing one of the two centrality criteria:

- Calculate the centrality for every pair of nodes;
- Link with the highest centrality is removed from the network. In case of a tie, one is randomly picked;
- Centrality matrix is updated;
- Two previous steps are repeated until any link is left in the network.

3. Dendrogram

Similarly to the Ravasz, Girvan-Newman algorithm does not predict the best partition. Again, modularity is used to determine the optimal cut in the dendrogram.

Regarding the complexity of the algorithm, the limiting step is the centrality calculation. If link betweenness is chosen, the complexity is $O(LN)$. The overall complexity is obtained by multiplying the previous by the number of times the centrality matrix has to be calculated – L (until all links are removed). This means $O(L^2N)$ or $O(N^3)$ (sparse network) is the final complexity.

Respecting Ravasz and GN algorithms, it is important to ask whether hierarchical structure is really present in real networks or if the algorithms are imposing it. Are there nested modules inside bigger ones? Is it possible to assess, *a priori*, if a network has this structure?

One way to check whether hierarchical modularity is present is by analyzing the clustering coefficient:

$$C(k) \sim k^{-1} \tag{50}$$

This dependence with the node's degree lets us identify whether such pattern is present. In many real networks this phenomenon is present: scientific collaboration, metabolic and citation networks. As expected, under degree-preserved randomization, community structure disappears. Resembling Erdős-Rényi random networks, where these structures are not present.

1.2.5.2. Modularity Maximization

Based on the hypothesis a random network does not have community structure, the local modularity concept was formulated [2]. It compares the partition of a given network with the analogous degree-preserved randomization.

Considering a network with N nodes and L links, and a partition of it with n_c communities, each with N_c nodes and L_c links:

$$c = 1, \dots, n_c \tag{51}$$

Local modularity compares the number of links between the real network wiring of a subgraph C_c and the randomly rewired subgraph:

$$M_c = \frac{1}{2L} \sum_{(i,j) \in C_c} (A_{ij} - p_{ij}) \tag{52}$$

p_{ij} results from randomly rewiring the whole network, maintaining the expected degree of each node:

$$p_{ij} = \frac{k_i k_j}{2L} \tag{53}$$

If $M_c = 0$, then the subgraph is randomly wired (thus, no community structure). If $M_c < 0$, C_c has less edges than expected and should not be considered a community.

Manipulating (52), a simplified formula is obtained:

$$M_c = \frac{L_c}{L} - \left(\frac{k_c}{2L}\right)^2 \tag{54}$$

Generalizing local modularity to the whole network, the best partition of the graph is identified. This way, network's modularity becomes the sum over all communities of (54):

$$M = \sum_{c=1}^{n_c} \left[\frac{L_c}{L} - \left(\frac{k_c}{2L}\right)^2\right] \tag{55}$$

Similarly, the higher the modularity, the higher the quality of the partition of the network. It can take positive, null or negative values. Whenever the whole network is defined as one community:

$$(k_c = 2L \wedge L_c = L) \Longrightarrow M = 0 \tag{56}$$

In the extreme case each node is a single community, $L_c = 0$ and the modularity becomes negative. Consequently, none of the previous structures can be classified as a community.

Several algorithms use modularity to partition a network.

Greedy Algorithm

Greedy algorithm maximizes modularity at each step [24]:

1. At the beginning, each node belongs to a different community;
2. The pair of nodes/communities that, joined, increase modularity the most, become part of the same community. Modularity is calculated for the full network;
3. Step 2 is executed until one community remains;
4. Network partition with the higher modularity is chosen.

In terms of computational complexity, since modularity variation can be calculated in constant time, step 2 requires $O(L)$ calculations. After merging communities, the adjacency matrix of the network is updated in a worst-case scenario of $O(N)$. Each merging event is repeated N-1 times. Thus, overall complexity is: $O\big((L+N)N\big)$ or $O(N^2)$, in a sparse graph.

Although modularity is computationally suited and an accurate way of community detection, two pitfalls should be highlighted: resolution limit and modularity maxima.

Resolution Limit

First, a way to calculate network's modularity variation when communities A and B are merged is introduced [2]:

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{k_A k_B}{2L^2} \tag{57}$$

This means two communities should be joined whenever:

$$\Delta M_{AB} > 0 \Leftrightarrow \frac{l_{AB}}{L} > \frac{k_A k_B}{2L^2} \tag{58}$$

Assuming there is one link between communities A and B:

$$1 > \frac{k_A k_B}{2L} \tag{59}$$

In other words, when communities A and B are connected even only by one link, they will be merged if their size is lower than a threshold. Becoming impossible to detect communities below a certain size.

Assuming,

$$k_A \approx k_B = k \tag{60}$$

$$1 > \frac{k^2}{2L} \Leftrightarrow k < \sqrt{2L} \tag{61}$$

The resolution limit depends on the size of the network. One way to overcome this limitation is by subdividing larger communities into smaller ones and partition them.

Modularity Maxima

The fourth hypothesis discussed in *Community Hypothesis*, rely on the assumption that higher modularity implies a better partition of the network. Although, in some graphs, significantly different partitions may have similar modularity. This becomes a relevant issue when the number of nodes in the network increases. Becoming harder to separate network's best partition from the lower quality ones. In the case of the algorithms that optimize modularity this is a central issue, once they iterate until its variation is lower than an input threshold.

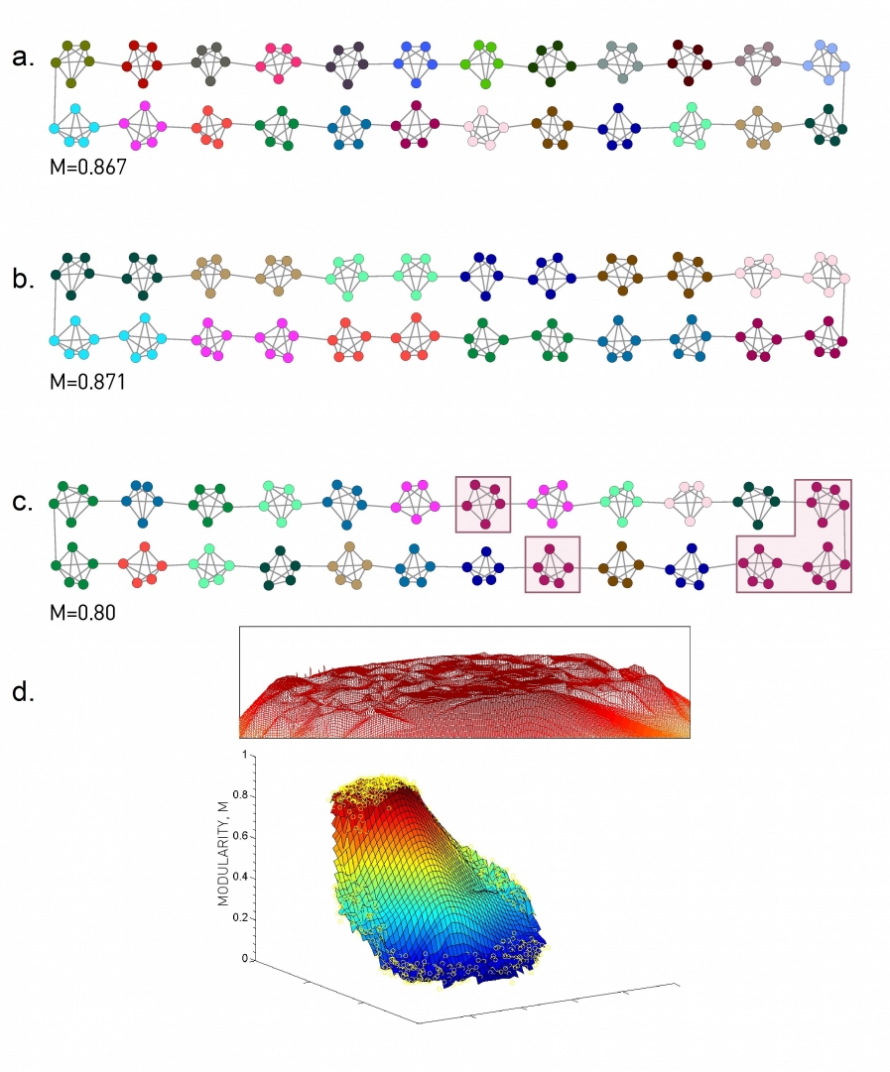

Figure 9 Significantly different partitions of the same network can have similar modularity *[2]* (Section 9.4).

Upon analyzes of the network in Figure 9, the number of links inside each cluster is approximated to:

$$k_c \approx \frac{2L}{n_c} \tag{62}$$

Considering $k_A = k_B = k_c$ and applying (57) to the previous network, modularity variation is calculated in terms of the number of nodes n_c:

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{k_A k_B}{2L^2} = \frac{l_{AB}}{L} - \frac{2}{n_c^2} \tag{63}$$

Merging two random clusters of the previous network in the same community will decrease modularity at most $\frac{2}{n_c^2}$. In the limit, this variation is undetectable:

$$\lim_{n_c \to \infty} \frac{2}{n_c^2} = 0 \tag{64}$$

Empirically, the best partition should be the one that groups each 5-node cluster in different communities. Although, modularity increases by 0.003 whenever two adjacent communities are merged. Moreover, if random

5-node clusters are assigned to a community, even if they are not directly connected, it results in a modularity variation close to 0 around the one detected for the optimal partition. This complies with the vision of the plateau in the modularity graph that may distort the choice of the best partition in Figure 9 (d). This plateau explains why a large number of modularity maximization algorithms can quickly detect high modularity partitions – they are not unique.

Modularity optimization algorithms are part of a larger set of problems that are solved by optimizing a quality function. Next, clique-based and statistical inference algorithms are presented.

1.2.5.3. Clique-Based Methods

Some nodes, in real networks, are part of multiple communities. Consequently, they cannot be correctly categorized in only one. This led to the arousal of two algorithms: *clique percolation* [25] and *link clustering* [26].

Clique Percolation Algorithm

Also called *CFinder*, it identifies communities as sets of adjacent cliques. Proceeding the following way:

1. An $N_{clique} \times N_{clique}$ matrix is created. Each entry O_{ij} is the number of shared nodes between i and j k-cliques. k is picked such that we get the richest community structure (with widely distributed cluster sizes);
2. If the number of shared nodes between 2 cliques is $k-1$, then they are joint. This step is performed in the whole network;
3. Step 2 is performed until no more adjacent cliques are possible to aggregate to the same community. Generally, maximal cliques are found implementing the following pseudocode:

Algorithm 1 Detecting Maximum Cliques

```
Require:
R: candidate vertices;
Q: current clique;
Ensure:
p: vertex in R;
Γ: neighborhood of a given vertex;
function expand (R, Q)
  repeat
    Q_p = Q ∪ {p}
    R_p = R ∩ Γ(p)
    If R_p ≠ {} then
      expand (R_p, Q_p)
    else
      return Q_p
    end if
    R = R − {p}
  until R ≠ {}
```

12: **end function**

It is important to state that even random networks may contain big connected components detectable by clique-based algorithms. To assess if a community is present by chance, the analysis of the randomly rewired network should be done.

$$p_c(k) = \frac{1}{[(k-1)N]^{\frac{1}{k-1}}} \tag{65}$$

Whenever the probability of connection between nodes exceeds p_c (which varies with the size of the network and of the clique), it is expected to observe a big component.

Finding cliques in a network is a task that grows exponentially with its size. Although, once the goal is not finding maximum cliques, but k-cliques, they can be identified in polynomial time. However, if large cliques are expected to be present, $O(e^N)$ algorithms may perform better.

Link Clustering Algorithm

Instead of clustering nodes like Ravasz algorithm, Link Clustering uses hierarchical clustering to partition the network in communities accordingly to the similarity between edges. This way, when nodes can belong to more than one community, edges are restricted to a single one. This property is usually verified in real networks.

Before introducing the algorithm, a measure of proximity between pairs of links is defined. Each entry of the similarity matrix is calculated the following way:

$$S\big((i,k),(j,k)\big) = \frac{|n_+(i) \cap n_+(j)|}{|n_+(i) \cup n_+(j)|} \tag{66}$$

$n_+(i)$ represents the number of nodes to which i is connected (including itself). S is the fraction of nodes shared by the 2 non-common nodes present in each link, among all the possibilities.

The overall procedure can be resumed as:

1. Calculation of a similarity matrix;
2. Cluster links with the highest similarity;
3. Apply single-linkage and merge communities;
4. Cut the dendrogram using, for instance, modularity.

The algorithm is divided in 2 steps. First, the calculation of a similarity matrix and, second, hierarchical clustering. Each contributes differently to the overall complexity:

- The first is dependent on the number of links present in the nodes i and j of the pairs being compared. This requires $max(k_i, k_j)$ steps. In case of a scale-free network, the complexity is $O(N^{\frac{2}{\gamma-1}})$. This estimate is determined by the size of the largest node $k_{\max}$;

- Hierarchical clustering step does not run in $O(N^2)$ (like in the Ravasz algorithm), but in $O(L^2)$ because links are being compared instead of nodes. In sparse graphs, it can be approximated to $O(N^2)$.

From the combination of the 2 previous steps, final complexity is:

$$O\left(N^{\frac{2}{\gamma-1}}\right) + O(N^2) = O(N^2) \tag{67}$$

1.2.5.4. Statistical Inference

Three approaches for community finding were already described: hierarchical clustering, modularity maximization and clique-based methods. Statistical inference is the last class of algorithms still not presented. Similarly to clique-based methods, it allows community overlapping.

Yang et al [27] proposed in 2012 a new model-based approach called Community-Affiliation Graph Model. They defined a new way of representing network's structure by relating each node to the community they belong. Each community has an associated probability of two random contained nodes being connected. Based on this model, they proposed a community finding algorithm.

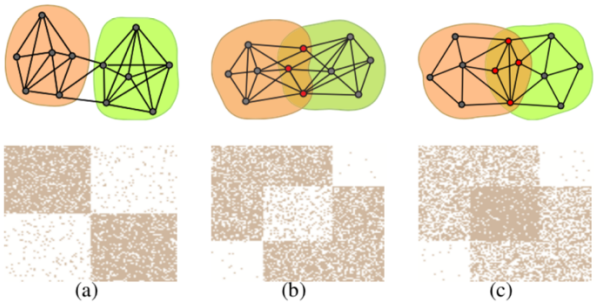

Figure 10 Networks (top) and respective adjacency matrices (bottom) *[27]*.

AGM Model

It distinguishes of any other by considering community overlapping and assuming that nodes in those areas, are more likely to be connected (Figure 10). This is an intuitive consequence of their assumption: the probability of 2 nodes being connected is equal to the sum of the probabilities of the communities they share. Before going through the mathematical basis, critical notation is introduced:

Let $B(V, C, M)$ be the bipartite graph that associates each node to the set of communities it belongs. V is the set of nodes, C contains all the communities and M englobes all the edges present in the model, such as: $(u, c) \in M$. u belongs to V and c to the set of communities C. Finally, given $B(V, C, M)$ and $\{p_c\}$, the Community-Affiliation Graph Model generates a graph $G(V, E)$ by creating an edge between u and v, with probabilities $p(u, v)$ (Figure 11). This probability is defined as:

$$p(u, v) = 1 - \sum_{k \in C_{uv}} (1 - p_k) \tag{68}$$

As predicted, $p(u,v)$ changes with the number of communities the nodes share. As higher C_{uv}, the lower $\sum_{k \in C_{uv}}(1-p_k)$ becomes and closer to 1 $p(u,v)$ gets. $C_{uv} \subset C$ and C_{uv} is the set of communities both nodes share:

$$C_{uv} = \{c | (u,c), (v,c) \in M\} \tag{69}$$

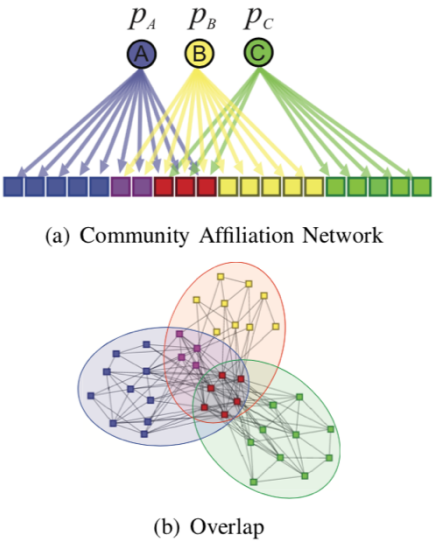

Figure 11 Community Affiliation Model (top) and the original network (bottom) *[27]*.

