Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Comparing community detection algorithms in graphs

Estudiante: Manuel Fandiño García
Dirección: Elena María Hernández Pereira

A Coruña, Tuesday 28th June, 2022.

To my family for always being there

Acknowledgements

To the people that helped and supported me along the journey and especially to my tutor, for having the greatest patience I have ever seen.

Abstract

The detection of communities in graphs has been a very discussed topic in recent years due to the raise of social networks. This topic is not a simple one, and as such, many different solutions have been proposed over the years to try and find communities in these networks. In this project we compare the results of different community detection algorithms when applied to a real live graph. This comparison is made using a set of quality metrics that will give us information of the algorithm's performance. With this information at hand we have seen that these algorithms perform very differently from each other and that the graph to which they are applied is a very important part in the results they return.

Resumo

A detección de comunidades en grafos foi un tema moi discutido nos últimos anos debido ao auxe das redes sociais. Este tema de estudo non é sinxelo, e por iso propuxéronse moitas solucións diferentes ao longo dos anos para tratar de atopar comunidades nestas redes. Neste proxecto comparamos os resultados de diferentes algoritmos de detección de cando se aplican a un grafo real. Esta comparación realízase empregando un conxunto de métricas de calidade que nos darán información do rendemento dos algoritmos. Coa informacion obtida, vimos que estes algoritmos teñen un comportamento moi distinto entre si e que o grafo ao que se aplican é unha parte moi importante nos resultados que devolven.

Keywords:

- Graph
- Community detection
- Louvain
- Modularity
- Label propagation
- Connected components

Palabras chave:

- Grafo
- Detección de comunidades
- Louvain
- Modularidad
- Propagación de etiquetas
- Componentes conexas

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many real life networks, such as, social networks, cities, stock exchanges and emails can be represented using graphs. This is usually done in such a way as to make them easier to read, interpret and work with, as graphs have been studied for a very long time.

One of the biggest problems faced when using graphs to represent real world scenarios is finding the communities hiding underneath these massive graphs. The finding of this communities has become very attractive recently because of the rise of social networks such as Twitter, Facebook or Instagram, and studying the communities that form in them would give an idea as to how information moves in our world and how relationships between different elements affect it.

For this purpose a great many community detection algorithms have been developed over time. The existence of so many community detection algorithms can be attributed to the fact that there exist as many graphs as there are algorithms to find communities in them. Each of them applies a different definition of what a community is and a different way to find them in the graph; as such, it becomes very hard to determine which one is more optimal for finding the communities in any given real world graph.

Taking all of this information into account, the need to know which community detection algorithm is better for any given graph has become increasingly more attractive. As such, we set out to compare the results of various community detection algorithms applied to a real world graph. These results will be compared using various quality metrics applied to them and a Ground Truth, which will give us an idea of which one performs better when applied to graphs of the same nature as the one used for our testing.

1.1 Objectives

The main objective of this project is to perform a comparison between different community detection algorithms in graphs on a public dataset, in order to identify their strengths and weaknesses. This objective can be divided in the following sub-objectives:

1. Attain knowledge of graph theory as well as community detection algorithms.
2. Analyze and determine which quality metrics are better to compare the results of the algorithms.
3. Analyze the use of these community detection algorithms in real world datasets.
4. Select a real world dataset upon which to execute the algorithms.
5. Apply these algorithms to a real world dataset and compare their results.
6. Determine which ones perform better under the selected real world dataset.

1.2 Outline

The document has been divided into five sections, each of which presents a part of the project.

The first section, Introduction, presents the motivations for this project and the objectives to be achieved.

The second section, Domain Description, presents terminology of the field of study of the project, as well as descriptions of community detection algorithms and metrics for the comparison of their results.

The third section, Neo4j and Dataset, describes the tools used to perform the tasks that comprise this project as well as the dataset used for testing. We also include in this section the methodology used and the cost estimates for this project.

The fourth section, Development, explains the process followed to prepare and import the Dataset into Neo4j.

The fifth section, Results and Discussion, deals with the tests performed on the dataset with the algorithms defined above, and a comparison of the results of these tests.

And finally, in the sixth section, Conclusions, we will discuss the conclusions found after the completion of the project and the work that could be done in the future.

Chapter 2

Domain Description

This chapter will cover the basics of the domain and will explain the different community detection algorithms used. To evaluate the quality of each of these algorithms, different quality metrics will be described.

2.1 Graph

A graph is defined as a pair G = (V,E), where V is a finite set of nodes or vertices and E is a set of pairs of elements, belonging to V, called edges. Edges can be directed or undirected depending on whether the pairs (u,v) are ordered or not. Directed edges are represented as lines ending in an arrow while undirected edges are represented as simple lines [1]. In some cases the edges have a value or weight that represents the strength of the relationship.

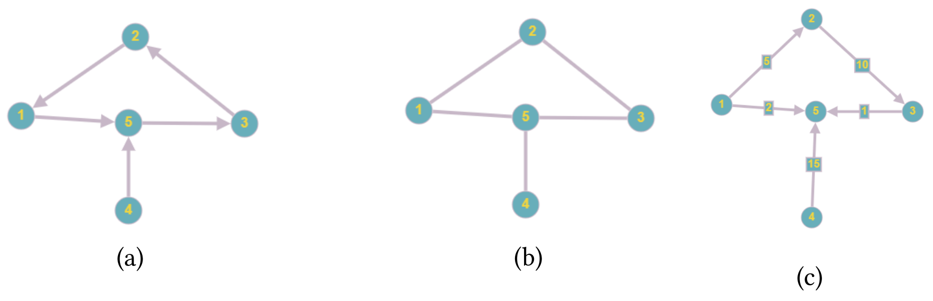


Figure 2.1: Example of (a) directed graph, (b) undirected graph, and (c) weighted directed graph

In addition to weight, both nodes and edges can incorporate information in the form of properties.

Among the properties of a graph, adjacency or neighborhood and incidence or degree stand out. Two nodes are said to be adjacent (or neighbors) if they are connected by an edge. The adjacency of a node u ϵ V is expressed as:

$$N(u) = v\epsilon V|(u,v)\epsilon E \tag{2.1}$$

The degree of a node is defined as the number of nodes adjacent to it and is formally expressed as:

$$g(u) = |N(u)| \tag{2.2}$$

Graphs are represented by adjacency lists and adjacency matrices. An adjacency list is a list containing all the nodes of the graph that maintains for each node a list of its adjacent nodes. The adjacency list is used when the graph is sparse and/or large, since the memory it occupies is proportional to the number of edges in the graph.

An adjacency matrix allows to represent an undirected graph G = (V,E) in a matrix defined as A = $[a_{uv}]$ of size n x n, where n is the total number of nodes, such that the entry a_{uv} will have a value of 1 if there is an edge between nodes u and v, and will have a value of 0 if there is no such edge. If the graph is weighted the matrix will store the value of the weight associated with the connection. The adjacency matrix will be used when the graph is dense, because it will occupy less memory since it does not depend on the number of edges but on the number of nodes.

In a graph, a path is a sequence of adjacent nodes. Formally, if a pair of nodes u, v ϵ V, not adjacent to each other, can be connected by a sequence of edges, a path is said to exist between them. A path is expressed as (u,z_1),(z_1,z_2),(z_2,z_3)...(z_{n-1},v) where $x_i \epsilon V$ and the pairs $(z_{i-1}, z_i)\epsilon E$. The length of such a path is the number of nodes that compose the sequence (n). The shortest path between two nodes is the one that travels the least number of edges to get from one node to another.

A cycle is a simple closed path, i.e. a path that starts and ends at the same node and where no nodes are repeated, except the first one.

For a graph G = (V,E) three nodes form a triangle when there are three edges that join them so that they form a cycle. Formally for three nodes u, v, z ϵ V, the triangle would be formed by the edges (u,v), (v,z) and (z,u) ϵ E.

2.2 Communities in graphs

Community detection aims to identify, using the topology of a graph, groups of densely connected nodes that share common characteristics within the set of nodes in the graph. Although this task has a specific goal, there is no universal definition of a community.

