Decentralized Diffusion-Controlled Algorithm for Community Detection

Initialization and Resolution Study

ADRIÁN RAMÍREZ DEL RÍO
Abstract

Community detection in graphs has been an important research topic for many fields. The aim of community detection is to extract from graphs those groups of nodes that present more connections between them than with the rest of the network. Detecting such groups at different scales can help understanding the global behaviour of the system. However, recent studies have shown that real-world graphs follow power-law distributions for degree and community sizes. Specifically, these graphs present many small communities but just a few large ones. This unbalanced community size distribution poses a great challenge for community detection algorithms.

Most of the existing methods are based on global approaches that require information about the network to be processed as a whole. Thus, those techniques can not be applied when the graph is too big to fit into one single machine, or in distributed setting when the graph is partitioned among multiple machines. To solve this limitations, a completely decentralized community detection algorithm is presented. It is based on diffusion, following a vertex-centric approach that allows each node to decide the diffusion rates based on local information. It adds as well a mechanism for controlling the diffusion speed through a customizable function.

We evaluate the algorithm with a variety of graphs with different levels of imbalance and community structures. Our algorithm is able to detect (almost) perfectly the communities when the imbalance between community sizes is not extreme. We show as well how the sizes of the detected communities can be controlled by the diffusion strategy, allowing for better detection of finer or coarser resolutions in hierarchical graphs. The algorithm is also compared to other two well-known existing methods, achieving similar results in most of the cases though with a higher computation time.

Keywords
community detection; unbalanced communities; adaptive diffusion; decentralized algorithm;
Sammanfattning


Vi utvärderar algoritmen genom flera olika grafer med olika nivåer av obalans och gemenskaps strukturer. Vår algoritm kan (nästan) felfritt upptäcka gemenskaper där obalansen mellan dem inte är för stor. Vi visar även hur storleken på de hittade gemenskaperna kan kontrolleras genom diffusions strategin, som tillåter bättre upptäckt av finare eller grövre resolution av hierarkiska grafer. Algoritmen kan också jämföras med två befintliga, välkända metoder, vilka ger liknande resultat i de flesta fallen men tar längre tid att genomföra.

Nyckelord
Gemenskap detektering, obalanda gemenskaper, adaptativ diffusion, decentraliserad algoritm;
## Contents

1 **Introduction**  
1.1 Background ........................................... 1  
1.2 Problem ............................................. 3  
1.3 Purpose ............................................. 3  
1.4 Delimitations ....................................... 4  
1.5 Outline ............................................. 4  

2 **Community detection in graphs**  
2.1 Basic concepts ..................................... 7  
2.1.1 Graphs ........................................... 7  
2.1.2 Communities and their importance .............. 8  
2.1.3 Communities in real world networks ............ 8  
2.2 Existing methods for community detection .......... 9  
2.2.1 Traditional methods ............................. 10  
2.2.2 Optimization of modularity function .......... 10  
2.2.2.1 Louvain method .............................. 11  
2.2.3 Random walks and diffusion .................... 11  
2.2.3.1 Infomap method ............................. 12  
2.2.3.2 Diffusion method ............................ 12  
2.2.4 Limitations ..................................... 13  
2.3 Validation Metrics .................................. 13  
2.3.1 Community scoring metrics .................... 13  
2.3.2 Partition Similarity Metrics .................. 15  

3 **Decentralized Diffusion-Controlled Algorithm**  
3.1 Motivation ......................................... 19  
3.2 Variables ......................................... 20  
3.3 The method ....................................... 21  
3.3.1 Node structure ................................ 22  
3.3.2 Initialization phase ........................... 22  
3.3.3 Gather phase .................................. 22
CONTENTS

3.3.4 Apply phase ........................................... 23
3.3.5 Scatter phase ........................................ 24
3.4 Initialization strategies ........................................ 24
3.5 Diffusion strategies ........................................ 27
3.6 Stopping criteria ........................................ 28
3.7 Further modifications ........................................ 30
3.8 Other considerations ........................................ 30

4 Experimental procedure ........................................ 33
4.1 Computer-generated benchmarks ............................... 33
  4.1.1 GN benchmark ........................................ 34
  4.1.2 LFR benchmark ........................................ 34
  4.1.3 Deterministic hierarchical network models ............ 36
4.2 Datasets .................................................. 37
  4.2.1 Dataset 1: LFR generated graphs ..................... 37
  4.2.2 Dataset 2: Unbalanced communities .................. 38
  4.2.3 Dataset 3: Hierarchical and balanced communities .... 42
  4.2.4 Dataset 4: Hierarchical and unbalanced communities ... 42

5 DDCA performance ........................................ 47
5.1 Effect of initialization strategy ............................. 47
5.2 Effect of controlled diffusion ............................. 50
5.3 Detection of unbalanced communities .................... 51
5.4 Effect of diffusion strategy ................................ 54
  5.4.1 Balanced hierarchical graphs ......................... 54
  5.4.2 Unbalanced hierarchical graphs ...................... 58
5.5 Comparison to other detection algorithms .................. 59
  5.5.1 Results on Dataset 1 ............................... 61
  5.5.2 Results on Dataset 2 ............................... 62
  5.5.3 Results on Dataset 3 ............................... 62
  5.5.4 Results on Dataset 4 ............................... 63

6 Conclusions ........................................ 67
6.1 Conclusions ........................................ 67
6.2 Discussion ........................................ 68
6.3 Future work ........................................ 69
List of Figures

3.1 Comparison of different diffusion functions on $d_{out}$ .......................... 27
3.2 Effect of $a$ on the diffusion strategies .................................................. 29

4.1 Impact of $\mu$ and $\beta$ of LFR benchmark on cluster detectability ........... 35
4.2 Hierarchical structure of a deterministic scale-free network ..................... 37
4.3 Degree and community size distributions for a graph from Dataset 1 ....... 39
4.4 Extremely unbalanced graphs from Dataset 2 ......................................... 41
4.5 Unbalanced graphs from Dataset 2 ......................................................... 41
4.6 Graph $H_3$ seen at different levels of the hierarchy .................................. 43
4.7 Graph $I_5$ seen at the two different levels of the hierarchy ...................... 45
4.8 Graph $I_7$ seen at the two different levels of the hierarchy ...................... 45

5.1 Comparison of performance of DDCA initializations on Dataset 1 .......... 48
5.2 Comparison of convergence times of DDCA initializations on Dataset 1 ... 49
5.3 Effect of the controlled diffusion on Dataset 1 ......................................... 50
5.4 Results of the different initializations for DDCA on Dataset 2 ............... 51
5.5 Precision and recall of the detected communities in extremely unbalanced graphs of Dataset 2 ................................................................. 53
5.6 Results of DDCA-RSPC on $H_1$ ............................................................... 55
5.7 Results of DDCA-RSPC on $H_2$ ............................................................... 56
5.8 Results of DDCA-RSPC on $H_3$ ............................................................... 57
5.9 Results of DDCA-RSPC on micro-communities for Dataset 4 ............... 58
5.10 Results of DDCA-RSPC on macro-communities for Dataset 4 ............ 59
5.11 Comparison of performance of DDCA and existing methods on Dataset 1 ................................................................. 60
5.12 Comparison of the execution time of DDCA and existing methods on Dataset 1 ................................................................. 61
5.13 Comparison of performance of DDCA and existing methods on Dataset 2 ................................................................. 63
# List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Definition of graph variables and their symbols</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Chosen values for LFR benchmark parameters in Dataset 1</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>Values of LFR benchmark parameters used for graphs from Dataset 2</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>Generated graphs for Dataset 2</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Goodness measures for graphs of Dataset 2</td>
<td>41</td>
</tr>
<tr>
<td>4.5</td>
<td>Properties of the generated graphs in Dataset 3</td>
<td>42</td>
</tr>
<tr>
<td>4.6</td>
<td>Values of LFR benchmark parameters used for graphs from Dataset 4</td>
<td>43</td>
</tr>
<tr>
<td>4.7</td>
<td>Properties of the generated graphs for Dataset 4</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Results of DDCA-RSPC on Dataset 2</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Results of DDCA-RSPC for different diffusion strategies on graph $H_3$</td>
<td>56</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparison of performance of DDCA and existing methods on H1</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Comparison of performance of DDCA and existing methods on H2</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparison of performance of DDCA and existing methods on H3</td>
<td>64</td>
</tr>
<tr>
<td>5.6</td>
<td>Comparison of performance of DDCA and existing methods on micro-communities of Dataset 4</td>
<td>64</td>
</tr>
<tr>
<td>5.7</td>
<td>Comparison of performance of DDCA and existing methods on macro-communities of Dataset 4</td>
<td>65</td>
</tr>
</tbody>
</table>
List of Algorithms

1 DDCA: Initialization phase ........................................... 23
2 DDCA: Gather phase ................................................... 23
3 DDCA: Apply phase ..................................................... 24
4 DDCA: Scatter phase ................................................... 25
Chapter 1

Introduction

This chapter presents very briefly the background of community detection in graphs as well as the identified problems in the existing solutions. The objective of the thesis is as well stated along with the delimitations of the project. At the end of the chapter the outline of the whole thesis is described.

1.1 Background

Graphs are mathematical models for the representation of a set of objects that correspond to abstractions (also called vertices or nodes) and the relations or interactions between pairs of them. These related pairs of nodes is what are called edges (also links). Graphs are a very powerful tool for representing the interactions present in the real world in a simple way, and thus the reason why they are a widely studied subject in many branches of science, from mathematics to biology, computer and social sciences, etc [1]–[4]. Sometimes graphs are also referred to as networks, so both words are used interchangeably.

Community structure is considered as one of the important characteristics that most of real world graphs exhibit. These communities (sometimes also called clusters) are groups of nodes that present a high number of links between them (high internal connectivity) and very few links pointing to nodes outside the group (low external connectivity) [5]. The detection of these communities has been a very active field of research in the recent years for many different science and engineering disciplines, since the identification of these groups can help understanding the global behaviour of the whole network.

There are many applications of community detection in graphs. One example found in the computer science field is automatic document organization where identifying meaningful groups of similar documents makes easier, efficient and effective the access to the information. Another example is word sense
disambiguation as finding groups of related words can help on deciding the sense of a word with several meanings.

The community detection problem consists in identifying clusters of nodes with no other information than the topology (the network itself), although looking for the groups with the optimal values of internal and external connectivity is an NP-hard problem and becomes unfeasible even for small-sized graphs. For this reason, the majority of the modern community detection methods are based on approximation algorithms to perform the optimization of objective functions that reward internal connectivity and penalize the external one.

Random walks approaches have been widely used to detect communities. A random walk is a stochastic process describing the probability of paths through the network. The process starts at a given node, and each step moves to one of the neighbours (all neighbours have equivalent probability to be chosen as next). In the presence of a community structure, a random walker gets stuck in regions with high density of edges, i.e. regions of highly connected nodes, and can escape those regions via one of the few edges pointing outside of the region, though with a very low probability.

Random walks can be interpreted as diffusion processes of flows through the network where the links act as “pipelines”. For example, one could think of the process as a color spreading through the graph: it starts at one node with an specific amount of the color and distributes it among its neighbours (nodes it is linked to); afterwards, the neighbours do the same so that the color spreads to more and more nodes and the process continues iteratively until each node has an amount of color that does not change (the process reaches the stationary distribution). If several colors are spread from different nodes known as seeds at the same time, we can at the end select for each node the color with the maximum amount as the community it belongs to.

Most of the real networks have been found to present a set of characteristics that makes the detection harder. Firstly, they happen to be scale-free[6], meaning that the distribution of the nodes’ degree (number of links of a node) follows a power-law (many nodes with few neighbours and few nodes with a lot of neighbours). Secondly, there is a great heterogeneity of communities which have different internal connectivity strengths and with sizes that follow as well power-law distributions (very small communities and few big ones) [6], [7]. When the gap between sizes of the smallest and the biggest communities is very big, it is said that the network presents an unbalanced community structure.
1.2 Problem

The imbalance found in real world networks makes existing algorithms to struggle with the detection of some clusters, since they normally have bounds on the sizes of the communities they can detect. Some approaches suffer from a resolution limit [8], such that they are unable to detect communities that are too small in comparison to the size of the whole network, even when these groups of nodes present a high number of internal connections and low number of external ones with respect to the size of the cluster (good community structure). Methods that suffer from this problem tend to merge these small communities into bigger ones. On the other hand, other approaches suffer from the opposite problem, presenting upper limits on the sizes of the communities they can detect. In other words, they are not able to detect communities that are very big in comparison to the size of the network. This is known as the field-of-view-limit and the detection methods that suffer from it tend to split the biggest communities into smaller ones [9]. Note that both limitations are not exclusive as some methods could have difficulties detecting both small and very big communities.

Another drawback present in most of the existing algorithms is that they are based on global approaches, so as they require information about the whole network to carry out the detection process, and accordingly, they need to fit the whole graph in memory. These type of methods present also the disadvantage of being harder to define as distributed algorithms in comparison to solutions based on inspecting locally different regions of the graph. Furthermore, global approaches are also used usually in terms of the optimization process, where all the communities are assumed to be detectable under the same global value for the optimization parameters.

The question now is how can we design a community detection algorithm that overcomes the stated limitations and how can we evaluate it.

1.3 Purpose

In order to overcome the limitations of existing algorithms stated above, a novel algorithm is designed: the Decentralized Diffusion-Controlled Algorithm (DDCA) for community detection in graphs. DDCA follows a vertex-centric approach (rather than a global one) where each node is responsible for computing the community it belongs to by gathering information about its neighbours, thus obtaining a fully decentralized algorithm that can be easily distributed and run in parallel with each node operating independently.

The main idea behind DDCA is to perform a diffusion process of several colors concurrently. At the beginning, a number of nodes are selected as seeds
and given a certain amount of a different color each. Then the colors start flowing through the network (passing from a node to its neighbours), but unlike traditional diffusion, we let the nodes decide the amount of each color that they should giveaway based on the information of their neighborhood. This is equivalent to local tuning of the parameters of the optimization process which is performed at each node, instead of imposing it globally as existing methods. As the parameters are tuned for each community in base of the neighbourhood information at each node, the method should be able to identify communities regardless their size or the size of the graph and be less prone to suffer from the resolution limit or the field-of-view limit. The algorithm includes, as well, a resolution mechanism controlling the diffusion process so that it is possible to decide the granularity level of the communities that are wanted to be detected.

