Community Finding with Applications on Phylogenetic Networks

Luís Rita[1,3,4]

Supervision Team: Bruno Gonçalves[3], Alexandre Francisco[1,2], João Carriço[3], Vítor Borges[4]

(1) Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
(2) INESC-ID, R. Alves Redol 9, 1000-029 Lisboa, Portugal
(3) Instituto de Microbiologia, Instituto de Medicina Molecular, Faculdade de Medicina, Universidade de Lisboa, Lisboa, Portugal
(4) Departamento de Doenças Infeciosas, Instituto Nacional de Saúde Dr. Ricardo Jorge, Lisboa, Portugal

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 to visualize the results in a minimalist way, in any browser or device.

Louvain, Infomap and Layered Label Propagation (LLP) algorithms were implemented, tested and benchmarked against real and synthetic networks. An interface was implemented and tested using Cytoscape.js and D3.js (SVG and Canvas).

For the first time, these algorithms were implemented in JavaScript. 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, LLP performed better. It is advantageous to use higher gamma parameters for highly mixed Girvan-Newman (GN) alike networks, oppositely, to what was verified in well separated ones. The increase in the nodes' average degree enhanced 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 (LFR) implementation, the execution time remains linear. In the thesis implementation, it is exponential.

Along with *PHYLOViZ*, *Phyl* allows us to promptly handle infectious diseases. 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

Introduction

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

Graphs are compact ways of representing all components and their relations in arbitrarily complex systems. Graph theory comprehend all the tools that have been used to analyze these structures.

Although networks are ubiquitous, their mathematical representation – graphs, it is still not a standard. Sometimes due to lack of reliable data or for privacy issues.

Next Generation Sequencing methods, developed in the last years, have been proving us with 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. Once these phenomena frequently require 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 events.

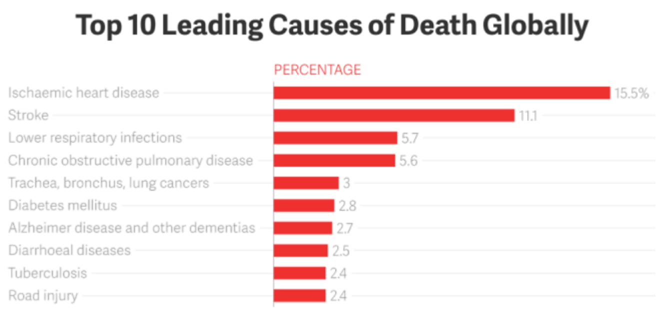

Figure 1 – Infections are still one of the major causes of death globally. Data collected by WHO between 2000-2015 from a population of 90 000 people [1].

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

To answer the previous 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. Having said this, based on the network under analysis, one should test a wide variety of algorithms with varying parameters, to find out which ones perform the best. Below are some strategies that are commonly used to perform community finding.

Maximum Clique – Every subgraph completely connected is considered a community. This was considered a definition too rigid and has been deprecated.

Strong Community – Every vertex in a given community has more edges connected to vertices in the same community than outside:

$$k_i^{int}(C) > k_i^{ext}(C) \qquad (1)$$

$k_i^{int}(C)$ and $k_i^{ext}(C)$ are, respectively, the number of internal and external links of a given node i in a cluster C.

Weak Community – The sum of internal edges in a given community is superior than from inside outwards:

$$\sum_{i\in C} k_i^{int}(C) > \sum_{i\in C} k_i^{ext}(C) \qquad (2)$$

In the case of small networks, one may decide to apply the previous rules to perform community detection. The problem, at this stage, is that brute-force approaches looking for communities as defined above is not scalable. The number of possible partitions grows faster than exponentially with the network's size. Bell Number illustrates this:

$$B_N = \frac{1}{e}\sum_{j=0}^{\infty}\frac{j^N}{j!} \qquad (3)$$

N is the total number of nodes in the network. Community finding algorithms are imperative.

Hierarchical Clustering – it presents 2 approaches: agglomerative (Ravasz algorithm [1]) and divisive (GN algorithm [2] [3]). The 1st implies the definition of a similarity matrix representing the distance between nodes. After, a group similarity criterion is defined to allow us to calculate the distance between clusters (each future node consists of a group of previously joint nodes due to high similarity). The algorithm stops when 1 single cluster remains. A dendrogram is returned. The choice of the best partition can be done using modularity – a measure that compares link density that would be expected by chance with the one present in the network. GN algorithm focuses in gradually eliminating edges which are the most likely to be connecting nodes from different communities (until none is left). *Link betweenness* or *random-walk betweenness* usually are used to determine which edges to remove. The 1st is proportional to the number of shortest paths, between all pairs of nodes in the network, the specified link belongs. In the 2nd case, instead of considering shortest paths, random walks between every combination of 2 nodes in the network is applied. Finally, the algorithm returns a dendrogram with the most likely partitions. Again, modularity can be used to determine the best.