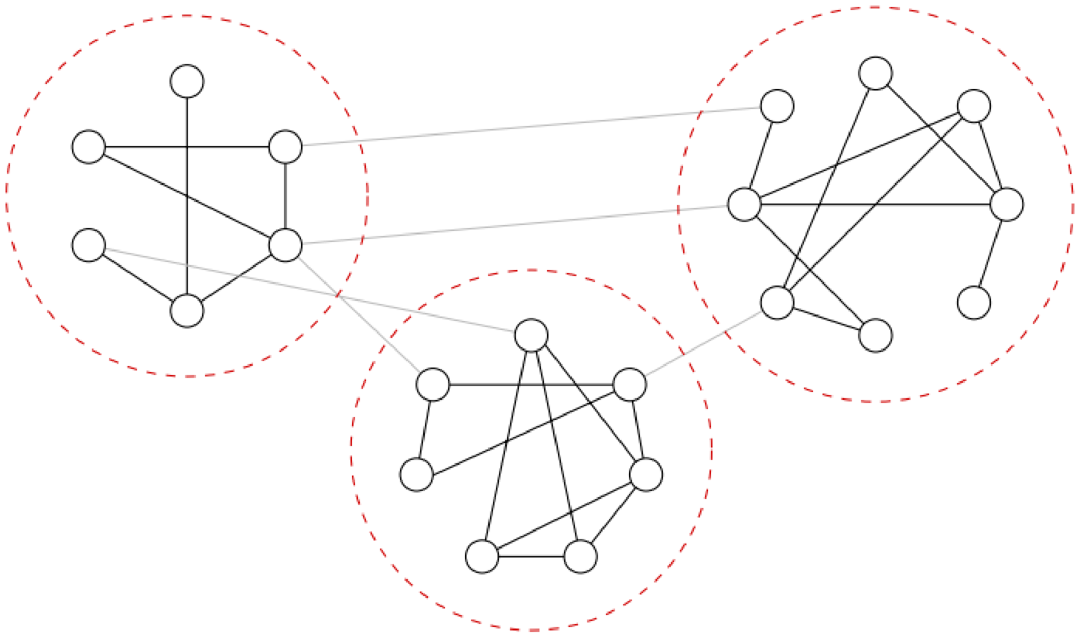


Figure 2.2: Example of a graph with communities

Beyond the fact that a community must be formed by nodes that share properties with each other and play similar roles, all the community detection algorithms published in the last decade present their own definition of community driven by the techniques and metrics used in the algorithm.

In recent years, various criteria have been presented to indicate what is considered a community, for example, in 2003 Moody and White [2] proposed that a community is formed in such a way that when a random node is removed from it, it does not disappear.

Another example of how to define what constitutes a community would be the definition given by Girvan and Newman [3], by which they proposed that a community must have more connections between the nodes that form it than with those outside of it.

And finally, another approach to defining a community could be comparing the similarity of the nodes in the graph and grouping them using that criteria. One example of this line of thinking would be Stanley Wasserman and Katherine Faust [4], who defined a metric by which one could compute the distance between two nodes and by taking that into account and the neighbors of the nodes, one could separate them into communities.

2.3 Measures for community detection

The task of detecting communities within a graph is so complex that many different measures have been created with the aim of finding the distinct characteristics used by the different algorithms to identify a set of nodes as a community. In this section we define a couple of the measures used.

2.3.1 Density

The density $\delta(G)$ is a measure that indicates the number of relations or edges that a graph G has.

A graph G = (V,E) is considered dense if the number of relations is close to the maximum number of possible relations, that is, a graph in which all the nodes are related to each other, which would be represented as:

$$|E| = n * (n-1) \tag{2.3}$$

A graph is considered sparse when the number of relations is low, close to an empty graph without edges [5].

Density is formally represented as:

$$\delta(G) = \frac{m}{n(n-1)} \tag{2.4}$$

where *m* represents the number of edges of the graph G and *n* represents the number of nodes of the graph.

2.3.2 Centrality

The centrality measure generally refers to the importance of the nodes in a graph and was first introduced by Alex Bavelas in 1948 [6]. There are different ways to calculate this measure which are explained below:

Degree centrality

The degree centrality determines the importance of a node with respect to other nodes, based on the number of relationships it has with them. This number of relationships represents the number of edges that start from the node and connect it to other nodes. This measure was defined by Linton Freeman [7].

Formally for a node u ϵ V, the degree centrality is defined as:

$$k_u = \sum_v (u, v) \; \forall v \epsilon V \tag{2.5}$$

where (u,v) ϵ E.

The average degree centrality of the graph is defined as:

$$c_g = \frac{1}{n} \sum_v k_v \forall v \epsilon V \tag{2.6}$$

Betweenness centrality

The measure of betweenness centrality proposed by Freeman [7], quantifies the frequency or number of opportunities in which a node acts as a connection within a path between two given nodes.

This type of centrality helps us to differentiate between nodes that are in the center of a community (those through which many shortest paths pass) and those that are on the outside of a community, or even those that do not belong to any community. Nodes with high betweenness, have a great importance in the community to which they belong and in the graph in general, since they can become the controllers of the data flow in the graph.

Formally, the betweenness centrality of a node z ϵ V, is defined as:

$$C_B(z) = \sum_{u,v \epsilon V} \frac{g_z(u, v)}{g(u, v)} \tag{2.7}$$

where g(u,v) represents the total number of shortest paths between nodes u and v, while $g_z(u, v)$ is the number of shortest paths between nodes u and v that pass through node z.

Eigenvector centrality

This measure, also called prestige score, measures the transitive influence or prestige of a node within a graph [8]. The value of this measure for a node is obtained through the number of paths that pass through that node, penalizing the longest distance connections with other nodes.

It is formally defined as:

$$C_B(u) = \frac{1}{\lambda} \sum_{v \epsilon Adj(u)} C_V(v); \;\; C_V(v) = \frac{1}{\lambda} \sum_{z \epsilon G} a_{v,z} C_V(z) \tag{2.8}$$

where a_{vz} is the entry in the adjacency matrix for u and v, $lambda$ is a constant and Adj(u) is the set of all nodes adjacent to u.

This measure is the basis of the PageRank measure [9], which is used in the Google search engine to optimize webpage searches.

2.3.3 Modularity

Modularity, formulated by Girvan and Newman [3], is defined as the difference between the number of links between nodes in a community and the number of links expected in an equivalent random graph. Formally, modularity (M) is defined as:

$$M = \sum_{i=1}^{k} (e_{ii} - a_i^2) \; where \tag{2.9}$$

$$e_{ii} = \frac{|(u,v) : u\epsilon V_i, \; v\epsilon V_i, \; (u,v)\epsilon E|}{|E|} \tag{2.10}$$

$$a_i = \frac{|((u,v) : u\epsilon V_i, \; (u,v)\epsilon E)|}{|E|} \tag{2.11}$$

where e_{ii} is the percentage of edges in community i, a_i is the percentage of edges with a node belonging to community i. The values returned by this metric are always between -1 and 1, where values close to -1 indicate a poorly formed community with too many outgoing edges, while values close to 1 indicate a well-formed community, with more edges connecting nodes inside than outside.

Although this measure is commonly used to compare the quality of the divisions in communities of different algorithms, it is also used as a measure in certain algorithms as an aid when incorporating nodes into a community.

2.4 Community detection algorithms

The fact that there doesn't exist a single universal definition for what constitutes a community in a graph gives rise to varied and very different algorithms that aim to give a definition for a community and divide the graph accordingly.

In this section we will give definitions to the community detection algorithms that we will compare in this project.

2.4.1 Louvain Algorithm

This algorithm, developed by Blondel and collaborators [10] is based on the optimization of the modularity measure. Its objective is to distribute the nodes into communities and then to evaluate them with respect to the density of relationships with the rest of the communities. The algorithm is divided into two phases:

1. Initially, as many communities as nodes are generated. For each node in each community, the modularity gain is determined by adding this node to the community of each of its neighbors. The node joins the community with the highest modularity gain.

2. A new graph is then created in which the nodes are the communities created in the previous phase and the process is repeated until convergence is achieved.

Below is the pseudocode of the Louvain algorithm, where **C** represents the identified communities, **M** the modularity, **max_iterations** defines the maximum number of iterations that the algorithm will execute before being forced to stop and **threshold** is the parameter that defines the stopping criteria of the algorithm.

```
Input: G = (V,E), threshold, max_iterations
Output: communities (C), modularity (M)

i = 0

while true and i < max_iterations do
  changes = false
  C = init(G) //each node isa community
  $M_{prev}$ = calculate_modularity(G,C)
  while true do
    foreach v $\epsilon$ V do
      calculate_modularity_between_neighbours(v)
      C = move_to_best_community(v)
        $M_{prev}$ = M
        M = calculate_modularity(G,C) //recalculate modularity
        if M - $M_{prev}$ <= threshold then
            break
        changes = true
    if changes then
        break
    i += 0

return C, M

```

The algorithm has a computational complexity of O(n*log(n)), where n is the number of nodes in the graph.

This algorithm can be especially useful in large graphs, since it uses modularity based on a heuristic, which has a low computational complexity, making the execution of the algorithm faster than other algorithms that do not use heuristics. In dense graphs, it allows the identification of community hierarchies, so that both communities and subcommunities can be studied.

This algorithm has been used in different fields: to predict the appearance of new species in a complex prebiotic graph [11], to track the growth of communities in dynamic social graphs [12] or to obtain partitions of social graphs [13].

2.4.2 Label Propagation Algorithm

This algorithm was proposed by Raghavan and collaborators [14], and its operation is based on using the structure of the graph to propagate labels that indicate to which community each node of the graph belongs. Instead of trying to optimize a measure such as modularity, labels are assigned to nodes depending on the most repeated labels among their neighbors.