Algorithms discussed in the previous sections, based on the underlying assumptions, would more likely identify a community in overlapping regions or join together both clusters as a single community [2].

As a consequence of the flexibility of the model, it is possible to represent a wide number of network topologies, with non-overlapping, overlapping and nested communities (Figure 12).

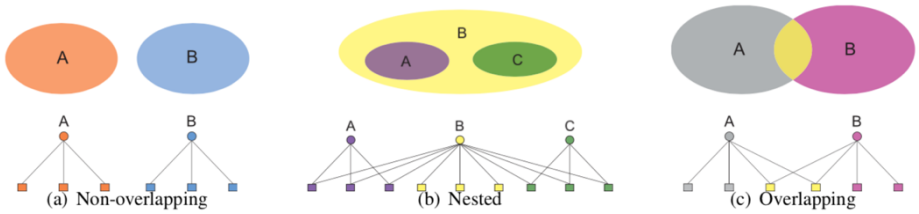

Figure 12 AGM is able to represent non-overlapped (a), nested (b) and overlapped networks (c) *[27]*.

Community Finding Algorithm

The goal is the find the best network partition by maximizing the likelihood $L(B,\{p_c\}) = p(G|B,\{p_c\})$ of the underlying graph G:

$$arg\max_{B,\{p_c\}} L(B,\{p_c\}) = \prod_{(u,v)\in E} p(u,v) \prod_{(u,v)\notin E} \big(1-p(u,v)\big) \tag{70}$$

The problem is solved by employing coordinate ascent strategy, iterating over B and $\{p_c\}$. This way, when one of the variables is updated, the other is kept fixed.

While keeping B fixed and updating $\{p_c\}$, the following optimization problem is solved (considering $0 \leq p_c \leq 1$):

$$arg\max_{\{p_c\}} \prod_{(u,v)\in E} \left(1 - \prod_{k\in C_{uv}} (1-p_k)\right) \prod_{(u,v)\notin E} \left(\prod_{k\in C_{uv}} (1-p_k)\right) \tag{71}$$

Although this problem is not convex, it can be made convex by taking the logarithm of the likelihood and proceeding to the following variable substitution:

$$e^{-x_k} = 1 - p_k \tag{72}$$

The likelihood maximization problem has now a different shape:

$$arg\max_{\{x_c\}} \sum_{(u,v)\in E} log\,(1 - e^{-\sum_{k\in C_{uv}} x_k}) - \sum_{(u,v)\notin E} \sum_{k\in C_{uv}} x_k \tag{73}$$

The resulting variable is constrained to:

$$x_c \geq 0 \tag{74}$$

A globally optimal solution can now be found applying Newton's method or gradient descent.

The last part of the optimization process focuses in finding B that maximizes the likelihood of the model. A different iterative approach is used, once B is not a quantifiable variable like $\{p_c\}$. A method called Metropolis-Hastings is used [28]. Stochastically, it updates B, whenever a certain transition maximizes the likelihood's ratio:

$$max\left(1, \frac{L(B', \{p_c\})}{L(B, \{p_c\})}\right) \tag{75}$$

Metropolis-Hastings randomly updates the bigraph using the following transitions (Figure 13):

- Leave – Considering Figure 13, node u is removed from the actual community. The final set of edges is: $M' = M \backslash \{(u, c)\}$. Assuming, $\{(u, c)\} \in M$;
- Join – Node u joins a community c. Updated version of M is: $M' = M \cup \{(u, c)\}$. Assuming, $\{(u, c)\} \notin M$;
- Switch – 1 node switches communities. This can be described as: $M' = (M \backslash \{(u, c_1)\}) \cup \{(u, c_2)\}$. Assuming, $\{(u, c_1)\} \in M$ and $\{(u, c_2)\} \notin M$.

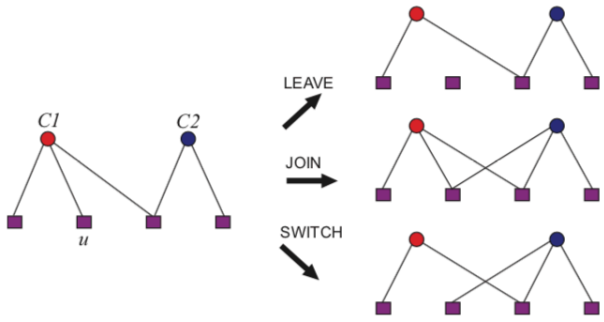

Figure 13 Metropolis-Hastings procedure to find the bigraph that maximizes the likelihood of the model *[27]*.

Resuming, a random B is generated at first place. In each iteration i, the transition is validated ($B_{i+1} = B_i'$) based on the ratio of log-likelihoods. In case the condition is not satisfied, then no update is made ($B_{i+1} = B_i$). Being B_i' the bigraph after applying Metropolis-Hastings procedure at iterate i.

Regarding the computational complexity of AGM algorithm, it is not indicated for networks with millions of nodes. The time complexity of the algorithm is $O(N^2)$ [27]. Being N the number of nodes in the network.

1.2.6. Benchmark

An effective way to benchmark algorithms is partitioning networks whose community structure is known. Girvan-Newman, Lancichinetti-Fortunato-Radicchi and Community-Affiliation benchmarking approaches are described in the thesis. The first two are detailed in the *Methodology* section, and the last one is described below. Still in this section, tools to assess the accuracy and to measure the speed of community finding algorithms are presented.

1.2.6.1. Community-Affiliation Graph Model

AGM can be used as a community finding algorithm, but also as a benchmark network generator [27]. What distinguishes this method from the other two is the ability to generate graphs accounting to community overlap.

The user starts by deciding the number of nodes and communities in the network. Then, the probabilities of two nodes inside a given community being connected by an edge are defined. In order to create an even more realistic network, it can also consider an extra community designated ε-community which adds an extra probability for nodes from different communities to connect:

$$\varepsilon = \frac{2|E|}{|V|(|V|-1)} \tag{76}$$

Based on the parameters of the model, a synthetic network is generated.

Measures to assess algorithm's partitions correctness are introduced next.

1.2.6.2. Accuracy

Normalized Mutual Information [29], Average F1 Score [27], Omega Index [30] and Accuracy in the Number of Communities [27] are commonly used to test the accuracy of community finding algorithms respecting the ground-truth communities present in the network. The first method is described in the *Methodology* section and the last three are below.

Average F_1 Score

F_1 score is defined as the average of the F_1 score of the best matching ground-truth community for each detected one, and the F_1 score of the best estimated against each ground-truth community. This can be mathematically defined as:

$$F_1 = \frac{1}{2}(\bar{F}_g + \bar{F}_d) \tag{77}$$

Where:

$$\bar{F}_g = \frac{1}{|C^*|}\sum_i max_j F_1(C_i, \hat{C}_j) \tag{78}$$

$$\bar{F}_d = \frac{1}{|\hat{C}|}\sum_j max_j F_1(C_j, \hat{C}_i) \tag{79}$$

And $F_1(C_i, \hat{C}_j)$ is the harmonic mean of precision and recall of C_i and $\hat{C}_j$:

$$F_1(C_i, \hat{C}_j) = \left(\frac{recall^{-1} + precision^{-1}}{2}\right)^{-1} = 2 \times \frac{precision \times recall}{precision + recall} \tag{80}$$

Contingency tables are usually used to represent the number of true/false negatives/positives. From those, precision and recall are calculated. The first measure is given by the ratio of true positives over the sum of true and false positives. Recall is calculated dividing the number of true positives over the sum of those with the false negatives. This statistical definition can be extrapolated to community finding context by considering the community each node is assigned, against its ground-truth.

Omega Index

It estimates the accuracy of the detected number of shared communities by a pair of nodes versus its theoretical value. It is particularly relevant when working with covers in which community overlapping is present.

$$\frac{1}{|V|^2}\sum_{u,v \in V} \mathbf{1}\{|C_{uv}| = |\hat{C}_{uv}|\} \tag{81}$$

C_{uv} and $\hat{C}_{uv}$ are, respectively, the ground-truth and the detected communities in the network. V is the total set of vertices that u and v belong.

Accuracy in the Number of Communities

It determines how the number of detected communities relates to the one found in the ground-truth partition of the network.

$$1 - \frac{||C^*| - |\hat{C}||}{|C^*|} \tag{82}$$

1.2.6.3. Speed

In terms of speed, it is expected the overall time of execution of the algorithms to vary accordingly to the computing machine and size of the network. Nevertheless, computational complexity provides us an upper bound regarding the number of steps each algorithm will have to execute.

Table 1 compares all community finding algorithms in the thesis. Analyzing Table 1, it should become clear the fastest algorithms are Louvain and Infomap. On the opposite side, *CFinder* is the least efficient.

Table 1 Community finding algorithms' time complexity.

Algorithm	Nature	Time Complexity	Reference
Ravasz	Hierarchical Agglomerative	$O(N^2)$	[22]
Girvan-Newman	Hierarchical Divisive	$O(N^2)$	[23] [17]
Greedy Modularity	Modularity Optimization	$O(N^2)$	[24]
Clique Percolation (*CFinder*)	Overlapping Communities	$Exp(N)$	[25]
Link Clustering	Hierarchical Agglomerative Overlapping Communities	$O(N^2)$	[26]
AGM	Statistical Inference Overlapping Communities	$O(N^2)$	[27]
Louvain	Modularity Optimization	$O(L)$	[31]
Infomap	Flow Optimization	$O(NlogN)$	[32]
LP	Semi-Supervised	$O(L)$	[33] [34]
LLP	Semi-Supervised	$O(L)$	[35]

1.3. Phylogenetics

The term phylogeny was used for the first time by Haeckel in 1866. Phylogenetics aims to trace an evolutionary relation between taxa. Calling it taxa, we generalize the claim to all kind of taxonomic groups, such as strains, species or populations.

Based on visual representations of descendance, named phylogenetic trees, Charles Darwin was the first to popularize these structures. Regarding its shape, the most recent taxa are represented as leaves, ancestors as internal nodes and a common ancestor in the top of the tree occupying the position of the root. In order to overcome the limited representation of the relations among nodes in phylogenetic trees, a generalized concept started to be used – phylogenetic networks. Those are particularly useful when in the presence of reticulate events: hybridization, horizontal gene transfer, recombination, gene duplication or loss.

In the 19th century, similarity between taxa was based on morphological characteristics. Nowadays, the attempt to classify organisms based on their morphology is called phenetics. With the discovery of DNA and the advent of protein and nucleic acids analysis, a variety of sequencing methods is now available. Those lead to a boom in the quantity and quality of data available to infer more precise evolutionary relations. A new field of study emerged – Molecular Phylogenetics.

1.3.1. Molecular Phylogenetics

Molecular phylogenetics uses DNA, RNA or protein sequences to infer similarity among taxa. This choice is based on the expected evolutionary time separating them and the structure best conserved over time but that still captures evolution.

Multiple phenomena can be in the origin of the evolutionary patterns present in phylogenetic trees. Tree bifurcation is accurately explained by 2 events:

Founder Effect – the migration of organisms to new and isolated regions;

Vicariance – abrupt events can be in the origin of the separation of organism belonging to the same region. Leading, just like the above, to the isolation of lineages of species, resulting in speciation.

At DNA level, taxa divergence/convergence is explained by 2 phenomena: mutation and recombination events.

Many algorithms were developed to trace and represent phylogenetic trees based on evolutionary data. In spite of the previous reference to a standard type of phylogenetic trees – with leaves, nodes and a root – there are a couple of alternative representations.

1.3.2. Data Acquisition

Data acquisition for phylogenetic analysis strongly relies on DNA sequencing. Frederick Sanger, a pioneer in protein and DNA sequencing (2 times Nobel Prize winner), developed a sequencing method that due to its reliability and easiness of execution, was used in the 1st generation of DNA sequencers. This would remain the standard sequencing method from the 1980s until middle 2000.

With the development of shotgun sequencing, the development of Next Generation Sequencing alternatives was triggered. This would lead to a row of high-throughput methods which strongly decreased the costs and time of sequencing. This made possible analyzing longer pieces of DNA, but also multiple ones simultaneously. Completely sequenced chromosomes and genomes were the result of NGS methods.

All these developments were proved to be crucial in metagenomics, medicine, forensics, molecular and evolutionary biology. Once the microorganisms in the onset of most infections (bacteria and virus) have the shortest genomes, they were the first to be sequenced. This way, microbiology had its role emphasized and its development led to the creation of new labs and genetic databases.

Some standard and replicable typing methods are now widely used to infer phylogenetic relations. The most common is Multi Locus Sequence Typing (MLST). Instead of considering the complete DNA sequence of each strain, 7 housekeeping loci are usually determined. Each locus contains around 400-500 nucleotides and belong to a gene, a regulating or a non-coding region. Progressively, more housekeeping loci are being used. The main advantage of this approach is to buffer the effect of recombination in taxa evolution. Inference algorithms based on this method do not count the number of differences between analogous segments of different strains, but a binary output of their similarity is returned. cgMLST, rMLST and wgMLST and SNP are some other techniques used to assess taxa similarity.

Depending on the scope of the analysis and data availability, one inference method can be chosen over the other. Figure 14 highlights the variation of discriminatory power based on the method. For short-term outbreak situations, SNP analysis is usually chosen.

In spite of all data and powerful sequencing techniques available, there is a lack of software to analyze it in useful time frames.

Source: https://en.wikipedia.org/wiki/DNA_sequencing, [36], [37]

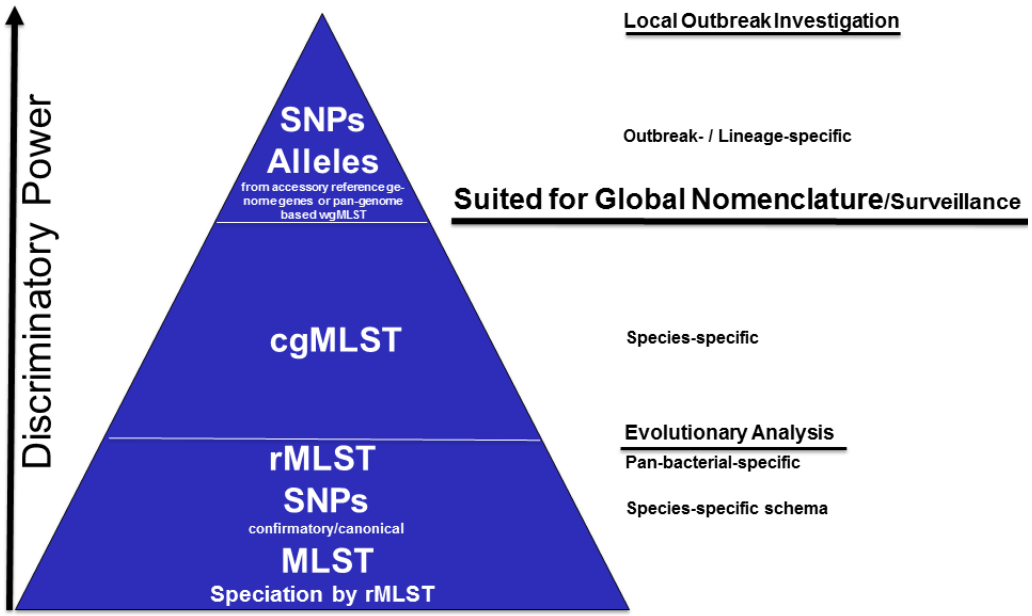

Figure 14 Discriminatory power of different typing methods *[36]*.