Modularity Maximization – this algorithm proposes the use of modularity itself as an optimization quantity at every stage of the algorithm. At the beginning, every node is assigned to a different community. Every connected pair of nodes in the network is analyzed and the network's modularity determined (considering this change). The duo that maximizes modularity is chosen and joined as part of the same community. Modularity is recorded. This is done until there is no increase in modularity. In the case of big networks, this algorithm struggles in detecting communities below a given size. Moreover, during its execution, it may find significantly different partitions with similar modularity. Depending on the stopping threshold, it can become unreliable. Many algorithms depend on similar strategies, but different quality functions to optimize.

Clique-Based Methods – *Clique Percolation* [4] and *Link Clustering* [5] are 2 possible approaches. Their major strength resides in allowing nodes to belong to more than 1 community. The 1st one considers communities groups of nodes whose connections form k-cliques. The 2nd applies a similar method to Rasvaz algorithm, but it clusters links instead of nodes.

Statistical Inference – Yang et al [6] proposed a new model-based approach that takes into consideration community overlapping in certain parts of the graph. By finding out the unknown parameters of the model, it becomes possible to unravel the communities in the graph. The algorithm maximizes the likelihood:

$$arg \max_{B,\{p_c\}} L(B,\{p_c\}) = \prod_{(u,v)\in E} p(u,v) \prod_{(u,v)\notin E} (1-p(u,v)) \qquad (4)$$

Let B be the bipartite graph which assigns each node to a different community in the original model and $\{p_c\}$ the probability associated to each cluster which translates in the frequency that links between nodes of the same module will be present. $p(u,v)$ is the probability 2 edges will appear connected:

$$p(u,v) = 1 - \sum_{k\in C_{uv}} (1-p_k) \qquad (5)$$

This introduction broadly covers all types of community finding algorithms available. In the thesis, it was implemented Louvain, Infomap and LLP algorithms which are described in the section below.

In a world where data processing is migrating from our devices to the cloud, both due to affordability and computational power reasons, it was clear the need to empower the user with server-side tools.

Methods

Community Finding

Louvain

This algorithm is divided in 2 phases: Modularity Optimization and Community Aggregation [7]. After the first step is completed, the second follows in the pipeline. Louvain will iteratively go through both until we get an optimized partition of the network.

$$M = \frac{1}{2m}\sum_{i,j}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j) \qquad (6)$$

Let m be the sum of the weights of all edges in the graph, A_{ij} the entry for nodes i and j in the adjacency matrix, k_i the degree of node i and c_i the community it belongs.

Modularity Optimization – the algorithm will randomly order all nodes in the network such that, one by one, it will remove and insert it in a different community. This will continue until no significant variation (input parameter) of modularity is verified. This variation is given by:

$$\Delta M = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + 2k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \qquad (7)$$

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 sum of the weights of all links incident in node i, $k_{i,in}$ the sum of the weights of links connecting node i and nodes in the community C and m is the sum of the weights of all edges in the graph.

Community Aggregation- After finalizing the first pass, all nodes belonging to the same community are merged into a single giant one and the links connecting these will be formed by the sum of the ones previously connecting nodes from the same different communities. From now on, there will also be present self-loops that represent the sum of all links in a given community before being collapsed into a single one.