The execution of the label propagation algorithm begins with the assignment to each node of a different label, which represents the community to which it belongs. In each iteration of the algorithm, a node is chosen and is assigned the majority label among its neighbors; the node then becomes part of the community represented by that label. This process will be repeated until all the nodes have the majority label among all their neighbors or until we reach a previously defined number of iterations that determines the end of the algorithm's execution.

For the initial assignment of the labels, we can use seeds assigned by the user, an heuristic, or they can be assigned randomly. These seeds will indicate which label each node will have in the first iteration of the algorithm. The heuristics used for label assignment can be based on different factors such as the weight of the nodes in the graph or the name of the nodes themselves.

The more the labels propagate through the graph, the more groups of densely connected nodes quickly converge to a single label, thus creating each of the communities. This can be a problem for graphs with densely connected nodes as a single label could quickly spread throughout the graph, dominating the rest and becoming the majority label for all nodes.

A pseudocode for this algorithm is described below. where **max_iterations** defines the maximum number of iterations that the algorithm will execute before being forced to stop:

```
Input: G = (V,E), max_iterations
Output: G
//initialize the labels of all nodes in the graph
for node u ϵ V:
  u.label = x
i = 0
end = false
repeat:
  randomize_nodes(V)
    for u ϵ V:
      u.label = mayority_neighbours(u)
    if labels_converge()
      end = true
    else
      i += 1;
until end or i > max_iterations

return G

```

The computational complexity of this algorithm is O(imn) where n is the number of nodes of the graph, m is the number of edges of the graph, and i is the number of iterations performed by the algorithm.

This algorithm can be used in large graphs as a first algorithm to find an initial distribution of communities. It is especially useful in weighted graphs as it makes the selection of the label to assign to each node simpler. This method can become extremely fast as it can be parallelized, thus dramatically increasing the speed of assigning labels to nodes.

Among the applications of this algorithm are the assignment of names to people in television news [15], the prediction of defects in software [16] or the identification of overlapping communities [17].

2.4.3 Strongly Connected Components Algorithm

This algorithm was proposed by Sambasiva Rao Kosaraju but published by Micha Sharir in 1981 [18]. Its objective is the identification of fully connected nodes representing a community. This means, nodes form a community if there is a path between all pairs of nodes [19]. The algorithm makes use of depth-first search to identify the different connected components.

A pseudocode of this algorithm is presented below:

```
Input: G = (V,E)
Output: components

STACK = create_stack()
//start arrays with enough size to fit all the nodes
visited[] = startVisitedArray(n)
components[] = startArrayComponents(n)
numComponents = 0

for all u ϵ V
  if !visited[u]
    depth_first_search(u)

depth_first_search(u):
  visited[u] = true
  for all neighbor of u
    if (!visited[neighbor])
      depth_first_search(neighbor)
  STACK.push(u)


reverse_depth_first_search(u):
  component[numComponents].add(u)
  visited[u] = true
  for all reverse_neighbor of u //we take the relationships
backwards
    if (!visited[reverse_neighbor])
      reverse_depth_first_search(reverse_neighbor)


main():
  for all u ϵ V
    if !visited[u]
      depth_first_search(u)

  for all u ϵ V //empty visited array
    visited[u] = false
  end for

  while !STACK.isEmpty()
    u = STACK.pop()
    if(!visited[u])
      component = reverse_depth_first_search(u)
      components.add(component)
      numComponents++

  return components
```

The algorithm with this implementation has a complexity of O(n+m) where n represents the number of nodes of the graph and m the number of edges.

This algorithm is commonly used to check that the graph does not contain islands (components that have no connection with the rest of the graph), which provides a first approximation of the distribution of communities in the graph at the simplest level. These communities can be used as inputs for other algorithms to perform a more in-depth analysis of the communities.

Some examples of the use of this algorithm are: analysis of the flow of ecological subsystems [20], mathematical analysis of the asymptotic behavior of a graph [21] or the automatic discovery of secondary targets in reinforced training [22].

2.4.4 Weakly Connected Components Algorithm

This algorithm was described by Robert Endre Tarjan in 1974 [23]. The goal of the algorithm is to find weakly connected components in an undirected graph, which are defined as sets of nodes connected to each other by one or more edges and which have at least one path by which they can reach any other node in the component. This algorithm uses breadth-first search to find components, as opposed to depth-first search used to find strongly connected components.

Pseudocode for this algorithm is included below:

```
Input: G = (V,E)
Output: components

//start arrays with enough size to fit all the nodes
visitedNodes[n] = startArrayNodes()
component[n] = startArrayNodes()
components[n] = startArrayComponents()

for all u $\epsilon$ V
  if !visitedNodes[u] then
    connectedNodes = breadthFirstSearch(u)
    visitedNodes.add(connectedNodes)
    component.add(connectedNodes)
  end if
  components.add(component)
  emptyArray(component)
end for

return components
```

This implementation of the algorithm has a computational complexity of O(n+m), where n is the number of nodes in the graph and m is the number of edges.

The most common use for this algorithm is in the preprocessing of graphs before applying more complex algorithms, in order to check if the graph in question is completely connected or if it is divided into different subgraphs that are not connected between them. This is especially useful for algorithms that require the graph to be fully connected, as it helps us to know if they can be applied to the graph or not.

Some examples of the use of this algorithm are: characterization and citation graph mining of computer literature [24] and duplicate Database detection [25].

2.5 Quality metrics

To determine the quality of the communities identified by the different algorithms, a set of metrics are used. These metrics will allow us to compare the mentioned algorithms and to determine which one best represents the behavior of the graph in terms of communities. Among these metrics, modularity (described in section 2.2.3), performance, clustering coefficient, coverage, and a set of structural metrics will be used.

2.5.1 Performance

This metric consists of the count of the edges between nodes of the same community divided by the pairs of nodes that are composed of a node of that community and a node that does not belong to it, which are not joined by an edge. Formally it is defined as:

$$perf(G) = \sum_{i=0}^{|C|} \frac{f(C_i) + g(C_i)}{\frac{1}{2}n(n-1)}, where \tag{2.12}$$

$$f(C_i) = \sum_{u,v} \{(u,v) \epsilon E | u, v \epsilon C_i\} \tag{2.13}$$

$$g(C_i) = \sum_{u,v} \sum_{i>j} |\{\{u,v\} \notin E | u \in C_i, v \in C_j\}| \tag{2.14}$$

where G is the graph to be analyzed, |C| is the number of communities, C_i is a community of the graph, $f(C_i)$ computes the number of edges between nodes of the community and $g(C_i)$ counts the number of pairs containing a node that belongs to that community and a node that doesn't, which are not joined an edge.

The values returned by this metric are in a range between 0 and 1, where higher values indicate a distribution of communities in which there are many more edges between nodes within communities than edges between a node that belongs to that community and one that doesn't, which makes it a good distribution of communities. Values closer to 0 indicate a distribution that contains many edges connecting nodes from different communities, which indicates that it is not a good distribution.

One problem with this metric is that complex graphs tend to be more sparse, which makes the component g(C_i) tend to be larger and thus causes the equation to return indiscriminately larger values. This can cause a community distribution to return a high value, which would indicate good quality, independent of the quality of the community.

2.5.2 Clustering coefficient

This metric quantifies the strength of a node's connection with its neighbors. Its value for a given node is obtained from information on the number of triangles and the number of paths of length two of which the node is a part.

Formally defined as:

$$C_c = \frac{N_{\Delta}c}{P_2c + N_{\Delta}c} \tag{2.15}$$

Where $N_{\Delta}c$ is the number of triangles in the community and P_2c is the number of paths of length 2 in the community.

When the value of this metric is close to 1 it indicates that the community contains strongly connected nodes, while if it is close to 0 it indicates that the nodes in the community have few relationships with each other.

It is common to calculate the mean of this metric for all of the nodes in a community and using it as the value of the metric for that community. It is represented as avg C_c

2.5.3 Coverage

The coverage metric is defined as the ratio of the total number of edges between nodes in the same community to the total number of edges in the graph.

$$Cov(G) = \frac{m_C}{m} \tag{2.16}$$

where m_C is the total number of edges between nodes of the same community and m is the total number of edges of the graph.

Intuitively it can be seen that this metric will have a value of 1 when there are no edges between nodes of different communities, while it will be closer to zero the more edges join nodes of different communities. Therefore, this metric helps us to see how far apart are the communities found by a detection algorithm.

2.5.4 Structural metrics

There are a set of metrics that allow to analyze the structure of the communities identified by the community detection algorithms. These metrics are called structural metrics and refer to the size of the communities. Among their utilities is that of identifying dominant communities, large communities with a high standard deviation value, that eclipses all other communities.

The structural metrics chosen for this project are: average, lowest and highest number of components, which refers to the number of nodes in the communities found with the community detection algorithms; and the standard deviation of the number of components.

We chose these metrics because they compliment the information given by the other metrics chosen nicely and help us draw better conclusions from the results obtained.

Chapter 3

Neo4j & Dataset

Although there are several options for working with graphs, the choice of Neo4j as the development tool for this project was motivated by its ease of use, its complete documentation, and its large community of users.