The purpose of this thesis is to depict DDCA and different versions for its initialization as well as different diffusion strategies. It presents as well the results of the evaluation of DDCA on different sets of graphs with various properties and discusses the observed behaviour, capabilities and limitations of the algorithm.

1.4 Delimitations

The study is limited to the evaluation of a predefined set of initialization and diffusion methods for the designed algorithm on unweighted and undirected graphs with non-overlapping communities. The aim is to study the controlled diffusion mechanism behavior in specific demonstrative cases that can make us understand it.

Although naturally designed for distributed graph processing, the project will not include the implementation and testing of the algorithm in these type of frameworks but the collaboration on a C++ implementation and the evaluation on graphs that can fit in a single-machine RAM.

1.5 Outline

The content of this master degree thesis is structured as follows:

Chapter 2 provides the knowledge necessary to understand the rest of this thesis. First, some preliminary concepts as graphs and communities are explained. Then, it provides a background of the different areas covered by the thesis, such as community detection algorithms in graphs and their limitations and validation metrics.

Chapter 3 depicts the Decentralized Diffusion-Controlled Algorithm (DDCA) for community detection with its different initialization and diffusion strategies.
Chapter 4 describes the different generated datasets that are used to test the properties of DDCA. Chapter 5 presents the evaluation of the experiments run on the datasets described in the previous chapter. Chapter 6 concludes the thesis by summarizing the work done and discussing further possible improvements.
Chapter 2
Community detection in graphs

This chapter introduces the basic concepts and knowledge required to understand the performed work and obtained results. It starts defining simple concepts as what a graph or a community are and properties of real world graphs. Then it briefly presents existing community detection techniques and validation metrics.

2.1 Basic concepts

In this section the reader will find the basic definitions to understand what graphs and communities are and why their study is important.

2.1.1 Graphs

A graph, or a network, is a mathematical model used to represent entities and their interactions. The entities are called nodes or vertices whereas their interactions are called links or edges, and a commonly used visual representation of a graph consists of dots (nodes) connected by lines (links). As an example, we can think of the WWW, where the different web-pages would play the role of nodes and the hyperlinks that they contain pointing to other webs would form the edges. We also find examples in many other disciplines, like biology (protein-protein interaction networks), electrical engineering (power-grid networks), physics (molecular systems networks), and a long etc. [1]–[4].

Mathematically, a graph \( G(V,E) \) can be represented by the set of vertices \( V \) and the set of edges \( E = \{(u,v): u,v \in V\} \) meaning \( u \) interacts with \( v \), and depending on the type of interaction, whether it is reciprocal or not, the network is said to be directed or undirected, respectively. The interaction can also present different strengths for the connections (called weights) and hence there is a distinction between weighted and unweighted graphs. The number of connections
that a node presents is called the *degree* of the node; and in the case of weighted graphs it is also defined the *strength* of the node as the sum of the strengths of each of its connections.

Another way to represent graphs is by the so-called *adjacency matrix* $A$, which is a matrix where the entrance $A_{ij}$ is 1 when the nodes $i$ and $j$ are connected and 0 if they are not (for weighted graphs the value is the strength of the connection instead of 1). This notation is very widely used since it allows to express different properties of the networks with simple formulas.

### 2.1.2 Communities and their importance

Most of the real world graphs studied by researchers and companies present the interesting property of having a community structure. This means that the nodes of the network are connected in such way that groups of them are found to be more strongly connected than others. These groups are called communities and the main way to define them is as groups of nodes where there are a lot of internal connections (between nodes of the same group) while there are few external ones (between nodes from different groups) [5]. These two characteristics are often referred to as *internal cohesion* and *separability*, and although there have been stated different formal definitions of what a community is [1], [5], [10], all of them are conceptually based in these two properties.

The community structure can as well take different forms. Sometimes, clusters are disjoint sets of nodes, in other words, nodes can only belong to one community; other times, nodes can have many memberships and hence the graph is said to have overlapping communities. In other cases, the community structure takes a hierarchical form, in which the groups of higher levels can be splitted in smaller communities to form a lower level in the hierarchy.

The topic of finding communities in networks has become so popular because the identification of these groups may provide important information about the organization of the network. Knowing the group structure can uncover the internal organization of the graph and allow the detection of hidden relationships between nodes that would be too hard to find by empirical inspection. It can help, for example, to classify the vertices based on the properties of their community or the role they play inside their group [1]. From the examples given before, we could identify web-pages sharing a topic or proteins with similar functions.

### 2.1.3 Communities in real world networks

The majority of networks coming from real world scenarios follow strong self-organizing principles that makes them non-random and interesting. On one hand, real networks are found to be *scale-free* [6], which means that the node’s degree
distribution follows a power-law, or in other words, that we find many nodes with few connections and few nodes with a lot of them. On the other hand, there is also a great heterogeneity of communities which come with very different internal/external connectivity and sizes. In particular, the sizes of the communities in real networks have also been found to follow a distribution close to a power-law, i.e. a lot of small communities and very few communities with a huge number of nodes [6], [7].

The problem to be solved is to identify the communities with no other information than the topology, this is, the network itself. The described characteristics of real networks generate an imbalance in the sizes of the clusters that have to be detected, and methods have to be able to identify very small and very big communities at the same time. The greater the gap between the smallest and the biggest communities, the more unbalance the community structure is.

This imbalance is one of the main reasons causing the limitations found by the existing community detection methods. There are some methods that suffer from the resolution limit [8] as they can only detect communities up to a minimum size, meaning that communities which are smaller than this limit are not detectable by the algorithm which will tend to find solutions with larger communities, only exploring coarse partitions. The complementary limit is also found, this is, some algorithms have upper bounds on the sizes of the communities detectable for them, so the communities with sizes beyond this limit are not identified. This is known as the field-of-view limit and the detection methods that suffer from it tend to split the biggest communities into smaller ones [9]. Note that both limitations are not mutually exclusive as some methods could have difficulties detecting both small and very big communities.

Trying to optimize the internal and external connectivity of each possible partition of the set of nodes on order to find the best communities is $NP$-hard and hence prohibitive for real world networks which tend to be very big and grow larger and larger every day. For this reason, many different approaches for community detection exist and are usually based on approximate optimization and the use of heuristics.

2.2 Existing methods for community detection

Due to the popularity of the field, a large number of community detection approaches and methods have been developed throughout the years. Nevertheless, some of them can be identified as the most widely-known and we will expose them in this section.
2.2.1 Traditional methods

The most traditional approach to detect clusters in graphs is to study their spectral properties. By analyzing the eigenvalues of the adjacency matrix (or other derived matrices like the Laplacian), it is possible to detect a gap between a group of them and the rest. One can then apply traditional clustering techniques on the projection of the vertices into an Euclidean space where each vertex is represented by coordinates corresponding to the eigenvectors of the outlying eigenvalues [11].

The main drawback of spectral methods is their complexity, since the operations they require (finding the eigenvalues of a matrix) are very time consuming, and their application becomes unfeasible when the graphs grow too large. Another problem of these methods is that they are not always reliable, specially when the network under study is very sparse [1] (those where the number of edges is very small, in the order of the number of vertices).

Other traditional but more recent approach is to fit a generative model on the graph. The standard model used is known as stochastic block model (SBM) [12], which generates a graph where nodes are connected according to a probability function that depends on their groups membership. The major disadvantages found in this type of methods is that the number of communities has to be specified in advance and as for spectral methods, the computation times tend to be too high.

2.2.2 Optimization of modularity function

A rather different approach, and usually more efficient, is to optimize a target function describing the quality of the identified communities over the set of possible clusterings. The most commonly used quality function is the modularity (Equation (2.4)) [13], [14], although it has been shown that finding the maximum of the modularity function is \textit{NP}-complete (no algorithm can find it in polynomial time unless \( P = NP \)) [15]. Due to this, approximation algorithms, such as greedy optimization based on heuristics or simulated annealing techniques, are needed for large networks.

Despite the popularity of the modularity metric, it comes with several problems that generate fundamental flaws in the detection algorithms that optimize it. Firstly, it has been stated that random graphs without real community structure may present partitions with high modularity, even when it has been designed to capture the difference between randomness and non-randomness. Secondly, it has been showed that modularity optimization methods suffer from the resolution limit [8]. In an attempt to alleviate this unwanted property, a number of resolution techniques have been proposed, like introducing a resolution parameter to the modularity equation [16] or designing multiscale variants for community detection algorithms [9].
2.2. EXISTING METHODS FOR COMMUNITY DETECTION

2.2.2.1 Louvain method

As an example of a method based on modularity optimization, and probably the most well-known, one can find the Louvain method [17]. It performs a greedy optimization of the modularity function through an iterative process which is described next.

First of all, each node is initialized so that it forms its own community, giving the trivial solution where there are as many communities as nodes in the network. Afterwards, a two-steps process is repeated: In the first stage, each node’s neighborhood is inspected, and for each neighbour, the difference of the current modularity with that obtained by changing the node’s community to that of the neighbor’s is computed. Then, the node is placed on the community that provided the maximum gain (in case of negative maximum change, the node stays in its original community). This process is repeated sequentially for all nodes, until none of them change their community membership, meaning the algorithm has reached a local maxima. In the second phase, a new (weighted) network is derived where the nodes are the communities obtained at the end of the previous stage. The edges between the new nodes are given weights equal to the sum of the weights of the edges linking the corresponding two communities. The edges between nodes in the same community are transformed into self-loops, with weight equal to the sum of the weights of the internal edges. Once this second phase is complete, the first step can be reapplied and the whole process iterated.

Some advantages of the algorithm are its intuitiveness and simplicity, which make it a really fast algorithm (“near linear on typical and sparse data” [17]). Moreover, the algorithm provides different levels of granularity of the communities, naturally incorporating a notion of its hierarchy, and making it more robust to the resolution limit. Lancichinetti and Fortunato [18] obtained very good results in their comparative analysis when considering the bottom level of the hierarchy (smallest clusters), however Fortunato itself points out that there is no guarantee that this deepest level would provide the best partition, so the results should be studied at different resolutions [1].

2.2.3 Random walks and diffusion

Other approaches are based on dynamical processes that may reveal the community structure of the network. Most of this methods make use of the properties of random walk dynamics. A random walk can be thought as the process of a walker through the network that at each step picks randomly one of the connections of the current node it is visiting and moves to the selected neighbour; then the process is repeated. Hence, random walks are stochastic process that describe the probability of paths through the network from a starting node, thus giving information of what
regions are more or less likely to reach. Methods that use random walks to find communities are based on the assumption that the walker would get trapped inside communities, as the number of connections with different clusters is small and the probability of the random walker jumping out of the community is low [19].

### 2.2.3.1 Infomap method

One of the most well known methods of this category, based on the idea of optimally compressing the information of a random walk taking place in the graph. The simplest description is a sequence of the vertices (each described by a code-word) reached by the random walk. In order to find this description, the map equation is proposed, which “provides the description length of an infinitely long random walk taking place on the network”[20]. To find the partition that minimizes this description length, a computational search (using a deterministic greedy search algorithm) refined with simulated annealing is performed. This method is commonly called Infomap [1], [21].

Unlike methods based on modularity optimization, Infomap is not known to suffer from the resolution limit, since it tries to find dense substructures locally, it is suited for detection at the finest scale [20]. Nonetheless, this comes with the drawback that if the graph under consideration has slowly mixing communities (the random walk needs a lot of time to cover the whole community), the algorithm will suffer from the field-of-view-limit and will tend to an over-partitioned solution [9].

### 2.2.3.2 Diffusion method

A different way to interpret random walks is as diffusion processes of flows through the network where the links act as “pipelines”. Thus, the ideas behind random walks can be used to diffuse information through the network in such a way that members of the same communities are more likely to get it [22]. For example, one could think of the process as a color spreading through the graph: it starts at one node with an specific amount of a specific color and distributes it among its neighbours (nodes it is linked to); afterwards, the neighbours do the same so that the color spreads to more and more nodes and the process continues iteratively until the stationary distribution is reached (there are no more changes in the amounts of color held by nodes) or the process is stopped. If several colors are spread from different nodes which are called seeds, at the same time, we can at the end of the process select for each node the color with the maximum amount as the community it belongs to [23].

The benefit of algorithms based on diffusion strategies is that they are usually decentralized and hence they are potentially better suited for distributed networks
where a global inspection of the network is not possible. The main problem, as for Infomap, are graphs with slow mixing communities.

2.2.4 Limitations

Apart from the limitations already stated regarding the limits on the sizes that the algorithms can detect, it is possible to identify some other.

On one hand, most of existing algorithms consisting in the optimization of some objective function are based on global approaches and require information about the whole network to work. This is sometimes a serious restriction, as for very large graphs that do not fit in memory the detection algorithms have to be able to execute in distributed frameworks where only local information is available for the algorithm.

On the other hand, the optimization processes followed by some existing algorithms try to fit every community in the graph under the same global values of the parameters, instead of letting each community to decide the values of these parameters independently.

2.3 Validation Metrics

Since communities are naturally found in real-world graphs, a different number of metrics have been defined over the years in order to evaluate their quality. Some of them measure the internal connectivity, others measure the external one, and some others combine information on both [24]. These type of metrics are called community scoring metrics.

There are also another set of metrics that are interesting in the graph clustering field, which are known as partition similarity metrics. Unlike community scoring metrics, these type focus on comparing two different clusterings over the same graph. Their main use is to compare the results obtained by a community detection algorithm against a known ground truth.

2.3.1 Community scoring metrics

Also known as standalone-quality metrics, they measure the goodness of a partition of the graph in different clusters (not necessarily disjoint) by evaluating how well internally connected they are (in terms of the number of connections between nodes of the same cluster) and/or how well are clusters separated from each other (in terms of the number of connections between clusters). The advantage of these type of measures is that no information about ground truth clusters is needed in order to evaluate the quality of a clustering of the graph.
Chapter 2. Community Detection in Graphs

Conductance

The conductance is defined as the fraction of the connections going outside the cluster, i.e., the number of links between a node in the cluster and a node outside it, divided by the total number of links with at least one endpoint inside the cluster (Equation (2.1)).

