UCLouvain

LOUVAIN
School of Management

Louvain School of Management

Graph node clustering base on attraction/repulsion matrices.

An experimental comparison.

Author: Aghemio Pier-Gustavo
Supervisors: Saerens Marco and Courtain Sylvain
Academic year 2019-2020

Acknowledgements

I would like to thanks my two supervisors Pr. *Marco Saerens* and Mr. *Sylvain Courtain*, for their help and patience throughout the realization of this thesis. I also want to thanks my uncle and brother for welcoming me while I wrote this thesis.

Contents

Chapter 1

Introduction

A *network* is a collection of interconnected objects. Many things in the real or abstract world can be conceptualized as networks. It can be a road system, social interaction between members of a species, computers linked together, etc. Networks can be found in every field [Newman, 2010].

There are many aspects to study in networks. Let's illustrate with an ant colony. The objects composing the network are the ants, and two ants are connected if they interacted together. When studying an ant colony, we can focus on the nature of the individual components, *e.g.* study an ant's anatomy. We could also study the nature of the connection, *e.g.* "how ants interact?", "Why did they interact?". In this thesis, we are interested in the third aspect of networks; We focus on the *pattern* of connections. The pattern of connections can tell us a lot about a network; it has a big effect on the system's behavior. For example, the pattern of connections will determine how quickly information spread across a network. In the ant colony case, it will determine how fast the colony will react, with a sufficient number of ants, to an invader [Newman, 2010].

In this thesis, we focus on community detection on graph[1], *i.e.* we assign the objects constituting a network into different groups [Newman, 2010]. The investigation of community structure in networks has received a great deal of attention recently. Although, the concept of community is *ill-defined* [Chakraborty et al., 2017]. This is partly since there are different sorts of communities. Due to the heterogeneity in communities, there is a huge variety of algorithms specialized in detecting one community type. They are using different metrics that underline better the type of cluster they seek cluster. [Chakraborty et al., 2017].

[1]Graph refers to the mathematical representation of a network.

To detect those communities and ultimately create our groups, clusters, we are using **attraction/repulsion** matrices. Each element of an attraction/repulsion matrix quantifies two objects' likelihood to belong in the same group. When there is an attraction between an object i and an object j, the element q_{ij} of the attraction/repulsion matrix, $\mathbf{Q}$, will be positive; And if there is a repulsion between an object i and an object j, the element q_{ij} will be negative [Bagon and Galun, 2011, Courtain et al., 2020][2]. Attraction/repulsion matrices can be seen as maps that algorithms use to find clusters on networks. Our aim here is to find the attraction/repulsion matrix that will yield the best result when plugging into two specific algorithms.

The attraction/repulsion matrices we tested are based upon different similarity and dissimilarity matrices. Additionally, we applied various **transformations** to those matrices, such as centering, to obtain more attraction/repulsion matrices.

We plug our various attraction/repulsion matrices into two different clustering algorithms, the well-known *kernel k-means* and the *attraction repulsion soft clustering* developed in [Courtain, 2017, Courtain et al., 2020]. The kernel k-mean algorithm is a well-established algorithm; it is made to run with a kernel. A kernel[3] is a different kind of matrix that guides the clustering algorithm. The attraction-repulsion soft clustering (ARSC) is a new type of algorithm inspired by [Lehmann and Hansen, 2007, Rossi and Villa-Vialaneix, 2010] and was developed to use an attraction/repulsion matrix to perform its cluster partitioning. The advantages of the ARSC are that it scales easily on large and/or sparse graphs, it is short and easy to implement, it gives a fuzzy membership, and it is competitive with state-of-the-art algorithms [Courtain et al., 2020].

The kernel k-means (KKM) algorithm is a nonoverlapping[4] community clustering algorithm, whereas the ARSC, as mentioned above, can assign an object to different groups; it is a fuzzy-algorithm[5] [Courtain, 2017, Courtain et al., 2020, Fouss et al., 2016]. Compared to other well-established algorithms, the competitiveness of the ARSC is already proven in [Courtain, 2017, Courtain et al., 2020].

[2]As of the date of publication of this thesis the paper [Courtain et al., 2020] has not been published yet. This thesis is working on the same algorithm as presented in Courtain *and al.*

[3]See chapter 6

[4]This is one of the various sorts of communities mentioned earlier. See chapter 7

[5]See chapter 7

As already stated, this thesis aims to plug new attraction/repulsion metrics in the ARSC algorithm to make it more effective. To attain this goal, this thesis will answer the three following questions:

- Which attraction/repulsion matrix achieves the most competitive results with the *attraction repulsion soft clustering* algorithm?
- Which attraction/repulsion matrix achieves the most competitive results with the *kernel k-means* algorithm?
- Which of the two algorithms, ARSC and KKM, is the most competitive when plugged with attraction/repulsion matrices? And are they competing against well-established algorithms?

To answer those questions, we will run a series of clustering experiments with the ARSC, and the KKM injected with divers attraction/repulsion matrices. The algorithms plugged with different attraction/repulsion matrices will run onto various datasets. For each of those datasets, we not only know the number of clusters they contain, but we also know each object true class, *i.e.* the cluster for which an object belongs. Knowing this, we can use performance metrics to assess the efficiency of each algorithm against one another.

We start with an overview of possible applications in management (see chapter 2). Thereafter, we will introduce the basic concepts of network and the different notations (see chapter 3). In the next three chapters, we will then expose the different similarities, dissimilarities (see chapter 4 5), and the kernels (see chapter 6) measures we will plug into the ARSC and the KKM algorithms. We will then introduce the two algorithms, the KKM and the ARSC (see chapter 6 8). After this, we will explain how we will evaluate their performances (see chapter 9). We then detail the clustering experiments (see chapter 10) and analyze the results (see chapter 11) before concluding.

Chapter 2

Application in management

As we have seen in the introduction, networks can be found everywhere in various forms, and they have numerous aspects of studying. In our case, we focus on the *pattern* of connections between the components and networks' structure. This mathematical approach has many real-world applications that provide a great deal of valuable information and save a considerable amount of time [Newman, 2010]. We will take a peek at the possibilities the study of networks offers to the worlds of Marketing, Finance, and the study of innovation clusters.

2.1 Marketing

A perfectly tailor-made marketing strategy for every consumer is financially and logistically impossible. Graph partitioning comes in very handy, as it allows marketers to group consumers sharing similar consumption patterns [Fernandes et al., 2019, Jones et al., 2013]. Those clusters of customers are equivalent to market segments [Berry and Linoff, 2004]. Telecommunication companies have widely used market segmentation via clustering to segment their customers [Berry and Linoff, 2004]. They can develop a marketing strategy tailor for those groups, thus developing closer relations with their customers. They can use this market segmentation to optimize the design of their marketing strategy [Huang et al., 2007].

Furthermore, they can find the customer who has the greatest network value, the most influential customer in a cluster for each of those groups. Now more than ever, potential clients are seeking the "expertise" of existing clients. The "expert" can be a random stranger over the internet [Jones et al., 2013]. Picking a restaurant bases upon the comments on Google is a perfect example of this consumer behavior.

In [Fernandes et al., 2019], they demonstrate the technique of community detection as a marketing tool described in the paragraph above. They detected consumer clusters, based upon their consumption patterns, in seven different markets. For each cluster, they found the customers with the most connections within their cluster, the customer who has the biggest influence. Those influencers are the most valuable as they can send positive feedback waves into the network if they are satisfied with their company. By taking advantage of those influential customers, companies can better connect with their diverse communities.