In this chapter we describe the Neo4j Graph Data Platform, as well as the dataset on which the detection of communities has been performed, the methodology followed in this project and the cost estimates.

3.1 Neo4j

Neo4j Graph Data Platform [26] is a suite of applications created by Emil Eifrem in 2007 with the aim of creating a non-relational database [27] more efficient than the alternatives that existed at the time and the platform includes the Neo4j Graph Data Science [28] an analytics workspace for graph data, the graph visualization and exploration tool Bloom [29], the Cypher query language, and numerous tools, integrations and connectors to help developers and data scientists build graph-based solutions with ease.

At the core of the Neo4j Graph Data Platform is the Neo4j Graph Database, a native graph data store built from the ground up to leverage not only data but also data relationships. Unlike other types of databases, it connects data as it's stored, enabling queries much more complex to be executed much more quickly.

Finally, Neo4j Bloom is an easy-to-use graph exploration application for visually interacting with Neo4j graphs. Bloom gives graph novices and experts alike the ability to visually investigate and explore graph data from different business perspectives.

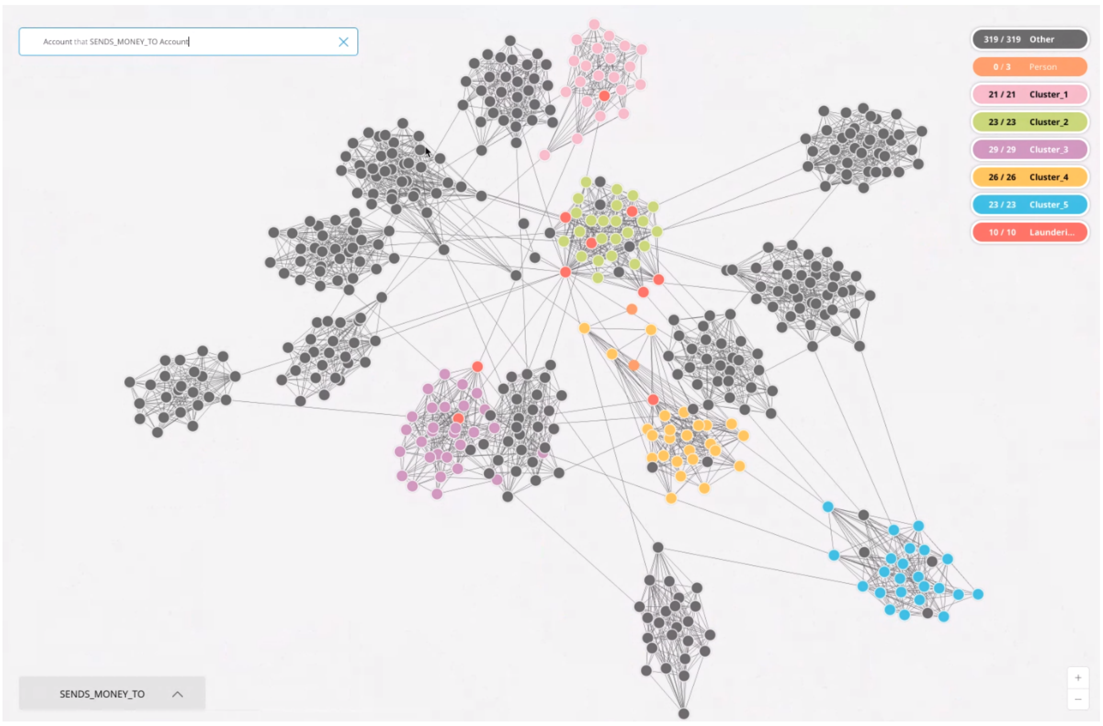


Figure 3.1: Example of graph representation in Neo4j Bloom

The operation of Neo4j involves the creation of graphs from a data set and the execution of algorithms on the created graph.

The query language used by Neo4j is Cypher[30], created in 2011 by Andres Taylor. This language bases its syntax on ASCI-art, making the language very visual and the queries easy to formulate. The nodes in this language are represented in brackets, while the relationships are represented with an arrow (directed or not) with the type of the relationship in square brackets. An example of a query in this language would be:

```
MATCH (u:Item)-[:RELATED]-(v:Item)
WHERE u.comm = v.comm
RETURN u.id
```

This example query contains the three most common keywords in Cypher, which are: MATCH, used before describing the search pattern for nodes, relations or a combination of both. In this example, the nodes u and v of type Item connected with an edge of type RELATED; WHERE, that adds more restrictions to the search. In our example it compares that the comm attributes of the nodes u and v are equal; and RETURN, that defines what data will be returned and its format. For our example it will return the id attribute of the node u.

3.2 Data Set

Several repositories specialized in network datasets suitable for community analysis were consulted for the choice of the dataset, such as: Colorado Index of Complex Networks[31], Institute for Web Science and Technologies (WeST) [32], Stanford Network Analysis Project (SNAP) [33] or Canadian Institute for Cybersecurity[34].

The repository of choice for this final degree project was SNAP, as it contains a large number of high quality datasets. More specifically, within SNAP we have focused on the Stanford Large Network Dataset Collection which, as its name suggests, contains large network datasets.

The chosen dataset is called email-Eu-core network[35], core of a larger set, named email-EuAll[36], and represents the emails exchanged by members of a European development institution between October 2003 and May 2005. This set was collected for a study by Jure Leskovec, Jon Kleinberg and Christos Faloustsos.[37] on how graphs evolve over time.

The email-Eu-core set contains 1005 nodes, representing the employees of the institution, conveniently anonymized; and 25571 edges, representing the emails exchanged between members of the organization. These employees belong to 42 different departments within the institution.

The following figure represents the whole graph as seen in Neo4j Bloom, showing the nodes and relationships that form it:

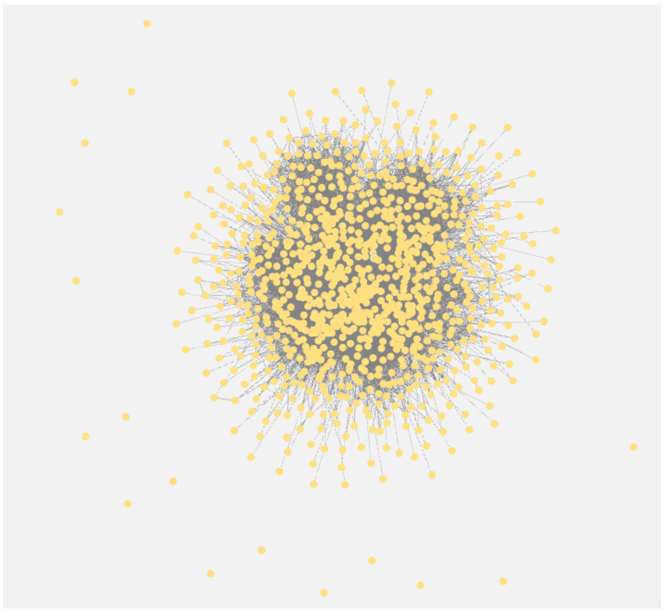


Figure 3.2: Representation of the Dataset in Neo4j

As it can be seen, if we want to view the whole graph, the information is lost due to the fact that there are so many relationships and nodes to be presented at a given time.

In the next figure we show just a part of the Dataset; more precisely, we represent the relationships that join three nodes with the rest of the graph, incoming and outgoing:

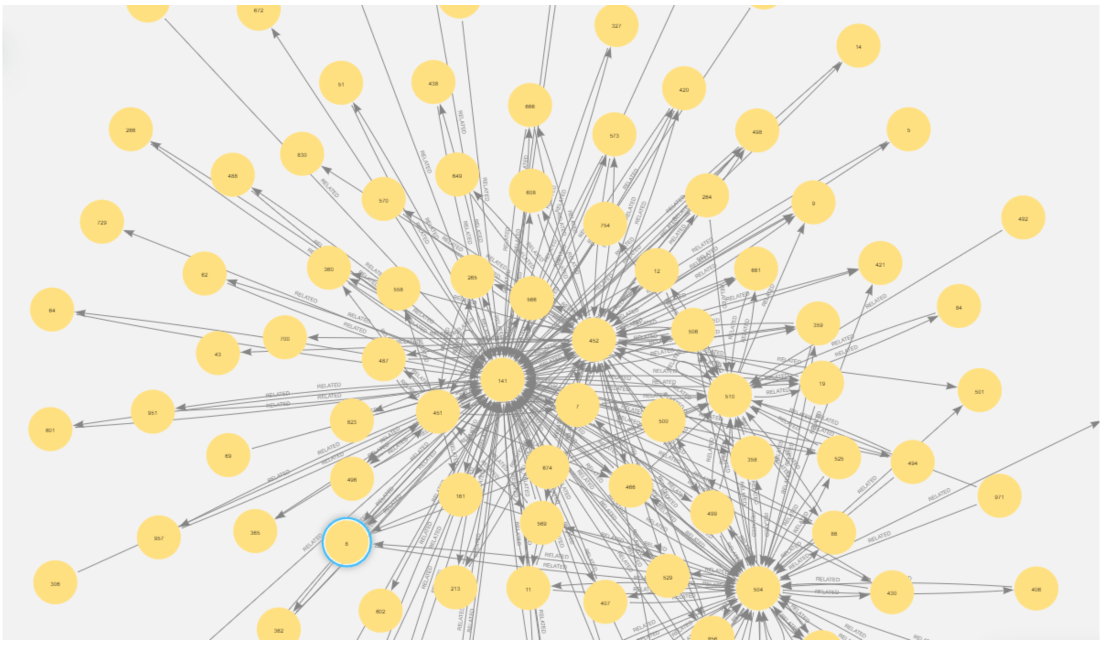


Figure 3.3: Representation of part of the Dataset in Neo4j

Here we can see that even though we only represent the relationships that concern just three nodes, we get a huge amount of outgoing and incoming relationships.