For a given graph $G(V, E)$ with adjacency matrix $A$, the conductance [1], [24] of a cluster $S \subset V$ is defined by

$$
\phi(S) = \frac{\sum_{i \in S, j \notin S} A_{ij}}{\sum_{i \in S, j \notin S} A_{ij} + \sum_{i, j \in S} A_{ij}}
$$

The conductance focuses on measuring the quality of a cluster in terms of how well it is separated from the rest of the network, although it includes information of both internal and external connectivity. It takes the value 0 when the cluster is completely isolated (disconnected component from the rest of the graph) and starts growing as the number of external connections increases, becoming 1 when there are only external connections (cluster formed by non-connected nodes).

The conductance for the whole graph can then be computed as 1 minus the average of the conductance values of all $k$ clusters (Equation (2.2)), so that its value ranges from 0 (worst) to 1 (optimal).

$$
\phi(G) = 1 - \frac{1}{k} \sum_k \phi(S_k)
$$

Note that other definitions of conductance can be found in the literature [25], [26], though the concept behind all of them is the same: to measure the level of isolation of a cluster from the rest of the network.

Coverage

Unlike conductance, the coverage measures the quality of a whole graph divided in different clusters and it focuses on how well internally connected these clusters are. It is defined as the fraction of edges between nodes of the same cluster with respect to the total number of edges in the graph (Equation (2.3)) [26].

Let $A$ be the adjacency matrix of a graph $G(V, E)$ and $s(i)$ a function that gives the cluster to which the node $i \in V$ belongs, we define the coverage as

$$
\theta = \frac{\sum_{i,j} A_{ij} \delta(s(i), s(j))}{\sum_{i,j} A_{ij}}
$$

(2.3)
where $\delta(a,b)$ is the Kronecker delta function (equals 1 if $a = b$ and 0 otherwise).

This metric, as the conductance (Equation (2.2)), falls in the range $[0, 1]$, but focuses on the strength of internal cluster connectivity. As for the conductance, the optimal value 1 is achieved when all clusters are completely isolated, and the worst value 0 when there are no connections between nodes of the same cluster.

**Modularity**

This goodness metric was first proposed by Girvan and Newman [13], [14] and has gained a lot of attention recently, making it one of the most well-known measures for accounting cluster quality as well as one of the most widely used as objective function for optimization methods [24].

Let $A$ be the adjacency matrix of a graph $G(V,E)$, with a total number of $m$ edges, the modularity of the graph with respect to a node-to-cluster mapping is expressed as [1], [14], [17]

$$Q = \frac{1}{2m} \sum_i \sum_j \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(s(i), s(j))$$

(2.4)

where $k_i$ is the degree of node $i$ and the $\delta(s(i), s(j))$ expression acts as in the coverage definition (Equation (2.3)).

Modularity focuses on comparing the density of internal links against the same (expected) quantity if edges were to be distributed at random, so it is a bit different from the other two presented, as it includes information based on a network model.

As pointed out by some researchers, there are large discrepancies between the different community goodness measures depending on which property they focus on, so it is a good practice to consider several of them when trying to compare results from community detection algorithms [24].

### 2.3.2 Partition Similarity Metrics

In the case that one has information about the underlying communities present in the graph, it is interesting to study the ability of a community detection algorithm to find the “correct” clusters or ground truth. This is done by comparing the results obtained by the algorithm against the ground truth, in order to quantify their agreement. Many approaches exist, including metrics based on pair counting, cluster matching or information recovery theory [1].
Formally, these metrics evaluate the similarity between the real set of communities $X = \{x_1, x_2, \ldots, x_{qX}\}$ and the set of communities detected by a graph clustering algorithm $Y = \{y_1, y_2, \ldots, y_{qY}\}$ over the same graph $G$, with $n$ vertices and $m$ edges. Each $x_i, y_j$ is a set of nodes which forms a cluster, and since we only consider non-overlapping communities, any two $x_i, x_j \in X$ satisfy $x_i \cap x_j = \emptyset$ and the same for any two clusters in $Y$.

**Adjusted Rand Index (ARI)**

This metric, based on pair counting, is the corrected-for-chance version of the Rand Index ($RI$) which is defined as [27]:

\[
RI = \frac{a_{00} + a_{11}}{a_{00} + a_{01} + a_{10} + a_{11}} = \frac{a_{00} + a_{11}}{\binom{n}{2}}
\]  

(2.5)

where

$a_{11} = |i, j \in V \text{ s.t. } x(i) = x(j) \land y(i) = y(j)|$ is the number of pairs of nodes which are in the same cluster in both $X$ and $Y$,

$a_{00} = |i, j \in V \text{ s.t. } x(i) \neq x(j) \land y(i) \neq y(j)|$ is the number of pairs of nodes that fall in different clusters in both $X$ and $Y$,

$a_{01} = |i, j \in V \text{ s.t. } x(i) = x(j) \land y(i) \neq y(j)|$ is the number of pairs of nodes where both belong to the same cluster in $X$ but to different clusters in $Y$,

$a_{10} = |i, j \in V \text{ s.t. } x(i) \neq x(j) \land y(i) = y(j)|$ is the number of pairs of nodes that fall in different clusters in $X$ but the same one in $Y$.

A simple interpretation is that $a_{11}$ and $a_{00}$ count the number of pairs of nodes in which $X$ and $Y$ agree whereas $a_{01}$ and $a_{10}$ count the number of disagreements, hence the rand index is the fraction of agreements among all pairs of nodes.

However, the $RI$ does not guarantee that a stronger disagreement will yield values closer to 0, so the Adjusted Rand Index ($ARI$) was designed to solve this by subtracting the expected value of the $RI$ and normalizing [1], [26], [28]:

\[
ARI(X, Y) = \frac{RI - E[RI]}{\max(RI) - E[RI]}
\]  

(2.6)

This metric ranges from 0 when the similarity between two clusterings $X$ and $Y$ is expected due to chance, to 1 when they completely agree. Although
this metric is very commonly chosen to compare partitions, it also presents a number of problems as pointed out by Fortunato and Hric [1].

**Mutual Information (MI)**

The mutual information metric comes from the information theory field and is defined as in Equation (2.7),

$$MI(X, Y) = \sum_{x \in X} \sum_{y \in Y} P(x, y) \log \left( \frac{P(x, y)}{P(x)P(y)} \right)$$  \hspace{1cm} (2.7)

where $P(x, y) = |x \cap y|/n$ and $P(x) = |x|/n$.

Alternatively, one can write it as the compact form $MI(X, Y) = H(X) + H(Y) - H(X|Y)$ where $H(X) = -\sum_{x \in X} P(x) \log P(x)$ is the Shannon entropy and $H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} P(x, y) \log P(x, y)$ is the joint Shannon entropy [26].

As comes clearly from Equation (2.7), the metric is symmetric on the input and it can be interpreted as the amount of information that we get about $X$ from knowing $Y$ (and vice versa).

The mutual information, however, presents some problems as has being stated by Fortunato [1], since for a given partition $X$, any new partition created by splitting some of the clusters in $X$ would share the same amount of mutual information with $X$ despite being very different from each other.

**Normalized Mutual Information (NMI)**

The NMI is the normalized version of the mutual information (2.3.2), developed to solve the problems present on the latter. It can be expressed as

$$NMI(X, Y) = \frac{2MI(X, Y)}{\sqrt{H(X)H(Y)}}$$  \hspace{1cm} (2.8)

although alternative definitions can be found depending on the normalization strategy used [29]. Since it is a normalized value, it ranges from 0 (independent solutions) to 1 (identical). Although a better option than MI, the normalized version still suffers from sensitivity to the number of detected communities and might give higher values for larger clusterings (greater number of communities) regardless they are more similar to the ground truth or not.
Nonetheless, whatever problems it may present, it has become the standard metric to compare community detection algorithms results along benchmarks [1], [18], [26], [30], [31].
Chapter 3

Decentralized Diffusion-Controlled Algorithm

This chapter describes the Decentralized Diffusion-Controlled Algorithm (DDCA) for community detection in graphs which has been developed in collaboration with the SCS department from KTH and is the main concept under study in this master thesis.

First, the motivation (section 3.1) and the core phases of the algorithm (section 3.3) are presented. Afterwards, we describe the different initialization (section 3.4) and diffusion strategies (section 3.5). Next we comment some possible criteria to check convergence (section 3.6) and finally we end the chapter by describing some possible further modifications (section 3.7) and other considerations regarding the algorithm and its implementation (section 3.8).

3.1 Motivation

The main motivation for developing a new community detection algorithm is to overcome the limitations found by the existing approaches. Our first objective is to develop a completely decentralized algorithm that can be easily run on distributed graphs, performing the optimization with only local information. Specifically, the algorithm will take a node-centric approach, such that each node only requires to know information about its direct neighbours. Moreover, we want to develop a method that performs the optimization at the community level instead of globally.

We present the Decentralized Diffusion-Controlled Algorithm for community detection in networks or DDCA. To achieve a complete decentralization, it uses the concepts of the diffusion processes through networks (section 2.2.3.2) and performs a diffusion of several colors concurrently. Each of these colors actually represents a different community. The diffusion process is first initialized,
selecting a set of nodes as seeds that will be given a different color each, while non-seed nodes remain uncolored. All seeds will receive the same fixed amount of color and then, iteratively, a diffusion process for each color takes places, so that they start spreading through the network competing for dominance of the different regions. The process finishes when one of the convergence criteria is met.

Besides, the algorithm follows a vertex-centric approach where each node takes care of deciding to which community it belongs to by the information contained in its neighborhood, thus getting a local rather than global approach.

Furthermore, each node is responsible of deciding how valuable a color is in its region and computing a give-away ratio that controls the amount of color that is diffused to the neighbours. This can be thought as an optimization process where the parameter tuning is performed by node and community according to the information of the surrounding nodes. As the parameters are tuned for each community in base of the neighbourhood information at each node, the method should be able to identify communities regardless their size or the size of the graph and be less prone to suffer from the resolution limit or the field-of-view limit.

Additionally, the algorithm can take different functions for the diffusion process, making nodes more “selfish” (keep the majority of the colors) or “generous” (give away most of the colors amount) at giving their colors away, in order to be able to detect communities at different resolutions (for example in a hierarchical structure). It should be noticed, however, that the algorithm does not include any mechanism to infer the possible hierarchy structure, but just of getting finer or coarser resolutions.

3.2 Variables

In this section we present a series of mathematical denotations that will be used throughout the rest of the thesis to describe graphs and their properties, which are summarized in Table 3.1. As was introduced in section 2.1.1, a graph $G(V,E)$ is defined by its set of vertices $V$ and its set of edges $E$, and can be represented by an adjacency matrix $A$ where $A_{uv}$ is the weight of the connection between nodes $u, v \in V$ (0 if they are not). We will use equivalently the notation $w(u,v) = A_{uv}$.

For each node $u \in V$, its set of neighbours is defined as $N_u = \{v : v \in V, (u,v) \in E\}$ and its degree or number of neighbors is denoted as $k_u = |N_u|$. The strength is the sum of the weights of the connections and is defined by $s_u = \sum_{v \in N_u} w(u,v)$. In the case of unweighted graphs, $s_u = k_u$.

If each node in the graph is given a color taken from a set of colors $CL = \{c_1, ..., c_l\}$, then for each $c_k \in CL$ and node $u \in V$ we can define the internal strength of $u$ with respect to color $c_k$ as $s^i_u(c_k) = \sum_{v \in N_u} w(u,v) \delta(c,c(v))$ and the external as $s^e_u(c_k) = \sum_{v \in N_u} w(u,v)(1 - \delta(c,c(v)))$, where $\delta$ is the Kronecker
3.3 The method

This section depicts the Decentralized Diffusion-Controlled Algorithm (DDCA) in detail. First, the variables controlled by each node are presented (section 3.3.1). Later, the initialization steps of the method are described (section 3.3.2). Finally, DDCA is presented in three phases, following the Gather-Apply-Scatter (GAS) programming model [32]. This model consists in defining three functions or stages: the gather phase, where nodes recollect information about their neighbours; the apply stage, where the recollected information is used to modify the information of the vertex; and the scatter phase, which is responsible of the flow of information from the node to its neighbours. These three stages are iteratively run by each node until one of the convergences criteria is met. At this point, the dominant color of each node indicates the community that it belongs to. Note that the process is started with a certain amount of colors that should be equal or higher to the number of communities to detect, as the algorithm is able to discard colors, but not of creating new ones.

### Table 3.1: Definition of graph variables and their symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_u$</td>
<td>${v : v \in V, (u,v) \in E}$</td>
<td>Set of neighbours</td>
</tr>
<tr>
<td>$k_u$</td>
<td>$</td>
<td>N_u</td>
</tr>
<tr>
<td>$s_u$</td>
<td>$\sum_{v \in N_u} w(u,v)$</td>
<td>Strength</td>
</tr>
<tr>
<td>$s_{in}^u(c_k)$</td>
<td>$\sum_{v \in N_u} w(u,v)\delta(c,c(v))$</td>
<td>Internal strength w.r.t. $c_k$</td>
</tr>
<tr>
<td>$s_{out}^u(c_k)$</td>
<td>$\sum_{v \in N_u} w(u,v)(1 - \delta(c,c(v)))$</td>
<td>External strength w.r.t. $c_k$</td>
</tr>
<tr>
<td>$d_{in}^u(c_k)$</td>
<td>$s_{in}^u(c_k)/s_u$</td>
<td>Internal ratio w.r.t. $c_k$</td>
</tr>
<tr>
<td>$d_{out}^u(c_k)$</td>
<td>$s_{out}^u(c_k)/s_u$</td>
<td>External ratio w.r.t. $c_k$</td>
</tr>
</tbody>
</table>

delta, and $c(u)$ the color of node $u$. Note that the sum of the internal and external relative strengths should be equal to the strength of the node independently of the color, i.e. $s_u = s_{in}^u(c_k) + s_{out}^u(c_k) \forall c_k \in CL$.