1.3.3. Phylogenetic Trees

In this section, a couple of algorithms to design phylogenetic networks are presented. Prim and Kruskal algorithms are 2 greedy approaches for tracing phylogenetic trees. Those are a particular type of networks in which each node only has one parent. They can be also called minimum spanning trees.

Prim's Algorithm

Prim's implementation can be described in 3 steps [38]:

- Initialize a tree with a random vertex from the initial graph;
- Grow the tree by connecting the previous vertex to another linked by an edge with the lower weight;
- Previous step is repeated until all vertices are present in the tree.

In case the initial graph is represented by an adjacency matrix, time complexity is $O(|V^2|)$.

Kruskal's Algorithm

One major difference between Kruskal's and Prim's algorithm is that the former allows us to handle a disconnected graph [39]. Based on the number of disconnected components, it will find a minimum spanning tree for each. Consequently, resulting a forest with the sum of each edge weight minimized.

In a concise way, this is how the algorithm works:

- Create a forest F where each node is considered an isolated tree;
- Create a list S with all possible edges in the network and with the respective weights associated;
- While S is not empty:
  → Remove the edge with minimum weight from S;

→ Add it to forest F if it connects 2 different trees.

In terms of its time complexity, these two algorithms are equally fast for sparse graphs, but slower than other more sophisticated ones.

Based on a distance matrix, one can use hierarchical clustering methods, Neighbor-Joining or Fitch-Margoliash methods to categorize data. The former includes: UPGMA, WPGMA, Single Linkage and Complete Linkage algorithms.

Moreover, Maximum Parsimony, eBURST, goeBURST, Maximum Likelihood and Bayesian Inference are some of the other popular methods among phylogenetics.

https://en.wikipedia.org/wiki/Phylogenetics; https://en.wikipedia.org/wiki/Phenetics;

1.3.4. PHYLOViZ

PHYLOViZ offers the user a scalable way of analyzing phylogenetic data. It is available as a cross-platform JAVA software [40] and a web application [37]. The latter includes a RESTful API that allows for programmatic access and analysis of data. This thesis focuses in the online version.

Implementation

PHYLOViZ Online is a Node.js application widely supported by most browsers [37]. It uses goeBURST [41] to build Minimum Spanning Trees for different data inputs. Plus, it allows users to exchange visual representations of the generated data and provides access to a public database with hundreds of microbial genetic profiles and respective auxiliary data.

Along with this user-friendly application, the RESTful API allows to retrieve, upload data and run available tree algorithms, from any browser or device.

Input Data Types

Due to the number of databases and sequencing data formats available, it is fundamental to allow the input of the broadest range. PHYLOViZ Online supports [37]:

- A tab-delimited file format for MLST, cgMLST, wgMLST, MLVA and SNP analyses;
- FASTA files with sequences of the same length or aligned to the same length;
- Newick files already containing the tree topology and branch lengths.

The user can also upload auxiliary data in a tab-delimited format, such as epidemiological, demographic or temporal. Some examples of these are antibiotic resistance, serotype… In the case of tab-delimited files, identical column headers in the profile data and auxiliary files relate both. For FASTA and Newick formats, identifiers from both files are searched in the first column of the metadata.

Data visualization

There are four options for visualizing data: tree visualization, table visualization, interactive distance matrix and sequence viewer [37]. The first mode is provided by VivaGraphJS JavaScript library. It renders thousands of nodes in phylogenetic trees using machine's GPU acceleration hardware, assigned to WebGL JavaScript API. DataTables library organizes the databases in dynamic tables. Matrix representation and pie charts are result of Data Driven Documents (D3.js) execution. BioJS MSA Viewer represents multiple sequence alignments of the FASTA input.

2. Related Work

Before *Methodology*, it is important to situate this thesis in the state of the art. This way, research and review components along the next sections will be now identified.

Louvain, Infomap and Layered Label Propagation algorithms were presented for the first time by *Blondel et al* in 2008 [31], *Rosvall et al* in 2009 [32] and *Raghavan et al* in 2007 [33], respectively. Several implementations, in Python and C++, were made available online by the authors. The main goal of the thesis was to implement the 3 algorithms, for the first time, in JavaScript. Louvain's implementation was based on a version available in NPM (though, it was adapted to convey phylogenetic data analysis, as well as to perform faster than the original one). Infomap was implemented by modification of Louvain so that it was able to optimize a different quality function. LLP was designed from zero.

An algorithm to generate GN synthetic networks was developed for the first time, in JavaScript, in the thesis. The original version was part of a paper published by Girvan and Newman in 2002 [17]. In 2008, Lancichinetti, Fortunato and Radicchi, using C++ programming language, implemented a second version [42]. The same implementation also generates a different class of networks called LFR networks. This was not reimplemented in the thesis, but it was used to perform benchmark testing.

Finally, all data generated was plotted with the most up-to-date and stable JavaScript versions of D3 (version 4) and Cytoscape (version 3.2.22). They are still not a standard in the web due to problems of backward compatibility.

JavaScript programming language was repeatedly emphasized before once it is becoming a web standard due to its flexibility, scalability and for being supported by many browsers and devices. Thus, it is fundamental for online data analysis.

Wrapping it all, an innovative web application was built. It allows the user to explore 3 CF algorithms, 5 widely used networks (or an uploaded one) and to visualize the results using 1 out of 3 possible big-data friendly frameworks.

3. Methodology

3.1. Community Finding

3.1.1. Louvain Algorithm

Louvain is an unsupervised algorithm (does not require the input of the number of communities nor their sizes before execution) divided in 2 phases: *Modularity Optimization* and *Community Aggregation* [31]. After the first step is completed, the second follows. Both will be executed until there are no more changes in the network and maximum modularity is achieved.

$$M = \frac{1}{2m}\sum_{i,j}(A_{ij} - p_{ij})\delta(c_i, c_j) = \frac{1}{2m}\sum_{i,j}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j) \tag{83}$$

A_{ij} is the adjacency matrix entry representing the weight of the edge connecting nodes i and j, $k_i = \sum_j A_{ij}$ is the degree of node i, c_i is the community it belongs, δ-function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise. $m = \frac{1}{2}\sum_{ij} A_{ij}$ is the sum of the weights of all edges in the graph.

Modularity Optimization

Louvain will randomly order all nodes in the network in *Modularity Optimization*. Then, one by one, it will remove and insert each node in a different community C until no significant increase in modularity (input parameter) is verified:

$$\Delta M = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{84}$$

Let Σ_{in} be the sum of the weights of the links inside C, Σ_{tot} the sum of the weights of all links to nodes in C, k_i the sum of the weights of all links incident in node i, $k_{i,in}$ the sum of the weights of links from node i to nodes in the community C and m is the sum of the weights of all edges in the graph.

One way to further improve the performance of the algorithm is by simplifying (84) and calculating ΔMm instead of the complete expression:

$$\Delta M = \frac{k_{i,in}}{m} - \frac{2\sum_{tot} k_i}{(2m)^2} \Leftrightarrow \Delta M m = k_{i,in} - \frac{\sum_{tot} k_i}{2m} \quad (85)$$

While $k_{i,in}$ and Σ_{tot} need to be calculated for each trial community, $\frac{k_i}{2m}$ is specific of the node that is being analyzed. This way, the latter expression is only recalculated when a different node is considered in *Modularity Optimization*.

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 will also generate self-loops which are the sum of all links inside a given community, before being collapsed into one node.

Thus, by clustering communities of communities after the first pass, it inherently considers the existence of a hierarchical organization in the network (Figure 15).

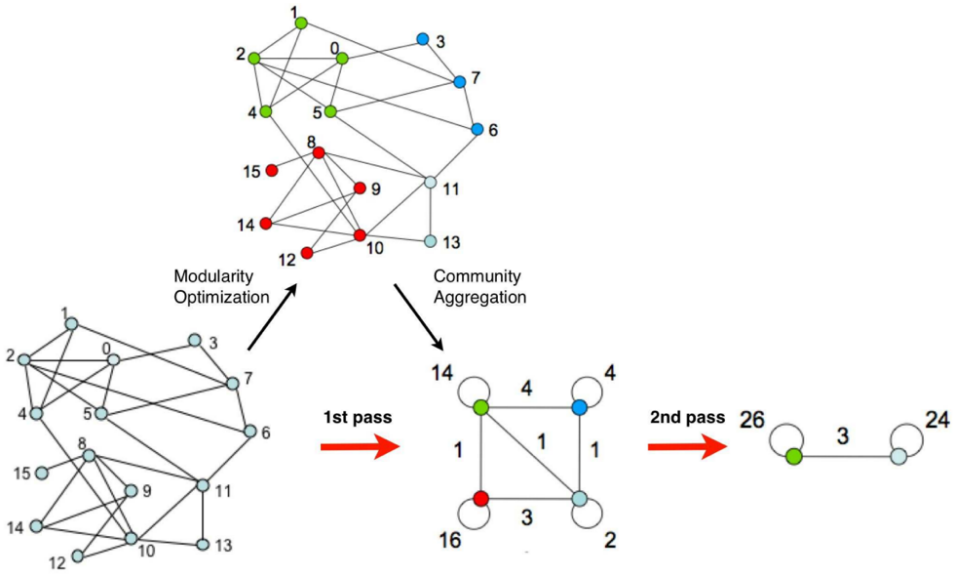

Figure 15 Sequence of steps followed by Louvain algorithm *[31]*.

Regarding computational complexity, Louvain execution is not limited for the computational time, but for storage requirements. The most demanding step is *Modularity Optimization* which time complexity scales with L. Overall, in a sparse network $N \approx L$, it is assumed to be $O(N)$. Thus, allowing the analysis of networks with millions of nodes (Algorithm 2).

Algorithm 2 Louvain

Require:
$G^0 = (V^0, E^0)$: initial undirected graph. V^0 is the initial set of vertices. E^0 is the initial set of edges;
θ: modularity improvement threshold.
Ensure:
M: resulting module;
Mod: resulting modularity;

δMod: modularity variation using Equation 4;
k_i: sum of the weight of all edges connecting node i;
$k_{i,j}$: weight of the edge connecting nodes i and j.

$m = \sum k_{i,j}, (i,j) \in E^0$
$k = 0$ // Iteration number.
repeat
  // Attributing a different community to each node.
  for all $i \in V^k$ **do**
    $M_i^k = \{i\}$
  end for
  Compute $Mod_{new} = Mod(M)$ using Equation 3
  repeat
    $Mod = Mod_{new}$
    Randomize the order of vertices.
    for all $i \in V^k$ **do**
      $best\_community = M_i^k$
      $best\_increase = 0$
      for all $M' \in C^k$ **do**
        $M_i^k = M_i^k \backslash \{i\}$
        $\Sigma_{tot}{}^{M_i^k} = \sum_{\alpha \in M_i^k} k_\alpha - k_i$; $\Sigma_{tot}{}^{M'^k_i} = \sum_{\alpha \in M'^k_i} k_\alpha + k_i$;
        $\Sigma_{in} = \Sigma_{tot}{}^{M'^k_i} - \sum k_{i,j}, (i,j) \in E^k, i \in C_i^k \; and \; j \notin C_i^k$
        $k_{i,in} = \sum_{\alpha \in M'^k_i} k_{i,\alpha}$
        If $\delta Mod_{M_i^k \to M'^k_i} > best\_increase$ then
          $best_{increase} = \delta Mod_{M_i^k \to M'^k_i}$
          $best_{com} = M'^k_i$
          $M'^k_i = M'^k_i \cup \{i\}$
        else
          $M_i^k = M_i^k \cup \{i\}$
        end else
      end for
    end for
    Compute $Mod_{new} = Mod(M)$ using Equation 3
  until No vertex movement **or** $Mod_{new} - Mod < \theta$
  // Calculate updated modularity
  Compute $Mod_{new} = Mod(M)$ using Equation 3
  If $Mod_{new} - Mod < \theta$ then
    break;
  end if
  $Mod = Mod_{new}$
  // Merge communities into a new graph
  $V^{k+1} \leftarrow C^k$
  $E^{k+1} \leftarrow e(C_u^k, C_v^k)$
  $G^{k+1} = (V^{k+1}, E^{k+1})$
  $k = k + 1$
until break

3.1.2. Infomap Algorithm

In spite Louvain and Infomap algorithms optimize a quality function, the way the network is perceived differs. While the first focuses in the structural aspect, Infomap partitioning is based on the flow induced by the pattern of connections [32].

Considering a sender pretends to communicate a random path inside a network to a receiver, the following is assumed:

The size of this message is intended to be minimized. An immediate strategy would be to attribute to each node a different name (code) and send to the receiver the corresponding sequence. The latter would be able to decodify the message based on a proper codebook. Considering the path is described in binary language by N codes of the same size, the minimum length L of each code word is:

$$L = \log_2 N \tag{86}$$

Another way to describe the same path, in a more concise way, is using Huffman coding. This approach consists in associating codes of different lengths to each node, depending on the ergodic node visit frequencies of a random path (the average node visit frequencies of an infinite-length random walk). By itself, Huffman coding is already an efficient way to transmit sporadically or discontinuously separate nodes that constitute the random path we pretend to codify. In fact, Shannon's source coding theorem states that when we use n codewords to describe n states of a random variable X, the minimum description length (MDL) is given by the entropy of the random variable itself:

$$H(X) = -\sum_{1}^{n} p_i log\ (p_i) \tag{87}$$

It is possible to make the description more efficient if the goal is to transmit the full path or significant sequences of the belonging nodes. To achieve this, the network is partitioned in modules and, for each module, a different code book is defined. This allows different nodes to be given the same identifications. Which not only implies the definition of a module codebook for each module (including an exit code), but also an index codebook to flag anytime a random walker enters in a different module. In the case of these layered descriptions, the MDL is given by a weighted average of the frequency each module and index codebooks are used:

$$L(M) = qH(Q) + \sum_{m=1}^{n_m} p_{\odot}^{m} H(P_m) \tag{88}$$

Being $q = \sum_{j=1}^{m} q_j$ the sum of the exit probability for each community, $H(Q)$ the average code length of the movement between communities, $p_{\odot}^{i} = q_i + \sum_{\beta \in i}^{m} p_\beta$ the stay probability for a random walk in a community c and $H(P_m)$ the average code length of a module codebook m.

$$H(Q) = -\sum_{i=1}^{m} \frac{q_i}{\sum_{j=1}^{m} q_j} log\left(\frac{q_i}{\sum_{j=1}^{m} q_j}\right) \tag{89}$$

$$H(P_m) = -\frac{q_i}{q_i + \sum_{\beta \in i}^{m} p_\beta} \sum_{i=1}^{m} log\left(\frac{q_i}{q_i + \sum_{\beta \in i}^{m} p_\beta}\right) - \sum_{\alpha \in i} \frac{p_\alpha}{q_i + \sum_{\beta \in i}^{m} p_\beta} log\left(\frac{p_\alpha}{q_i + \sum_{\beta \in i}^{m} p_\beta}\right) \quad (90)$$

Inserting (89) and (90) in (88):

$$L(M) = \left(\sum_{m \in M} q_m\right) log\left(\sum_{m \in M} q_m\right) - 2 \sum_{m \in M} q_m \, log(q_m) - \sum_{\alpha \in V} p_\alpha \, log(p_\alpha) + \sum_{m \in M} (q_m + \sum_{\alpha \in m} p_\alpha) \, log\left(q_m + \sum_{\alpha \in m} p_\alpha\right) \quad (91)$$

Considering an unweighted and undirected network, q_m is the exit probability of a module m and p_α the relative weight w_α that is computed dividing the total weight of the edges connected to α by twice the total weight of all links in the graph.