3.3 Methodology

The methodology followed in this project was the spiral methodology, first defined by Barry Bohem [38]. This methodology is one of the most important ones for software development and is based on the definition of different tasks that will comprise a phase of work. When all the tasks are completed the results are evaluated, the risks assessed, and it is decided if the next phase begins or not if the results are satisfactory. In our case, the risks to be assessed are purely determining if a determined algorithm can be run efficiently with our current graph and if some changes can be made to adapt it. Typically, each phase is divided into four tasks: determine the objectives, risk analysis, develop and test; and planning.

For our purposes the planning task has been divided into the following stages:

1. Acquisition of knowledge about graph theory and the different community detection algorithms that exist.

2. Analysis of different community detection algorithms and selection of the most relevant ones.

3. Analysis of different public datasets and selecting the most suitable one for the community detection task.

4. Analysis of the quality metrics of community detection algorithms and selecting the most appropriate for the comparison.

5. Implementation of community detection algorithms and their quality metrics.

6. Testing and evaluation of the selected algorithms on the basis of the metrics chosen above. This stage will be divided into four separate task, one for each algorithm we will test.

    6.1. Test and evaluate the Louvain Algorithm

    6.2. Test and evaluate the Label Propagation Algorithm

    6.3. Test and evaluate the Strongly Connected Components

    6.4. Test and evaluate the Weakly Connected Components

The selection of the working dataset was done by searching for a set of relevant size taking into account the chosen community detection algorithms.

The different iterations of the work were performed for the implementation, testing, and evaluation phases of the community detection algorithms. In each iteration, the implementation of an algorithm, its testing and the quality metrics were evaluated.

We understand by implementation of an algorithm, the adaptation of an existing algorithm to work on the selected dataset. The testing phase focuses on the execution of the algorithm with different configuration parameters.

The last iteration consisted of evaluating the results obtained in terms of the communities identified and their quality.

The following figure depicts a simple Gantt chart of the organization of these phases and how they where executed.

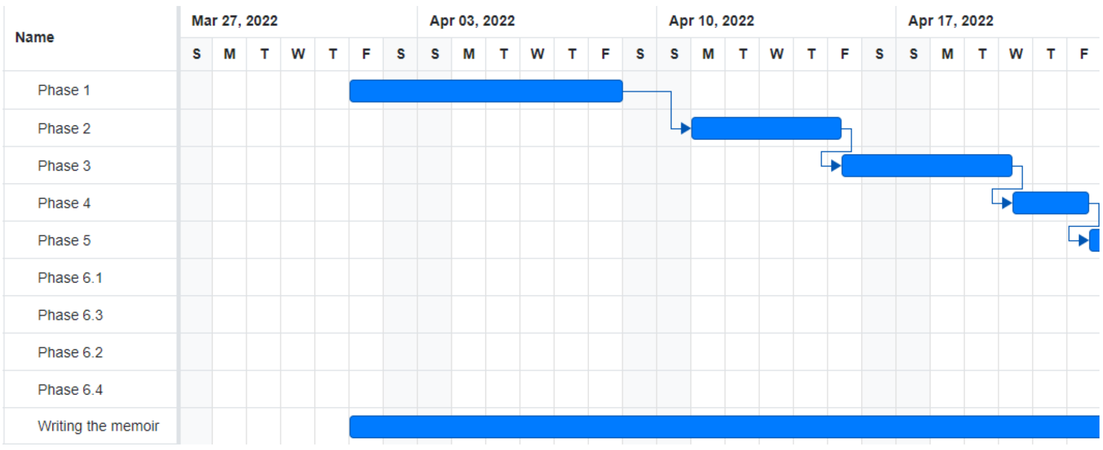


Figure 3.4: First half of theGantt chart

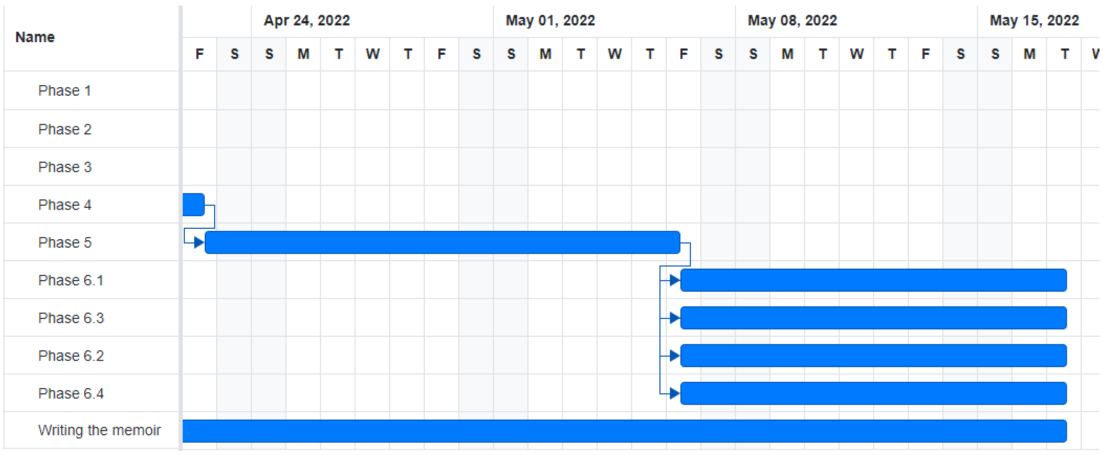


Figure 3.5: Second half of theGantt chart

3.4 Cost Estimates

To make an estimate of the cost of this project, only one developer will be taken into account. The developer in question is tasked with importing the dataset into a database, conducting the execution of the algorithms, applying the quality metrics to those results, and drawing conclusions from those quality metrics.

The time cost is calculated using a salary of 15€/hour and the estimation and associated costs are represented in the following table:

Phase	Aprox. effort (hours)	Cost (€)
1	48	720
2	33	495
3	24	360
4	21	315
5	75	1125
6	60	900
	Total: 261	3895

Table 3.1: Personnel cost estimate

For the realization of this project computers and other resources were necessary, all of them having costs associated with them that are represented in the following table:

Element	Cost
Internet	75
Electricity	100
Development equipment	1500
Total:	1675

Table 3.2: Resource cost estimate

The total cost comes to an estimate of 5570€.

Chapter 4

Development

In this chapter we will explain the preparation of the data for further processing, the creation of the corresponding graph as well as the creation of the existing relationships, and the configuration of the different community detection algorithms that will be studied.

4.1 Dataset setup

From the email-Eu-core [35] dataset we have used all of the nodes and relationships present within it for the testing of the algorithms used in this project. More over we have used the community distribution described in this dataset to setup a Ground Truth to be used as reference for the quality of the results we get.

The preparation of the dataset to be imported into the database used for the testing of the different algorithms was rather simple, as the data was already anonymized and put into a format suitable to be imported into Neo4j.

The changes made to the data were to add names to the columns of data in the files that represent the relationships between nodes and the nodes that comprise each community as defined by the Ground Truth; and transforming those files to CVS format. This is done to make importing them to Neo4j much easier.

It's necessary to import the dataset into a Neo4j database because all of the algorithms we want to test are executed on a graph created from a database and their results are saved, also, on the database. Another reason to import the dataset into a database is the simplicity with which we can execute queries on the nodes and relationships; and also the ease with which we can implement the different quality metrics.

4.2 Importing the dataset

In order to import the dataset into our Neo4j database, we had to follow the steps detailed bellow:

First, we created an index on the id property of the nodes. This helps with the efficiency of the queries we will execute to create the graph, the import of the dataset and with the execution of the algorithms; but it also affects the size of the database and write times since it is creating a redundant copy of the property.

The index is created with the command bellow:

```
CREATE INDEX FOR (c:Item) ON (c:id)
```

Next, we will import all of the nodes that will comprise the database, as well as the community they belong to in the Ground Truth, which we will write as a property in the node. The community to which they belong is given to us in one of the files that comprises the dataset and lists all of the employees in one column and their respective department in the next.