The internal and external strengths can be normalized, obtaining the internal and external ratios $d_{in}^u(c_k) = s_{in}^u(c_k)/s_u$ and $d_{out}^u(c_k) = s_{out}^u(c_k)/s_u$, which sum up to 1 independently of the color. These two ratios range between 0 and 1 and give the fraction of internal/external strength of a surrounding color.
3.3.1 Node structure

Next we will describe the variables that the nodes held and that are used by the algorithm in the different stages. Let \( u \in V \) be a node from the graph, we define:

- **\( u\).neighbours = N_u \)**
  The set of neighbours of the node.

- **\( u\).weights = \{ (v, w(u, v)) : v \in N_u, w \in R^+ \} \)**
  Mapping between the neighbours and the weight of the edge linking to each of them. In case of unweighted graphs, all connections get \( w = 1 \).

- **\( u\).strength = s_u \)**
  The strength of the node as the sum of all the weights of its connections.

- **\( u\).colors = \{ (c_i, a_i) : c_i \in CL, a_i \in [0, M] \} \)**
  The colors repository for each node as a mapping between the colors kept by the node and the amount of each of them. Here \( CL \) indicates the set of all initial colors and \( M \) the maximum available amount of each, which is fixed. It is worth noting that the node does not necessarily keep all the possible colors, but just those where the amount is greater than zero.

- **\( u\).ratios = \{ (c_i, r_i) : c_i \in C, r_i \in [0, 1] \} \)**
  Mapping of the give-away ratio computed for each possible color.

- **\( u\).dominant = c(u) \)**
  The current dominant color.

3.3.2 Initialization phase

The network is first initialized selecting a set of nodes as seeds that will be given a different color each, while non-seed nodes remain uncolored. All seeds will receive the same amount of color \( M \) which is fixed. The initialization_strategy function is the one responsible of sampling the seeds from the graph and can take many different approaches as defined in section 3.4. The amount of selected seeds is defined by the sample parameter \( s \in (0, 1] \), which is the fraction of nodes that should be picked. The pseudo-code of this process is presented in Algorithm 1.

3.3.3 Gather phase

In this stage, each node simply collects all amounts of colors coming from neighbours and adds it to the amounts kept in the repository. The receive function makes the necessary operations to gather all pairs of colors and corresponding
Algorithm 1: Initialization phase

**Data:** A graph $G(V, E)$; 
- a fixed amount for each color $M$; 
- a sample parameter $s$.

```plaintext
seeds = initialization_strategy(G, s)
for $u \in V$ do
    if $u \in seeds$ then
        $c = \text{new\_color}()$
        $u.colors[c] = M$
        $u.dominant = c$
    else
        $u.dominant = NA$
    end
end
```

Algorithm 2: Gather phase

**Data:** A node $u \in V$ from the graph $G(V, E)$.

```plaintext
for $v \in u\text{.neighbours}$ do
    for $(c_i, a_i) \in receive(u, v)$ do
        if $c_i \in u.colors$ then
            $u.colors[c_i] += a_i$
        else
            $u.colors[c_i] = a_i$
        end
    end
end
```

amounts $(c_i, a_i)$ that were sent from node $v$ to $u$. The pseudo-code of this stage is presented in Algorithm 2.

### 3.3.4 Apply phase

The *apply phase* has two main purposes: to decide the dominant color for the node and to compute the giveaway ratios for each color in the node. To select the dominant color, it simply checks for which of them the node has the higher amount and picks that one as dominant.

To compute the giveaway ratios, first the neighborhood is inspected, and the strength of each surrounding color is calculated as the sum of the connections to nodes with that color as dominant. Then the external ratio $d_{out}$ is calculated for
Algorithm 3: Apply phase

Data: A node $u \in V$ from the graph $G(V,E)$.

// Update dominant color
$u.dominant = \text{argmax}(u.colors)$

// Compute strengths of surrounding colors
for $v \in u.neighbours$ do
    strengths[$v.dominant$] += $u.weights[v]$
end

// Compute giveaway ratios
for $(c_i, a_i) \in u.colors$ do
    if $c_i \in \text{strengths}$ then
        $d_{out} = 1 - \text{strength}[c_i]/u.strength$
        $u.ratios[c_i] = \text{diffusion_strategy}(d_{out})$
    else
        $u.ratio[c_i] = 1$
    end
end

For each neighbouring color. This value can then be inputted to a function that decides the giveaway ratio based on a diffusion strategy. The basic choice for the diffusion strategy is the identity function which we call linear diffusion, though other two strategies are proposed in section 3.5.

For those colors that are not found in the neighborhood, the give-away ratio is set 1, which means that all the amount of those colors will be diffused.

3.3.5 Scatter phase

In the scatter phase, every node decides, for each color it possesses, the amount that needs to be sent to each neighbor. For that, it computes the amounts of each color to send to each neighbor based on the giveaway ratios (computed in the apply phase) and the strength of the connections. These amounts are then subtracted from the colors repository and sent to the neighbours. The send function makes the necessary operations to send an amount of color $c_i$ from node $u$ to node $v$. The pseudo-code of this phase is presented in Algorithm 4.

3.4 Initialization strategies

It was previously introduced the concept of seed node, this is, those nodes that will be given colors at the very beginning and from which the diffusion process
3.4. Initialization Strategies

**Algorithm 4:** Scatter phase

**Data:** A node \( u \in V \) from the graph \( G(V,E) \).

for \( v \in u.\text{neighbours} \) do
  for \( (c_i,a_i) \in u.\text{colors} \) do
    \( \text{amount} = a_i \times u.\text{ratio}[c_i] \times u.\text{weights}[v] / u.\text{strength} \)
    \( u.\text{colors}[c_i] \leftarrow \text{amount} \)
    \( \text{send}(u,v,c_i,\text{amount}) \)
  end
end

will start. However, we did not discuss how these seeds should be picked. One possibility is to choose all nodes as seeds and start competition between all of them from the beginning, similarly as is done in the Label Propagation Algorithm (LPA) [33]. However, the more seeds we chose, the more computationally complex the algorithm gets, so an approach where only a subset of the nodes is initialized (hence the name seed) is preferred. It turns out that there are lots of different strategies that can be used, and here we will discuss some of them that we consider intriguing, and that we want to compare in order to study how this initialization process affects DDCA.

**High Clustering Coefficient Sample (HCCS)**

The clustering coefficient (CC) of a node [34] expresses how close its neighbours are to form a clique (complete graph) by computing the fraction of neighbours that are also connected between them among all possible connections. More formally, the CC of a vertex \( u \in V \) is defined as

\[
CC_u = \frac{|\{v,w): v,w \in N_u, (v,w) \in E\}|}{k_u(k_u-1)} \tag{3.1}
\]

The method consists in computing the CC of each node and then selecting those with highest value according to a sample parameter \( s \in (0,1] \) which indicates the fraction of nodes that should be selected as seeds. For example, in a network of 100 nodes, \( s = 0.2 \) would result in selecting the 20 vertices with the highest CC.

Intuitively, since communities are well connected internally and separated from each other, nodes with high CC would lay inside communities and thus selecting those nodes as seeds could be a good starting point for the diffusion process, as colors will collide at the boundaries of the clusters. Nonetheless, this initialization comes with the computational overrun of having to compute the clustering coefficient for every node in the network.
High Page Rank Sample (HPRS)

Following a similar approach, we could ask ourselves what happens if we change the CC for other measure that takes into account the importance of a node inside the network. One such measure, and one of the most extensively known, is the PageRank (PR) [35] which is defined for each node \( u \in V \) by the Equation (3.2)

\[
PR_u = \frac{1 - d}{n} + \sum_{v \in N_u} \frac{PR_v}{k_v}
\] (3.2)

where \( d \), also known as the dumping factor, is usually set to 0.85.

As in the previous example, the selected seeds are those nodes with highest PR, and the amount of them to choose comes given by a sample parameter. Computing the PR also adds some overhead, although usually less than computing the CC.

High Degree Sample (HDS)

The idea in this method is the same, but now we take nodes with the highest degree (also known as hubs) as seeds. These type of nodes have a significantly larger number of links that the rest of nodes in the network, and they naturally emerge in scale-free networks.

The difference that this initialization may provide is that hubs help to effectively spread the information through the network during the diffusion process, however hubs tend to act as bridges between communities, so colors would have to compete for dominance of the regions of the graph from the very beginning of the diffusion process.

Random Sample (RS)

The simplest of all approaches is to just select a sample of nodes completely at random and let those nodes act as seeds.

The main advantage of this approach is its simplicity. Nevertheless, it comes with the drawback that in the presence of very unbalanced communities (in terms of sizes), most of the selected seeds will belong to the biggest communities while the smallest ones will probably not get any members to start as seeds.

One seed per community

This approach is rather different than those presented before. Here we adopt a more idealistic approach in which the user is able to give one seed per community beforehand, and the diffusion process starts from there. Of course, this is a much more restricted use-case since prior information is needed to run the algorithm, however studying it can help understand better
the behaviour of the process. When the seeds are selected at random, we will refer to it as Random Seed Per Community (RSPC) initialization.

3.5 Diffusion strategies

As we commented in section 3.3.4, the diffusion strategy can be replaced by defining any function \( D(d_{out}) \) where \( D(0) = 0 \) (keep all color if the neighborhood fully supports it) and \( D(1) = 1 \) (discard all color if it is not in the neighborhood). Note that the colors actually represent communities, so when the neighborhood of a node supports strongly a certain color, it means that this is the community adopted by the majority of the node’s neighbours. Since the give-away ratios (or diffusion rates) control the diffusion process, we expect to be able to influence the overall behaviour of the algorithm by changing the strategy.

We will now present three alternative diffusion functions that we want to test in order to see if the size of the identified communities can be controlled by changing them, thus giving a sense of resolution to the algorithm. A visual comparison of the three of them is found in Figure 3.1.

Figure 3.1: Comparison of different diffusion functions on \( d_{out} \).

**Linear diffusion**

This strategy is the more basic one and uses the value of \( d_{out} \) itself as the give-away ratio by defining the diffusion function as the identity function (Equation (3.3)).

\[
D_{linear}(d_{out}) = d_{out}
\]  

When there is a lot of agreement with the rest of the neighborhood, \( d_{out} \) will become close to 0 and so will the giveaway ratio, hence making the node
keep most of the color. On the contrary, while less neighbours support the color, the closer the give-away ratio will get to 1 making the node send away higher amounts. In the case of full uncertainty \( d_{out} = 0.5 \), the node will diffuse half of the amount and keep the other.

**Fast diffusion**

The idea behind this strategy is to turn the nodes into more generous givers. For that we define the function showed in Equation (3.4). It is controlled by the diffusion coefficient \( a \), for which higher values will make the diffusion process faster and faster, i.e. the nodes will get higher giveaway ratios and most of the color will be distributed to the neighbours except if there is a strong support to that specific color in the neighborhood (see Figure 3.2a).

\[
D_{fast}(a,d_{out}) = |1 - e^{-ad_{out}}|
\]  

(3.4)

**Slow diffusion**

The inverse strategy of the fast diffusion, where we try to make the nodes more selfish, is defined by Equation (3.5) and is as well controlled by a tunable coefficient \( a \). As before, higher values of the coefficient will increment the egoism of the nodes, making the give-away ratios very small and therefore making the diffusion process slower, i.e. most of the nodes will try to keep their colors unless they are surely worthless (see Figure 3.2b).

\[
D_{slow}(a,d_{out}) = \frac{|1 - e^{ad_{out}}|}{e^a}
\]  

(3.5)

It is worth noting that a slower or faster diffusion does not necessarily mean a faster or slower convergence. The terminology refers only at the speed of the spread process (how fast the colors move through the network).

### 3.6 Stopping criteria

In this section the reader can find different conditions that are checked by the algorithm after each iteration to decide whether the diffusion process has converged to a solution or not.

**Color changes**

The more natural and simple way to check convergence is to keep count of the number of nodes that change their dominant color during an iteration. In other words, at the moment at which all nodes decide that they do not
3.6. Stopping criteria

(a) Fast diffusion.  
(b) Slow diffusion.

Figure 3.2: Effect of a coefficient on the fast and slow diffusion strategies.

want to change their color, the algorithm can be stopped. We could call this situation *strong convergence*.

Although this condition looks very natural, there could be some graphs where it is not a plausible situation. For example, if there were some nodes in the boundary of a community acting as bridges to other communities, it could happen that the stream of colors that they receive goes back and forth, making the nodes alternate their dominant color indefinitely.

This drawback could be partially mitigated by relaxing the condition and just check that the number of color changes is less than a specified percentage of all the nodes. Although promising, this approach has not been implemented and remains as a research area for future studies.

**Number of communities**

To solve the problems of the previous condition, a softer one can be introduced. We can focus on the community level rather than the node one and keep track of the number of communities detected in each iteration. If the number of detected communities remains the same for (previously) fixed number of iterations, then we can assume that the algorithm has enter a state of *soft convergence*.

**Early stopping**

Even with the two previously described conditions, we might still have cases where neither strong nor weak convergence is attained. To prevent the algorithm from indefinitely keep running, we set a parameter that controls the maximum number of iterations. In case the algorithm reaches this point, we refer to it as *divergence*.

Note, however, that this does not mean that the algorithm would not
converge theoretically in case one does not limit the number of iterations.

### 3.7 Further modifications

Here we describe some other modifications of the algorithm that were implemented but were not tested in the end and which might as well be interesting for further research.

**Look-ahead**

The original design of DDCA included a look-ahead step inherited from the Tovel algorithm [23]. This is a step performed during the computation phase where instead of checking just the amounts of each color owned by a node, the information about the colors in the neighbours is also taken into account, which is somehow an estimation of the amount of each color that could come to the node in the next iteration. However, this approach was substituted by the one defined in section 3.3 to reduce computation time and make the algorithm simpler.

### 3.8 Other considerations

Note that by design, the algorithm is suitable for any kind of graph, i.e. unweighted/weighted and/or undirected/directed. This is a very nice characteristic since most of the algorithms require some adjustments from the original idea to work on other types of graphs.