The rise of the internet and social networks such as Facebook has given marketer access to a vast network of potential customers; it gives them access to huge advertising space. New marketing strategies have been developed to get the most out of social platforms; one is **viral marketing**. Viral marketing relies on the people composing social networks to promote the company instead of promoting itself. It is a snowball effect, and the potential reach is enormous. However, it is not so simple to kick-start. There is no magic receipt; a marketing campaign base upon this technique can be a total flop. Hopefully, some tricks increase the chance of success for marketers. One of them is to target those users qualified as a trust influence in their cluster of friends, families, etc. Once marketers have identified those "influencers", they target them with the content they want them to share. Here, cluster analysis on the graph comes into plays as it can identify the different clusters and the influential nodes in a social network such as Facebook [Huang et al., 2019].

Clustering in marketing can also be useful for product recommendation. Looking at a similar customer's purchasing pattern, we can refine product recommendations, increasing the customer basket [Berry and Linoff, 2004]. After this small peek into the possibilities in the world of marketing, let us now look at the possible applications in finance.

2.2 Finance

In the financial world, clustering analysis can have some good uses; let's review some of them. The performance of a company is characterized by one single digit, the stock price. The value of this number is based upon available information that is constantly debated. Companies are always interacting with one another, but we do not always know-how. We know that when stock prices of one company change, other companies' stock prices are going to be impacted too. There is some degree of correlation between stock prices. We can use this correlation to build an **asset** graph, where the stock represents individuals, and the links are given by the degree

of correlation. Using those asset graphs, we can build a taxonomy of the financial market. The network analysis allows us to depict a clearer picture of the financial market with publicly available information [Onnela et al., 2004, Onnela et al., 2003]. It allows us to cluster together stocks without requiring any information about the stock themselves; hours of extensive research can be skipped.

Yet, stocks are already classified into big categories known as an *industry*. It is possible to recreate those big categories using time series clustering. In doing so, the interest is once we have recreated our clusters, we can find which stock has the biggest influence on the rest of the industry [Wittman, 2002].

Extending from those industry clusters, we can see which industries have a high interdependence. Knowing concurrent fluctuation is not that useful, but we can use this interdependence to build predictive models across industries [Wittman, 2002].

Last but not least, it can be used for anomalies detection. An anomaly is a pattern in the data that do not confront the expected behavior of the data. Anomaly detection is a wide field with many applications ranging from intrusion detection to fraud detection. By finding irregularities in the financial network, we can prevent organizations from making an illegal profit from *insider trading*. Insider trading is when people make an illegal profit by taking advantage of information before they are publicly available [Chandola et al., 2009]. In [Courtain et al., 2019], they did just that by using a distance measure[1] called the *free energy*[2].

2.3 Innovation clusters

Innovation clusters have captured a great deal of attention from academics to policymakers. Indeed, innovation clusters promise high innovations and better competitiveness, thus improving a regions' economy [Simmie, 2004, Hamdouch, 2007]. Yet, the definitions of an innovation cluster are still unclear; there are almost as many definitions as there are authors. In [Hamdouch, 2007], A. Hamdouch makes an account of all the leading definitions of innovation clusters. After synthesizing the various definitions, he states that innovation clusters are an ensemble of organizations and institutions that are defined, *i.* through geographical features that can greatly vary, *ii.* interact formally and/or informally and *iii.* contribute collectively to the achievement of all kind of innovations within a given industry or domain of

[1] Distance measures are introduce in chapter 5.

[2] It should be noted that for this example, there is no clustering involved. However, it is relevant in the sense that this type of fraud uses graph theory.

activity.

Innovation clusters are a very complex multidisciplinary topic that still needs researching. A key aspect of innovation clusters is the actors' ability to engage in valuable relationships. How do the different actors connect, and how strong are those connections? This influences the shape of the network, *i.e.* the way the different actors connect. There are numerous ways a network can be interconnected if the interconnections' distribution is completely random, a network if-then qualify a random network. Random networks should correspond to the image of a purely competitive market [Hamdouch, 2007].

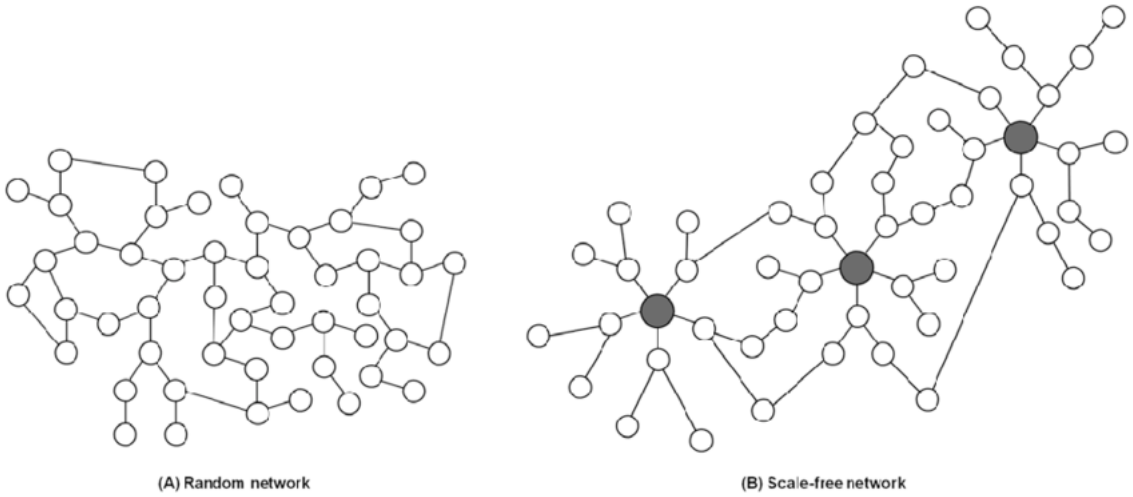


Figure 2.1: A random network and a scale free network [Seo et al., 2013].

Two types of networks have been associated with the possible configuration of innovation clusters, small-world networks and scale-free networks. They are essentially composed of small pockets of highly interconnection elements and some connection between elements that are far apart. Moreover, scale-free networks are dynamic because elements can enter and exit the network over time. Scale-free network and small-world networks are basically the same, except that small-world networks are not expected to change over time [Hamdouch, 2007]. Figure 2.1 illustrate the difference between a random network and a scale-free network.

L. Flemming *et al.* worked on a co-authorship patent to investigate the effects of collaboration networks on innovation in [Fleming et al., 2007]. They failed to find evidence that small-world networks enhance innovation. Nevertheless, they make several theoretical, methodological, and empirical contributions to understanding small-world networks and innovation. Studies aiming to identify the network structures that boost innovations can help decision-makers take the right action to build up innovation. The secret to successfully emulate innovation clusters lies definitely

not only in their network structure. Yet, it might give some insight into how to, to policymakers.

In the next chapter, we will introduce basic notions about networks. This will be the beginning of our journey around the theoretical framework of this thesis.

Chapter 3

Fundamentals of network

This chapter will introduce the essential foundation for the study of networks. The tools and notations laid down in this section will be key in understanding the following theoretical developments. Most of the theory developed in this chapter will come from graph theory. Graph theory is the branch of mathematics that studies networks [Newman, 2010].

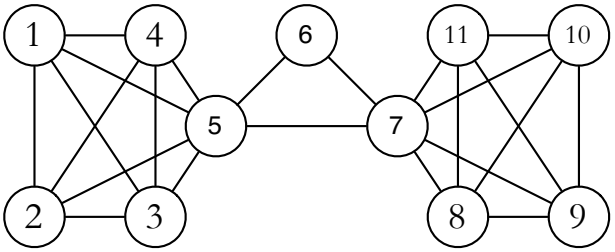


Figure 3.1: A small network composed of 11 nodes and 23 edges, $G = (11, 23)$.