The process is done using the following command:

```
LOAD CSV WITH HEADERS
FROM "file:///email-Eu-core-department-labels.csv" AS row
CREATE (:Item {id:row.NodeId,Department:row.Department});
```

The next task is to import the relationships that connect the nodes of the dataset. This is done using the other file in the dataset that represents in the first column the employee that sent the email is represented and in the next column the employee that received it.

The relationships are created in the following command:

```
LOAD CSV WITH HEADERS FROM "file:///email-Eu-core.csv" AS row
MATCH (p1:Item {id:row.FromNodeId}),(p2:Item {id:row.ToNodeId})
CREATE (p1)-[:RELATED]->(p2);
```

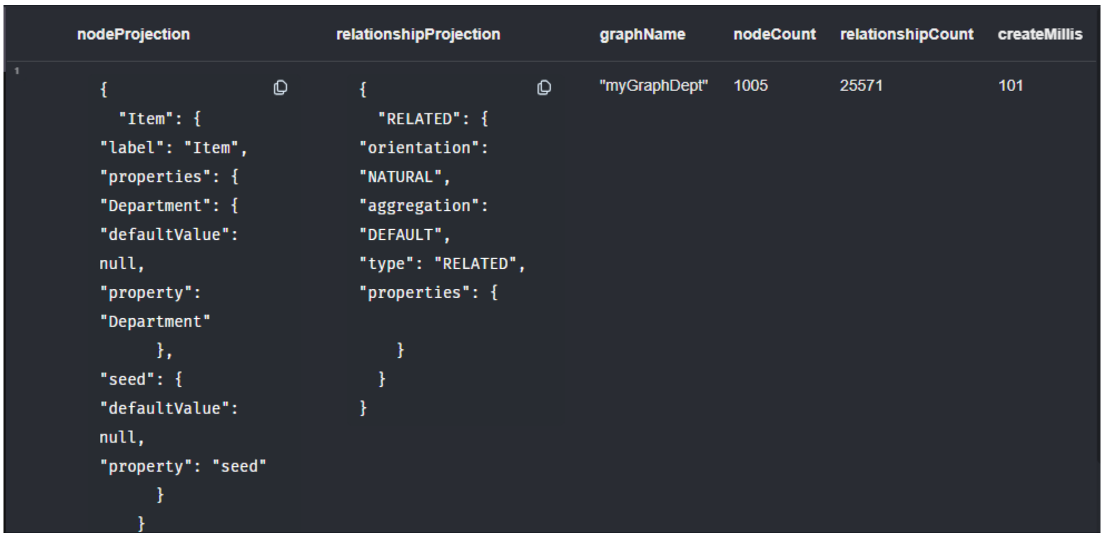
We will also need to assign to each node a random label that we will need later in the execution of the Label Propagation Algorithm. As Neo4j doesn't have the option for creating random properties, we created a small Python program to create a file with two rows, one with the nodes in order and the next with their random labels. This file is then imported into our database with the next command:

```
LOAD CSV WITH HEADERS FROM "file://email-eu-core-randomLabels.csv"
    AS row
MATCH (u:Item {id:row.nodeId})
set u.seed = row.label
```

Finally we will create the graph upon which we will execute the algorithms with the command:

```
CALL gds.graph.create(
    'myGraph',
    'Item',
    'RELATED',
    {
      nodeProperties: ['Department','seed']
    }
)
```

After creating the graph we can see this output in Neo4j informing us of the properties of the graph created:



```
nodeProjection          relationshipProjection     graphName       nodeCount  relationshipCount  createMillis
1
{                       {                          "myGraphDept"   1005       25571              101
  "Item": {               "RELATED": {
"label": "Item",        "orientation":
"properties": {         "NATURAL",
"Department": {         "aggregation":
"defaultValue":         "DEFAULT",
null,                   "type": "RELATED",
"property":             "properties": {
"Department"
      },                    }
"seed": {                 }
"defaultValue":         }
null,
"property": "seed"
      }
    }
```

Figure 4.1: Output of the graph creation process

In the figure above we can see the information given to us when creating a graph. It shows the name and properties of the nodes added to the graph in the nodeProjection column, the name and properties of the relationships added to the graph in the relationshipProjection column; the name of the graph, the number of nodes and relationships it has, and the time it took to create in milliseconds.

As was explained in section 3, we need to asses if each algorithm can be run efficiently with our graph. After due review of our graph and the algorithms to be executed, we found the execution time and results to be satisfactory and decided to not make any changes to either the graph or the algorithms to be run.

Chapter 5

Results and discussion

This chapter will detail the tests performed with the different algorithms; present the results of the algorithms as well as the quality metrics, and compare the community detection algorithms in view of the values of these metrics. The tests of the algorithms were divided into four iterations, one per algorithm, as well as a first establishing the values of the parameters we will use to make the testing and applying the quality metrics to the Ground Truth provided to us in the Dataset.

5.1 Algorithm Configuration Parameters

Different parameters and values have been used for the execution of the algorithms. The parameters used in the execution of each algorithm are described below:

The Louvain algorithm is configured using the following parameters:

1. Iteration Limit: this parameter determines how many iterations the algorithm will perform. Its purpose is to stop its execution if convergence is not reached.
2. Tolerance: this parameter determines the minimum increase in the modularity measure to continue with the execution of the algorithm.
3. Maximum Levels: this parameter determines the maximum number of divisions made to the graph that are later condensed.

The configuration of the Label Propagation algorithm is done through the following parameters:

1. Iteration Limit: as for Louvain's algorithm, this parameter determines the maximum number of iterations the algorithm will perform.
2. Label Property: this parameter indicates which property of the network nodes will be used as the initial value of the labels assigned by the algorithm.

The two Connected Component algorithms do not require the initial configuration of any parameters.

5.2 Ground Truth

In order to determine the goodness of the results of the different community detection algorithms, it is necessary to compare them with the original distribution of communities in the Dataset, that is, the Ground Truth. To do this, it is necessary to calculate the quality metrics on this distribution and compare these metrics with those calculated on the results of the different community detection algorithms.

The following table presents the results of the quality metrics on Ground Truth:

Quality Metric	**Value**
Performance (Perf)	0.95
Coverage (Cov)	0.36
Modularity (Mod)	0.68
Average clustering coefficient (avg C_c)	0.32
Number of Communities (# C)	42
Average number of components (avg N_C)	24
Largest Number of Components (max N_C)	109
Lower Number of Components (min N_C)	1
Standard deviation (μ N_C)	24.22

Table 5.1: Quality metric results for the Ground Truth

Analyzing the values in the table, the high value of the performance metric stands out, due to the existence of a large number of small communities, which are isolated or poorly connected with the rest of the communities. It is also important to highlight the low value of the coverage metric compared to the performance value. This is due to the fact that the number of edges of the network linking nodes from different communities is high. The value of the mean and standard deviation of the number of components of the communities allows us to determine the size range of the communities, which is between 1 and 48 nodes.

The analysis of all the metrics as a whole indicates that most of the communities in Ground Truth are small in size and not very connected to the rest of the communities, while there are other much larger communities that are connected to each other.

5.3 First Iteration

The first iteration will be performed with Louvain's algorithm, which will be executed with the following values for the parameters described in section 4.1:

Execution	1	2	3	4	5	6
Iteration Limit	1000	1000	1000	1000	1000	1000
Maximum Levels (maxL)	10	10	10	1000	1000	1000
Tolerance (T)	10^{-3}	10^{-5}	10^{-7}	10^{-3}	10^{-5}	10^{-7}

Table 5.2: Parameter values for Louvain algorithm

In Neo4j the execution of the algorithms will be done using the Graph Data Science library through commands, as shown below:

```
CALL gds.louvain.write('myGraph', { writeProperty: 'community',
    maxIterations: 1000, tolerancia: $10^{-3}$, maxLevels: 10 })
YIELD communityCount
```

where:

- **write** is the execution option that writes on a property of the database nodes to which community they belong.
- **myGraph** indicates the previously created graph on which the algorithm is executed.
- **writeProperty** indicates on which property of the database nodes the result of the algorithm will be written

The rest of the parameters are the concrete configuration parameters of the Louvain algorithm.

The algorithm provides the number of communities identified through the output parameter communityCount.

The results of the six different executions of the Louvain algorithm are presented below:

maxL	T	t (ms)	Perf	Cov	Mod	avg C_c	# C	avg N_C	max N_C	min N_C	μ N_C
10	10^{-3}	301	0.89	0.59	0.43	0.09	122	7	199	1	31.42
	10^{-5}	515	0.89	0.59	0.43	0.10	121	8	198	1	29.74
	10^{-7}	326	0.90	0.57	0.42	0.07	179	5	127	1	20.87
1000	10^{-3}	1526	0.87	0.59	0.43	0.3	38	26	221	1	55.17
	10^{-5}	1213	0.87	0.59	0.42	0.42	27	37	225	1	63.24
	10^{-7}	1879	0.86	0.59	0.43	0.42	27	26	217	1	52.80

Table 5.3: Louvain algorithm results

where T is the execution time of the algorithm in milliseconds.