It should also be fairly easy to extend the algorithm to allow the discovery of overlapping communities, since nodes end up having amounts of several different colors, and choosing several dominant colors would result into multiple memberships. However, this extension lies out of the scope of the project and hence has not been implemented nor tested.

Another benefit that the algorithm presents is its complete decentralization. Since there are no central structures and nodes just need information about their neighbours in order to operate, it makes the algorithm extremely suitable for distributed graph processing tools such as Pregel [36], GraphLab [37], PowerGraph [32] or GraphX [38]. This tools execute algorithms on networks following a think-like-a-vertex philosophy whilst they distribute the computation of a single vertex-program (just like DDCA) over a cluster of machines, which makes them the best scalable option when the networks under study grow too big.

Nonetheless, development and testing of the algorithm in such frameworks is left for further research and the implementation of the algorithm and all the different strategies described in this chapter was done in the C++ programming
language. Since efficiency was one of the main concerns and this language proofs to be fast while providing object-oriented and data abstraction capabilities. An open-source implementation of the algorithm in C++ can be found online at the author’s Github account \(^1\).

\(^1\)https://github.com/AdrianRamirezRio/masterThesis
Chapter 4

Experimental procedure

This chapter starts presenting different methods for generating artificial benchmark graphs for community detection and then describes the different datasets generated with those methods. Some of the properties of the generated graphs are presented as well as some visualizations.

4.1 Computer-generated benchmarks

Due to the great number of different methods for community detection in graphs available, the natural question of how to validate them and compare them arises. This is, we should be able to check how precise an algorithm is when recovering the communities of a benchmark network whose community structure is known beforehand.

One may obtain this information by interpreting the metadata (non-topological features) of real network, however accessing this type of information has not been always easy, thus way throughout the years, a number of computer-generated benchmarks have been designed by the community.

In the recent years, the access to real-world examples has become easier, however there are hints that the groups inferred from the metadata may not have a good correspondence, in general, with the topological community structure. Hric et al [39] showed how for a set of real-world annotated networks the similarity between the partitions found by several different methods was far from being closely related to the ground truth communities extracted from metadata. For this reason, artificial benchmarks are still widely used to test the capabilities of the developed algorithms.

Following, we will present different approaches for the generation of artificial graphs with planted community structure together with their benefits and limitations.
4.1.1 GN benchmark

A very simple way to build a network with a planted community structure is the one proposed by Girvan and Newman [40]. Based on the idea that two vertices have higher probability to be joined by a link when they belong to the same community, only two possible values for the edge probability are defined, \( p_{in} \) and \( p_{out} \) for edges pointing inside or outside communities, respectively. Note that not necessarily \( p_{in} \) and \( p_{out} \) sum up to 1, as the expected average degree is \( k = k_{in} + k_{out} = p_{in} \cdot n_C + p_{out} \cdot n_C \cdot (q - 1) \), where \( q \) is the number of clusters and \( n_C \) their size (a number \( q \) of equally-sized communities is assumed). In the original version, Girvan and Newman fix the number of nodes to 128 (4 groups of 32 nodes) and the average degree to 16, getting the relationship \( p_{in} + 3p_{out} = 1/2 \).

The usual approach is to tune the value \( k_{out} = 96p_{out} \) to get values for \( p_{in} \) and \( p_{out} \). As one increments the value of \( k_{out} \), it gets more difficult to differentiate communities from one another, and as long as \( p_{in} > p_{out} \) (\( k_{out} < 12 \)), one can expect to do better than random assignment.

Although very widely used [1], this benchmark suffers from a lot of limitations that make it too simple for algorithm comparison. It does not provide heterogeneity neither in the degree of the vertices nor in the size of the communities, hence not following the skewed distributions often reported to be found in real graphs.

4.1.2 LFR benchmark

To solve the limitations of the GN benchmark, Lancichinetti et al. [41], [42] proposed a new strategy to build a network with an introduced community structure that tries to reproduce skewed distributions for the degree and the communities sizes, to account for the heterogeneity often found in real networks, distributions which are assumed to be power-laws with exponents \( \gamma \) and \( \beta \) respectively.

As input parameters the algorithm takes these two exponents, the number of nodes \( n \), the average degree \( k \) and the mixing parameter \( \mu \) which controls the level of mixing between communities. The degree of every node is decided according to a power-law distribution with exponent \( \gamma \) (where the minimum and maximum degree are chosen so that they comply with the average degree \( k \)) and shares a fraction \( 1 - \mu \) of its edges with the other nodes of the same community and a fraction \( \mu \) with the rest of nodes in the graph. The sizes for the communities are chosen from a power-law distribution with exponent \( \beta \) with the restriction that they sum up to \( n \).

In Figure 4.1 we show examples of graphs generated with the LFR benchmark for different values of \( \mu \) and \( \beta \), as well as the number of communities \( (C) \), and
4.1. COMPUTER-GENERATED BENCHMARKS

Figure 4.1: Impact of $\mu$ and $\beta$ on cluster detectability in graphs generated with LFR benchmark. Visualization uses “Spring” layout from networkx[43] Python’s library.

the conductance ($\phi$) for each graph. We observe how by incrementing the value of $\mu$, the communities get more and more fuzzy. It is also interesting the impact of $\beta$, which controls the heterogeneity of community sizes, hence the greater the value the more diverse the communities. One may ask why the conductance of the graphs does not change for different values of $\beta$ when we keep $\mu$ constant, and the explanation is that as the mixing parameters are strongly concentrated by construction around the value $\mu$, all communities have a conductance close to this value [1] (the conductance shown here is the average of the conductance of all communities).

For small values of $\mu$, communities are well separated and can be easier to detect, and as $\mu$ increases, it becomes harder and harder. However, there exists a limit on the value of $\mu$ beyond which one can not expect better performance than random guessing. Though it is commonly mistaken that the threshold should be $\mu \leq 1/2$, in Fortunato’s survey [1] the reader can find a full theoretical proof
that, in order to have detectable communities, Equation (4.1) should be fulfilled for every community $C$

$$\mu \leq 1 - \frac{K_C}{2m}$$

(4.1)

where $K_C$ is the sum of the degrees of vertices inside the community. That is, the limit $\mu_{\text{max}} = \min_{C} \left( 1 - \frac{K_C}{2m} \right)$, can be defined so that $\mu \leq \mu_{\text{max}}$ must be satisfied in order to have detectable communities.

Note that the original version of the benchmark [41] (the one described in this section) could only produce undirected and unweighted graphs with no overlapping between communities, though in the more recent version [42], it was extended for such cases.

The importance of the LFR benchmark for algorithm testing and comparison is such that is considered a standard for the research community [1]. It has been used to compare the performance of different algorithms [18], [26] and developers have also used it to demonstrate competitiveness of new algorithms against existing ones [31], [44]–[46].

### 4.1.3 Deterministic hierarchical network models

Another type of network that can be constructed so that the degree distribution follows a power-law was proposed in [47] through a deterministic iterative process. By construction, these graphs present a hierarchical structure that can be interpreted as a planted partition at different levels to test the resolution capabilities of detection algorithms [16].

The building process is as follows: The algorithm takes as input the base $b$ and the replication factor $r$. It starts with a complete graph of $b$ vertices, which constitutes the basic module of the network, and selects one of the vertices as the center. In the next step, it creates $b - 1$ replicas of the module and links the peripheral nodes (those that are not selected as centers) to the center of the original module, which becomes also the center of the new module. This process is iterated as many times as indicated by $r$, taking as the basic module the one generated in the previous step. In this way, each of the modules generated can be thought of as a community, formed by lower-level modules which also act as communities, having in the end a hierarchical group structure.

It must be noticed that the number of nodes grows exponentially with respect to the replication factor, and the final network has a total of $b^r$ nodes. The number of (non-trivial) hierarchies in such networks is $r - 1$, and the number of communities and their sizes for each level $i \in \{1, r - 1\}$ are $b^{r-i}$ and $b^i$ respectively.

An example of a network with $b = 10$ and $r = 3$ can be found in Figure 4.2 where the planted communities at the two levels of the hierarchy are also showed.
4.2 Datasets

This section presents the different generated benchmarks and the properties of the graphs contained in them.

4.2.1 Dataset 1: LFR generated graphs

Since the GN benchmark was considered too simple, the first tests tackled directly graphs generated with the LFR benchmark (section 4.1.2), which present scale-free topologies and diversity of community sizes. The dataset is a replica of the unweighted and undirected benchmark used by Lancichinetti and Fortunato [18] where the value of the mixing parameter $\mu$ is gradually incremented in order to compare the performance of the algorithms.

Since there are a lot of parameters to tune, they were chosen according to those presented by the authors [1] and are outlined in Table 4.1. Two different graph sizes are generated, small graphs with 1000 nodes and bigger graphs with 5000; two configurations for the community sizes are used as well, one where communities are of “small size” (from 10 to 50 nodes) and one with communities
twice as big (20 to 100 nodes); the mixing parameter $\mu$ is tested from 0.1 to 0.8 at increments of 0.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>${1000, 5000}$</td>
<td>Number of nodes</td>
</tr>
<tr>
<td>$k$</td>
<td>20</td>
<td>Average degree</td>
</tr>
<tr>
<td>$k_{\text{max}}$</td>
<td>50</td>
<td>Maximum allowed degree</td>
</tr>
<tr>
<td>$\mu$</td>
<td>${0.1, 0.2...0.8}$</td>
<td>Mixing parameter</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>2</td>
<td>Degree dist. exp.</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
<td>Community size dist. exp.</td>
</tr>
<tr>
<td>$[\text{min}_C, \text{max}_C]$</td>
<td>${S=[10, 50], B=[20, 100]}$</td>
<td>Community size range</td>
</tr>
</tbody>
</table>

Table 4.1: Chosen values for LFR benchmark parameters in Dataset 1.

With the described configuration, we generated a total of 320 undirected LFR benchmark graphs (10 graph realizations for each value of $\mu$) using the implementation provided by the authors and freely available at Fortunato’s personal page\(^1\). Since the code only runs one realization of the benchmark (only produces one graph with given parameters) a higher order Python \(^{48}\) script that automates the process was implemented. Additionally, all the produced graphs were tested to comply with the theoretical limit presented in Equation (4.1) to ensure that the planted community structure is theoretically correct.

In Figure 4.3 we can find an example of one of the generated graphs together with its degree and community size distributions. The scale-free and heterogeneity of communities can be appreciated.

Later, we used this graphs to evaluate the performance of DDCA (with the different initialization strategies) to check the impact of the chosen initialization method on the performance. We also compare these results to those obtained by the well-known algorithms Louvain and Infomap, which were presented in section 2.2.

### 4.2.2 Dataset 2: Unbalanced communities

The dataset described in the previous section was good to test the performance of the algorithms in general, but did not include any extremely unbalanced communities, i.e. graphs where the difference between the sizes of the bigger and smaller communities is of order of magnitudes. To test the behaviour of the algorithm in such type of graphs, a second dataset focused on exploiting this property was generated.

\(^1\)https://sites.google.com/site/santofortunato/inthepress2
Figure 4.3: Visualization of a LFR-generated graph from Dataset1 with 1000 nodes and $\mu = 0.1$. Plotted together the degree and community size distributions.

We used, as well as in section 4.2.1, the LFR benchmark software, but this time we tuned the parameters accordingly to our objectives. To allow very unbalanced and heterogeneous communities, we had to tune the parameters as shown in Table 4.2.

We decide to increment the number of nodes to 10000 and set $\beta$ to 1.5, so that communities are more diverse. The average degree was also increased to 100, while $\gamma$ was left the same (2). Then, the maximum degree of the nodes, $k_{\text{max}}$, was tested at two levels, 900 and 9000, so that we generate unbalanced graphs in different orders of magnitude. As well as in Dataset 1, the mixing parameter $\mu$ was tested at different levels, although this time we restricted it to just three values (0.1, 0.3 and 0.5) for simplicity.

Only one realization of the benchmark was generated this time, leading to a total of 6 undirected and unweighted graphs, where again the theoretical
Parameter & Value
\[ n \] & 10000 \\
\[ k \] & 100 \\
\[ k_{\text{max}} \] & \{900, 9000\} \\
\[ \mu \] & \{0.1, 0.3, 0.5\} \\
\[ \gamma \] & 2 \\
\[ \beta \] & 1.5 \\

Table 4.2: Chosen values of LFR benchmark parameters in Dataset 2.

conditions for having detectable planted community structure were checked for each of them.

<table>
<thead>
<tr>
<th>Graph</th>
<th>( n )</th>
<th>( m )</th>
<th>( k_{\text{avg}} )</th>
<th>( k_{\text{max}} )</th>
<th>( C )</th>
<th>( \text{min}_C )</th>
<th>( \text{max}_C )</th>
<th>( \mu )</th>
<th>( \mu_{\text{max}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_1 )</td>
<td>10k</td>
<td>460k</td>
<td>91.95</td>
<td>7898</td>
<td>18</td>
<td>19</td>
<td>7895</td>
<td>0.1</td>
<td>0.12</td>
</tr>
<tr>
<td>( G_2 )</td>
<td>10k</td>
<td>436k</td>
<td>87.18</td>
<td>6361</td>
<td>18</td>
<td>17</td>
<td>5681</td>
<td>0.3</td>
<td>0.35</td>
</tr>
<tr>
<td>( G_3 )</td>
<td>10k</td>
<td>500k</td>
<td>100.10</td>
<td>8463</td>
<td>21</td>
<td>19</td>
<td>4334</td>
<td>0.5</td>
<td>0.51</td>
</tr>
<tr>
<td>( G_4 )</td>
<td>10k</td>
<td>481k</td>
<td>96.20</td>
<td>899</td>
<td>68</td>
<td>28</td>
<td>810</td>
<td>0.1</td>
<td>0.84</td>
</tr>
<tr>
<td>( G_5 )</td>
<td>10k</td>
<td>481k</td>
<td>96.16</td>
<td>895</td>
<td>66</td>
<td>29</td>
<td>887</td>
<td>0.3</td>
<td>0.87</td>
</tr>
<tr>
<td>( G_6 )</td>
<td>10k</td>
<td>493k</td>
<td>98.56</td>
<td>893</td>
<td>59</td>
<td>30</td>
<td>812</td>
<td>0.5</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 4.3: Generated graphs for Dataset 2.