3.1 Basic notions

A *network* also commonly known as a *graph* is, in mathematics, a finite no-null set N of n nodes, also called vertices and a set E of m edges, also called links that pairs up nodes [Newman, 2010, Fouss et al., 2016]. Throughout this thesis, n will denote the number of nodes, vertices, of a network, and m its number of edges, links. A graph with a collection of N node and E edge will be denoted as $G = (N, E)$. Figures 3.1 (taken from [Mantrach et al., 2010]) is a graphical representation of a network.

A network can be a road network with its edges as the streets and the nodes its intersections. Another example of a network is the well-known Facebook with its friendship network; in this instance, a friendship link is an edge and an individual a

vertex [Fouss et al., 2016, Huang et al., 2019, Newman, 2010].

In this thesis, all the graphs we will work with will have one edge between each node. Networks with more than one edge between two vertices are called a *multiedge* graph. Moreover, this thesis's studied graph will not contain any *self-edge*, also known as *self-loop*, *i.e.* an edge starting and finishing at the same node. The networks studied in this thesis are commonly referred to as *simple graphs* as they do not contain any *self-edge* and *multi-edge* [Newman, 2010, Fouss et al., 2016].

A network can be directed, meaning that its edges have a direction. The edges flow from a node, i, to another node, j, for all $(i, j) \in E$. Or a network can be undirected, meaning that an edge can be taken in both directions; the edge between (i, j) is the same as (j, i) [Fouss et al., 2016, Newman, 2010]. In the frame of this thesis, all graphs are undirected.

A *weight* w_{ij} can be attributed to the edges of a graph G. The weight can represent the degree of affinity or similarity between two vertices. A weight can either be positive or negative. There are no specific rules on weight [Fouss et al., 2016, Newman, 2010]. In this thesis, we will be working with weighted and unweighted graphs. To sum up, there are two different types of graphs we will be dealing with: unweighted undirected simple graph and weighted undirected simple graph.

3.2 Adjacency matrix

There are numerous ways in which a network can be represented mathematically. One of them is with the *adjacency* matrix. The adjacency matrix $\mathbf{A}$ with a size $n \times n$ of a simple graph with element a_{ij} is such that [Fouss et al., 2016, Newman, 2010],

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge between nodes } i \text{ and } j, \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The *adjacency* matrix of the network in figure 3.1 is represented in figure 3.2. There are two noticeable features on the adjacency matrix in figure 3.2, the diagonal elements of the matrix are all equal to 0, and the matrix is symmetric. This is because the represented network is a *simple undirected* graph.

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array}\right] \end{array} \tag{3.2}$$

In the instance of a *weighted* graph, we can redefine the element of $\mathbf{A}$ as,

$$a_{ij} = \begin{cases} w_{ij} & \text{if there is an edge between nodes } i \text{ and } j, \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

3.3 Volume & degree

The *volume* of a graph is the sum of all the elements of a graph's adjacency matrix. It is defined as follow,

$$vol(G) = \sum_{i,j=1}^{n} a_{ij} = a_{\bullet\bullet} = 2m \tag{3.4}$$

The **degree** d_i of a nodes i correspond to the number of edges that are connected to it [Fouss et al., 2016, Newman, 2010]. In the instance of an undirected graph, it is defined as

$$d_i = \sum_{j=1}^{n} a_{ij} = a_{i\bullet} \tag{3.5}$$

Another important concept is the degree vector $\mathbf{d}$, of size $n \times 1$, defined as

$$\mathbf{d} = \mathbf{A}\mathbf{e} \tag{3.6}$$

with $\mathbf{e}$ as the unit vector of size $n \times 1$.

The **degree** vector contains the degree of every vertex. We can also define a matrix **D** containing all of the degrees on its diagonal, the rest being zeros.

3.4 Laplacian matrix

The *Laplacian* matrix is closely related to the adjacency matrix; its properties can tell lots of useful information about a network structure. The Laplacian matrix comes from the *diffusion* process, which is a process of spread across a network [Newman, 2010]. The Laplacian matrix has a size $n \times n$ and is defined as,

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{3.7}$$

The Laplacian matrix from the adjacency matrix 3.2 is the following,

$$\mathbf{L} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 4 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & 6 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 6 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 4 \end{array}\right] \end{array} \tag{3.8}$$

For a weighted network, the elements of the graph Laplacian are defined as

$$l_{ij} = \begin{cases} d_{ij} & \text{if } i = j, \\ -w_{ij} & \text{if } i \neq j \text{ and there is an edge between nodes } i \text{ and } j, \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

The Laplacian matrix cannot have any negative eigenvalues. however, they can be equal to zero, and there is always at least one eigenvalue equal to zero. The Laplacian is, therefore semi-definite. It is also doubly centered, *e.i.* the sum of the elements for each column or row is equal to 0,

$$\mathbf{eL} = 0 \quad \text{and} \quad \mathbf{e}^{\mathrm{T}}\mathbf{L} = 0^{\mathrm{T}} \tag{3.10}$$

Now that the basic notions of graph theory have been laid out, we will move to the

next chapter. We will introduce our various attraction/repulsion matrices. We will see that the adjacency matrix is the basic bloc to construct our similarity matrices hence our attraction/repulsion matrices. In contrast, the Laplacian matrix is more the basic bloc to build dissimilarity matrices.

Chapter 4

Attraction repulsion affinity matrices

The first aim of this thesis is to seek alternative attraction/repulsion matrices to plug into the ARSC[1] algorithm to yield better results. The secondary aim is to plug the same attraction/repulsion matrices into the KKM[2] algorithm to see if it yields good performances. Those matrices act like maps that tell the algorithms how to form the clusters. The interest of clustering with attraction/repulsion matrices is that many data frames are attraction/repulsion matrices. A correlation matrix is an attraction/repulsion matrix.

In this chapter, we first introduce the similarities used to create the attraction/repulsion matrices. Thereafter, we will explain the transformations that we operate over those matrices to obtain more attraction/repulsion matrices.

4.1 Similarity measures

As mentioned above, we will initialize two different algorithms with various attraction/repulsion matrices. All of those matrices will be based on some **similarity** measures. Let us begin with the concept of similarity.

A similarity measure is a way to quantify the similarity between a vertex i and j [Newman, 2010, Lü and Zhou, 2011]. To quantify the similarity between two nodes, one could look at the attributes of the two nodes. However, here in Graph theory, we only work with the structure of the data, so we want to quantify the similarity between two nodes by looking at the connection patterns. Are the two nodes "structurally" close in the network [Fouss et al., 2016]?

[1] Attraction Repulsion Soft Clustering
[2] Kernel k-means

There are three intuitions behind the concept of similarity [Fouss et al., 2016]:

- The similarity between two objects is related to their commonality. The more commonality they share, the more similar they are.
- Symmetrically, the similarity between two objects is related to the differences between them. The more differences they have, the less similar they are.
- The maximum similarity between two objects is reached when the two objects are identical, no matter how much commonality they share.

Not all similarity measures satisfy those three concepts.

Nodes can be considered similar if they have certain proximity in the network or are highly connected by an indirect path. They are also considered similar if they share some common (sub)structure, if they have a similar role in the network, and if they influence the network in the same way [Fouss et al., 2016].

The type of similarity measures selected is considered *local similarity* measures. A local similarity only looks at the neighborhoods of two nodes [Fouss et al., 2016].