This way, by compressing the description length of the network, this is partitioned in modules which reflect the dynamics of the network. This is particularly useful when the connections between nodes represent a flow of a given quantity and not only the similarity between those (Algorithm 3).

Infomap approach clarifies how community finding algorithms can be also used in compression problems.

Algorithm 3 Infomap

Require:
$G^0 = (V^0, E^0)$: initial undirected graph. V^0 is the initial set of vertices.
E^0 is the initial set of edges;
θ: quality improvement threshold.
Ensure:
M: resulting module;
L: resulting MDL;
δL: MDL variation;
w_u: sum of the weight of all edges connecting node u;
$w_{u,v}$: weight of the edge connecting nodes u and v.

$w = 0$ // Iteration number.
repeat
  for all $u \in V^k$ **do**
    $p_u = \frac{degree(u)}{|E^k|}$
  end for
// Attributing a different community to each node.
  for all $u \in V^k$ **do**
    $M_u^k = \{u\}$
    $p^{M_u^k} = \sum_{\alpha \in M_u^k} p_\alpha$
    $q^{M_u^k} = \sum w_{u,v}, (u,v) \in E^k, u \in C_u^k \; and \; v \notin C_u^k$
  end for
  Compute $L_{new} = L(M)$ using Equation 5
  Repeat
    $L = L_{new}$
    Randomize the order of vertices.
    for all $u \in V^k$ **do**

If $M'^k_u = argmin\left(\delta L_{M^k_u \to M'^k_u}\right) < 0$ then
$M^k_u = M^k_u \backslash \{u\}$; $M'^k_u = M'^k_u \cup \{u\}$
$p^{M^k_u} = \sum_{\alpha \in M^k_u} p_\alpha - p_u$; $p^{M'^k_u} = \sum_{\alpha \in M'^k_u} p_\alpha + p_u$
update $q^{M^k_u}$; update $q^{M'^k_u}$
end if
end for
Compute $L_{new} = L(M)$ using Equation 5
until No vertex movement or $L - L_{new} < \theta$
// Calculate updated modularity
Compute $L_{new} = L(M)$ using Equation 3
If $L - L_{new} < \theta$ then
break;
end if
$L = L_{new}$
// Merge communities into a new graph
$V^{k+1} \leftarrow C^k$
$E^{k+1} \leftarrow e(C^k_u, C^k_v)$
$G^{k+1} = (V^{k+1}, E^{k+1})$
$k = k + 1$
until break

3.1.3. Layered Label Propagation Algorithm

Layered Label Propagation algorithm [35] development was strongly based in the old Label Propagation [33]. The latter starts by assigning a different community to each node present in the network. Then, at the beginning of each round, the order of the nodes is randomized. In each round, every node will be assigned the community more represented among the immediate neighbors. LP finishes as soon as no changing in communities happens or after reaching a given number of iterations. It does not depend on the optimization of any objective function and it does not require prior information regarding the communities present in the network (unsupervised). Although, it can be input (semi-supervised) with an approximate set. By definition, this algorithm is inherently local, (approximately) linear in the number of edges and requires few passes in the graph. These are fundamental features to allow the analysis on real networks. The problem of finding an optimal partition has been proven to be equivalent to find local minima of the Hamiltonian for a kinetic Potts model [35]. Although the problem has a trivial globally optimal solution that considers the whole network as a single cluster, it is not of our interest. Label Propagation avoids such solution because of the inherent dynamics of local search. While this is normally considered a drawback, in this case it is a fundamental feature of LP algorithm to work.

In spite of using a similar approach, LLP not only takes into account the nodes in the neighborhood, but also the ones in the remaining network. The iterative process is the same, the difference resides in the value it optimizes. In this case, the assigned community will be the one that maximizes:

$$k_i - \gamma(v_i - k_i) \tag{92}$$

Being k_i the number of nodes with label λ_i in the neighborhood of a given node and v_i the total number in the whole graph labeled the same way. γ is the resolution parameter that buffers the number of nodes belonging to a given community in the neighborhood, by its usual presence across the whole network. Whenever it approaches

0, LLP reduces to LP. It is expected that for low values of γ, the algorithm will highlight few, big and sparse clusters. For higher values, clusters become smaller, denser and increase their number. It is relevant to say that there is no *a priori* measure to state which value could be better for the network under analysis (Algorithm 4).

Similarly to LLP, other algorithms have been proposed that execute the same sequence of steps but maximizing different values like modularity.

In all the previous community finding algorithms, parallel implementations, making using of multiple processors, can significantly speed up their execution. Nevertheless, this will not be considered along the thesis.

Algorithm 4 Layered Label Propagation

Require:
$G^0 = (V^0, E^0)$: initial undirected graph. V^0 is the initial set of vertices. E^0 is the initial set of edges;
$max_{iteration}$: maximum iteration number;
γ: resolution parameter.
Ensure:
M: resulting module;
k_i: number of neighbors having label λ_i;
v_i: overall number of nodes having label λ_i.

$k = 0$ // Iteration number.
// Attributing a different community to each node.
for all $i \in V^0$ **do**
  $M_i^0 = \{i\}$
end for
repeat
  Randomize the order of vertices.
  for all $u \in V^k$ **do**
    for all $i \in labels(u)$ **do**
      If $k_i(u) - \gamma(v_i(u) - k_i(u)) > k_{i-1}(u) - \gamma(v_{i-1}(u) - k_{i-1}(u))$ **then**
        $M_u^k = M_u^k \backslash \{u\}$; $M'^k_u = M'^k_u \cup \{u\}$
      end if
    end for
  end for
  $k = k + 1$
until No vertex movement **or** $k < max_{iteration}$

3.2. Benchmark

Although a network structure can be completely defined by its wiring diagram, different algorithms (based on distinct assumptions) will partition the same graph in different ways. Consequently, it is important to understand which algorithms are the most reliable.

An accurate way to gain this insight is by partitioning networks whose community structure is already known. GN and LFR benchmarking techniques have been giving an important contribution to the topic.

All benchmark tests were repeated 10 times. The mean values and corresponding confidence intervals at 95% were registered. These 2 types of synthetic networks were used to benchmark community finding algorithms, in terms of accuracy and speed.

3.2.1. GN Network

Historically, GN was the first synthetic benchmark network to be designed. In their paper, Girvan and Newman described it as a set of 128 nodes divided in 4 groups, each one with exactly 32 nodes [17]. Then, a mixing parameter, chosen by the user, defines the probability of connecting nodes in different communities. This probability is given by the value μ.

$$\mu = \frac{k^{ext}}{k^{int} + k^{ext}} \tag{93}$$

The authors also considered each node to be connected to, exactly, 16 different ones (Figure 16). The algorithm hereby implemented not only allows the mixing parameter to vary, but also the average degree (Algorithm 5). This led to an increased number of benchmark tests using these networks.

Algorithm 5 GN Benchmark

Require:
mix: mixing parameter.
deg: degree of each node.
Ensure:
E: resulting set of edges.
V: resulting set of nodes.
A: adjacency matrix.
T: resulting set of temporary selectable nodes.
// Generating nodes of GN network.
$k = 0$; // Iteration number.
$V = \{\}$;
repeat
  If $k < 32$ **then**
    $V = V \cup \{id: k, group: 1\}$
  else if $k < 64$
    $V = V \cup \{id: k, group: 2\}$
  else if $k < 96$
    $V = V \cup \{id: k, group: 3\}$
  else
    $V = V \cup \{id: k, group: 4\}$
  end if
  $k = k + 1$
until $k < 128$
// Generating internal edges to each group of GN network.
Randomize the order of the set of vertices V maintaining them ordered by blocks of ground-truth communities.
$i = 0$;
$A^{initial} = [0]_{128\times128}$
$E^{initial} = \{\}$;
repeat

$T = \{\}$;
$j = 0$;
repeat
If $V_j.group === V_i.group$ then
$T = T \cup \{T_j\}$
end if
$j = j + 1$
until $j < 128$
$j = 0$;
loop:
repeat
$j = round(rand * (T.length - 1))$ // Generates random position in T array.
repeat
$T = T \backslash \{T_j\}$
If $T === \{\}$ then
break loop;
end if
$j = round(rand * (T.length - 1))$
until $A^{initial}_{(V_i.id)(T_j.id)} === 1$ or $degree(T_j) \geq deg \times (1 - mix)$
$E^{initial} = E^{initial} \cup \{source: V_i.id, target: T_j.id, value: 1\}$
$A_{(V_i.id)(T_j.id)} = 1$; $A_{(T_j.id)(V_i.id)} = 1$
until $degree(V_i) < deg \times (1 - mix)$
If $(i + 1)\%32 === 0$ then
$aux = 0$; // Auxiliary variable.
$k = 0$;
repeat
If $V_i.group === V_k.group$ then
$aux = aux + (deg \times (1 - mix) - degree(V_k))$;
end if
$k = k + 1$
until $k < 128$
If $aux > 0$ then
$i = i - 32$;
else
$E^{final} = E^{final} \cup E^{initial}$; $A^{final}_{(V_i.id)(T_j.id)} = A^{final}_{(V_i.id)(T_j.id)} + A^{initial}_{(V_i.id)(T_j.id)}$
end if
$E^{initial} = \{\}$;
$A^{initial}_{(V_i.id)(T_j.id)} = [0]_{128\times128}$;
end if
$i = i + 1$;
until $index < 128$
// Generating external edges to each group of GN network.
Randomize the order of the set of vertices V.
$i = 0$;
$A^{initial2} = [0]_{128\times128}$
$E^{initial2} = \{\}$;
repeat
$T = \{\}$;

$j = 0$;
repeat
If $V_j.group$! == $V_i.group$ then
$T = T \cup \{T_j\}$
end if
$j = j + 1$
until $j < 128$
$j = 0$;
loop:
repeat
$j = round(rand * (T.length - 1))$ // Generates random position in T array.
repeat
$T = T \backslash \{T_j\}$
If T === {} then
$i = -1$;
$A^{initial2} = [0]_{128\times128}$
$E^{initial2} = \{\}$;
break loop;
end if
$j = round(rand * (T.length - 1))$
until $A^{initial2}_{(V_i.id)(T_j.id)}$ === 1 or $degree(T_j) \geq deg \times mix$
$E^{initial2} = E^{initial2} \cup \{source: V_i.id, target: T_j.id, value: 1\}$
$A_{(V_i.id)(T_j.id)} = 1$; $A_{(T_j.id)(V_i.id)} = 1$
until $degree(V_i) < deg \times mix$
$A^{final}_{(V_i.id)(T_j.id)} = A^{final}_{(V_i.id)(T_j.id)} + A^{initial2}_{(V_i.id)(T_j.id)}$
$E^{final} = E^{final} \cup E^{initial2}$
$i = i + 1$;
until $index < 128$

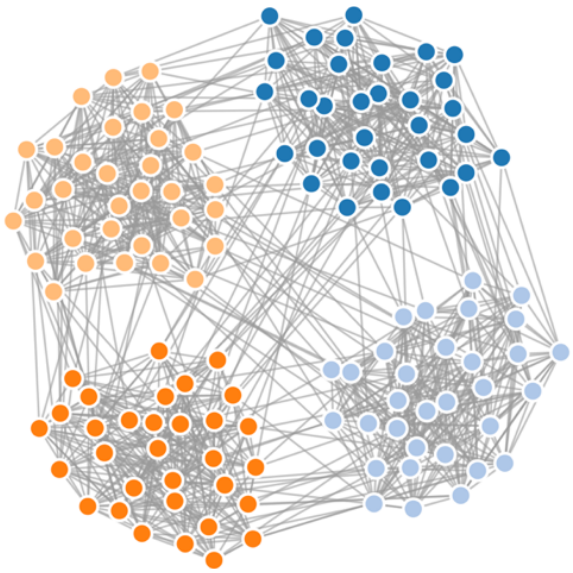

Figure 16 GN synthetic network. $N = 128$, $\mu = 0.1$ and $k = 16$.

Obtained using D3.js and SVG.

3.2.2. LFR Network

LFR benchmark networks [43] [17] try to approximate the real ones. This means that not only they consider a power-law distribution of community sizes with coefficient ζ (94), but also a power-law distribution of node degrees with coefficient γ (95):

$$p_{N_c} \sim {N_c}^{-\zeta} \tag{94}$$

$$p_k \sim k^{-\gamma} \tag{95}$$

This technique starts by generating N isolated nodes. Then, each one is assigned to a different community such that following the correct size distribution of the network. Based on the nodes' degree distribution, a degree k is attributed to each node. Next, each one will be assigned with $(1-\mu)k_i$ internal edges and the remaining of the connections will be linked to nodes from outside the respective cluster. Finally, all links (based on nodes' internal and external degrees) are determined randomly (Figure 17).

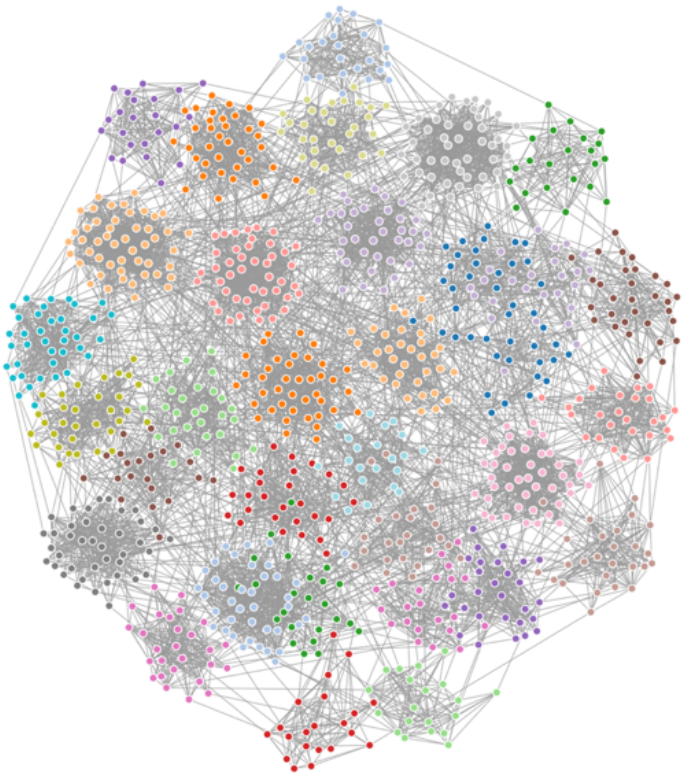

Figure 17 LFR synthetic network. $N = 1000$, $\mu = 0.1$, $k_{avg} = 15$, $k_{max} = 50$, $c_{min} = 20$ and $c_{max} = 50$. Obtained using D3.js and SVG.