Algorithm 1 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;
$\theta$: modularity improvement threshold.
Ensure:
$M$: resulting module;
$Mod$: resulting modularity;
$\delta Mod$: 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 $j$.
1:  $m = \sum k_{i,j}, (i,j) \in E^0$
2:  $k = 0$ // Iteration number.
3:  repeat
4:    // Attributing a different community to each node.
5:    for all $i \in V^k$ do
6:      $M_i^k = \{i\}$
7:    end for
8:    Compute $Mod_{new} = Mod(M)$ using Equation 3
9:    repeat
10:     $Mod = Mod_{new}$
11:     Randomize the order of vertices.
12:     for all $i \in V^k$ do
13:       $best\_community = M_i^k$
14:       $best\_increase = 0$
15:       for all $M' \in C^k$ do
16:         $M_i^k = M_i^k \backslash \{i\}$
17:         ${\Sigma_{tot}}^{M_i^k} = \sum_{\alpha \in M_i^k} k_\alpha - k_i; {\Sigma_{tot}}^{M'^k_i} = \sum_{\alpha \in M'^k_i} k_\alpha + k_i;$
18:         $\Sigma_{in} = {\Sigma_{tot}}^{M'^k_i} - \sum k_{i,j}, (i,j) \in E^k, i \in C_i^k \; and \; j \notin C_i^k$
19:         $k_{i,in} = \sum_{\alpha \in M'^k_i} k_{i,\alpha}$
20:         If $\delta Mod_{M_i^k \to M'^k_i} > best\_increase$ then
21:           $best_{increase} = \delta Mod_{M_i^k \to M'^k_i}$
22:           $best_{com} = M'^k_i$
23:           $M'^k_i = M'^k_i \cup \{i\}$
24:         else
25:         $M_i^k = M_i^k \cup \{i\}$
26:         end else
27:       end for
28:     end for
29:   Compute $Mod_{new} = Mod(M)$ using Equation 3
30:   until No vertex movement or $Mod_{new} - Mod < \theta$
31:  // Calculate updated modularity
32:  Compute $Mod_{new} = Mod(M)$ using Equation 3
33:  If $Mod_{new} - Mod < \theta$ then
34:    break;
35:  end if
36:  $Mod = Mod_{new}$
37:  // Merge communities into a new graph
38:  $V^{k+1} \leftarrow C^k$
39:  $E^{k+1} \leftarrow e(C_u^k, C_v^k)$
40:  $G^{k+1} = (V^{k+1}, E^{k+1})$
41:  $k = k + 1$
42:  until break
```

Infomap

Similarly to Louvain algorithm, network partitioning is done using a quality function. In this case, we pretend to minimize the minimum description length of the network [8]:

$$L(M) = qH(Q) + \sum_{m=1}^{n_m} p_{\odot}^{m} H(P_m) \qquad (8)$$

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

Expanding each term:

$$L(M) = \left(\sum_{m\in M} q_m\right) log\left(\sum_{m\in M} q_m\right) - 2\sum_{m\in M} q_m\, log(q_m) - \sum_{\alpha\in V} p_\alpha\, log(p_\alpha) + \sum_{m\in M} (q_m + \sum_{\alpha\in m} p_\alpha)\, log\left(q_m + \sum_{\alpha\in m} p_\alpha\right) \quad (9)$$

Let q_m be 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.

Algorithm 2 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;
    $\theta$: quality improvement threshold.
    Ensure:
    $M$: resulting module;
    $L$: resulting MDL;
    $\delta L$: 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 $v$.
1:  $w = 0$ // Iteration number.
2:  repeat
3:    for all $u \in V^k$ do
4:      $p_u = \frac{degree(u)}{|E^k|}$
5:    end for
6:  // Attributing a different community to each node.
7:    for all $u \in V^k$ do
8:      $M_u^k = \{u\}$
9:      $p^{M_u^k} = \sum_{\alpha\in M_u^k} p_\alpha$
10:     $q^{M_u^k} = \sum w_{u,v}, (u,v) \in E^k, u \in C_u^k \text{ and } v \notin C_u^k$
11:   end for
12:   Compute $L_{new} = L(M)$ using Equation 5
13:   Repeat
14:     $L = L_{new}$
15:     Randomize the order of vertices.
16:     for all $u \in V^k$ do
17:       If $M'^k_u = argmin\left(\delta L_{M_u^k \to M'^k_u}\right) < 0$ then
18:         $M_u^k = M_u^k \backslash \{u\}$; $M'^k_u = M'^k_u \cup \{u\}$
19:         $p^{M_u^k} = \sum_{\alpha\in M_u^k} p_\alpha - p_u$; $p^{M'^k_u} = \sum_{\alpha\in M'^k_u} p_\alpha + p_u$
20:         update $q^{M_u^k}$; update $q^{M'^k_u}$
21:       end if
22:     end for
23:     Compute $L_{new} = L(M)$ using Equation 5
24:   until No vertex movement or $L - L_{new} < \theta$
25:   // Calculate updated modularity
26:   Compute $L_{new} = L(M)$ using Equation 3
27:   If $L - L_{new} < \theta$ then
28:     break;
29:   end if
30:   $L = L_{new}$
31:   // Merge communities into a new graph
32:   $V^{k+1} \leftarrow C^k$
33:   $E^{k+1} \leftarrow e(C_u^k, C_v^k)$
34:   $G^{k+1} = (V^{k+1}, E^{k+1})$
35:   $k = k + 1$
36: until break
```

LLP

LLP algorithm development was strongly based in the old Label Propagation (LP) [9]. The latter starts by assigning a different community to each node present in the network and then, iteratively, it modifies the belonging community based on the dominant community of the nodes in the immediate neighborhood. In each iterate, this step will be executed for each node, in a previously randomized order. At the end, it is expected that regions of the network strongly connected will preferentially be grouped into a single community.

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 [10]. The iterative process is the same, the difference resides in the value it pretends to optimize. In this case, the assigned community will be the one that maximizes:

$$k_i - \gamma(v_i - k_i) \quad (10)$$

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.

This approach pretends to buffer the number of nodes belonging to a certain community (in the neighborhood), by its usual presence across the whole network.