4.1.1 Direct similarity

This similarity simply takes as a similarity score, the affinity a_{ij} of the **adjacency** matrix, for the nodes i and j [Fouss et al., 2016]. It simply is the adjacency matrix.

$$[\mathbf{S}_d]_{ij} \triangleq a_{ij} \tag{4.1}$$

The direct similarity is non-negative, as is the adjacency matrix, and it is symmetrical as we only work with an undirected graph. As it is non-negative, the adjacency matrix is only an attraction matrix, not an attraction/repulsion matrix, as it only underlines the attraction between nodes. In fact, all similarities are attraction matrices. They will only be attraction/repulsion matrices after we apply a transformation.

The direct similarity has a fatal flaw; it does not consider the nodes neighborhoods; it does not underline structural equivalences [Fouss et al., 2016]. The similarity introduces from now on will overcome this flaw by taking into account the node's neighborhoods.

Before moving on to the next similarity, let us first look again at the adjacency matrix 3.2, *i.e.* the direct similarity matrix, of the network in figure 3.1. Looking

at the adjacency matrix, we could intuitively define 3 different clusters: $\{1, 2, 3, 4\}$, $\{5, 6, 7\}$ and $\{8, 9, 10, 11\}$ or $\{1, 2, 3, 4, 5\}$, $\{6\}$ and $\{7, 8, 9, 10, 11\}$.

The similarity between nodes 5 and 6, $[\mathbf{S}_d]_{5,6}$, is similar to the similarity between nodes 4 and 5, $[\mathbf{S}_d]_{4,5}$. Looking at the network in figure 4.1 we can see that nodes 1 to 5 form a stronger community clique than 5 and 6. Shouldn't the similarity between node 4 and 5 be greater than the similarity between 5 and 6, *i.e.* $[\mathbf{S}_d]_{4,5} > [\mathbf{S}_d]_{5,6}$? As this is not the case, we might be fooled into forming a group $\{5, 6, 7\}$. Figure 4.1 shows us how the graph should be partitioned.

Moreover, let us imagine that there is no edge between nodes 4 and 5. In that case the direct similarity would be equal to 0, $[\mathbf{S}_d]_{4,5} = 0 = a_{4,5}$. Even though, without an edge between nodes 4 and 5, the community formed by nodes 1 to 5 would still be strong. Shouldn't there be at least some degree of similarity between vertices 4 and 5, *i.e.* $[\mathbf{S}_d]_{4,5} > 0$?

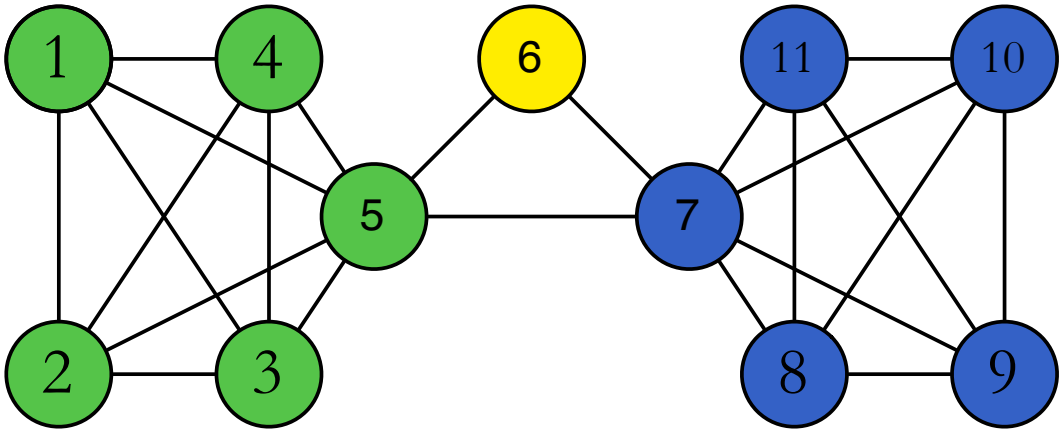


Figure 4.1: Network 3.1 with its cluster highlight.

4.1.2 Common neighbors

To underline the structural equivalence, the shared neighborhood, between two nodes we can simply count the number of common neighbors between two vertices. For an *undirect* network, the number n_{ij} of common neighbors between i and j is given by,

$$[\mathbf{S}_{cn}]_{ij} = \sum_{k}^{n} a_{ik} a_{kj} \tag{4.2}$$

Which corresponds to the ijth element of the squared adjacency matrix $\mathbf{A}^2$ [Fouss

et al., 2016][Newman, 2010]. This similarity measure gives us a positive and semi-define matrix. The whole common neighbors matrix can be computed as follow,

$$\mathbf{S}_{cn} = \mathbf{A}\mathbf{A}^{\mathbf{T}} = \mathbf{A}^2 \tag{4.3}$$

As we only work with undirected graph, the adjacency matrix is symmetrical, thus $\mathbf{A}\mathbf{A}^{\mathbf{T}} = \mathbf{A}^2$.

$$\mathbf{S}_{cn} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 4 & 3 & 3 & 3 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 4 & 3 & 3 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 3 & 4 & 3 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 3 & 3 & 4 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 6 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 6 & 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & 4 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & 3 & 4 & 3 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & 3 & 3 & 4 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & 3 & 3 & 3 & 4 \end{array}\right] \end{array} \tag{4.4}$$

The matrix 4.4 is the common neighbors similarity matrix of figure 3.1. Here, we see that the similarity score is the number of neighbors that two vertices share. Just by looking at this matrix we can easily spot 3 different clusters, $\{1, 2, 3, 4, 5\}$, $\{6\}$ and $\{7, 8, 9, 10, 11\}$.

How can we interpret the resulting similarities? How does $[\mathbf{S}_{cn}]_{4,5} = 3$ score in comparison with $[\mathbf{S}_{cn}]_{5,6} = 1$? We know that a higher number means a higher affinity. However, we can hardly say how similar two nodes are with the common neighbors measure [Newman, 2010]. This is why most of the following measures have some normalization that places the similarity on a 0 to 1 scale.

4.1.3 Cosine similarity

Introduce by Salton in [Salton, 1989], the *Cosine* similarity regards the ith and jth rows (or columns) of the adjacency matrix as vector and compute the cosine angle as a similarity measure [Newman, 2010]. The cosine similarity between nodes i and j is,

$$[\mathbf{S}_{cos}]_{ij} = \frac{\sum_k^n a_{ik}a_{kj}}{\sqrt{\sum_k^n a_{ik}^2}\sqrt{\sum_k^n a_{kj}^2}} \tag{4.5}$$

The above metric provides a similarity scale between 0 and 1. A similarity of 0 means that the two vertices have no neighbor in common, and 1 indicates that they share the exact same neighbors [Newman, 2010]. We can easily compute the cosine similarity matrix with the following equation,

$$\mathbf{S}_{cos} = \frac{\mathbf{A}^2}{\mathbf{d}_{cos}\mathbf{d}_{cos}^{\mathbf{T}}} \tag{4.6}$$

With $\mathbf{d}_{cos} = \sqrt{\mathbf{A}^2\mathbf{e}}$.