In Table 4.3 some properties of the generated graphs can be found, such as the number of nodes and edges, the average and maximum degree, the number of communities (\( C \)), their minimum and maximum size (\( \text{min}_C, \text{max}_C \)) as well as the approximated mixing parameter and the theoretical limit for it in order to have detectable communities (\( \mu_{\text{max}} \)) like was explained in section 4.1.2.

Additionally we present in Table 4.4 some goodness measures of the planted community structure by LFR (conductance and coverage) as well as the average clustering coefficient of the network.
### 4.2. Datasets

<table>
<thead>
<tr>
<th>Graph</th>
<th>$\phi$</th>
<th>$\theta$</th>
<th>$CC_{avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>0.82</td>
<td>0.98</td>
<td>0.77</td>
</tr>
<tr>
<td>$G_2$</td>
<td>0.54</td>
<td>0.79</td>
<td>0.57</td>
</tr>
<tr>
<td>$G_3$</td>
<td>0.33</td>
<td>0.50</td>
<td>0.84</td>
</tr>
<tr>
<td>$G_4$</td>
<td>0.82</td>
<td>0.90</td>
<td>0.63</td>
</tr>
<tr>
<td>$G_5$</td>
<td>0.54</td>
<td>0.70</td>
<td>0.38</td>
</tr>
<tr>
<td>$G_6$</td>
<td>0.33</td>
<td>0.50</td>
<td>0.21</td>
</tr>
</tbody>
</table>

Table 4.4: Goodness measures for graphs of Dataset 2.

Figure 4.4: Visualization of the ground truth communities of graphs from Dataset 2 with extreme imbalance ($k_{max} \approx 9000$).

Figure 4.5: Visualization of the ground truth communities of graphs from Dataset 2 with high imbalance ($k_{max} \approx 900$).

The reader can also find visualization for the six generated graphs; in
Figure 4.4 we find the three graphs with $k_{\text{max}} \approx 9000$ and in Figure 4.5 the other three with $k_{\text{max}} \approx 900$. In both cases, we can appreciate how the community detectability is degraded as $\mu$ is incremented, and visually check the diversity of communities (from very big to very small).

In this dataset, we run DDCA with random seed expansion, i.e., a seed expansion strategy as described in section 3.4 where one seed is selected at random from each ground truth community, and linear diffusion.

### 4.2.3 Dataset 3: Hierarchical and balanced communities

We are not only interested in studying the performance of the algorithm in the type of graphs previously described. As we have stated before, it is also important to check the capabilities of algorithms to find clusters at different hierarchical levels.

One way to obtain graphs with such hierarchical structure is to use the network construction algorithm described in section 4.1.3. Hence, we implemented the method and generated the graphs which are described in Table 4.5.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$b$</th>
<th>$r$</th>
<th>$n$</th>
<th>$m$</th>
<th>$k_{\text{avg}}$</th>
<th>$\overline{CC}$</th>
<th>$\text{min}_C$</th>
<th>$\text{max}_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_1$</td>
<td>4</td>
<td>6</td>
<td>4096</td>
<td>13173</td>
<td>6.43</td>
<td>0.74</td>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>$H_2$</td>
<td>5</td>
<td>5</td>
<td>3125</td>
<td>10904</td>
<td>6.98</td>
<td>0.56</td>
<td>5</td>
<td>625</td>
</tr>
<tr>
<td>$H_3$</td>
<td>10</td>
<td>4</td>
<td>10000</td>
<td>39951</td>
<td>7.99</td>
<td>0.53</td>
<td>10</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 4.5: Properties of the generated graphs in Dataset 3.

We observe how by tuning the base ($b$) and the replication factor ($r$) we can generate graphs with various properties, obtaining different sizes and levels of hierarchy, with varying sizes for the underlying communities. We get as well diversity in the average degree of the graphs and the inter-connectivity between them.

As an example, a visualization of $H_3$ and its ground truth communities at different scales is presented in Figure 4.6. It can be appreciated how the recursive way of building the network generates a hierarchy of balanced communities (at each level, all of them have the same size), which makes it a good starting point to study the resolution capabilities of DDCA. Also, we can see how the deeper we go into the hierarchy the lower the conductance gets, indicating that the smaller communities are fuzzier and harder to detect.

### 4.2.4 Dataset 4: Hierarchical and unbalanced communities

To take things to the next level, we want to build networks where the planted community structure is hierarchical at the same time as unbalance. To do
4.2. Datasets

(a) 1st level (bottom).
(b) 2nd level.
(c) 3rd level (top).

Figure 4.6: Visualization of the ground truth communities of $H_3$ seen at the different levels of the hierarchy.

so, we make use of an extension of the LFR benchmark software available at Fortunato’s personal page\footnote{https://sites.google.com/site/santofortunato/inthepress2} that allows the introduction of a two-level hierarchy, while maintaining the scale-free and community size heterogeneity properties. Due to this limitation of only two levels of hierarchy, we talk about micro-communities (bottom level) embedded into macro-communities (top level).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>10k</td>
<td>Number of nodes</td>
</tr>
<tr>
<td>$k$</td>
<td>100</td>
<td>Average node degree</td>
</tr>
<tr>
<td>$k_{max}$</td>
<td>${200^{5-8},500^{1-4}}$</td>
<td>Max. node degree</td>
</tr>
<tr>
<td>$\mu_m$</td>
<td>${0.1^{1,3,5,7},0.3^{2,4,6,8}}$</td>
<td>Mixing param. for micro-comm.</td>
</tr>
<tr>
<td>$\mu_m$</td>
<td>${0.2^{1,3,5,7},0.3^{1,3,5,7}}$</td>
<td>Mixing param. for macro-comm.</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>2</td>
<td>Exponent for degree dist.</td>
</tr>
<tr>
<td>$\beta$</td>
<td>${1^{1-4},1.5^{5-8}}$</td>
<td>Exponent for comm. size dist.</td>
</tr>
<tr>
<td>$\text{min}_{C_m}$</td>
<td>5</td>
<td>Min. macro-community size</td>
</tr>
<tr>
<td>$\text{max}_{C_m}$</td>
<td>${100^{1,2},400^{5,6},1k^{3,4},4k^{7,8}}$</td>
<td>Max. micro-community size</td>
</tr>
<tr>
<td>$\text{min}_{C_M}$</td>
<td>$1k$</td>
<td>Min. macro-community size</td>
</tr>
<tr>
<td>$\text{max}_{C_M}$</td>
<td>${4k^{5,6},9k^{1-4,7,8}}$</td>
<td>Max. macro-community size</td>
</tr>
</tbody>
</table>

Table 4.6: Chosen values of LFR benchmark parameters used to generate graphs from Dataset 4. The super indices $i$ showed in some values indicate the graphs $I_i$ for which generation the parameter took that value.

In Table 4.6 the parameters used for the generation of the eight graphs that conform Dataset 4 are presented. The different values of the parameters are chose
as similar as possible to those chosen for Dataset 2 (Table 4.2) although some tuning was needed in order to get the desired imbalance between communities of the different graphs.

<table>
<thead>
<tr>
<th>Graph</th>
<th>n</th>
<th>m</th>
<th>$k_{avg}$</th>
<th>$k_{max}$</th>
<th>$C_m$</th>
<th>$C_M$</th>
<th>$[\text{min} C_m, \text{max} C_m]$</th>
<th>$[\text{min} C_M, \text{max} C_M]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>10k</td>
<td>499k</td>
<td>99.70</td>
<td>499</td>
<td>47</td>
<td>3</td>
<td>[157,314]</td>
<td>[1751,5194]</td>
</tr>
<tr>
<td>$I_2$</td>
<td>10k</td>
<td>494</td>
<td>98.74</td>
<td>500</td>
<td>45</td>
<td>4</td>
<td>[149,333]</td>
<td>[1874,3465]</td>
</tr>
<tr>
<td>$I_3$</td>
<td>10k</td>
<td>492k</td>
<td>98.34</td>
<td>496</td>
<td>39</td>
<td>2</td>
<td>[25,985]</td>
<td>[1538,8462]</td>
</tr>
<tr>
<td>$I_4$</td>
<td>10k</td>
<td>493k</td>
<td>98.65</td>
<td>500</td>
<td>47</td>
<td>5</td>
<td>[22,982]</td>
<td>[1158,3091]</td>
</tr>
<tr>
<td>$I_5$</td>
<td>10k</td>
<td>496k</td>
<td>99.27</td>
<td>200</td>
<td>98</td>
<td>6</td>
<td>[35,400]</td>
<td>[1004,2667]</td>
</tr>
<tr>
<td>$I_6$</td>
<td>10k</td>
<td>499k</td>
<td>99.80</td>
<td>200</td>
<td>103</td>
<td>6</td>
<td>[36,388]</td>
<td>[1160,2657]</td>
</tr>
<tr>
<td>$I_7$</td>
<td>10k</td>
<td>496k</td>
<td>99.28</td>
<td>200</td>
<td>34</td>
<td>3</td>
<td>[37,3847]</td>
<td>[1676,5433]</td>
</tr>
<tr>
<td>$I_8$</td>
<td>10k</td>
<td>499k</td>
<td>99.80</td>
<td>200</td>
<td>39</td>
<td>3</td>
<td>[37,1695]</td>
<td>[1695,4752]</td>
</tr>
</tbody>
</table>

Table 4.7: Properties of the generated graphs for Dataset 4.

We also show in Table 4.7 the main properties of the generated graphs, namely the number of edges and nodes, average and maximum degree, number of micro ($C_m$) and macro ($C_M$) communities, and the size ranges of the micro and macro communities. The generated graphs have a rather heterogeneous configuration of imbalance. For example, in $I_1$ the ratio between the biggest and smallest micro-communities is 2 whereas for $I_4$ is around 44 and for $I_8$ it gets as high as 100 (two orders of magnitude). With respect to the macro-communities, the differences are not so big but we still get small ratios (< 2 for $I_2$) and big ones (> 5 for $I_3$). We find as well diversity in the number of micro-communities (34 for $I_8$ and 103 for $I_6$) and in the number of macro-communities (2 for $I_3$ and 6 for $I_5$).

In Figure 4.7 and Figure 4.8, we find the visualizations of the ground truth communities in $I_5$ and $I_7$ as a clear example of the type of graphs that have been generated where the community imbalance can be visually appreciated.
4.2. Datasets

Figure 4.7: Visualization of the ground truth communities for the two levels of hierarchy in $I_5$.

(a) Bottom level.  
(b) Top level.

Figure 4.8: Visualization of the ground truth communities for the two levels of hierarchy in $I_7$.

(a) Bottom level.  
(b) Top level.
Chapter 5

DDCA performance

In the following sections we will describe the different experiments performed and results found over the datasets introduced in chapter 4. All the executions of the different algorithms exposed in this chapter were performed on a server machine provided by RISE SICS\(^1\) with 126 GB of RAM and Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz processors (no parallel computation was performed at algorithm level).

In all experiments involving DDCA, we set a maximum number of 150 iterations, no dominant color changes for strong convergence condition and 10 iterations with the same number of communities as soft convergence condition.

The execution of the different algorithms and versions of DDCA over the different datasets where automated with Python scripts and the visualizations of the results have been generated with seaborn and matplotlib [49] libraries and with the help of jupyter notebooks [50]. All the related code can be found on the github repository of the project \(^2\).

5.1 Effect of initialization strategy

In this experiment we want to test the different initialization strategies of DDCA and see how the selection of the seeds impacts the performance of the algorithm as well as its convergence. In order to do that, we run DDCA with linear diffusion and all the versions for initialization given in section 3.4 on the Dataset 1 (section 4.2.1).

For HCCS, HPRS, HDS and RS, we set a sample size of 0.15, thus selecting around 150 seeds for the graphs with \(n = 1000\) and 750 for the ones with \(n = 5000\). Note that since the minimum allowed size for a community is 10, the algorithm

\(^1\)https://www.sics.se/about-rise-sics
\(^2\)https://github.com/AdrianRamirezRio/masterThesis
always has the possibility of discarding colors until reaching the real number of communities (there are a maximum of $n/10$ communities). Additionally, we include an initialization where one node from each ground truth community is randomly selected as a seed (RSPC). For the cases of RS and RSPC that are affected by random choices, we execute 10 different runs and average the results. To measure the performance, a bunch of metrics are calculated over the resulted partitions, including those described in section 2.3.

In Figure 5.1 we show the performance in terms of the Normalized Mutual Information (Equation (2.8)). This type of visualization of the results has been widely used [1], [18] and consists in plotting the mixing parameter $\mu$ (x axis), against the average obtained $NMI$ of the algorithm (y axis). As the value of $\mu$ increases the communities become less internally connected and more mixed.
5.1. Effect of initialization strategy

Figure 5.2: Number of iterations of different initializations for DDCA on Dataset 1, separated by number of nodes and sizes of the communities.

between them hence more difficult to detect, so we expect a deterioration in the performance. At the same time, the results are presented in four different plots, one for each combination of graph and communities sizes.

We find that there is not much difference among the results obtained by the distinct seed-choosing strategies. This suggests that the algorithm is relatively robust to the selected seeds, although in Figure 5.2 we see that the number of iterations needed to converge does indeed depend on the initialization. The RSPC initialization shows the best convergence, probably because the number of initial seeds matches the number of ground-truth communities, whereas the rest of initialization strategies start with a higher number of seeds.
5.2 Effect of controlled diffusion

We are as well interested in studying how the controlled-diffusion (use of give-away ratios in the diffusion process) is affecting the results. For that, we run a version of the algorithm where the colors are sent to the neighbours in a non-controlled way (i.e. all the color possessed by the node is distributed to its neighbours) and we compare to the results of RSPC initialization presented in the former experiment. As well, the results are averaged over 10 different runs where we select the same nodes that where used by RSPC as seeds to start both algorithms in the same state.

We find that the controlling process is essential to the good performance of the algorithm (Figure 5.3), as a full diffusion process is only able to detect the ground truth communities when they are very well separated. A further study revealed...
that this behaviour might be explained because a non-controlled diffusion tends to underestimate the number of communities.