Algorithm 3 LLP

```
    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;
    $\gamma$: resolution parameter.
    Ensure:
    $M$: resulting module;
    $k_i$: number of neighbors having label $\lambda_i$;
    $v_i$: overall number of nodes having label $\lambda_i$.
1:  $k = 0$ // Iteration number.
2:  // Attributing a different community to each node.
3:  for all $i \in V^0$ do
4:      $M_i^0 = \{i\}$
5:  end for
6:  repeat
7:      Randomize the order of vertices.
8:      for all $u \in V^k$ do
9:          for all $i \in labels(u)$ do
10:             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
11:                 $M_u^k = M_u^k \backslash \{u\}$; $M'^k_u = M'^k_u \cup \{u\}$
12:             end if
13:         end for
14:     end for
15:     $k = k + 1$
16: until No vertex movement or $k < max_{iteration}$
```

Benchmark

All benchmark tests were repeated 10 times and the average and corresponding interval of confidence at 95% plotted. The following 2 synthetic networks were used to benchmark every algorithm, in terms of accuracy and speed.

GN Network

Historically, GN was the first synthetic benchmark network designed. In the paper Girvan and Newman described it for the first time [2], they considered a network of 128 nodes divided in 4 different groups, each one with exactly the same 32 nodes. Then, a mixing parameter chosen by the user defines the probability of connecting a pair of nodes from different communities. Nodes from different groups connect with a probability given by that value μ and the ones present in the same with $1 - \mu$. The authors considered each node would be connected to, exactly, 16 different ones. The algorithm hereby implemented allow not only the mixing parameter to vary, but also the average degree. This was important to allow an increased number of benchmark tests using these networks.

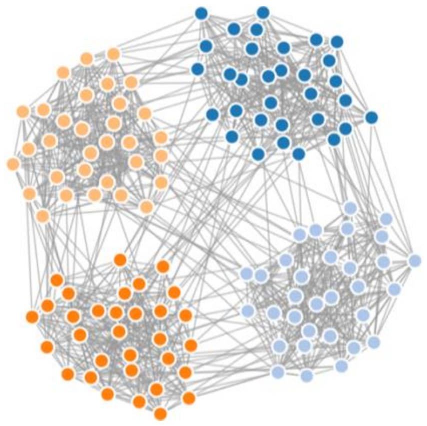

Figure 2 - GN synthetic network. $N = 128 \mid \mu = 0.1 \mid k = 16$. [D3.js] [SVG]

LFR Network

LFR benchmark networks, try to better resemble real networks [11] [2]. This means that not only they consider a power-law distribution of community sizes, but also a power-law distribution of node degrees. This way, the user chooses the exponents for each distribution.

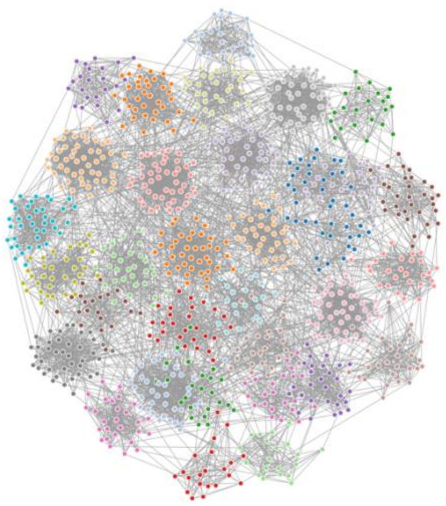

Figure 3 - LFR synthetic network. $N = 1000 \mid \mu = 0.1 \mid k_{avg} = 15 \mid k_{max} = 50 \mid c_{min} = 20 \mid c_{max} = 50$. [D3.js] [SVG]

Normalized Mutual Information

The clustering quality of the community finding algorithms was tested using NMI [12]. An algorithm was developed in JavaScript and tested with several networks. It receives an input of 2 arrays, with the same length, and returns the corresponding normalized mutual information.

$$NMI(Y,C) = \frac{2 \times I(Y;C)}{[H(Y) + H(C)]} \qquad (11)$$

$$I(Y;C) = H(Y) - H(Y|C) \qquad (12)$$

NMI is dependent on the mutual information I, the conditional entropy $H(Y|C)$ and the entropy of the labeled $H(Y)$ and clustered set $H(C)$.

Test Data

The higher the number of tests we performed using the implemented algorithms, the higher the certainty they are working properly for the broadest number of networks. This way, disconnected networks (Amazon) and small sized ones (Zachary's Karate Club) were additionally considered to validate algorithm's correctness.

Amazon Network

This network was obtained after crawling Amazon website [13]. Each node represent a given product available in the store and a connection between 2 is present whenever they are frequently bought together.