$$\mathbf{S}_{cos} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 0.75 & 0.75 & 0.75 & 0.61 & 0.35 & 0.20 & 0 & 0 & 0 & 0 \\ 0.75 & 1 & 0.75 & 0.75 & 0.61 & 0.35 & 0.20 & 0 & 0 & 0 & 0 \\ 0.75 & 0.75 & 1 & 0.75 & 0.61 & 0.35 & 0.20 & 0 & 0 & 0 & 0 \\ 0.75 & 0.75 & 0.75 & 1 & 0.61 & 0.35 & 0.20 & 0 & 0 & 0 & 0 \\ 0.61 & 0.61 & 0.61 & 0.61 & 1 & 0.29 & 0.17 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.35 & 0.35 & 0.35 & 0.35 & 0.29 & 1 & 0.29 & 0.35 & 0.35 & 0.35 & 0.35 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.17 & 0.29 & 1 & 0.61 & 0.61 & 0.61 & 0.61 \\ 0 & 0 & 0 & 0 & 0.20 & 0.35 & 0.61 & 1 & 0.75 & 0.75 & 0.75 \\ 0 & 0 & 0 & 0 & 0.20 & 0.35 & 0.61 & 0.75 & 1 & 0.75 & 0.75 \\ 0 & 0 & 0 & 0 & 0.20 & 0.35 & 0.61 & 0.75 & 0.75 & 1 & 0.75 \\ 0 & 0 & 0 & 0 & 0.20 & 0.35 & 0.61 & 0.75 & 0.75 & 0.75 & 1 \end{array}\right] \end{array} \tag{4.7}$$

In cosine matrix 4.7 we see that all elements are in a ratio between 0 and 1. With this, it is easy to determine which vertex are strongly similar and which are not. We see a stronger similarity between nodes 5 and 4, $[\mathbf{S}_{cos}]_{4,5} = 0.61$ than between nodes 5 and 6, $[\mathbf{S}_{cos}]_{5,6} = 0.29$. It is now easier to spot the right clusters, $\{1, 2, 3, 4, 5\}$, $\{6\}$ and $\{7, 8, 9, 10, 11\}$.

4.1.4 Adamic-Adar similarity

The *Adamic-Adar* similarity, introduced in [Adamic and Adar, 2003], is a refined version of the common neighbors. It gives more weight to low degree neighbors, *i.e.* it penalizes high degree neighbors. The Adamic-Adar similarity states that two vertices are more similar if they share a node with few neighbors in their common neighborhood. If something unusual is share between two nodes, the nodes are more likely to be connected. Friends tend to be similar [Feld, 1981]. Mathematically this is interpreted as the following [Adamic and Adar, 2003],

$$[\mathbf{S}_{aa}]_{ij} = \sum_{z \in \Gamma(i) \cap \Gamma(j)}^{n} \frac{1}{\log(d_z)} \tag{4.8}$$

With $\Gamma(i)$ is the neighborhood of i, thus $\Gamma(i) \cap \Gamma(j)$ being the shared neighborhood of i and j. So, z can only be a neighboring node share by i and j. d_z is the degree of

the node z. The Adamic-Adar similarity sums the logarithm degree of two nodes, i and j, shared neighbors.

$$\mathbf{S}_{aa} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 2.43 & 1.81 & 1.81 & 1.81 & 1.87 & 0.56 & 0.56 & 0 & 0 & 0 & 0 \\ 1.81 & 2.43 & 1.81 & 1.81 & 1.87 & 0.56 & 0.56 & 0 & 0 & 0 & 0 \\ 1.81 & 1.81 & 2.43 & 1.81 & 1.87 & 0.56 & 0.56 & 0 & 0 & 0 & 0 \\ 1.81 & 1.81 & 1.81 & 2.43 & 1.87 & 0.56 & 0.56 & 0 & 0 & 0 & 0 \\ 1.87 & 1.87 & 1.87 & 1.87 & 3.83 & 0.56 & 0.77 & 0.56 & 0.56 & 0.56 & 0.56 \\ 0.56 & 0.56 & 0.56 & 0.56 & 0.56 & 1.12 & 0.56 & 0.56 & 0.56 & 0.56 & 0.56 \\ 0.56 & 0.56 & 0.56 & 0.56 & 0.77 & 0.56 & 3.83 & 1.87 & 1.87 & 1.87 & 1.87 \\ 0 & 0 & 0 & 0 & 0.56 & 0.56 & 1.87 & 2.43 & 1.81 & 1.81 & 1.81 \\ 0 & 0 & 0 & 0 & 0.56 & 0.56 & 1.87 & 1.81 & 2.43 & 1.81 & 1.81 \\ 0 & 0 & 0 & 0 & 0.56 & 0.56 & 1.87 & 1.81 & 1.81 & 2.43 & 1.81 \\ 0 & 0 & 0 & 0 & 0.56 & 0.56 & 1.87 & 1.81 & 1.81 & 1.81 & 2.43 \end{array}\right] \end{array} \tag{4.9}$$

Unfortunately, like the common neighbors, this similarity does not give back a ratio, so it is hard to interpret the values. In the Adamic-Adar similarity matrix 4.9, we can see that $[\mathbf{S}_{aa}]_{5,7} = 0.77$, which is higher than $[\mathbf{S}_{aa}]_{5,8} = 0.56$. Vertices 5 and 7 only shares one neighbor, so do vertices 5 and 8. However, the shared neighbor is different, 5 and 7 share node 6. Whereas 5 and 8 share node 7. As node 6 has a lower vertex degree than vertex 7, this makes the Adamic-Adar similarity score $[\mathbf{S}_{aa}]_{5,7}$ greater than $[\mathbf{S}_{aa}]_{5,8}$, *i.e.* $[\mathbf{S}_{aa}]_{5,7} = 0.77 > [\mathbf{S}_{aa}]_{5,8} = 0.56$. Sharing node 6 as a neighbor makes vertices 5 and 7 more similar. Looking at figure 3.1 this is an obvious fact. Looking at matrix 4.9, we would assign once again the node to the clusters, $\{1, 2, 3, 4, 5\}$, $\{6\}$ and $\{7, 8, 9, 10, 11\}$.

4.1.5 Jaccard similarity

Jaccard similarity was introduced over a hundred years ago by Jaccard in [Jaccard, 1902]. It normalizes the common neighbors of two vertices with the join neighborhood of the two same vertices [Fouss et al., 2016, Zhou et al., 2009].

$$[\mathbf{S}_j]_{ij} = \frac{|\Gamma(i) \cap \Gamma(j)|}{|\Gamma(i) \cup \Gamma(j)|} \tag{4.10}$$

The Jaccard similarity gives back a ratio that will be higher if the two nodes share a lot of common neighbors with a maximum of 1 if they share the exact same neighbors. Here is Jaccard similarity matrix computation,

$$\mathbf{S}_j = \frac{\mathbf{A}^2}{(\mathbf{d}\mathbf{e}^\mathbf{T}) + (\mathbf{e}\mathbf{d}^\mathbf{T}) - \mathbf{A}^2} \tag{4.11}$$

We see with matrix 4.12, that the Jaccard similarity is quite close to the cosine similarity. We can still find our three correct clusters.

$$\mathbf{S}_j = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 0.60 & 0.60 & 0.60 & 0.43 & 0.20 & 0.11 & 0 & 0 & 0 & 0 \\ 0.60 & 1 & 0.60 & 0.60 & 0.43 & 0.20 & 0.11 & 0 & 0 & 0 & 0 \\ 0.60 & 0.60 & 1 & 0.60 & 0.43 & 0.20 & 0.11 & 0 & 0 & 0 & 0 \\ 0.60 & 0.60 & 0.60 & 1 & 0.43 & 0.20 & 0.11 & 0 & 0 & 0 & 0 \\ 0.43 & 0.43 & 0.43 & 0.43 & 1 & 0.14 & 0.09 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.14 & 1 & 0.14 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.11 & 0.11 & 0.11 & 0.11 & 0.09 & 0.14 & 1 & 0.43 & 0.43 & 0.43 & 0.43 \\ 0 & 0 & 0 & 0 & 0.11 & 0.20 & 0.43 & 1 & 0.60 & 0.60 & 0.60 \\ 0 & 0 & 0 & 0 & 0.11 & 0.20 & 0.43 & 0.60 & 1 & 0.60 & 0.60 \\ 0 & 0 & 0 & 0 & 0.11 & 0.20 & 0.43 & 0.60 & 0.60 & 1 & 0.60 \\ 0 & 0 & 0 & 0 & 0.11 & 0.20 & 0.43 & 0.60 & 0.60 & 0.60 & 1 \end{array}\right] \end{array} \tag{4.12}$$