3.2.3. Normalized Mutual Information

NMI is a variant of a common measure in information theory called mutual information. Mutual information accounts to the "amount of information" one can extract from a distribution regarding a second one (Figure 18). In the case of discrete distributions, mutual information of 2 jointly random variable X and Y is calculated as a double sum:

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p_{(X,Y)}(x,y)\log\left(\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)}\right) \tag{96}$$

Upon observation of (96), if X and Y are independent random variables, then:

$$\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)} = 1 \Longrightarrow \log\left(\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)}\right) = 0 \Longrightarrow I = 0 \tag{97}$$

A set of properties of mutual information result from definition (96).

Nonnegativity

Using Jensen's inequality one can show [44]:

$$I(X,Y) \geq 0 \tag{98}$$

Symmetry

By definition, $p_{(X,Y)}(x,y)$ is symmetrical. This implies:

$$I(X,Y) = I(Y,X) \tag{99}$$

Conditional and Joint Entropy Dependence

$$\begin{aligned}
I(X;Y) &= \sum_{x\in X,y\in Y} p_{(X,Y)}(x,y)\log\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)} \\
&= \sum_{x\in X,y\in Y} p_{(X,Y)}(x,y)\log\frac{p_{(X,Y)}(x,y)}{p_X(x)} - \sum_{x\in X,y\in Y} p_{(X,Y)}(x,y)\log p_Y(y) \\
&= \sum_{x\in X,y\in Y} p_X(x,y)p_{Y|X=x}(y)\log p_{Y|X=x}(y) - \sum_{x\in X,y\in Y} p_{(X,Y)}(x,y)\log p_Y(y) \\
&= \sum_{x\in X} p_X(x,y)\left(\sum_{y\in Y} p_{Y|X=x}(y)\log p_{Y|X=x}(y)\right) \\
&- \sum_{y\in Y}\left(\sum_{x\in X} p_{(X,Y)}(x,y)\right)\log p_Y(y) \\
&= -\sum_{x\in X} p_X(x)H(Y|X=x) - \sum_{y\in Y} p_Y(y)\log p_Y(y) = -H(Y|X) + H(Y) \\
&= H(Y) - H(Y|X)
\end{aligned} \tag{100}$$

See Figure 19.

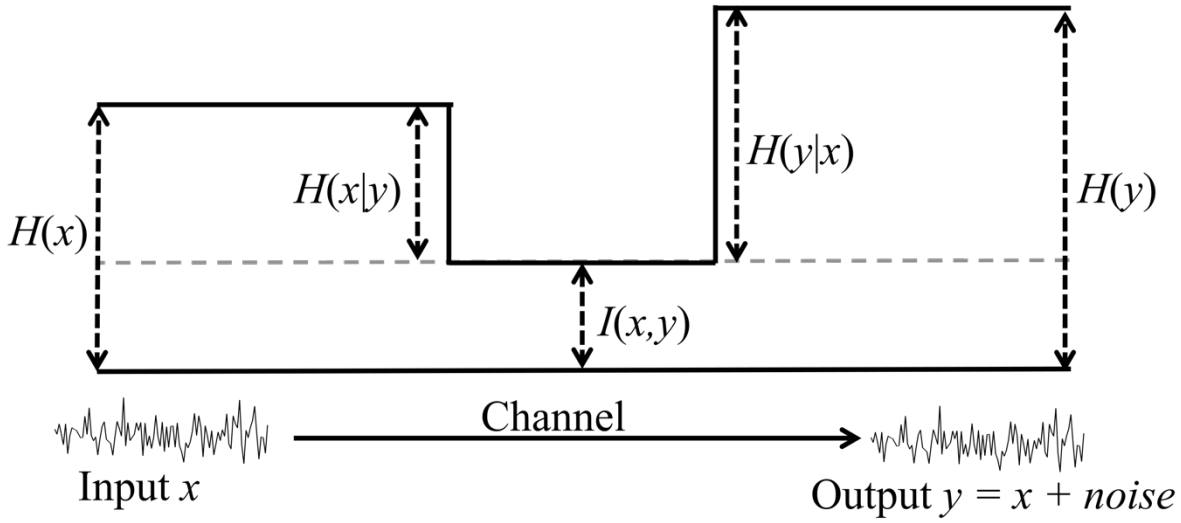

Figure 18 Analogy between signal transmission and the problem of community finding in networks with ground-truth modules *[45]*.

Clustering quality of the community finding algorithms was tested using a normalized measure of mutual information – NMI [29]. An algorithm was developed, in JavaScript, in the thesis and tested in several networks. It receives an input of 2 arrays, with the same length, and returns the corresponding normalized mutual information.

$$NMI(Y, C) = \frac{2 \times I(Y; C)}{H(Y) + H(C)} \tag{101}$$

NMI not only depends on the mutual information I, but also on the entropy of the labeled $\mathrm{H(Y)}$ and clustered set $\mathrm{H(C)}$.

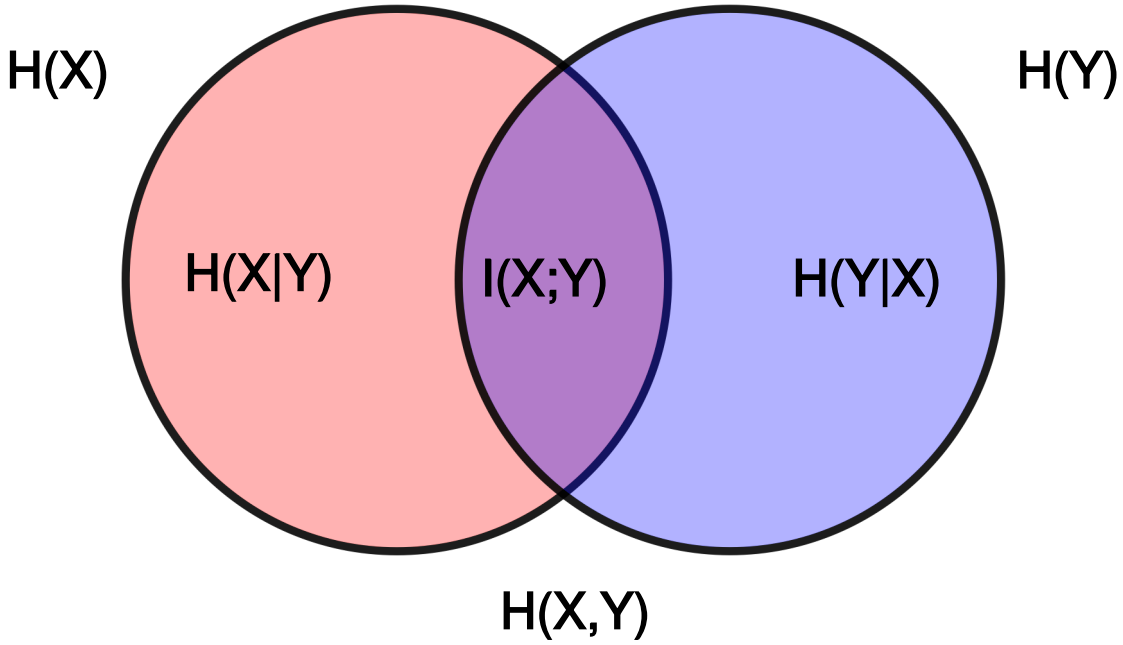

Figure 19 Venn diagram depicting the relation between different measures of entropy and mutual information *[45]*.

3.3. Test Data

The higher the number of tests performed using the implemented algorithms, the higher the certainty they are working properly for the broadest number of networks. This way, disconnected networks (sampled from a real Amazon network), small sized ones (Zachary's Karate Club) and a biological/phylogenetic network (*Staphylococcus aureus*) were additionally considered to validate algorithm's correctness.

3.3.1. Amazon Network

This network was obtained after crawling Amazon website [46]. Each node represents a given product available in the store and a connection between 2 is present whenever they are frequently bought together. The original network contains 334 863 nodes and 925 872 links. Only the first 5 000 links along the respective nodes were used in the tests (Figure 20).

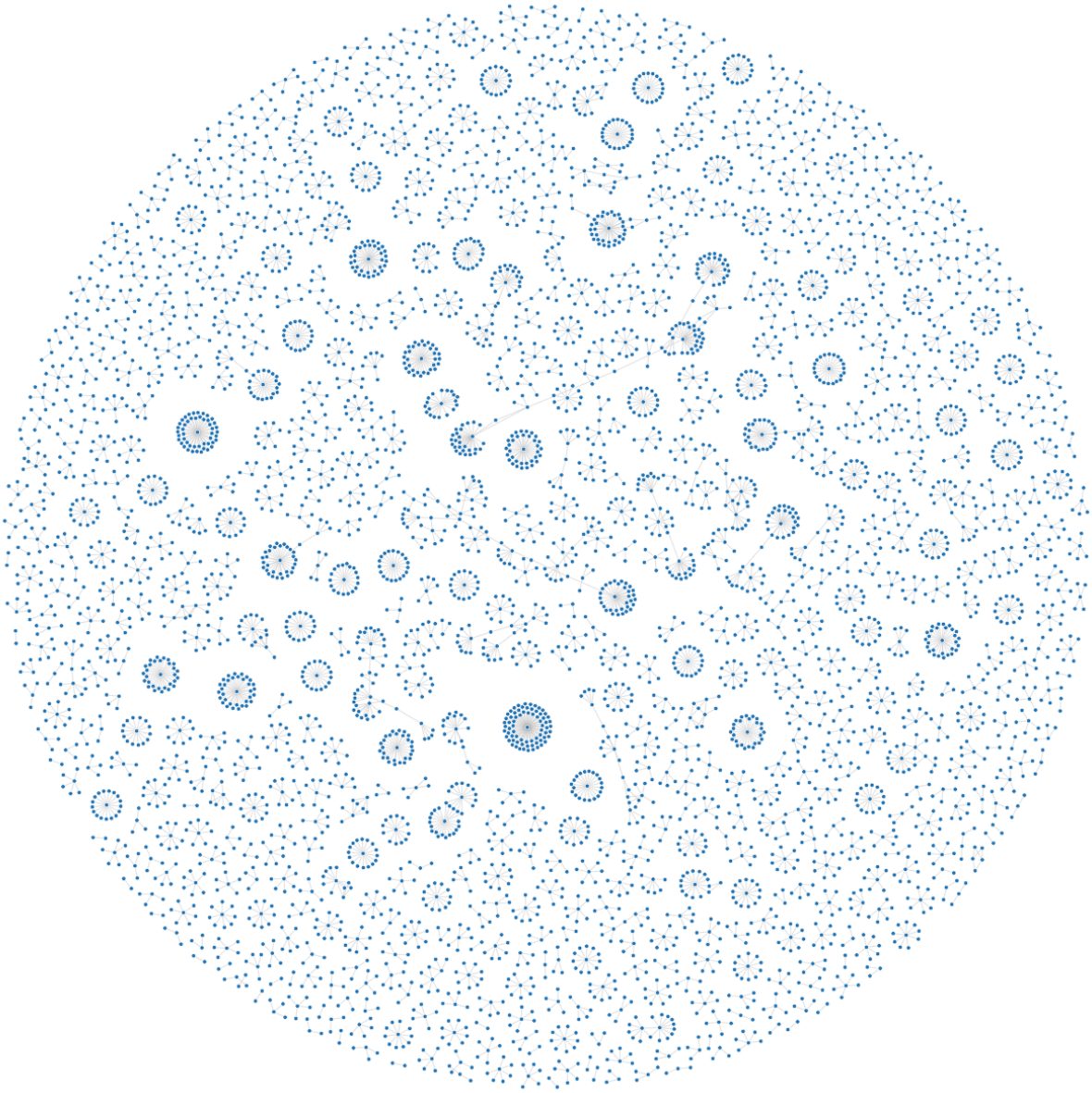

Figure 20 Amazon product co-purchasing network sampled from a 5 000 links graph. Obtained using D3.js and SVG.

3.3.2. Zachary's Karate Club Network

One of the most popular networks in network science is the Zachary's Karate Club [47]. It became popular after have being used, in 2002, in a paper of Michelle Girvan and Mark Newman [17]. This is a social network representing the 77 relations between 34 individuals from a university karate club.

Wayne W. Zachary studied this network for 3 years, from 1970 to 1972. During their study, a conflict arose and the club split in 2. Half the members formed a new a club with the instructor and the other half found a new instructor or gave up karate.

Each node is a member. Node 1 is the instructor and 34 is the president (Figure 21). Dark and light blue nodes represent the split inside the club that was created due to a conflict between them. Each link connects 2 members when they meet regularly outside the club.

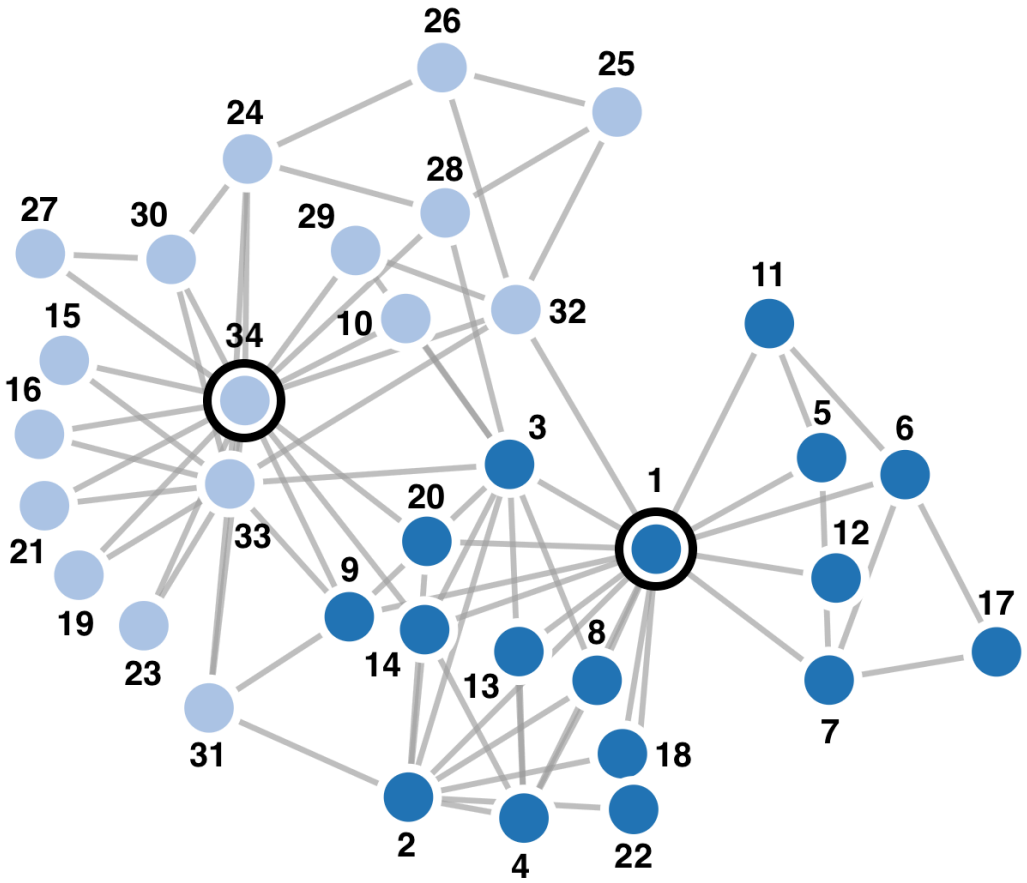

Figure 21 Zachary's karate club network. Obtained using D3.js and SVG.

3.3.3. Staphylococcus aureus

Last network used in the thesis is a microbiological sample of *Staphylococcus aureus* (Figure 22). The profile contains 7 *loci* (*arcC*, *aroE*, *glpF*, *gmk*, *pta*, *tpi* and *yqiL*) and 5199 strains. This network was partitioned in components and an SLV modified version was used to perform community finding.