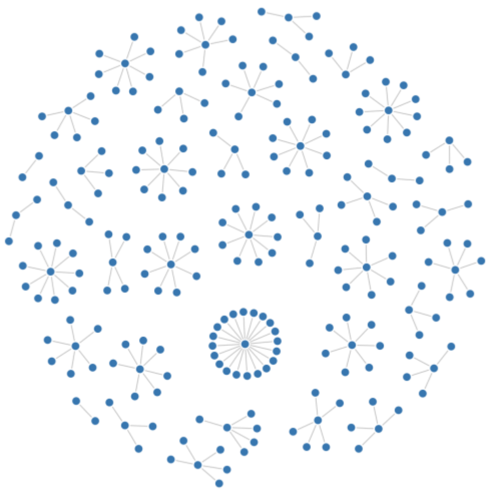

Figure 4 - Amazon product co-purchasing network sample from a 10 000 nodes graph. [D3.js] [SVG]

Zachary's Karate Club Network

One of the most well-known graphs in network science is the Zachary's Karate Club [14]. This represents the relations between individuals from a karate club. A link is present whenever to members meet regularly outside the club. The 2 nodes with a black circumference around are the president and the instructor. The dark and light blue represent the split that was created due to a conflict between the 2 in charge of the club.

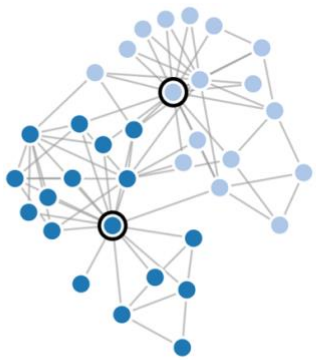

Figure 5 - Zachary's karate club network. [D3.js] [SVG]

Visualization

The user can opt by visualizing every network present above and below this point using D3 (https://d3js.org/) or Cytoscape (http://js.cytoscape.org/) 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.

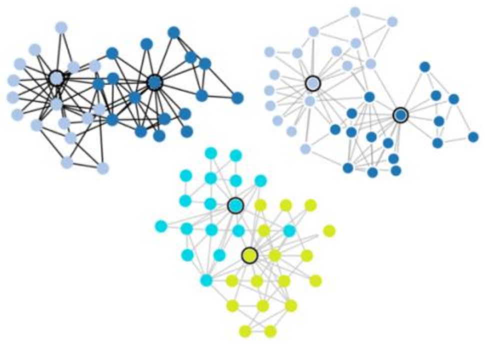

Figure 6 - Zachary's Karate Club network represented using D3.js Canvas (top-left), D3.js SVG (top-right) and Cytoscape.js (bottom).

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

Phylogenetics

After implementing the algorithms, test and benchmark them, phylogenetic data – *Staphylococcus* MLST profile, was analyzed.

1. *PHYLOViZ 2* was firstly installed. A MLST profile of Staphylococcus [], along with the corresponding metadata file [], was then uploaded to the platform. A minimum spanning tree was generated using goeBurst and following all the tiebreak rules. *PHYLOViZ* prints a set of clonal complexes which were defined based on the similarity among STs;
2. Copy-paste operation was manually performed to send CC data from *PHYLOViZ* to a text document. Data format is the same as in the Java tool;
3. This file is then uploaded to *Phyl* using the appropriate button on the interface. Original MLST data profile should also be uploaded;
4. The web application will now build an SLV network with all the strains present in the CC under study, considering *loci* information from the profile file;
5. Community finding algorithms run in the previous network;
6. The input metadata file is now returned with an extra column where every determined community is represented. For all nodes that were not part of the CC, the same random number was attributed (accounting that it should be different than any other already used by the strains in the CC).

Results

Web Application

In order to easily analyze and visualize the results from the execution of the algorithms, a web application was designed.

Its backend was conceived using Node.js and it runs in a Heroku server. The following functions are strictly performed in the cloud:

- Generation of GN and LFR benchmark networks. The latter are generated using a binary from a C++ implementation [15]. This was possible after importing a package from NPM which simulates the command line in the application;
- Amazon [13], Zachary's Karate Club [14] and *Staphylococcus* [] testing networks had their data properly processed in the cloud, so that it could be sent and displayed in the graphical interface;
- Execution of Louvain, Infomap and LLP algorithms is not performed in the user's browser. Although this may raise some privacy issues, it was not feasible to run Infomap in the user's device and still maintain the web application usable in all devices;
- Finally, 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 graphical interface (louvain-algorithm, infomap, layered-label-propagation, girvan-newman-benchmark, normalized-mutual-information and hamming-dist are the packages names);
- Finally, all benchmarking data was obtained after running, repetitively, 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 assured 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 – Cytoscape.js, D3.js (Canvas) or D3.js (SVG);

- *Results* merges in a single section all data obtained from a thorough analysis of every implemented algorithm. It dynamically represents data, so that one can analyze it individually and in a clearer way: considering one line at a time with or without IC 95% error bars. Whenever it is opportune, the same plot can be visualized considering GN or LFR benchmark networks. On the top of each plot, both axes are identified following the rule: (y-axis) x (x-axis). At the end of each line, additional information is provided in order to differentiate the curves. Under each x-axis, the user has the possibility to download the CSV file used in each plot;
- *Web tools* contain a small description of 2 tools that complement *Phylo*;
- *Events* describes 4 contests/workshops which were crucial for the conclusion of this thesis.