5.3 Detection of unbalanced communities

We are also interested on studying the behaviour of the algorithm and its different initializations in graphs where the communities are very unbalanced, that is, the difference between the smaller and bigger communities is very high. With that purpose in mind we designed the Dataset 2 (section 4.2.2). We run DDCA with the same initialization methods as in section 5.1, this time with a sample parameter of 0.01 (around 100 seeds selected).

![Graphs showing Adjusted Rand Index, Conductance, and Number of iterations for different strategies.](image)

**Figure 5.4: Results of the different initializations for DDCA on Dataset 2.**

In Figure 5.4 we find the Adjusted Rand Index value (Equation (2.6)) obtained by the different strategies. We choose this value instead of the Normalized Mutual Information because the ARI is more robust to results where the imbalance is
not detected, i.e. algorithms that try to put small clusters inside bigger ones are more penalized by \textit{ARI} than \textit{NMI}. As before, we observe that there are not great differences between the diverse initializations, except for RSPC in $G_1$ and HCCS in $G_2$ where results are bad in comparison to the rest of the initializations. Additionally we plot the conductance $\phi$ of the obtained partitions, where we also find very similar results among the different to choose the initial seeds.

Independently of the chosen initialization, DDCA has problems to detect the group structure when the difference in sizes is of two orders of magnitude ($G_1$, $G_2$ and $G_3$), and specially when the communities are very mixed ($G_3$). If we look at the run number of iterations, we observe that the algorithm has trouble converging in these graphs. On the other hand, when the difference between sizes is reduced to one order of magnitude, the algorithm does a pretty good job, specially with RSPC and RS initializations, with the former one presenting the faster convergence. The specific values of different monitored metrics obtained by RSPC are presented in Table 5.1.

<table>
<thead>
<tr>
<th>Graph</th>
<th>\textit{ARI}</th>
<th>\textit{NMI}</th>
<th>\textit{AMI}</th>
<th>$\phi$</th>
<th>$\theta$</th>
<th>\textit{#iters}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>0.427</td>
<td>0.633</td>
<td>0.500</td>
<td>0.302</td>
<td>0.488</td>
<td>150.0</td>
</tr>
<tr>
<td>$G_2$</td>
<td>0.872</td>
<td>0.913</td>
<td>0.871</td>
<td>0.455</td>
<td>0.343</td>
<td>150.0</td>
</tr>
<tr>
<td>$G_3$</td>
<td>0.370</td>
<td>0.725</td>
<td>0.616</td>
<td>0.297</td>
<td>0.210</td>
<td>150.0</td>
</tr>
<tr>
<td>$G_4$</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.817</td>
<td>0.897</td>
<td>4.5</td>
</tr>
<tr>
<td>$G_5$</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.538</td>
<td>0.700</td>
<td>20.5</td>
</tr>
<tr>
<td>$G_6$</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.333</td>
<td>0.500</td>
<td>13.5</td>
</tr>
</tbody>
</table>

Table 5.1: Results of DDCA with RSPC initialization on Dataset 2.

We tried a new run of DDCA with RSPC initialization increasing the number of maximum iterations to 500 to test whether it would be able to converge with more time allowed and get better results, however we obtained still the same: good convergence for $G_4$, $G_5$ and $G_6$ and no convergence for the other three graphs.

To further study what is going wrong in the identification of this very unbalanced graphs, we can measure the \textit{precision} and \textit{recall} of the resulted partition and the ground truth. The precision of a detected cluster is the fraction of its nodes that belong to the same ground-truth community. The recall of a detected cluster is the fraction of nodes from the ground-truth community that have been detected over the total size of the ground truth community. Note that it is possible to define in the same way the precision and recall for the ground-truth community.

In Figure 5.5a we find the precision recall of the detected communities by one of the runs of DDCA with RSPC initialization over the graph $G_1$. From the precision, we extract that none of the ground truth communities are correctly
5.3. Detection of unbalanced communities

Figure 5.5: Precision and recall of the detected communities in extremely unbalanced graphs of Dataset 2, sorted by the size of the detected communities.

identified except for one formed by 932 (third biggest one). The two biggest communities show the same behaviour, a very high precision but low recall, meaning that they are subsets of bigger ground truth communities that haven’t been fully detected. The middle sized communities are, in general, fully recovered as indicated by a perfect recall, though nodes of other communities are merged with them as the precision is lower. The rest of communities (mostly small ones) have a recall near to zero and a precision near to 0.5, meaning that these clusters are a mix of nodes coming from different ground truth communities. This explains the poor performance of the algorithm in terms of the \textit{ARI} metric, as it has been unable to identify properly most of the communities.

The same precision-recall chart for \(G_2\) is found in Figure 5.5b, although this time is harder to identify bad behaviours since the algorithm has performed a lot better. We identify again that the biggest community is not totally detected, with some small fraction of the nodes being relocated. On the other hand, the algorithm is no longer merging the smallest communities, thus explaining the good results obtained.

Similarly, analyzing the precision and recall for the results on \(G_3\) (Figure 5.5b) we find that the biggest community has been found almost entirely (high recall) but together with a lot of nodes from other communities (low precision). For the
second bigger community, we find low precision and recall, suggesting that these two biggest communities have been merged into a bigger one, and the community formed by 395 nodes is a remaining part of this merging. On the other hand, most of the small communities are being detected correctly.

5.4 Effect of diffusion strategy

In the experiments described in the following sections, we run DDCA with RSPC initialization (one seed per community selected at random) and tested the three versions of diffusion: linear, fast\((a = 25)\) and slow\((a = 25)\). The results shown are averaged among 10 different runs to avoid random fluctuations. We chose RS with one seed per community as initialization method as we have already seen that DDCA is robust to the selection of seeds in terms of detection performance, although the convergence times are much faster when it is initialized with one seed per ground truth community.

It is worth noting that in these set of experiments we do not focus on the ability of the algorithm to infer the hierarchical community structure, but on the ability of the algorithm to find communities at different resolutions, controlled by the number of selected seeds, and test whether the found communities match those found at that level of resolution in the hierarchy.

5.4.1 Balanced hierarchical graphs

We want to test the effect of the different diffusion strategies on balanced hierarchical graphs, i.e. graphs whose communities present a hierarchical structure (smaller clusters embedded in bigger ones) but where each level of the hierarchy contains communities of similar size. An example of this type of structure is that of the graphs generated for Dataset 3 (section 4.2.3).

The results obtained confirm the hypothesis that the number and sizes of the detected communities can be controlled via the type of diffusion strategy used and the number of selected seeds. In Figure 5.6, Figure 5.7 and Figure 5.8 we find the Adjusted Rand Index and the number of detected communities for each of the diffusion strategies at the different levels of resolution, which are then compared with the levels of the hierarchy in \(H_1\), \(H_2\) and \(H_3\).

Although the three graphs present different size and number of communities as well as different community internal and external density (section 4.2.3), the results that we obtain for the three graphs show the same behaviour for the three different diffusion strategies. Studying the Adjusted Rand Index values of the obtained results (figures 5.6a, 5.7a and 5.8a) we observe that the bottom levels’ communities are accurately recovered by the slow diffusion strategy with high
5.4. Effect of diffusion strategy

The slow diffusion obtaining the best results in the more fine-grained communities for the three graphs is an expected result. The reason behind this behaviour can be found if we focus on the number of communities detected (figures 5.6b, 5.7b and 5.8b). We see how the slow and linear diffusion strategies tend to detect a number of communities (very) close to the number of real ones, which makes sense since both of them start with that precise number of seeds. Nonetheless, for the smallest communities the behaviour is different, since we can appreciate how the linear diffusion is not able to retain the high number of communities and starts merging them, thus finding less and bigger ones, whereas the mechanism behind slow diffusion precisely allows to avoid this “over-merging”.

values of $ARI (> 0.85)$ that show that the algorithm is recovering the ground truth communities very accurately. On the other hand, the middle levels’ ones are better detected by the linear diffusion. None of the methods for color distribution seems to work specially well for the coarser communities, although these should be the better-defined ones according to the conductance measure (section 4.2.3). More specifically, we observe how the overall performance of the algorithm decreases as we try to recover communities of higher levels of the hierarchy.

Figure 5.6: Results obtained by DDCA with RSPC initialization on $H_1$ with the different diffusion strategies at the various levels of the hierarchy.
Figure 5.7: Results obtained by DDCA with RSPC initialization on $H_2$ with the different diffusion strategies at the various levels of the hierarchy.

Table 5.2: Results of DDCA with RSPC initialization and the different diffusion strategies on graph $H_3$ of Dataset 3.

<table>
<thead>
<tr>
<th></th>
<th>GT</th>
<th>$\text{diff}_{\text{slow}}$</th>
<th>$\text{diff}_{\text{linear}}$</th>
<th>$\text{diff}_{\text{fast}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>1000</td>
<td>1000</td>
<td>548.2</td>
<td>122.6</td>
</tr>
<tr>
<td>$\phi_1$</td>
<td>0.41</td>
<td>0.40</td>
<td>0.46</td>
<td>0.30</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>0.45</td>
<td>0.44</td>
<td>0.54</td>
<td>0.43</td>
</tr>
<tr>
<td>$C_2$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99.8</td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>0.59</td>
<td>0.42</td>
<td>0.58</td>
<td>0.32</td>
</tr>
<tr>
<td>$\theta_2$</td>
<td>0.65</td>
<td>0.48</td>
<td>0.64</td>
<td>0.45</td>
</tr>
<tr>
<td>$C_3$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>0.77</td>
<td>0.38</td>
<td>0.73</td>
<td>0.40</td>
</tr>
<tr>
<td>$\theta_3$</td>
<td>0.84</td>
<td>0.51</td>
<td>0.81</td>
<td>0.49</td>
</tr>
</tbody>
</table>

The counterpart is not however obtained. We would expect from the fast diffusion to be better suited for detecting bigger communities, thus to better detect the communities at the top of the hierarchy. The fast diffusion does get the best results on the top level communities of $H_1$ (Figure 5.6a) by a very small difference in comparison to the linear method, but obtains worst results than the linear diffusion in the other two graphs (figures 5.7a and 5.8a).
5.4. Effect of diffusion strategy

Figure 5.8: Results obtained by DDCA with RSPC initialization on $H_3$ with the different diffusion strategies at the various levels of the hierarchy.

A possible explanation is that the behaviour of the fast diffusion method is to try to get rid of useless colors, i.e. it will try to reduce the number of initial colors (seeds) until it finds a partition with bigger clusters than the other strategies. This is actually observed in the bottom levels of the hierarchy (figures 5.6b, 5.7b and 5.8b), and suggests that for the fast diffusion to work, one should provide it more seeds than real clusters, in other words, it is not suitable for one seed per community initialization (although it may work for others).

Another explanation for the bad results of the fast diffusion function is that it is more similar to the uncontrolled diffusion than the other two approaches, so it could be that the effect of the control is not strong enough.

In Table 5.2 the results of the experiment on $H_3$ are shown. Here we focus on goodness measure values such as conductance and coverage rather than partition similarity metrics. The linear diffusion obtains the best values for both conductance and coverage in all cases in comparison to the other diffusion methods. We even observe that the linear diffusion obtains higher values in the bottom level of the hierarchy (1000 communities of 10 nodes) than those of the ground truth communities, suggesting that there might be another hierarchical level of around 500 nodes that is not considered by the construction method of the graph as a real community structure.
5.4.2 Unbalanced hierarchical graphs

Although the graphs of Dataset 3 are a good starting point to check the capabilities of community detection algorithms in hierarchical structures, they lack a main property of real life networks: heterogeneity of communities in terms of sizes.

With that purpose we built Dataset 4 (section 4.2.4) and now we present the results obtained by DDCA with RSPC initialization for the three different diffusion strategies. As this type of graphs have only two hierarchical levels, we present the results over the eight graphs in one barplot for each resolution level, as shown in figures 5.10 and 5.9. As in the previous section, we study the impact of the diffusion strategy on the partition recovery metric $ARI$, as well as the impact on the number of identified communities.

![Figure 5.9: Results obtained by DDCA with RSPC initialization on the micro-communities of the graphs of Dataset 4 with the different diffusion strategies.](image)

Figure 5.9a shows the results for the bottom level of the hierarchy (micro-communities). We observe an outstanding performance for the linear diffusion strategy, which gets almost a perfect score in all of them. The slow strategy also gets very good results except for graph $I_7$ (the most unbalanced one with a ratio between sizes of bigger and smaller micro-community of 103), suggesting that this approach cannot handle extreme imbalance as well as the linear one. With respect to the fast strategy, it is again the worst-performing one except in graph $I_2$ where it also gets an almost perfect score. Again this seems to be explained by the fact that this approach discards initial colors (Figure 5.9b).
5.5. Comparison to other detection algorithms

However, the observed behaviour changes for the top level of the hierarchy (Figure 5.9). None of the three strategies seem to do an overall good job in recovering the planted community structure. The linear one poses itself as the best one again, except for graphs $I_5$ and $I_7$ where it is surpassed by the fast diffusion which gets almost perfect scores. In these two cases it looks like the expected effect of the fast diffusion (work better on bigger communities) is taking place, although in the rest of the graphs (except for $I_1$) it discards again some of the initial colors. The slow diffusion obtains the worst results on this top level, something that we already observed in the results of the previous section. Unexpectedly the best results for the top hierarchical level are obtained in graphs $I_5$ and $I_7$, which have a mixing parameter for the macro-communities higher than that for the micro ones. This suggest that higher values of the mixing parameter for the bottom-level communities affects the detectability of the macro-communities as well.

5.5 Comparison to other detection algorithms

Following the methodology of the previous experiments, we run the Louvain and Infomap methods over the graphs of the different datasets. We then compare the results obtained by DDCA against those obtained by these two algorithms which are known to have quite good capabilities to detect communities [18].
Figure 5.11: NMI of DDCA, Louvain and Infomap on Dataset 1, separated by number of nodes and sizes of the communities.

The code for these two algorithms is freely available at Lancichinetti’s webpage\(^3\), and the parameters were left to the default values set by the software.