The results obtained are very similar to each other except for the clustering coefficient, which shows variations between 0.07 and 0.42. The highest values coincide with high values of the parameter *Maximum number of levels* (maxL), which indicates that the increase in the value of this parameter implies an increase in connections and their density within the communities. This implies an improvement in the quality of these communities with respect to communities identified with lower values of this parameter.

With small values for the maxL parameter, many communities with a single node are identified, resulting in values close to 0 for the clustering coefficient. In turn, the mean of the number of nodes per community (μ N_C) reflects these results with low values.

For the remaining metrics, the following conclusions can be drawn:

1. The Performance (Perf) presents high values, close to 1, which indicates that all the communities have many more edges relating nodes inside the community than outside.

2. The Coverage (Cov) presents intermediate values, indicating that slightly more than half of the relationships in the entire network are formed by nodes of the same community.

3. The Modularity (Mod) presents values of 0.4, which indicates that the nodes of the communities have more relationships with nodes of the same community, but even so, they have a considerable number of relationships with nodes outside the community.

The difference between the Performance and Coverage metrics, which should normally present similar values, is due to the fact that there are in all iterations a high number of single-node communities that were not combined with any other because they did not provide sufficient modularity gain. These single-node communities cause an increase in the value of the Performance metric by having a large number of nodes outside the community to which they are not connected.

We can also see that, when using a lower value of the Maximum Levels parameter, the number of communities found is generally much higher than when using a higher value. This change is due to the fact that many of the single-node communities mentioned above have been combined together to create larger communities.

The communities found by the algorithm when using a higher value of the Maximum Number of Levels parameter have a higher standard deviation than those found with a lower value. This is due to the fact that there are still quite a few communities formed by a single node.

The algorithm execution time depends on the Maximum Levels parameter, since as more divisions are allowed, the number of iterations of the algorithm increases, thus increasing the algorithm execution time.

With respect to Ground Truth, it can be seen that executions with a higher Maximum Number of Levels value give community distributions closer to those of Ground Truth with similar values in the quality metrics.

5.4 Second Iteration

The second iteration will be performed with the Label Propagation algorithm, with the following values for the parameters described in section 4.1:

Execution	1	2	3
Maximum Iterations	1000	1000	1000
Label Property	Automatic	Ground Truth	Random

Table 5.4: Parameter values for the Label Propagation algorithm

The values of the Label Property parameter correspond to automatic, so that the algorithm chooses the label to propagate; Ground Truth value, in which the label corresponds to the one presented by the Dataset; random value, for which we assign each node a label randomly.

The results of the Label Propagation algorithm executions are presented below:

Labels	t (ms)	Perf	Cov	Mod	avg C_c	# C	avg N_C	max N_C	min N_C	μ N_C
Automatic	38	0.95	0.93	0.71	0.01	183	5	769	1	54.86
Ground Truth	30	0.88	0.93	0.72	0.07	36	28	760	1	126.34
Random	89	0.98	0.93	0.71	0.01	179	5	730	1	54.73

Table 5.5: Label Propagation results

where the Label column represents the value of the corresponding parameter.

These results show how the use of the Ground Truth value as a Label Property leads to a lower number of communities, close to the reference value (42). The rest of the metrics present similar values except those related to the number of communities, such as the mean number of nodes per community and the standard deviation. This indicates that the distribution of the nodes in the communities is similar in the three cases, although in one of them there is a lower number of communities.

This is reflected in the values of the last four columns of Table 4.5, which present a small mean size and a high standard deviation, indicating that most of the communities found have a size of 1 or close to 1, while there is a single large community comprising most of the nodes in the network.

More generally, the following conclusions can be drawn:

1. The Performance (Perf) has a very high value. This is because there are many isolated communities with a single node that will therefore have a high performance value.

2. The Coverage (Cov) presents a high value, which means that a large number of the edges of the network are within the identified communities. This is due to the fact that, since there is a community of such a large size, many of the edges of the network will be found within this community, which will greatly increase the value of this metric.

3. The Modularity (Mod) presents a high value because there is a large community that eclipses the rest and contains many relationships within the same community. Likewise, the number of relationships with nodes of another community is scarce, which results in this high modularity value that is not representative of the quality of the partition.

4. The Clustering Coefficient (C_c) presents low values for any of the executions, although its value increases slightly in the case of using as reference label that of the original Dataset. This is because communities that are not the largest return a minimum clustering coefficient value, which directly affects the value of the mean.

The execution time of this algorithm was very low for the different executions due to the low complexity of the algorithm and the composition of the network.

Comparing the results of this algorithm with those of the Ground Truth, we can see that we would only come close to the number of communities found in this one using the Ground Truth tags. But even so, the quality metrics do not come close in any of the executions to the results achieved with the Ground Truth.

This is due to the fact that in the Label Propagation Algorithm parts of the graph with many connections tend to have the same label, as the labels propagate much faster in those areas; and as we have seen, the graph is very dense with relationships. This results in a very large community comprised of all the nodes that have many connections with each other and then several small communities of one node that don't have relationships with other nodes.

5.5 Third Iteration

The third iteration will be performed with the Strongly Connected Components algorithm:

t (ms)	Perf	Cov	Mod	avg C_c	# C	avg N_C	max N_C	min N_C	μ N_C
41	0,77	0.97	0.72	0.004	203	5	803	1	56.29

Table 5.6: Strongly Connected Components Algorithm results

As can be seen, these results are similar to the Label Propagation algorithm ones, since a large community and a large number of communities with a small number of nodes are identified. This can be seen in the last four columns of the table.

Both Performance as well as Coverage and Modularity have high values due to the existence of a large size community that overshadows the results of these metrics in smaller size communities. While the clustering coefficient has a very small value due to the fact that there are many minimum size communities that also have a minimum value of the clustering coefficient.

Due to the simplicity of the Strongly Connected Components algorithm, the execution time is very low.

Comparing the quality metrics and the number of communities found with this algorithm with those found in Ground Truth, we can see that the results are very different, especially in the metrics of coverage, clustering coefficient and size of the largest community.

These results can be explained due to the fact that this algorithm only takes into account the direction of the relationships and if we can make paths from one node to another. In the case of our graph we can see that many nodes can be reached from one another in both directions due to the results of this algorithm.

5.6 Fourth Iteration

The fourth iteration will be performed with the Weakly Connected Components algorithm.

The results of running this algorithm are as follows:

t (ms)	Perf	Cov	Mod	avg C_c	# C	avg N_C	max N_C	min N_C	μ N_C
10	0.13	1	0.75	0.04	20	50	986	1	220.25

Table 5.7: Weakly Connected Components Algorithm results

As can be seen in these results, there are 20 communities of unconnected nodes with the rest of the network which, as in the previous algorithms, correspond to a large community containing most of the nodes of the network and several smaller communities with the rest of the nodes of the network.

With respect to quality metrics, the following conclusions can be drawn:

1. The Coverage (Cov) of the graph indicates that the distribution of communities is correct since the Weakly Connected Components, by definition, do not have edges joining them, so there will only be edges between nodes within the communities themselves.

2. The Performance (Perf) presents a very low value due to the fact that the number of communities found is very low and as such the component that counts the number of edges that don't connect to nodes outside of the community is in turn much lower.

3. The Modularity (Mod) has a value similar to that of the Strongly Connected Components algorithm, since the component that contains most of the nodes has a high modularity that makes the global modularity also high.

4. The Clustering Coefficient metric (C_c) presents a very low value since there are a large number of communities formed by a single node, which makes the average very low.

As can be seen, the execution time of the algorithm is the shortest of all the algorithms we have tested since it is also the simplest.

Finally, comparing these results with those of Ground Truth, we can see that they are not similar either in the number of communities or in the quality metrics.

Similarly to the Strongly Connected Components Algorithm, this algorithm only takes into account if a path can be drawn between two nodes, which explains the results found and how they are not similar to those of the Ground Truth.

5.7 Fifth Iteration

In this iteration, an analysis of the results from the different algorithms used in relation to the Ground Truth is presented.

Comparing the results obtained with the best parameter values, we can see that a pattern is repeated in all four algorithms: a large community is always found accompanied by a number of communities of very small size. This pattern coincides with the Ground Truth.

It can be seen, also, that the Label Propagation and Strongly Connected Components algorithms give very similar results in all their executions. Therefore, it can be said that these two algorithms work in the same way in this graph. This is due to the fact that by having many nodes with few relations to other nodes, a label spreads quickly through the nodes, and if these have a high number of relations with other nodes, the algorithm generates a large community and other small ones around it.

The Label Propagation, Strongly Connected Components and Weakly Connected Components algorithms return a distribution of communities in the network that is not very effective but can be used to identify the areas of the network with the most information exchange, since the relationships between nodes represent e-mail exchanges between employees.

The algorithm that performs best and gives the best results is the Louvain algorithm. This algorithm returns a distribution of communities similar to the one found in Ground Truth with good consistency.

But there are still cases in which it finds a high number of communities, a distribution very different from the one found in Ground Truth.