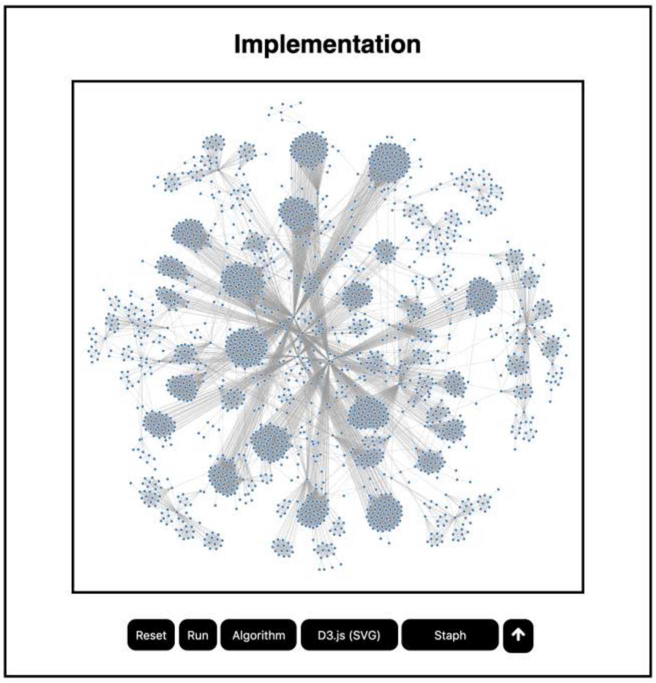

Figure 7 – *Phylo.*

Community Finding

In terms of <u>accuracy</u>, Louvain outperformed all others for weakly mixed ($\mu < 0.3$) networks. Being followed by Infomap, LP and LLP ($\gamma = 0.5$). By increasing mixing, LLP, which is able to detect communities based on the panorama of the whole network, consistently becomes the most effective. Comparing Louvain and Infomap, for mixing parameters higher than 0.5, their performance is similar. Based on the CI 95%, it is not possible to state Infomap performs better. Nevertheless, the curve reaches slightly higher NMI values. Another point to highlight is that Louvain and Infomap are very sensitive, thus unpredictable, in detecting communities in networks with mixing parameters around 0.5. LLP and LP do not present any specific value in which community detection is steeply affected. These conclusions are valid to GN and LFR networks.

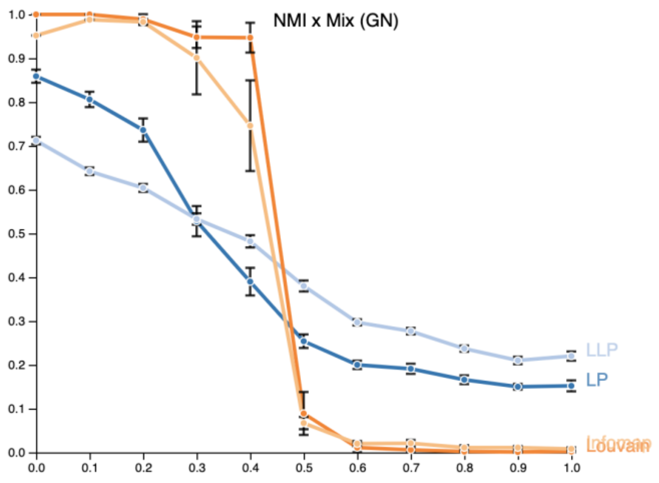

Figure 8 – Clustering quality of each algorithm. GN network under analysis.

Having considered only 2 gamma values for 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 for lightly mixed networks. Decreasing the performance, whenever we increase gamma for mixing parameters inferior to 0.5. For 0.5 or higher mixing parameters, the NMI between the detected partition and the original network is higher as long as we keep increasing gamma. In LFR networks, increasing gamma always leads to a decrease in the NMI. Nevertheless, better detection is achieved using higher gammas as long as we keep increasing mixing. Similarly to GN networks.

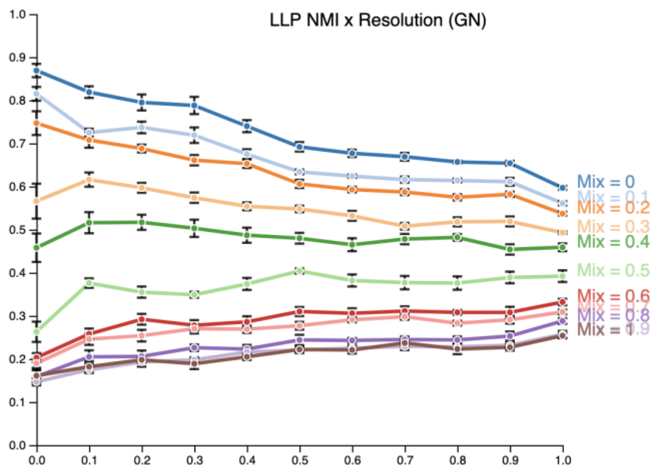

Figure 9 – LLP algorithm accuracy in terms of the resolution parameter. Analysis performed for mixing parameters 0-1 in GN networks.