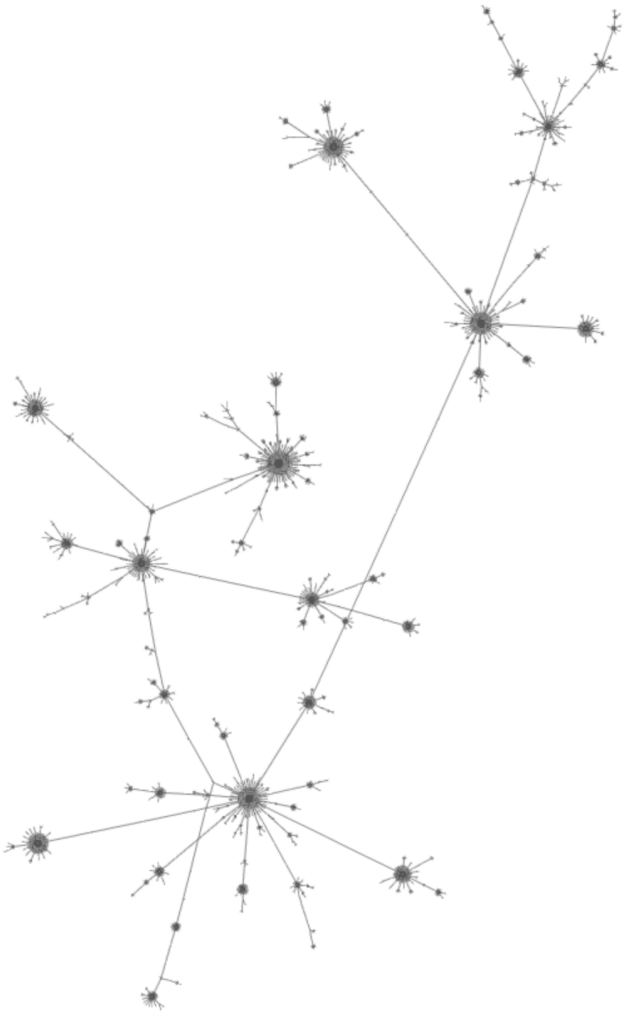

Figure 22 Minimum spanning tree generated using goeBURST implemented in *PHYLOViZ Online*.

3.4. Visualization

The most notable examples of older browser visualization frameworks are Prefuse, Flare and Protovis toolkits. The first was created in 2005 and required the use of Java, as well as a Java plug-in to allow rendering in the browser. Later, in 2007, Flare start becoming the web standard. It used ActionScript and required a Flash plugin for rendering. In 2009, a group from Stanford University created Protovis. This JavaScript library was able to generate SVGs from data sets. The same team, in 2011, interrupted Protovis development and focused in a new project called D3.js. Their project, a few years later, become the web standard to dynamically and efficiently represent data in every modern browser. D3.js not only avoids a monolithic implementation, turning data representation more flexible, but presents enhanced performances by making use of the most modern browser technologies: HTML5, CSS3 and JavaScript. Comparing with DOM standards, it uses a less verbose language and still allows the user to seamlessly interact with every element present in the webpage.

In 2015, a JavaScript library called Cytoscape.js was launched. It was the predecessor of the Java based Cytoscape, released in 2002. Cytoscape.js focuses not only in representing graphs but also in providing the user the tools to do analyze them.

Along with D3.js, Cytoscape.js performs seamlessly in the most modern browsers and in all devices. It does support a wide variety of graph properties: directed, undirected, loops… Although, it is not possible to use it for other applications apart from analyzing or representing networks.

In the thesis, the user can opt by visualizing every network using D3 or Cytoscape JavaScript libraries. In the case of the first framework, there is the possibility of visualizing data through a Canvas or an SVG element. All networks were plotted using a force directed implementation. This mode requires the user to define features like gravity, repulsion or spring constant which will affect how the network is displayed. Depending on the chosen parameters, node and link overlapping should be reduced in order to get an enhanced view of the graph (Figure 23).

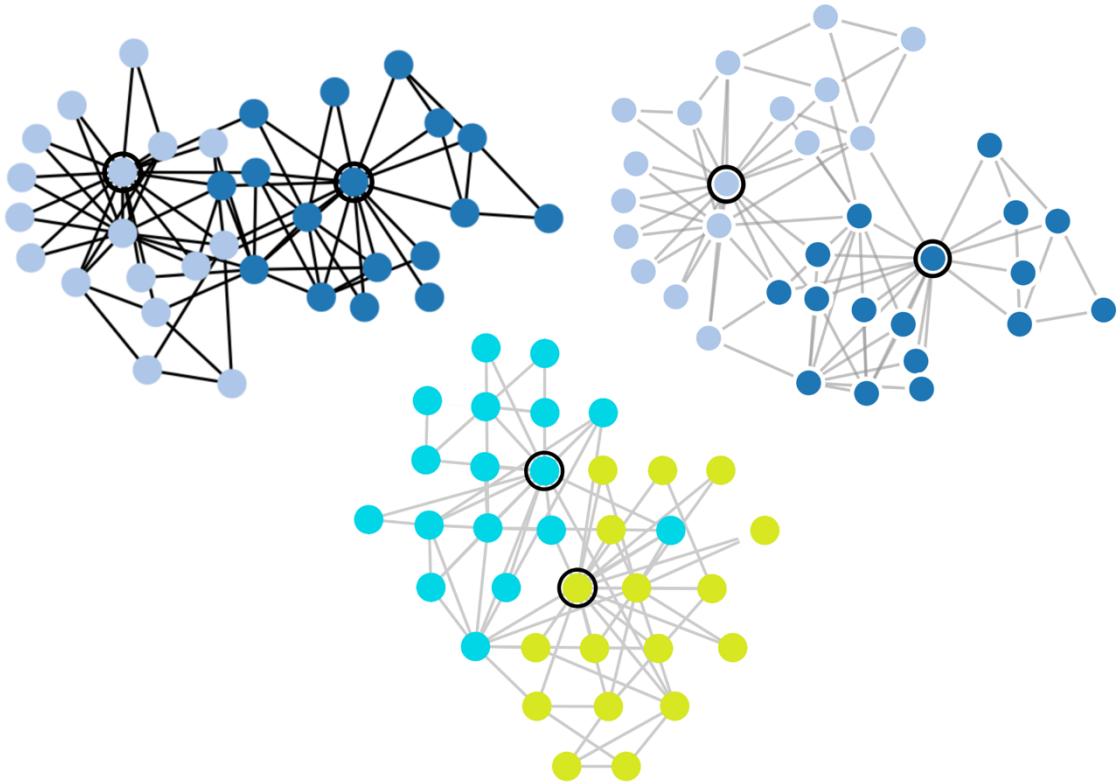

Figure 23 Zachary's Karate Club network represented using D3.js Canvas (top-left), D3.js SVG (top-right) and Cytoscape.js (bottom).

Regarding benchmark plotting, D3.js was selected. Its flexibility in terms of visual representation, performance and the number of practical examples available on the web, determined this choice.

3.5. Phylogenetics

After implementing the algorithms, test and benchmark them, phylogenetic data – *Staphylococcus aureus* MLST profile (Figure 24), was analyzed.

ST	arcC	aroE	glpF	gmk	pta	tpi	yqiL
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	26
3	1	1	1	9	1	1	12
4	10	10	8	6	10	3	2
5	1	4	1	4	12	1	10
6	12	4	1	4	12	1	3
7	5	4	1	4	4	6	3
...							

Figure 24 *Staphylococcus aureus* MLST profile.

1. *PHYLOViZ 2* was firstly installed. A MLST profile of *Staphylococcus aureus*, along with the corresponding metadata file (Figure 25), was then uploaded to the platform.

ST
1
2
3
4
5
6
7
...

Figure 25 Isolate data metafile.

A minimum spanning tree (Figure 26) was generated using *goeBURST* and following all tiebreak rules. *PHYLOViZ 2* prints a set of clonal complexes which were defined based on the similarity among STs;

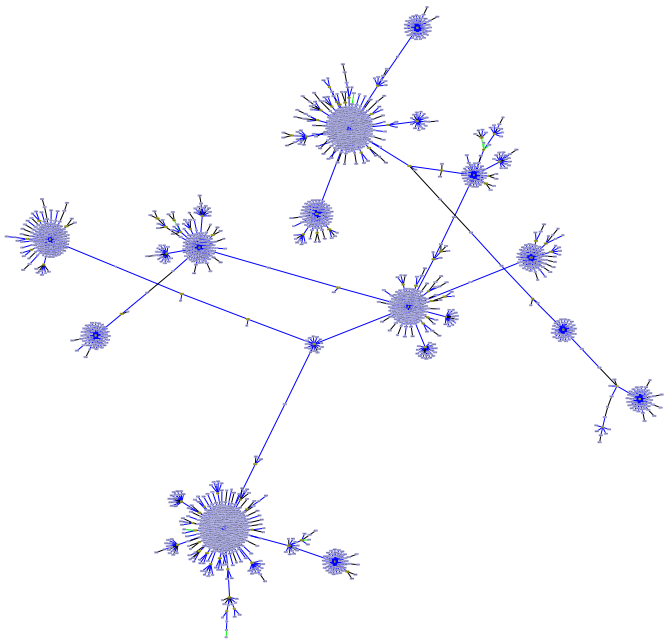

Figure 26 Minimum spanning tree of *Staphylococcus aureus* generated using *PHYLOViZ 2*.

2. Copy-paste operation was manually performed to send CC data from *PHYLOViZ 2* to a text file. Data format (Figure 27) is the same as provided by the Java tool;

```
CC 0 has 2209 STs:
ST 1 (1) 184 100 189 1735 (184 115 210 4684)
ST 4768 (1) 34 199 393 1582 (34 205 421 4533)
ST 2128 (1) 28 179 214 1787 (28 184 240 4741)
ST 1220 (1) 27 178 208 1795 (27 181 230 4755)
ST 1909 (1) 27 178 208 1795 (27 181 230 4755)
ST 1949 (1) 27 178 208 1795 (27 181 230 4755)
ST 2180 (1) 27 178 208 1795 (27 181 230 4755)
...
```

Figure 27 Data from the CC 0 of the *Staphylococcus aureus* network obtained using *PHYLOViZ 2*.

3. This file is then uploaded to *Phyl* using the appropriate button in the interface. Original MLST data profile should be also uploaded;

4. The web application will build an SLV network with all the strains present in the CC under study and considering *loci* information in the profile file;

5. Community finding algorithms were executed in the previous network;

6. The input metadata file is now returned with an extra column where every determined community is represented (Figure 28). For all nodes that were not part of the CC being studied, the same random number was attributed (accounting that it should be different than any other already used for strains in the CC).

```
ST      Community
1       0
2       5200
3       1
4       5200
5       2
6       3
7       5200
...
```

Figure 28 Final text file generated upon execution of *Phyl*. It contains an additional *Community* column, not present in the initial metadata file.

4. Results

4.1. Web Application

In order to facilitate the analyzes and visualization of the results after the execution of the algorithms, a web application was designed (Figure 29).

Its backend was conceived using Node.js and it runs in a Heroku server. An image of the app is available at Docker Hub. The following actions are strictly executed in the cloud:

- Generation of GN and LFR benchmark networks. The latter are generated using a binary from a C++ implementation [42]. This was possible after importing a package from NPM which simulates the command line in the application;
- Amazon [46], Zachary's Karate Club [47] and *Staphylococcus aureus* testing networks had their data properly processed in the cloud, so that it could be sent and displayed in the graphical interface;
- Once the execution of Louvain, Infomap and LLP algorithms is not performed in the user's browser, this may raise some privacy issues. Although, it was not feasible to run Infomap in the user side and still maintain the web application functional in all devices;
- The communication between *Phyl* and NPM was established in the server, so that it is possible to import and plot the statistics from each package uploaded to NPM in the application;
- Finally, all benchmarking data was obtained after running 10 times for each configuration, the algorithms in the cloud.

The frontend is both static and dynamic. The static counterpart uses HTML and CSS. On the other hand, the dynamic execution is coordinated by JavaScript.

Several sections were individualized in the interface:

- *Abstract* contains a concise description of the goals of the thesis;
- *Algorithms* shortly explains the core functioning of Louvain, Infomap and LLP;
- *Implementation* allows the user to actively test all the community finding algorithms against 5 different networks – GN, LFR, Amazon or Zachary's Karate Club or Staph. Using any of 3 available visualization options – D3.js (Canvas), D3.js (SVG) and Cytoscape.js;
- *Results* joins in a single section all data obtained from a thorough analysis of every algorithm implemented. It dynamically represents data, so that one can analyze it individually and in a clearer way: considering one line at a time with CI 95% error bars. The same plot can be visualized considering GN or LFR benchmark networks by selection of the respective radio button. On the top of each plot, both axles

are identified following the rule: (y-axis) x (x-axis). At the end of each line, additional information is provided in order to distinguish the curves. Under each x-axis, the user has the possibility to download the CSV file with the data used in each plot;

- *Web tools* contain a small description of 2 tools that complement *Phyl*;
- *Events* describes 4 contests/workshops which were crucial for the conclusion of this thesis.

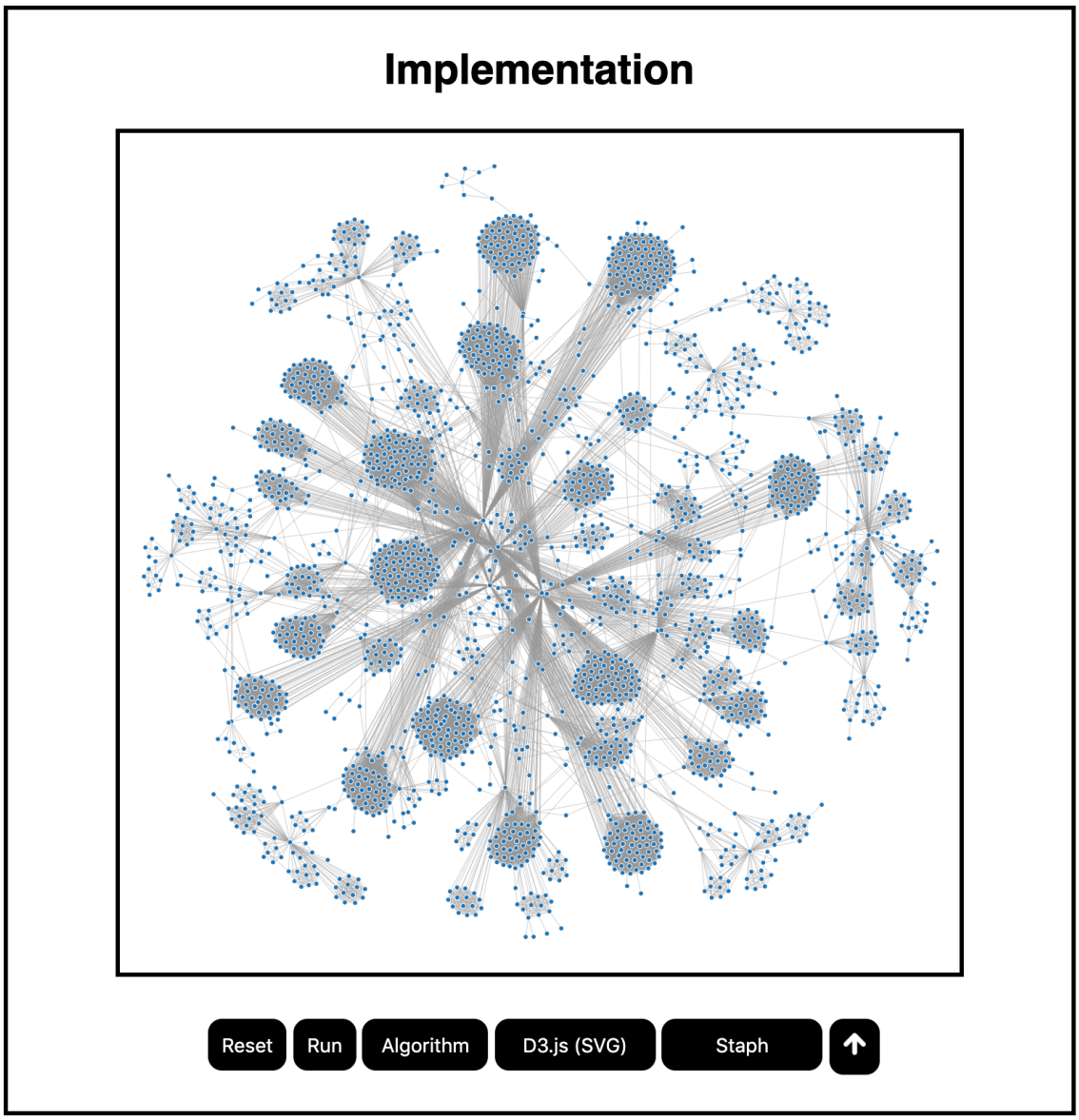

Figure 29 *Phyl*.