4.1.6 Leicht-Holme-Newman similarity

The *Leicht-Holme-Newman* similarity introduces in [Leicht et al., 2006] normalizes the common neighbors of two nodes by the expected number of such neighbors, *i.e.* the product of their degree [Zhou et al., 2009, Lü and Zhou, 2011]. The mathematical expression is the following,

$$[\mathbf{S}_{lhn}]_{ij} = \frac{|\Gamma(i) \cap \Gamma(j)|}{d_i d_j} \tag{4.13}$$

And here is the expression for the whole matrix,

$$\mathbf{S}_{lhn} = \frac{\mathbf{A}^2}{\mathbf{d}\mathbf{d}^\mathbf{T}} \tag{4.14}$$

With the Leicht-Holme-Newman similarity, nodes with a high degree are penalized by the multiplication factor. In the LHN (Leicht-Holme-Newman) matrix 4.15, we can see this in action with nodes 5 and 6. They both are equally similar to the vertices $N = 1, 2, 3, 4$. We also see the penalizing effect with the diagonals elements. Node 6 has the highest similarity. Unfortunately, LHN similarity doesn't have a clear and easy 0 to 1 scale. Looking at LHN matrix 4.15, it is a bit difficult to make out the clusters.

$$\mathbf{S}_{lhn} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 0.25 & 0.19 & 0.19 & 0.19 & 0.13 & 0.13 & 0.04 & 0 & 0 & 0 & 0 \\ 0.19 & 0.25 & 0.19 & 0.19 & 0.13 & 0.13 & 0.04 & 0 & 0 & 0 & 0 \\ 0.19 & 0.19 & 0.25 & 0.19 & 0.13 & 0.13 & 0.04 & 0 & 0 & 0 & 0 \\ 0.19 & 0.19 & 0.19 & 0.25 & 0.13 & 0.13 & 0.04 & 0 & 0 & 0 & 0 \\ 0.13 & 0.13 & 0.13 & 0.13 & 0.17 & 0.08 & 0.03 & 0.04 & 0.04 & 0.04 & 0.04 \\ 0.13 & 0.13 & 0.13 & 0.13 & 0.08 & 0.50 & 0.08 & 0.13 & 0.13 & 0.13 & 0.13 \\ 0.04 & 0.04 & 0.04 & 0.04 & 0.03 & 0.08 & 0.17 & 0.13 & 0.13 & 0.13 & 0.13 \\ 0 & 0 & 0 & 0 & 0.04 & 0.13 & 0.13 & 0.25 & 0.19 & 0.19 & 0.19 \\ 0 & 0 & 0 & 0 & 0.04 & 0.13 & 0.13 & 0.19 & 0.25 & 0.19 & 0.19 \\ 0 & 0 & 0 & 0 & 0.04 & 0.13 & 0.13 & 0.19 & 0.19 & 0.25 & 0.19 \\ 0 & 0 & 0 & 0 & 0.04 & 0.13 & 0.13 & 0.19 & 0.19 & 0.19 & 0.25 \end{array}\right] \end{array} \tag{4.15}$$

4.1.7 Sørsen similarity

The *Sørsen* similarity, introduce in [Sorensen, 1948], normalize the common neighbors of two nodes, not by the expected number of likewise neighbors, but simply by the sum of the two nodes degree [Zhou et al., 2009]. It is defined as,

$$[\mathbf{S}_s]_{ij} = \frac{|\Gamma(i) \cap \Gamma(j)|}{d_i + d_j} \tag{4.16}$$

And its expression for the whole matrix,

$$[\mathbf{S}_s]_{ij} = \frac{\mathbf{A}^2}{(\mathbf{de^T}) + (\mathbf{ed^T})} \tag{4.17}$$

It is mainly used for ecological community data [Zhou et al., 2009, Lü and Zhou, 2011].

$$\mathbf{S}_s = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 0.75 & 0.75 & 0.75 & 0.60 & 0.33 & 0.20 & 0 & 0 & 0 & 0 \\ 0.75 & 1 & 0.75 & 0.75 & 0.60 & 0.33 & 0.20 & 0 & 0 & 0 & 0 \\ 0.75 & 0.75 & 1 & 0.75 & 0.60 & 0.33 & 0.20 & 0 & 0 & 0 & 0 \\ 0.75 & 0.75 & 0.75 & 1 & 0.60 & 0.33 & 0.20 & 0 & 0 & 0 & 0 \\ 0.60 & 0.60 & 0.60 & 0.60 & 1 & 0.25 & 0.17 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.33 & 0.33 & 0.33 & 0.33 & 0.25 & 1 & 0.25 & 0.33 & 0.33 & 0.33 & 0.33 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.17 & 0.25 & 1 & 0.60 & 0.60 & 0.60 & 0.60 \\ 0 & 0 & 0 & 0 & 0.20 & 0.33 & 0.60 & 1 & 0.75 & 0.75 & 0.75 \\ 0 & 0 & 0 & 0 & 0.20 & 0.33 & 0.60 & 0.75 & 1 & 0.75 & 0.75 \\ 0 & 0 & 0 & 0 & 0.20 & 0.33 & 0.60 & 0.75 & 0.75 & 1 & 0.75 \\ 0 & 0 & 0 & 0 & 0.20 & 0.33 & 0.60 & 0.75 & 0.75 & 0.75 & 1 \end{array}\right] \end{array} \tag{4.18}$$

In the Sørsen similarity matrix 4.18, we can see that the results are very close to the cosine similarity matrix. We can identify the 3 clusters quite easily.

Now that we have reviewed the different similarity metrics that we are going to use. Let's move to the different transformations; we will apply over the similarity matrices.

4.2 Transformations

We subject the similarity matrices to different transformations to test whether a particular type of transformation would help them perform better. We have applied a modularity transformation; it will be the first transformation to be discussed. On top of that, we also have centered the similarity matrices, and we have centered and normalized the similarity matrix.

4.2.1 Modularity

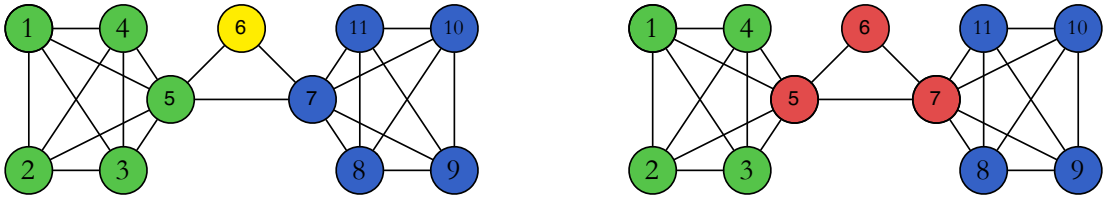
To introduce the modularity transformation, we must first understand what modularity is. The *modularity* is a measure that captures the quality of a graph partitioning, *i.e* how well a network has been clustered. As Chakraborty perfectly puts it in his 2017 article *"Metrics for Community Analysis: A Survey"*, the intuition behind this measure is the following: *"out of the total number of edges in the graph, how many of them are completely internal to a community with respect to the original graph, and how would this number differ if we consider a null graph, that is, a random graph with an identical degree distribution as the original graph?"* [Chakraborty et al., 2017]. Mathematically this gives us the following [Newman, 2010, Chakraborty et al., 2017],

$$\mathbf{Q} = \frac{1}{2m}\sum_{ij}^{n}(a_{ij} - \frac{d_i d_j}{2m})\delta(c_i, c_j) \tag{4.19}$$

With $\delta(c_i, c_j)$ as the Kronecker delta which return 1 if $c_i = c_j$, *i.e* if i and j are in the same cluster, else $\delta = 0$.