The influence of the nodes' average degree was tested to check how this affect the capacity of each algorithm to identify the communities present in the network. In the case of GN networks, until maxing parameters around 0.5, it enhances the capacity of Louvain, Infomap, LP and LLP to differentiate communities. After this value, NMI clustering quality is lower as higher is the node average degree. Supposing it is not expected the algorithms to identify the original communities with mixing parameters close to 1, this suggests that higher average degrees tends to eliminate the influence of noise in community detection. This same observation can be formulated for GN and LFR networks.

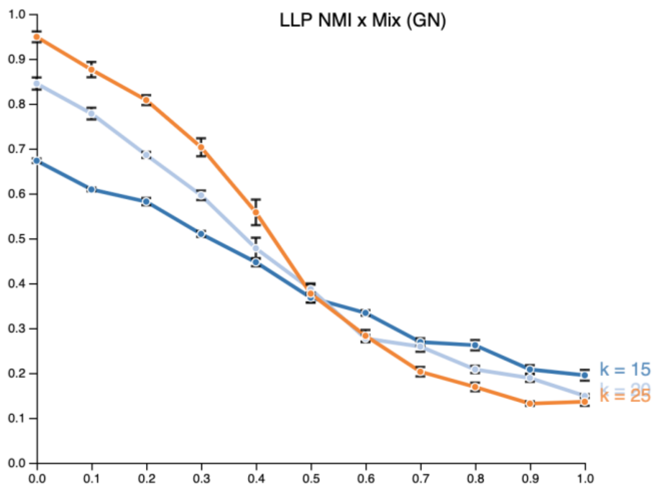

Figure 10 – LLP algorithm accuracy in terms of the mixing parameters. Analysis performed for 15, 20 and 25 nodes average degree GN networks.

In terms of speed, Louvain performs significantly better than any other. This happens due to the efficient step which allow the algorithm to calculate modularity variation without having to recalculate modularity at every step of the iteration in *Modularity Optimization* phase. Contrarily to what happens with Infomap, which recalculates minimum description length at the end of each iteration. This algorithm 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.

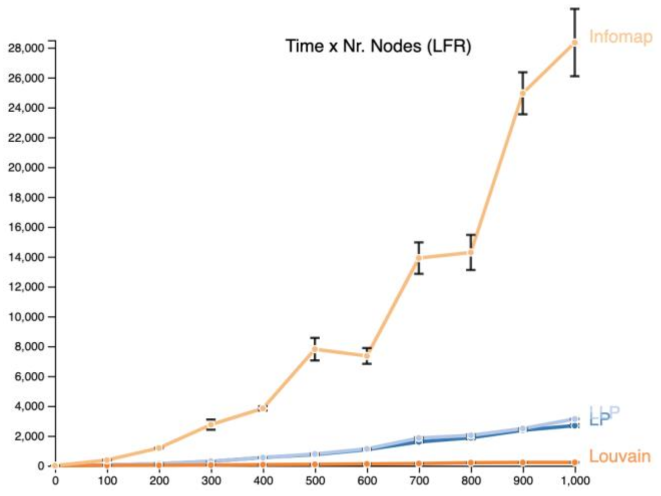

Figure 11 – Time each algorithm takes to finalize the analysis in terms of the size of the LFR network.

Benchmark

The previous GN and LFR networks were generated so the previous tests could be done. The algorithm to implement the former was developed along the thesis. The time it takes to generate such network increases exponentially with the nodes' average degree 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.

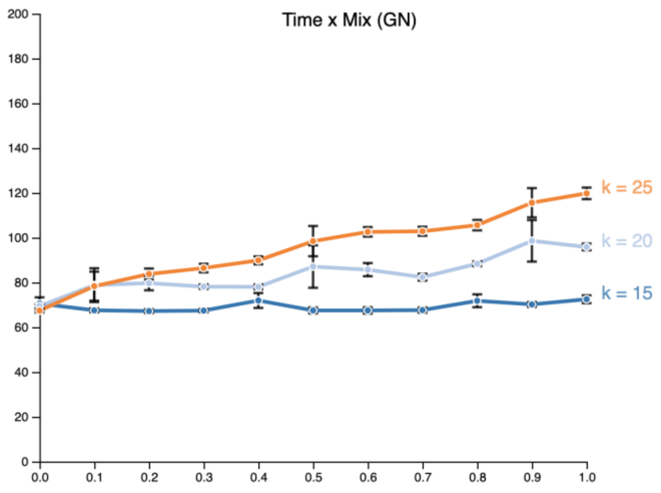

Figure 12 – Time needed to generate GN networks (using LFR implementation) in terms of the mixing parameter. Multiple average node degree networks analyzed.