Chapter 6

Conclusions

In this final degree work we have identified a variety of community detection algorithms and quality metrics with which to compare them. We have used algorithms on a real world dataset to identify its community structure and compared the results using the previously mentioned quality metrics. The results of this comparison have shown that each algorithm behaves very differently from the others and that its output it's greatly defined by the structure of the graph upon which it is applied.

6.1 Conclusions

The main objective of this project has been achieved as I have compared four different community detection algorithms with each other based on quality metrics applied to their results when executed on a real world dataset.

The following conclusions can also be drawn from the results of this work:

1. The use of community detection algorithms is highly affected by the choice of the graph on which they are going to be executed, since a graph containing many relationships between nodes will be more suitable for one algorithm than for another. Therefore, before executing a specific algorithm on a graph, we must know its structure and apply the one that best adapts to this structure.
   For example, in this work we have used a graph with many nodes connected to each other that form a big congregation with smaller ones surrounding it, thus resulting in most of the algorithms finding one big community and many smaller ones.

2. The community detection algorithms return very important and different information about the structure of the network, some tell us in which part of the network there are more connections between nodes, while others will tell us which parts of the network have a similar structure depending on how the communities are found.

For example with the Weakly Connected Components Algorithm, it finds a community where every node can be reached from any other node of the same community, thus giving us groups of nodes that may not have properties in common but are densely connected.

3. Analyzing the results provided by the algorithms that allow an initial configuration through parameters, it can be concluded that there are significant differences in the results obtained by varying these parameters. This fact makes it necessary to study the configuration parameters for each data set on which community detection algorithms are to be applied. The most significant case is the label propagation algorithm, where the parameter that establishes the starting labels constitutes the core of the algorithm.

6.1.1 Knowledge acquired

In this project I have used knowledge acquired through the different subjects studied in this degree; for example, I have used knowledge learned in the databases subject and concepts of graphs learned in algebra and discrete mathematics.

More over, during the time I've spent working on this project I have expanded my knowledge on those topics and have gained knowledge on other topics, such as:

1. Theoretical knowledge of communities in graphs and community detection algorithms.
2. Configuration and use of NoSQL databases.
3. Practical knowledge on the execution and the inner workings of community detection algorithms.
4. Interpretation of quality metrics applied to the results of the execution of an algorithm.

6.2 Future work

After the completion of this work, certain paths can be identified for the continuation of the work carried out.

The execution of the Louvain and Label Propagation algorithms with other configuration parameters would be interesting to see if the results become better or maybe worse if they are changed.

Seeing the results obtained for the four compared algorithms, it would be interesting to perform these tests on other community detection algorithms, using the metrics that were applied to those already compared and adding more. This would broaden our frame of reference for the interpretation of the results already found, as well as those we would find with these new algorithms and metrics.

Following this line, tests could also be run on other data sets with a different number and distribution of nodes and relationships, as this would give us an idea of how these algorithms react to other networks with different characteristics.

Bibliography

[1] K. H. Rossen, *Discrete Mathematics and Its Applications*, 7th ed. McGraw-Hill, 2012.

[2] J. Moody and D. White, “Structural cohesion and embeddedness: A hierarchical concept of social groups,” *American Sociological Review*, pp. 103–127, 2003. [Online]. Available: http://scholar.google.de/scholar.bib?q=info:2c7bh3rVpdcJ:scholar.google.com/&output=citation&hl=de&as_sdt=2000&ct=citation&cd=0

[3] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Phys. Rev. E*, vol. 69, no. 2, p. 026113, Feb. 2004. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.69.026113

[4] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*, ser. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[5] W. de Nooy, A. Mrvar, and V. Batagelj, *Exploratory Social Network Analysis with Pajek (Structural Analysis in the Social Sciences)*, illustrated edition ed. Cambridge University Press, 2005. [Online]. Available: http://www.amazon.com/Exploratory-Network-Analysis-Structural-Sciences/dp/0521602629

[6] A. Bavelas, “A mathematical model for group structures,” *Applied Anthropology*, vol. 7, no. 3, 1948. [Online]. Available: http://www.jstor.org/stable/44135428

[7] L. C. Freeman, “A set of measures of centrality based upon betweenness,” *Sociometry*, vol. 40, 1977.

[8] M. E. J. Newman, *Networks: an introduction.* Oxford; New York: Oxford University Press, 2010. [Online]. Available: http://www.amazon.com/Networks-An-Introduction-Mark-Newman/dp/0199206651/ref=sr_1_5?ie=UTF8&qid=1352896678&sr=8-5&keywords=complex+networks

[9] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Stanford Digital Library Technologies Project, Tech. Rep., 1998. [Online]. Available: citeseer.ist.psu.edu/page98pagerank.html

[10] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," 2008. [Online]. Available: http://arxiv.org/abs/0803.0476

[11] O. Markovitch and N. Krasnogor, "Predicting species emergence in simulated complex pre-biotic networks." *PloS one*, vol. 13, 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0192871

[12] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks." in *ASONAM*, N. Memon and R. Alhajj, Eds. IEEE Computer Society, 2010, pp. 176–183. [Online]. Available: http://dblp.uni-trier.de/db/conf/asunam/asonam2010.html#GreeneDC10

[13] J. M. Pujol, V. Erramilli, and P. Rodriguez, "Divide and conquer: Partitioning online social networks," *CoRR*, vol. abs/0905.4918, 2009. [Online]. Available: http://arxiv.org/abs/0905.4918

[14] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," Sep. 2007. [Online]. Available: http://arxiv.org/abs/0709.2938

[15] P. Pham The, T. Tuytelaars, and M.-F. Moens, "Naming people in news videos with label propagation," vol. 18, 2011. [Online]. Available: https://lirias.kuleuven.be/retrieve/148066

[16] Z.-W. Zhang, X.-Y. Jing, and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction," vol. 24, 2017. [Online]. Available: https://doi.org/10.1007/s10515-016-0194-x

[17] B. Huang, C. Wang, and B. Wang, "Nmlpa: Uncovering overlapping communities in attributed networks via a multi-label propagation approach," *Sensors*, vol. 19, no. 2, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/2/260

[18] M. Sharir, "A strong-connectivity algorithm and its applications to data flow analysis," vol. 7, pp. 67–72, 1981.

[19] M. Needham and A. E. Hodler, *Graph Algorithms Practical Examples in Apache Spark and Neo4j*, 1st ed. O'Reilly Media, Inc., 2019.

[20] S. J. Kazemitabar and H. Beigy, "Automatic discovery of subgoals in reinforcement learning using strongly connected components," M. Köppen, N. Kasabov, and G. Coghill, Eds. Springer Berlin Heidelberg, 2009.

[21] J. Treur, "Mathematical analysis of a network's asymptotic behaviour based on its strongly connected components," in *Complex Networks and Their Applications VII*, L. M. Aiello, C. Cherifi, H. Cherifi, R. Lambiotte, P. Lió, and L. M. Rocha, Eds. Springer International Publishing, 2019.

[22] S. Allesina, A. Bodini, and C. Bondavalli, "Ecological subsystems via graph theory: The role of strongly connected components," *Oikos*, vol. 110, pp. 164 – 176, 07 2005.

[23] R. E. Tarjan, "A new algorithm for finding weak components," *Information Processing Letters*, vol. 3, no. 1, pp. 13–15, 1974. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020019074900404

[24] Y. An, J. Janssen, and E. E. Milios, "Characterizing and mining the citation graph of the computer science literature," *Knowledge and Information Systems*, 2004.

[25] A. Monge and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," 1997.

[26] "Graph data platform | graph database management system | neo4j," 2007. [Online]. Available: https://neo4j.com/

[27] "Native graph database | neo4j graph database platform," 2007. [Online]. Available: https://neo4j.com/product/neo4j-graph-database/

[28] "Graph data science library | graph analysis algorithms | neo4j," 2007. [Online]. Available: https://neo4j.com/product/graph-data-science/

[29] "Bloom - neo4j graph database platform," 2007. [Online]. Available: https://neo4j.com/product/bloom/

[30] "The neo4j cypher manual v4.4," 2011. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/

[31] "The colorado index of complex networks," 2007. [Online]. Available: https://icon.colorado.edu/#!/

[32] "Institute for webscience and technologies," 2007. [Online]. Available: https://west.uni-koblenz.de/research/datasets

[33] "Stanford network analysis project," 2007. [Online]. Available: http://snap.stanford.edu/index.html

[34] "Canadian institute for cybersecurity," 2007. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[35] "email-eu-core network," 2007. [Online]. Available: http://snap.stanford.edu/data/email-Eu-core.html

[36] "Eu email communication network," 2007. [Online]. Available: http://snap.stanford.edu/data/email-EuAll.html

[37] L. Leskovec, J. Kleinberg, and J. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, 2007.

[38] B. Boehm, "A spiral model of software development and enhancement," *SIGSOFT Softw. Eng. Notes*, vol. 11, no. 4, p. 14–24, aug 1986. [Online]. Available: https://doi.org/10.1145/12944.12948