In Datasets 1 and 2 (flat structure), for the Louvain algorithm the bottom hierarchical level of the returned solution is taken in order to avoid, as far as possible, the resolution limit (as done before in [18], [26]).

On the other hand, for Datasets 3 and 4, the inferred hierarchical structure is mapped to the real one taking for each real level the inferred one that is closest to it. For these two datasets we use the hierarchical version of infomap [51] also available in the cited available software.

\(^3\)https://sites.google.com/site/andrealancichinetti/software
5.5. COMPARISON TO OTHER DETECTION ALGORITHMS

5.5.1 Results on Dataset 1

In Figure 5.11 we observe that the performance of DDCA is comparable to the one of Louvain and Infomap (actually almost the same as Louvain). Nevertheless, Figure 5.12, where we plot the computation time in seconds, shows that DDCA scales worse than the other two methods. Note that the differences in times could be explained by implementation-specific details, though the scaling capabilities are more likely to be explained by algorithm complexity.

For small graphs (1000 nodes), DDCA has low computation times (below a second) when the communities are not too fuzzy ($\mu < 0.6$), in some cases it even gets the faster results; however for graphs with very blended communities ($\mu \geq 0.6$) the times become too high in comparison to Louvain and Infomap (the main reason being that DDCA converges very slowly). When the number of
nodes is incremented to 5000, we observe a similar behaviour with respect to the
values of \( \mu \), however in these graphs DDCA has higher computation times than the
other two algorithms in all the cases, suggesting worse scaling in terms of graph
size. However, the benefits provided by the design of DDCA as a completely
decentralized algorithm makes it very easy to distribute and run in parallel, which
would greatly mitigate the scaling problems.

5.5.2 Results on Dataset 2

In this section we present a comparison between the benchmark algorithms
Louvain and Infomap against DDCA with RSPC initialization. The ARI and
conductance \( \phi \) of the detected partitions are shown in Figure 5.13. The results
found here are similar to those found for Dataset 1: DDCA and Louvain have
very close performance in terms of partition similarity measures and both of them
are able to perform (near) perfectly in the less unbalance graphs of the benchmark
in terms of ARI, whereas they struggle more than Infomap in \( G_1 \) and \( G_2 \). In \( G_3 \),
DDCA gets the best results, although none of the three algorithms is able to detect
accurately the group structure.

Interestingly, if we compare the performance in terms of ARI (partition
similarity measure) against the conductance (goodness measure), sometimes the
best performer in terms of one is the worst in terms of the other. This discrepancy
between the two types of measures, where an algorithm identifies communities
with good structural properties but does not yield good performance in terms of
partition similarity metrics, has already been noticed [26], [30].

5.5.3 Results on Dataset 3

This time we compare the results over the graphs of Dataset 3. DDCA is initialized
with RSPC and the three different diffusion functions, but we show only the best
result obtained by either of the functions. For Louvain and Infomap, we map the
inferred hierarchy into the real one, taking for each real level the inferred one that
gives the maximum ARI.

In Table 5.3 we find the comparison for the different levels of the graph \( H_1 \).
As it can be appreciated, DDCA gets the best results for the lower and middle
levels of the hierarchy, whereas Infomap is the best one finding the uppermost
ones where DDCA gets the poorer performance.

The results for \( H_2 \) are in line with the previous ones as shown in Table 5.4.

Finally, the results for \( H_3 \) (Table 5.5) confirm that DDCA is better recovering
the deepest levels but gives worst performance than the other 2 methods in the
highest levels.
5.5. Comparison to Other Detection Algorithms

![Graphs and Table]

(a) Adjusted Rand Index.
(b) Conductance.

Figure 5.13: NMI of DDCA, Louvain and Infomap on Dataset 2.

<table>
<thead>
<tr>
<th></th>
<th>DDCA</th>
<th>Louvain</th>
<th>Infomap</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottom</td>
<td><strong>0.955</strong></td>
<td>0.919</td>
<td>0.559</td>
</tr>
<tr>
<td>level₂</td>
<td><strong>0.887</strong></td>
<td>0.224</td>
<td>0.357</td>
</tr>
<tr>
<td>level₃</td>
<td><strong>0.746</strong></td>
<td>0.328</td>
<td>0.286</td>
</tr>
<tr>
<td>level₄</td>
<td>0.498</td>
<td>0.455</td>
<td><strong>0.829</strong></td>
</tr>
<tr>
<td>top</td>
<td>0.168</td>
<td>0.337</td>
<td><strong>0.439</strong></td>
</tr>
</tbody>
</table>

Table 5.3: ARI of DDCA, Louvain and Infomap on H1.

5.5.4 Results on Dataset 4

Here we compare the results of DCCA with RSPC initialization against the results of Louvain and Infomap in the graphs of Dataset 4. These graphs have a two-level hierarchy where we find micro-communities (lower level) and macro-communities (higher level).

The values that are shown for DDCA are the maximum ones obtained by either of the three diffusion functions. The results of Louvain and Infomap, which include an inferred hierarchy, are mapped to the real hierarchy taking the detected level that best matches the real one.

We find out that Louvain rarely detects the hierarchical structure, as it is only able to detect the two different levels for graph I₅. For the rest of the graphs, this method only detects one possible partition, meaning it is considering them having a flat hierarchy.
Chapter 5. DDCA Performance

### Table 5.4: ARI of DDCA, Louvain and Infomap on H2.

<table>
<thead>
<tr>
<th></th>
<th>DDCA</th>
<th>Louvain</th>
<th>Infomap</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottom</td>
<td>0.975</td>
<td>0.872</td>
<td>0.641</td>
</tr>
<tr>
<td>level2</td>
<td>0.940</td>
<td>0.674</td>
<td>0.243</td>
</tr>
<tr>
<td>level3</td>
<td>0.754</td>
<td>0.720</td>
<td>0.340</td>
</tr>
<tr>
<td>top</td>
<td>0.401</td>
<td>0.455</td>
<td>0.893</td>
</tr>
</tbody>
</table>

### Table 5.5: ARI of DDCA, Louvain and Infomap on H3.

<table>
<thead>
<tr>
<th></th>
<th>DDCA</th>
<th>Louvain</th>
<th>Infomap</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottom</td>
<td>0.855</td>
<td>0.731</td>
<td>0.796</td>
</tr>
<tr>
<td>level2</td>
<td>0.946</td>
<td>0.809</td>
<td>0.561</td>
</tr>
<tr>
<td>top</td>
<td>0.819</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

On the contrary, Infomap detects a two-levels hierarchy for the graphs with smaller mixing parameter ($I_1, I_3, I_5, I_7$) and a flat structure (no hierarchy) for the highly mixed communities ($I_2, I_4, I_6, I_7$).

In both cases, when the methods find flat hierarchies, the detected partitions correspond to the micro-communities while the macro-communities are not discovered.

### Table 5.6: ARI of DDCA, Louvain and Infomap on micro-communities of Dataset 4.

<table>
<thead>
<tr>
<th></th>
<th>DDCA</th>
<th>Louvain</th>
<th>Infomap</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>1.000</td>
<td>0.640</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_2$</td>
<td>1.000</td>
<td>0.979</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_3$</td>
<td>1.000</td>
<td>0.885</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_4$</td>
<td>1.000</td>
<td>0.896</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_5$</td>
<td>1.000</td>
<td>0.747</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_6$</td>
<td>1.000</td>
<td>0.958</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_7$</td>
<td>0.986</td>
<td>0.924</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_8$</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

In Table 5.6 we find the comparison in terms of the ARI performance metrics for the three algorithms in the micro-communities of the graphs. As shown, Infomap is able to fully recover all the micro-communities of the different graphs, and DDCA only fails in $I_7$ but with very good performance as well. The
5.5. Comparison to other detection algorithms

The performance of Louvain varies depending on the graph, but it gives worst results than Infomap and DDCA in all cases (except $I_8$ where all methods get the perfect solution).

Similarly, in Table 5.7 we find the comparison for the macro-communities. Now, the best results are obtained by DDCA, except in $I_1$ where Infomap gets the perfect partition. These results are a bit different than those obtained in Dataset 3 where the uppermost levels were worse identified by DDCA. This suggests that the deeper the hierarchy, the more difficult it is for DDCA to detect upper levels.

<table>
<thead>
<tr>
<th></th>
<th>DDCA</th>
<th>Louvain</th>
<th>Infomap</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>0.886</td>
<td>0.138</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_2$</td>
<td>0.177</td>
<td>0.122</td>
<td>0.124</td>
</tr>
<tr>
<td>$I_3$</td>
<td>0.586</td>
<td>0.051</td>
<td>0.051</td>
</tr>
<tr>
<td>$I_4$</td>
<td>0.408</td>
<td>0.357</td>
<td>0.300</td>
</tr>
<tr>
<td>$I_5$</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>$I_6$</td>
<td>0.263</td>
<td>0.148</td>
<td>0.136</td>
</tr>
<tr>
<td>$I_7$</td>
<td>1.000</td>
<td>0.521</td>
<td>0.420</td>
</tr>
<tr>
<td>$I_8$</td>
<td>0.270</td>
<td>0.259</td>
<td>0.259</td>
</tr>
</tbody>
</table>

Table 5.7: ARI of DDCA, Louvain and Infomap on macro-communities of Dataset 4.

Nevertheless, none of the algorithms presents a remarkably good performance in the macro-community level, suggesting that these are indeed harder to detect than the micro-communities.
Chapter 6

Conclusions

This chapter explains the conclusions obtained throughout the design, development, and evaluation of DDCA and proposes a number of improvements, extensions, or complements that may be of interest in order to continue the work.

6.1 Conclusions

The main objective was to design, implement and evaluate a decentralized algorithm for community detection that would be able to work on unbalanced graphs and to detect clusters at different resolutions. With that purpose we designed DDCA, which is completely decentralized and where nodes perform a diffusion process based solely on the information of their neighbours. The algorithm includes as well a mechanism to control the diffusion process which can be thought of as a resolution parameter.

To evaluate the implemented algorithm, we designed a series of computer-generated graphs which present different properties like balanced vs unbalanced communities, highly interconnected vs highly mixed ones and flat vs hierarchical structures.

Firstly, we studied the behaviour of the different initialization strategies, not finding any big differences between them and concluding that the method is robust to the selected seeds in terms of performance recovering the true communities. However, we found that the convergence times are highly dependent on how the selection of the seeds is done.

We also tested DDCA on balanced and unbalanced graphs, finding that it was able to recover the ground truth communities in almost all the cases, except for the extremely unbalanced graphs. We compare the results to those obtained by Louvain and Infomap (two well-known existing methods) finding that DDCA performs equally or better on not-extremely unbalanced graphs, although it shows
worst scaling capabilities. The Infomap method proofs to be the best in the extremely unbalanced generated graphs.

Finally, we evaluated the impact of the diffusion strategy on the results of DDCA. We showed how RSPC initialization (one random seed per ground truth cluster) is able to recover the communities at different resolutions of the hierarchy. We observe that the proposed diffusion functions behave differently and control the number and size of the detected communities. The slow diffusion proves itself the best option for detecting bottom levels of hierarchical structures, specially when the sizes of these bottom communities is very small with respect to the size of the graph. On the other hand, the fast strategy does not show to be specially suited for detecting top levels of the hierarchy, maybe because these are harder to discover than the lower ones. Overall, the linear proofs itself to be the best-working mode of the proposed diffusion methods, with the exception of the aforementioned case of bottom levels. We conclude that DDCA does a good job identifying the true communities in the bottom levels of the hierarchy, and starts downgrading as we move to higher levels.

The results of DDCA on hierarchical community structures are also compared to those obtained by Louvain and Infomap. We find that DDCA is better for deeper levels in the presence of balanced hierarchical structures, whilst the other two give better results for the uppermost levels. In the case of unbalanced hierarchical structures, DDCA obtains the best results overall in graphs with a two-level hierarchy. Nonetheless, it shall be noted that unlike done by Louvain and Infomap, the hierarchy structure of the networks is not inferred by DDCA, as it is only able to look at different resolutions.

## 6.2 Discussion

Although we were able to get a working version of the algorithm, it was a challenging job that required a lot of thinking and discussion. Even though we managed to design an algorithm that meets our performance expectations, the complexity of the algorithm limited a lot the ways to evaluate it, and is still an aspect where DDCA needs improvement. On the other hand, we observed the effect of the diffusion in line with our hypothesis that it can be used to find communities at different resolutions.

These limitations suggest that existing methods might be a better approach for detecting communities in the type of graphs evaluated in this thesis, as they can achieve similar results faster. The main benefits of DDCA reside on its decentralization so it must be tested on a distributed graph processing scenario to fully decide its benefits from existing methods.
6.3 Future work

As an important part of the future job remains the evaluation of DDCA in weighted and directed graphs to test whether its behaviour and performance extends to those types of networks. Likewise, extending it to allow the detection of overlapping communities is also a good next direction, as this reformulation of the community detection problem has gained a lot of importance in the recent years. Due to the design of the algorithm, the adaptation for this case should be fairly easy, as a node keeps track of several colors and one could just look those with greater amount (or a similar strategy) to get more than one membership for each node.

Another aspect of the algorithm that is worth investing more research time is a more clever definition of the diffusion strategy and its impact. We could, for example, construct a piecewise-defined function that takes a coefficient $a$ (as those presented in this work) that controls the diffusion effect, so that $a < 0$ gives slow diffusion, $a = 0$ gives linear, and $a > 0$ gives fast. This way it would be much easier to study the impact, as we could test DDCA at different levels of $a$. In a similar way, research could also be done by defining new diffusion functions, for example an s-shaped curve (resembling the softmax shape). It is as well a nice feature that should be included to make the algorithm infer the hierarchical community structure (as Louvain does) as an extension of the resolution control.

Initialization methods should also be further investigated, in order to try to find one that behaves as well as RSPC in convergence times but where there is no prior information needed to be inputted to the algorithm.

Another important improvement regarding the algorithm design that should be accomplished is to reduce the complexity of the algorithm to make it more suited for real-world networks where the number of nodes and edges is immense.

In a less scientific aspect and more related to engineering, an implementation and evaluation of DDCA on a distributed graph processing tool with real world graphs would give a much more realistic vision of its benefits and drawbacks.
Bibliography