The *modularity* gives us a value between -1 and 1 [Chakraborty et al., 2017]; it corresponds to the likelihood of the connection in a network. A positive number means that there is more connection than expected between nodes of the same type. A negative number that there are fewer [Newman, 2010]. So, modularity close to 1 corresponds to a strong community structure [Chakraborty et al., 2017]. $Q = 0.3$ is already an indicator of strong community structures [Clauset et al., 2004]. Numerous algorithms have been developed to maximize modularity [Chakraborty et al., 2017]. This is the case for the algorithms used in this thesis.

Figure 4.2 gives a good idea of how the modularity acts. It reaches a maximum of 0.41 with the clusters determine earlier in 7.1a. Graph partition 7.1b has a pretty god value of 0.32. However, its modularity being lower than partition 7.1a reveals to us that its partition is less optimal. Moreover, modularity takes a negative value if all edges are split into their own separated cluster, 4.2d, and it takes a 0 value if all nodes are grouped into one big cluster, 4.2d [Newman, 2010].



(a) $\mathbf{Q} = 0.41$ (b) $\mathbf{Q} = 0.32$

(c) $\mathbf{Q} = 0$ (d) $\mathbf{Q} = -0.1$

Figure 4.2: Different graph partitioning with its associate modularity values.

Let us now define one of the most important matrices concerning this thesis, the *modularity* matrix. It is considered an attraction/repulsion affinities matrix as it gives us the likelihood of two nodes being into the same cluster [Courtain, 2017, Newman, 2010]. The classic modularity matrix $\mathbf{Q}$ is defined as the following,

$$\mathbf{Q} = \mathbf{A} - \frac{d_i d_j}{2m} \tag{4.20}$$

From which we can derive the modularity measure as follow,

$$Q = \frac{1}{\text{vol}(G)} \sum_{k=1}^{m} \mathbf{u}_k^{\mathrm{T}} \mathbf{Q} \mathbf{u}_k \tag{4.21}$$

With $\mathbf{u}_k$ the binary membership vector for each cluster k, *i.e* $\mathbf{u}_{ik} = 1$ if $i \in C_k$, 0 if else. C_k is the k cluster of nodes on a graph G.

$$\mathbf{Q}_a = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} -0.35 & 0.65 & 0.65 & 0.65 & 0.48 & -0.17 & -0.52 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.65 & -0.35 & 0.65 & 0.65 & 0.48 & -0.17 & -0.52 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.65 & 0.65 & -0.35 & 0.65 & 0.48 & -0.17 & -0.52 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.65 & 0.65 & 0.65 & -0.35 & 0.48 & -0.17 & -0.52 & -0.35 & -0.35 & -0.35 & -0.35 \\ 0.48 & 0.48 & 0.48 & 0.48 & -0.78 & 0.74 & 0.22 & -0.52 & -0.52 & -0.52 & -0.52 \\ -0.17 & -0.17 & -0.17 & -0.17 & 0.74 & -0.09 & 0.74 & -0.17 & -0.17 & -0.17 & -0.17 \\ -0.52 & -0.52 & -0.52 & -0.52 & 0.22 & 0.74 & -0.78 & 0.48 & 0.48 & 0.48 & 0.48 \\ -0.35 & -0.35 & -0.35 & -0.35 & -0.52 & -0.17 & 0.48 & -0.35 & 0.65 & 0.65 & 0.65 \\ -0.35 & -0.35 & -0.35 & -0.35 & -0.52 & -0.17 & 0.48 & 0.65 & -0.35 & 0.65 & 0.65 \\ -0.35 & -0.35 & -0.35 & -0.35 & -0.52 & -0.17 & 0.48 & 0.65 & 0.65 & -0.35 & 0.65 \\ -0.35 & -0.35 & -0.35 & -0.35 & -0.52 & -0.17 & 0.48 & 0.65 & 0.65 & 0.65 & -0.35 \end{array}\right] \end{array} \tag{4.22}$$

Above is the modularity matrix for the networks adjacency matrix in figure 3.1. We see the modularity's value between nodes that are not in the same cluster are negative, except for vertex 5, 6, and 7. The nodes belonging to the same clusters, as determined earlier, have a positive modularity value. The modularity has transformed our direct similarity into an attraction/repulsion matrix. It has the same

effect as the other similarity measures. To obtain a modularity matrix for any similarities, we replace $\mathbf{A}$ by the similarity matrix $\mathbf{S}$ in equation 4.23.

$$\mathbf{Q} = \mathbf{S} - \frac{d_i^s d_j^s}{2m} \tag{4.23}$$

With d_i^s the similarity degree of the node i, $d_i^s = \sum_{j=1}^{n} s_{ij} = s_{i\bullet}$.

Now that we understand modularity and saw how the modularity matrix transformation affects our similarity matrices. Let us move on to the other transformations.

4.2.2 Double centering & normalizing

To center a matrix, we are going to use a centering matrix $\mathbf{H}$. When multiplied with a vector, it subtracts the mean of the vector components to every element of that vector, thus setting the mean to 0. When centering a $n \times n$ matrix, we can perform a right centering $\mathbf{HS}$, which removes the mean from each of the n columns of the matrix. Or we can perform a left centering $\mathbf{SH}$ that removes the mean from the n rows. However, performing one of those operations onto our similarity matrices will **break** the symmetry of the matrix. This is why we use a double centering transformation, so we keep the matrix symmetry. The row and column means are then set to zero [Grey, 1981, Marden, 1996, Izenman, 2008].
The double centering of our similarity matrices is performed as follow,

$$\mathbf{S}_c = \mathbf{HSH} \tag{4.24}$$

With,

$$\mathbf{H} = \mathbf{I} - \frac{\mathbf{e}\mathbf{e}^{\mathbf{T}}}{n} \tag{4.25}$$

$\mathbf{I}$ is the identity matrix of size $(n \times n)$.

The attraction/repulsion matrix 4.26 is the common neighbors similarity matrix after a double centering transformation.

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 2.41 & 1.41 & 1.41 & 1.41 & 0.87 & -0.04 & -1.13 & -1.59 & -1.59 & -1.59 & -1.59 \\ 1.41 & 2.41 & 1.41 & 1.41 & 0.87 & -0.04 & -1.13 & -1.59 & -1.59 & -1.59 & -1.59 \\ 1.41 & 1.41 & 2.41 & 1.41 & 0.87 & -0.04 & -1.13 & -1.59 & -1.59 & -1.59 & -1.59 \\ 1.41 & 1.41 & 1.41 & 2.41 & 0.87 & -0.04 & -1.13 & -1.59 & -1.59 & -1.59 & -1.59 \\ 0.87 & 0.87 & 0.87 & 0.87 & 3.32 & -0.59 & -1.68 & -1.13 & -1.13 & -1.13 & -1.13 \\ -0.04 & -0.04 & -0.04 & -0.04 & -0.59 & 1.50 & -0.59 & -0.04 & -0.04 & -0.04 & -0.04 \\ -1.13 & -1.13 & -1.13 & -1.13 & -1.68 & -0.59 & 3.32 & 0.87 & 0.87 & 0.87 & 0.87 \\ -1.59 & -1.59 & -1.59 & -1.59 & -1.13 & -0.04 & 0.87 & 2.41 & 1.41 & 1.41 & 1.41 \\ -1.59 & -1.59 & -1.59 & -1.59 & -1.13 & -0.04 & 0.87 & 1.41 & 2.41 & 1.41 & 1.41 \\ -1.59 & -1.59 & -1.59 & -1.59 & -1.13 & -0.04 & 0.87 & 1.41 & 1.41 & 2.41 & 1.41 \\ -1.59 & -1.59 & -1.59 & -1.59 & -1.13 & -0.04 & 0.87 & 1.41 & 1.41 & 1.41 & 2.41 \end{array}\right] \end{array} \tag{4.26}$$

To perform the normalize transformation to our matrices we simply applied an

elementwise division to the centered matrix. It goes as follow,

$$\mathbf{S}_{cn} = \frac{\mathbf{HSH}}{\max(\mathbf{HSH})} \tag{4.27}$$

The attraction/repulsion matrix 4.28 is the doubly centered common neighbors similarity matrix after normalization. We can see that all the values are now comprised between -1 and 1. The normalization transformation puts every similarity on a -1 to 1 scale.