4.2. Community Finding

In terms of accuracy, Louvain outperformed all others for weakly mixed networks ($\mu < 0.4$). It is followed by Infomap, LP and LLP ($\gamma = 0.5$) for $\mu < 0.3$ or Infomap, LLP and LP ($\gamma = 0.5$) for $0.3 \leq \mu \leq 0.4$. By increasing μ, LLP, which is able to detect communities based on the panorama of the whole network, consistently becomes the most effective for $\mu \geq 0.5$. Comparing Louvain and Infomap when $\mu \geq 0.5$, their performance is similar. Based on

the CI 95%, it is not possible to state one performs better than the other. Another point to highlight is that Louvain and Infomap are very sensitive, consequently, unreliable detecting communities in networks with $\mu \approx 0.5$. LLP and LP do not present any specific value in which community detection is steeply affected. These conclusions are all valid to GN and LFR networks (Figure 30).

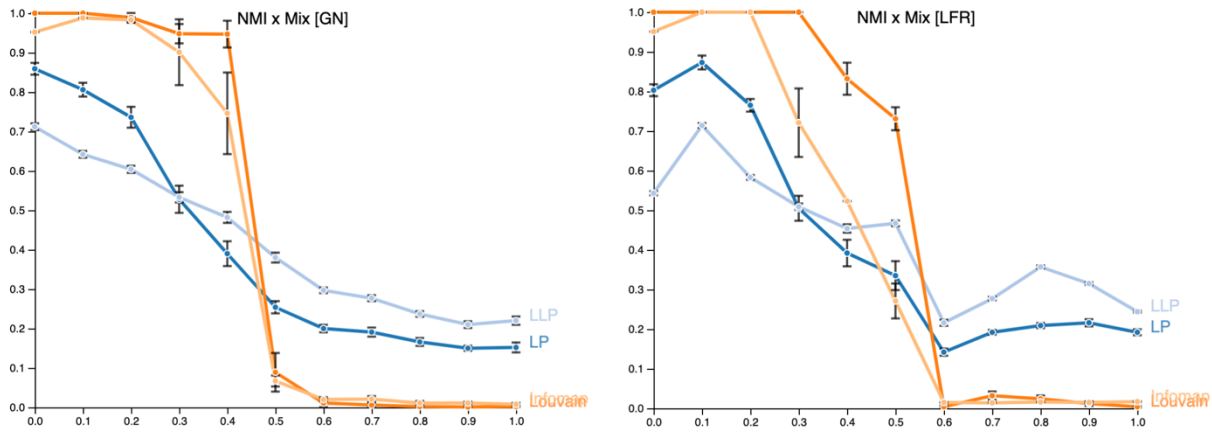

Figure 30 Clustering quality of each algorithm: Louvain (orange), Infomap (light-orange), LP (blue) and LLP (light-blue). GN and LFR networks were analyzed. Some error bars may not be totally visible due to their small values.

Having considered only 2 gamma values in LLP, a more precise analysis was needed to check whether there could be others achieving better results.

In the case of GN network, it is possible to detect communities with higher accuracy in networks less mixed. The performance decreases whenever we increase γ for $\mu < 0.5$. When $\mu \geq 0.5$, the NMI between the detected partition and the one in the original network is higher as long as we keep increasing γ. Another point to highlight is the possibility of detecting communities with higher precision in networks with higher μ using appropriate γ than in others with lower μ but with a non-optimal γ. This is valid for networks with $\mu > 0.5$ (Figure 31).

In LFR networks, increasing gamma always leads to a decrease in the NMI. Nevertheless, better detection is achieved using higher γ as long as we keep increasing μ. Last conclusion is generalizable to GN networks (Figure 31).

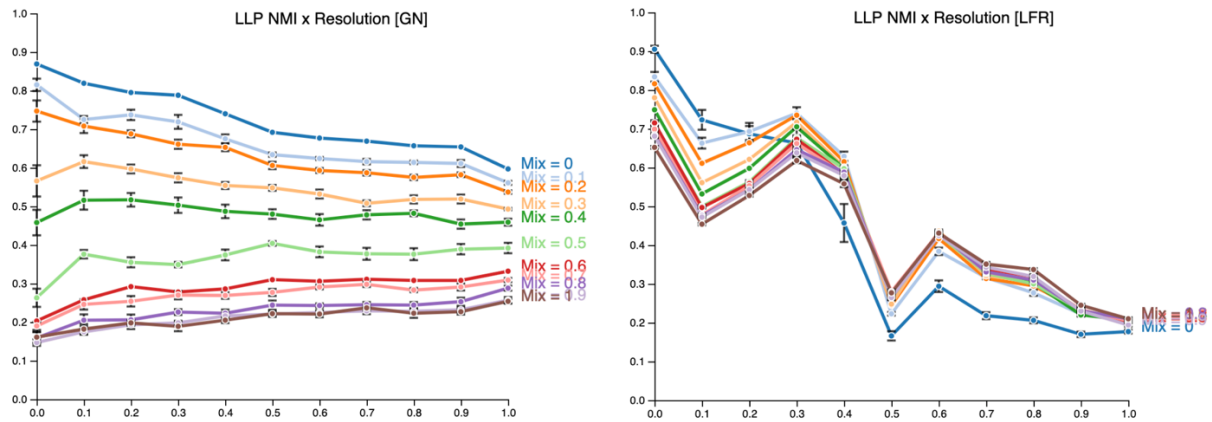

Figure 31 LLP algorithm accuracy in terms of the resolution parameter. Analysis performed for mixing parameters 0 – 1 (blue-brown) in GN networks. Some error bars may not be totally visible due to their small values.

The influence of the nodes' average degree was tested to check whether it affects the capacity of each algorithm to identify communities.

In the case of GN networks, for $\mu \leq 0.5$, it enhances the capacity of Louvain, Infomap, LP and LLP to differentiate communities. For $\mu > 0.5$, NMI clustering quality is lower as higher is the nodes' average degree. Assuming it is not expected the algorithms to identify the original communities for $\mu \approx 1$, this suggests that higher average degrees tends to eliminate the influence of noise in community detection. This observation can be applied to GN and LFR networks (Figure 32, Figure 33, Figure 34 and Figure 35).

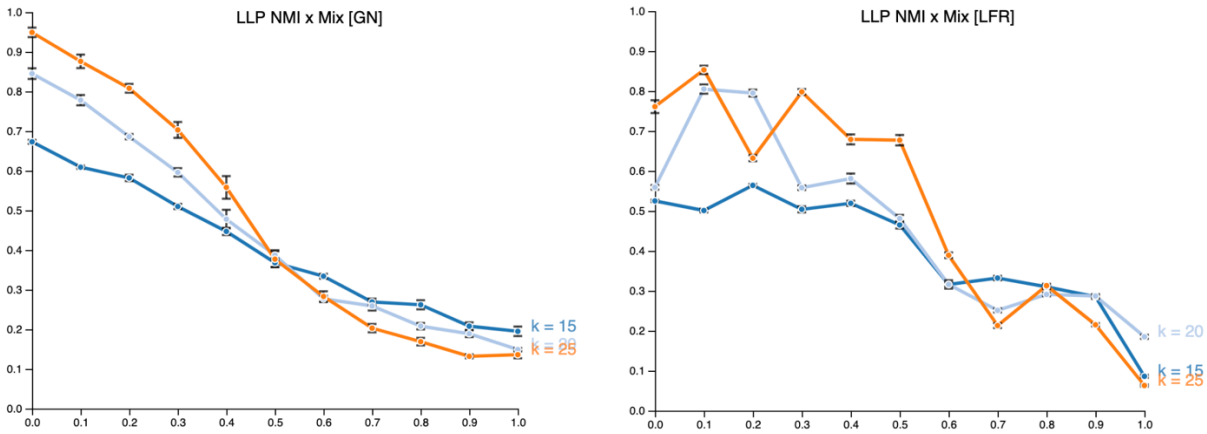

Figure 32 LLP algorithm accuracy in terms of the mixing parameter. Analysis performed in GN and LFR networks with nodes with the following average degrees: 15 (blue), 20 (light-blue) and 25 (orange). Some error bars may not be totally visible due to their small values.

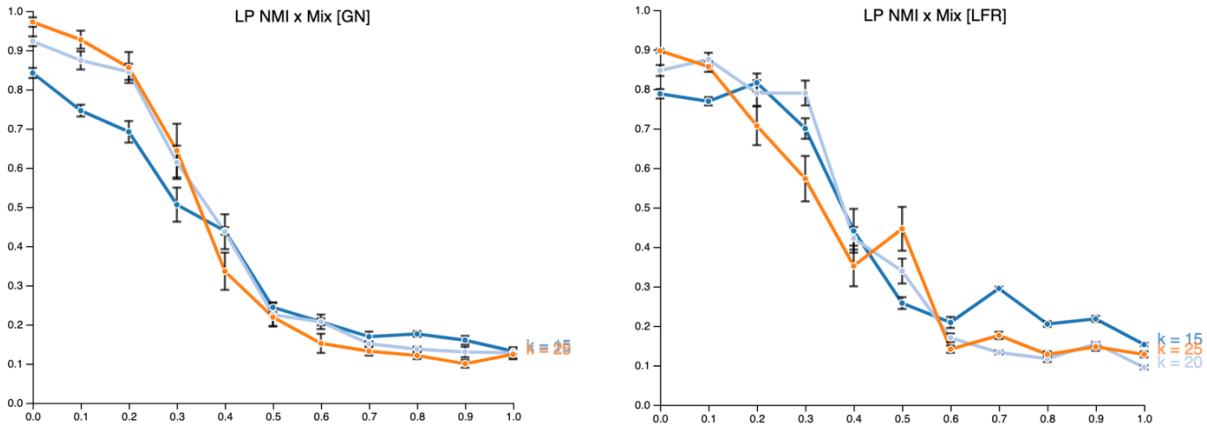

Figure 33 Label Propagation algorithm accuracy in terms of the mixing parameter. Analysis performed in GN and LFR networks with nodes with the following average degrees: 15 (blue), 20 (light-blue) and 25 (orange). Some error bars may not be totally visible due to their small values.

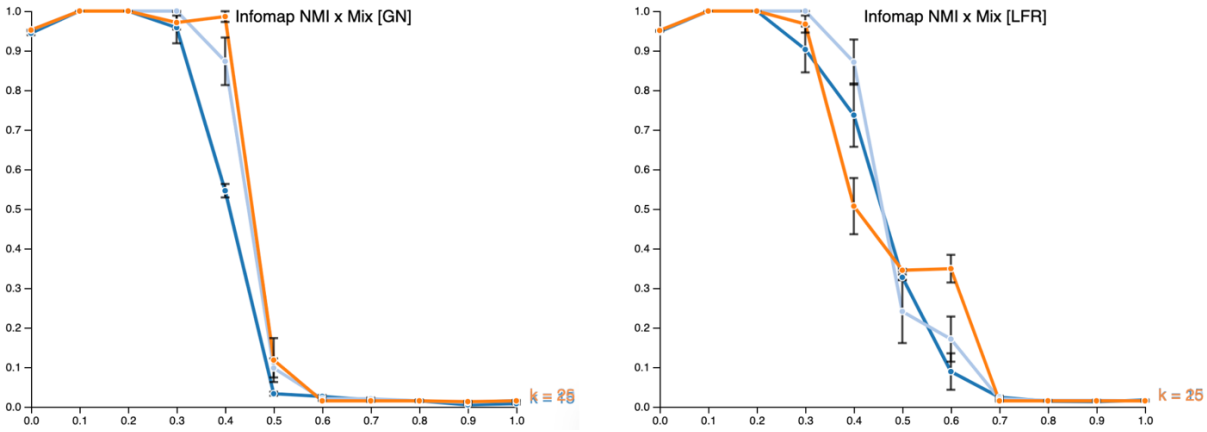

Figure 34 Infomap algorithm accuracy in terms of the mixing parameter. Analysis performed in GN and LFR networks with nodes with the following average degrees: 15 (blue), 20 (light-blue) and 25 (orange). Some error bars may not be totally visible due to their small values.

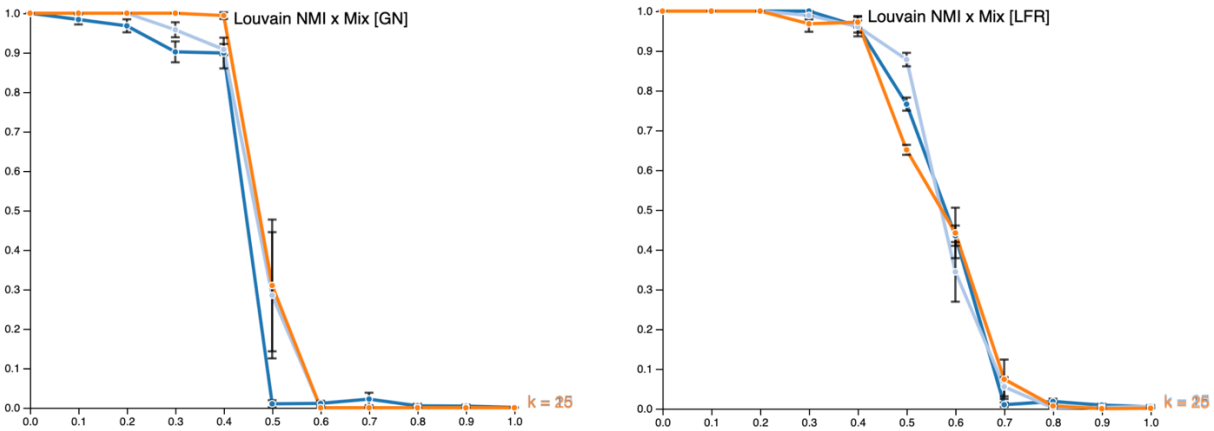

Figure 35 Louvain algorithm accuracy in terms of the mixing parameter. Analysis performed in GN and LFR networks with nodes with the following average degrees: 15 (blue), 20 (light-blue) and 25 (orange). Some error bars may not be totally visible due to their small values.

In terms of speed, Louvain algorithm performs significantly better than any other. This happens due to the efficient step which allows the algorithm to calculate modularity variation without having to recalculate modularity at every step of the iteration in *Modularity Optimization* phase (85). Contrarily to what happens with Infomap, which recalculates minimum description length at the end of each iteration (91). Infomap presented the poorest performance in the benchmark tests. Comparing LLP and LP, the fastest was the one that needed to optimize the simplest equation – LP (Figure 36).