The same analysis can be extrapolated for LFR benchmark generating algorithm. An additional test was performed. The execution time was tested against the number of nodes (input by the user) in the network. Both quantities are linearly proportional. This test was not considered in GN networks, once they assume a fixed number of nodes 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 node's maximum degree, the exponents of the node's degree distribution and community's size distribution, the maximum/minimum number of nodes per community, the number of overlapping nodes per community (covers were not considered along the thesis) and the number of memberships for the overlapping nodes [].

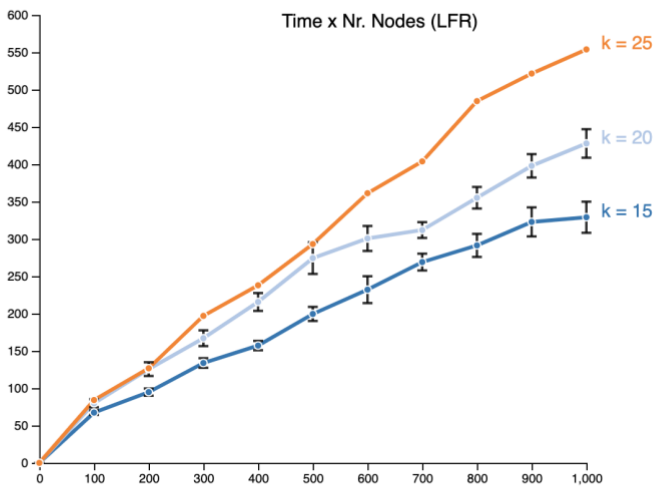

Figure 13 – Time needed to generate LFR benchmark networks in terms of the number of nodes. Multiple average node degree networks analyzed. LFR implementation used.

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 the resolution of your device). 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 loses 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* network and others with similar or higher number of nodes/edges would overflow the web application.

Phylogenetics

After running the golden standard community finding algorithm (Louvain), along with the best performant interface (D3.js, Canvas), an SLV network of *Staphylococcus* CC1 was returned. After adjusting the stopping parameters of this algorithm, such that the communities identified were the most pertinent following empirical criteria, a final network was obtained.

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.

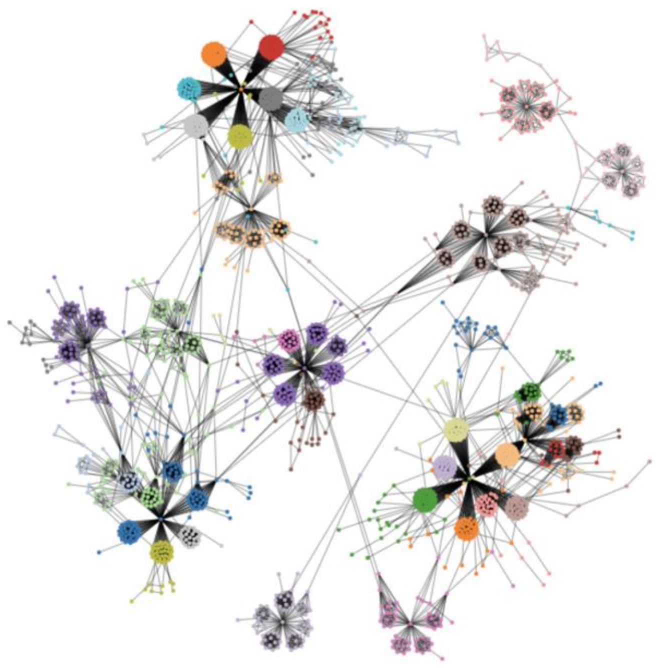

Figure 14 – Network obtained after running Louvain algorithm (until 1/10000 modularity variation) on CC1 of *Staphylococcus* MLST profile data. [D3.js] [Canvas]

Conclusion

A web application was conceived from scratch and is now fully functional and available online (working in any browser or device). It allows the user to run 3 different community finding algorithms, which are now available in NPM to download (after proper testing and benchmarking). Finally, observation of the results upon choosing one out of three frameworks can be also done in the app.

All goals were attained.

Next step should be the integration of these functionalities in a phylogenetic oriented analysis software like *PHYLOViZ*.

Acknowledgments

Truly grateful to my supervisors 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 MTiH); Mannheim, Germany (IMIM Industry & Innovation Day) and Heidelberg, Germany (AMR Hackathon 2018). From students and professors from some of the best European medical/engineering universities. AEICBAS 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] A.-L. Barabási, Network Science, United Kingdom: Cambridge University Press, 2016.

[17] 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].