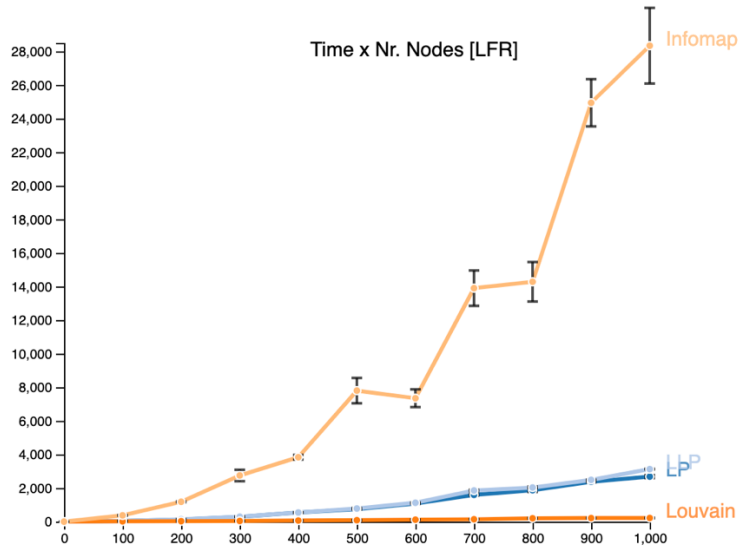

Figure 36 Time that Louvain (orange), Infomap (light-orange), LLP (light-blue) and LP (blue) took to finalize the analysis in terms of the size of the LFR network. Some error bars may not be totally visible due to their small values.

4.3. Benchmark

Previous GN and LFR networks were generated so the previous tests could be performed. The algorithm to create the former was implemented in the thesis. The time it takes to generate such networks increases exponentially with the average degree of the nodes and the mixing parameter. Becoming unfeasible to execute it for average degrees higher than 25, in an ordinary computer. Its performance was compared to the Fortunato's implementation which is highly scalable and allows the generation of networks with much higher number of links in shorter times. It was verified a linear increase in the time of execution for networks with progressively higher mixing parameters, contrarily to the one implemented in the thesis. Just like before, generating networks with more edges, requires additional computational power (Figure 37).

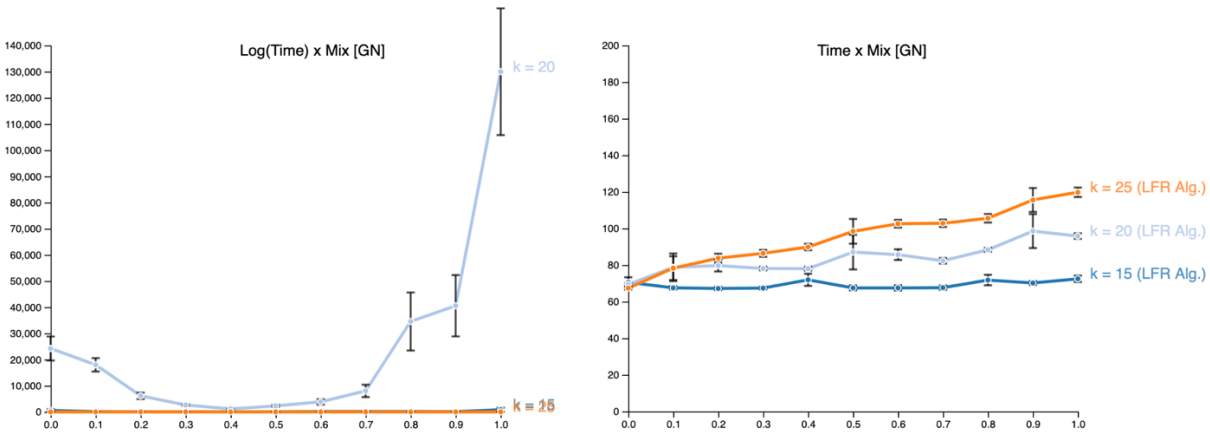

Figure 37 Time needed to generate GN networks (using thesis and LFR implementation) in terms of the mixing parameter. Analysis was performed in networks with nodes with the following average degrees: 15 (blue), 20 (light-blue) and 25 (orange). Some error bars may not be totally visible due to their small values.

Additional tests were performed in LFR networks. The execution time was tested against the number of nodes and the mixing parameter of the network. In both cases, it was verified a linearly proportional relation (Figure 38). The number of nodes was not considered in GN networks, once they assume a fixed number of elements per community and of communities. In spite only the mixing parameter and the number of nodes were analyzed, the user can still choose to adjust nodes' maximum degree, the exponents of the nodes' degree distribution and communities' size distribution, the maximum/minimum number of nodes per community, the number of overlapping nodes per community (covers were not considered in the thesis) and the number of memberships for the overlapping nodes [42].

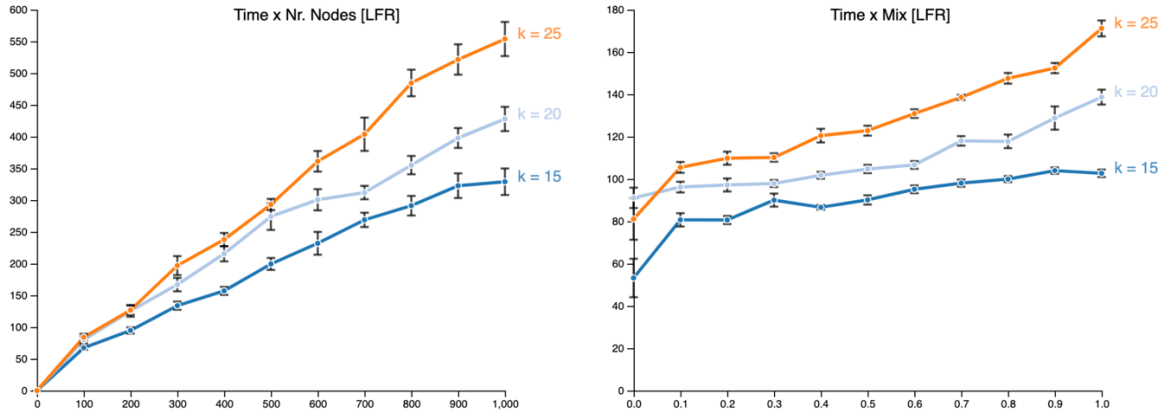

Figure 38 Time needed to generate LFR benchmark networks in terms of the number of nodes (left) and mixing parameter (right). Analysis was performed in networks with nodes with the following average degrees: 15 (blue), 20 (light-blue) and 25 (orange). Some error bars may not be totally visible due to their small values.

4.4. Visualization

D3.js (SVG) allowed the user to visualize all networks used in the web application, as well as to represent the benchmark plots with the highest resolution possible (limited by density of pixels of the device in use). In terms of performance, it is the second fastest.

D3.js (Canvas) is the fastest among the analyzed frameworks. Nevertheless, when compared to the previous, the nodes and links from the network lose resolution.

Cytoscape.js is advisable when network analysis will be performed along with its representation. There is a considerable number of algorithms, including community finding ones, that can facilitate the study of the graph. The lack of working examples and the slow processing of network data turn Cytoscape.js the less efficient. Representation of *Staphylococcus aureus* SLV network and others with similar or higher number of nodes/edges overflows the web application.

4.5. Phylogenetics

After running the golden standard community finding algorithm (Louvain), along with the best performant interface (D3.js with Canvas), an SLV network of *Staphylococcus aureus* CC0 was returned. After adjusting the stopping parameters of Louvain, such that the communities identified were the most pertinent following empirical criteria (every node of a densely connected region should belong to the same community), a final network was obtained (Figure 39).

Upon insertion of the generated metadata file in *PHYLOViZ Online*, one may expect to observe the MST with the respective nodes colored accordingly to the community each one belongs.

Figure 39 Network obtained after running Louvain algorithm (until 1/10000 modularity variation) on CC0 of *Staphylococcus aureus* MLST profile data. Obtained using D3.js and Canvas.

5. Conclusion

A web application – *Phyl* – was conceived from scratch and is now fully functional and available online (working in any browser or device).

6 algorithms implemented along the thesis are now available to download in NPM (after proper testing and benchmarking): *louvain-algorithm* (Louvain), *infomap* (Infomap), *layered-label-propagation* (LLP), *girvan-newman-benchmark* (GN benchmark), *normalized-mutual-information* (NMI) and *hamming-dist* (Hamming distance).

An image of the web application can be downloaded from Docker Hub.

A GitHub repository includes every implementation and a report clarifying all steps taken along the thesis.

Next steps include the integration of the functionalities developed in the thesis in a phylogenetic/surveillance-oriented analysis web application like *PHYLOViZ Online* or *INSaFLU*. Phylogenetic analysis and visualization tools will be implemented in the latter in the next months.

6. References

[1] "Top 10 Leading Causes of Death Globally," [Online]. Available: https://www.theatlas.com/charts/HkLaDreuW. [Accessed 12 May 2019].

[2] A.-L. Barabási, "Network Science Book," [Online]. Available: http://networksciencebook.com. [Accessed 15 May 2019].

[3] "Graph theory," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Graph_theory. [Accessed 12 May 2019].

[4] L. Euler, "The solution of a problem relating to the geometry of position," *Commentarii academiae scientiarum Petropolitanae,* vol. 8, pp. 128-140, 1741.

[5] "Seven Bridges of Königsberg," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg. [Accessed 12 May 2019].

[6] P. Erdős and A. Rényi, "On random graphs I.," *Publicationes Mathematicae (Debrecen),* vol. 6, pp. 290-297, 1959.

[7] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics,* vol. 30, no. 4, pp. 1141-1144, 1959.

[8] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science,* vol. 286, p. 509–512, 1999.

[9] S. A. Cook, "The complexity of theorem proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71)*, 1971.

[10] L. Fortnow, "The Status of the P versus NP Problem," *Communications of the ACM,* vol. 52, no. 9, p. 78–86, 2009.

[11] C. M. Institute, "P vs NP Problem," [Online]. Available: https://www.claymath.org/millennium-problems/p-vs-np-problem. [Accessed 15 May 2019].

[12] N. Viswarupan, "P vs NP Problem," Medium, 17 August 2017. [Online]. Available: https://medium.com/@niruhan/p-vs-np-problem-8d2b6fc2b697. [Accessed 15 May 2019].

[13] "P versus NP problem," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/P_versus_NP_problem. [Accessed 12 May 2019].

[14] J. Rosenberger, "P vs. NP poll results," *Communications of the ACM,* vol. 55, no. 5, p. 10, 2012.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, Cambridge, Massachusetts: MIT Press , 2003.

[16] S. A. Rice, "The identification of blocs in small political bodies," *American Political Science Review,* vol. 21, p. 619–627, 1927.

[17] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 99, no. 12, pp. 7821-7826, 2002.

[18] G. D. Bader and C. W. V. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC Bioinformatics,* vol. 4, no. 2, 2003.

[19] G. Ouyang, D. Dey and P. Zhang, "Clique-Based Method for Social Network Clustering," *Journal of Classification,* 2017.

[20] R. D. Luce, "Connectivity and generalized cliques in sociometric group structure," *Psychometrika,* vol. 15, pp. 159-190, 1950.

[21] A. Kahng, J. Lienig, I. Markov and J. Hu, VLSI Physical Design: From Graph Partitioning to Timing Closure, Springer, 2011.

[22] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai and A. L. Barabási, "Hierarchical Organization of Modularity in Metabolic Networks," *Science,* vol. 297, no. 5586, pp. 1551-1555, 2002.

[23] M. E.J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review. E, Statistical, nonlinear, and soft matter physics,* vol. 69, 2004.

[24] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review E,* vol. 69, no. 6, 2004.

[25] G. Palla, I. Derényi, I. Farkas and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature,* vol. 435, p. 814–818, 2005.

[26] Y.-Y. Ahn, J. P. Bagrow and S. Lehmann, "Link communities reveal multiscale complexity in networks," *Nature,* vol. 466, pp. 761-764, 2010.

[27] J. Yang and J. Leskovec, "Community-Affiliation Graph Model for Overlapping Network Community Detection," in *2012 IEEE 12th International Conference on Data Mining*, Brussels, Belgium, 2012.

[28] W. K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika,* vol. 57, no. 1, pp. 97-109, 1970.

[29] A. Lancichinetti, S. Fortunato and J. Kertesz, "Detecting the overlapping and hierarchical community structure of complex networks," *New Journal of Physics,* vol. 11, 2009.

[30] S. Gregory, "Fuzzy overlapping communities in networks," *Journal of Statistical Mechanics: Theory and Experiment,* vol. 2011, no. 02, 2011.

[31] V. D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech. (2008) P10008,* p. 12, 2008.

[32] M. Rosvall, D. Axelsson and C. T. Bergstrom, "The map equation," *The European Physical Journal Special Topics,* vol. 178, no. 1, pp. 13-23, 2009.

[33] N. Raghavan, R. Albert and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E 25th Anniversary Milestones,* vol. 76, no. 3, 2007.

[34] L. Šubelj, "Label propagation for clustering," in *Advances in Network Clustering and Blockmodeling*, New York, Wiley, 2018.

[35] P. Boldi, M. Rosa, M. Santini and S. Vigna, "Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks," in *WWW '11 Proceedings of the 20th international conference on World wide web*, 2011.

[36] "cgMLST- Genome-wide gene by gene microbial typing," Ridom Bioinformatics, [Online]. Available: https://www.ridom.de/seqsphere/cgmlst/. [Accessed 12 May 2019].

[37] B. Ribeiro-Gonçalves, A. P. Francisco, C. Vaz, M. Ramirez and J. Carriço, "PHYLOViZ Online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees," *Nucleic Acids Research,* vol. 44, no. W1, pp. 246-251, 2016.

[38] R. C. Prim, "Shortest connection networks And some generalizations," *Bell System Technical Journal,* vol. 36, no. 6, p. 1389–1401, 1957.

[39] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society,* vol. 7, no. 1, p. 48–50, 1956.

[40] A. P. Francisco, C. Vaz, P. T. Monteiro, J. Melo-Cristino, M. Ramirez and J. A. Carriço, "PHYLOViZ: phylogenetic inference and data visualization for sequence based typing methods," *BMC Bioinformatics,* vol. 13, no. 87, 2012.

[41] A. P. Francisco, M. Bugalho, M. Ramirez and J. Carrico, "Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach," *BMC Bioinformatics,* vol. 10, no. 152, 2009.

[42] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical review. E, Statistical, nonlinear, and soft matter physics.,* vol. 80, 2009.

[43] A. Lancichinetti, S. Fortunato and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical review. E, Statistical, nonlinear, and soft matter physics.,* vol. 78, no. 4, 2008.

[44] T. M. Cover and J. A. Thomas, Elements of Information Theory, Second Edition, New Jersey, USA: John Wiley & Sons, 2005.

[45] "Mutual information," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Mutual_information. [Accessed 12 May 2019].

[46] J. Yang and J. Leskovec, "Defining and Evaluating Network Communities based on Ground-truth," in *Proceedings of 2012 IEEE International Conference on Data Mining (ICDM)*, 2012.

[47] W. Zachary, "An Information Flow Model for Conflict and Fission in Small Groups," *Journal of anthropological research,* vol. 33, 1976.

[48] WHO, "The top 10 causes of death," 24 May 2018. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death. [Accessed 19 April 2019].

[49] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical Review E,* vol. 80, no. 5, 2009.