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 0.73 & 0.43 & 0.43 & 0.43 & 0.26 & -0.01 & -0.34 & -0.48 & -0.48 & -0.48 & -0.48 \\ 0.43 & 0.73 & 0.43 & 0.43 & 0.26 & -0.01 & -0.34 & -0.48 & -0.48 & -0.48 & -0.48 \\ 0.43 & 0.43 & 0.73 & 0.43 & 0.26 & -0.01 & -0.34 & -0.48 & -0.48 & -0.48 & -0.48 \\ 0.43 & 0.43 & 0.43 & 0.73 & 0.26 & -0.01 & -0.34 & -0.48 & -0.48 & -0.48 & -0.48 \\ 0.26 & 0.26 & 0.26 & 0.26 & 1 & -0.18 & -0.50 & -0.34 & -0.34 & -0.34 & -0.34 \\ -0.01 & -0.01 & -0.01 & -0.01 & -0.18 & 0.45 & -0.18 & -0.01 & -0.01 & -0.01 & -0.01 \\ -0.34 & -0.34 & -0.34 & -0.34 & -0.50 & -0.18 & 1 & 0.26 & 0.26 & 0.26 & 0.26 \\ -0.48 & -0.48 & -0.48 & -0.48 & -0.34 & -0.01 & 0.26 & 0.73 & 0.43 & 0.43 & 0.43 \\ -0.48 & -0.48 & -0.48 & -0.48 & -0.34 & -0.01 & 0.26 & 0.43 & 0.73 & 0.43 & 0.43 \\ -0.48 & -0.48 & -0.48 & -0.48 & -0.34 & -0.01 & 0.26 & 0.43 & 0.43 & 0.73 & 0.43 \\ -0.48 & -0.48 & -0.48 & -0.48 & -0.34 & -0.01 & 0.26 & 0.43 & 0.43 & 0.43 & 0.73 \end{array}\right] \end{array} \tag{4.28}$$

For each similarity matrix, we have 3 transformations. We have in total, for every different similarity presented above, 4 attraction/repulsion matrices. They will be injected into the KKM and the ARSC algorithms.

In the following chapter, we will introduce another type of measure, the *dissimilarity* measure. With measure will eventually construct *kernels* that will be used like our attraction/repulsion matrices.

Chapter 5

Distance metrics

In the previous chapter, we have introduced the concept of *similarity* between nodes. Here, we will introduce the converse, the *dissimilarity* metric. From the dissimilarity, we will be able to build kernels. We will introduce the concept of kernels in the next chapter. They are similarity matrices base upon a scalar product.

5.1 Basic notions

We have seen that the similarity between two nodes is an ill-defined concept. However, the dissimilarity coefficient, Δ_{ij} between two nodes i and j, has a set of strict characteristics under which it must comply to qualify as a *dissimilarity* measure [Fouss et al., 2016]. The set of characteristics are the following,

- **Non-negativity**: $\Delta_{ij} > 0 \quad \forall \quad i, j$
- **Symmetry**: $\Delta_{ij} = \Delta_{ji} \quad \forall \quad i, j$
- **Calibrate to zero**: $\Delta_{ij} = 0 \quad$ if and only if the object $\quad i \quad$ is the same as $\quad j$
- **Triangle inequality**: $\Delta_{ij} \leq \Delta_{ik} + \Delta_{kj} \quad \forall \quad i, j \quad$ and $\quad k$

A measure that complies with the three first characteristics is called a *dissimilarity* measure. If the dissimilarity coefficient respects the fourth characteristic, it qualifies as a *distance* measure [Fouss et al., 2016].

Furthermore, a measure is qualified as *Euclidean* if the nodes' configuration is such that the Euclidean distance between them is preserved, as they initially were [Fouss et al., 2016].

5.2 Distance measures

There is a wide variety of distance measures. Still, in this thesis's scope, we only use three different distance measures, the commute time, the logarithm forest, and the free energy distances. Those three distances have been well studied and are often used to make kernels plugged into the KKM algorithm. We will use them to compare the performance of our attraction/repulsion matrices.

5.2.1 Commute time distance

To define the *commute time distance*, we must first introduce the concept of a *random walk* on a graph. A *Markov chain* is a stochastic process, *i.e.* a mathematical process that evolves through time with a certain probability. The Markov chain is a special case of a stochastic process where the present state, $s(t)$, depends on the previous system's states, $s(t-1)$. Before $s(t-1)$, any states have no incidence whatsoever [Kemeny and Snell, 1976].

A random walk on a graph corresponds to the series of nodes visited by a random walker under a Markov chain process. The time step t, the state of the Markov chain corresponds to $s(t)$. If the random walk at time t is on node i, then $s(t) = i$. The probability that the random walk moves from i to the adjacent node j at time $s(t+1)$ is known and correspond to the following [Fouss et al., 2016, Saerens et al., 2004],

$$p_{ij} = P(s(t+1) = j | s(t) = i) = \frac{a_{ij}}{a_{i\bullet}} \tag{5.1}$$

Before introducing our dissimilarity quantity, the commute time distance, we need first to define one important quantity, the *average first passage time*. The *average first passage time* $m(i,k)$, is the expected number of *steps* a random walker starting from node i will take to arrive at node k. Its mathematical expression is the following [Fouss et al., 2016, Saerens et al., 2004],

$$m(i,k) = \begin{cases} m(i,k) = 1 + \sum_{j=1}^{n} p_{ij} m(i,k) & \text{for i} \neq k \\ m(i,k) = 0 & \text{for i = k} \end{cases} \tag{5.2}$$

Another equation to compute the average first passage time exist; it is based upon the Laplacian pseudo-inverse matrix $\mathbf{L}^+$. Its equation for an undirected graph corresponds to [Fouss et al., 2016],

$$m(i,j) = \sum_{k}^{n} = (l_{ik}^+ - l_{ij}^+ - l_{jk}^+ - l_{jj}^+) d_k \tag{5.3}$$

As $\mathbf{L}^+$ and $\mathbf{L}$ are doubly centered, we can simply derive $\mathbf{L}^+$ by taking the classic matrix inversion, it is the following [Fouss et al., 2016],

$$\mathbf{L}^+ = (\mathbf{L} + \frac{\mathbf{e}\mathbf{e}^{\mathrm{T}}}{n})^{-1} - \frac{\mathbf{e}\mathbf{e}^{\mathrm{T}}}{n} \tag{5.4}$$

We can now define the commute time distance, also known as the average commute time $n(i, j)$. It counts the average number of steps taken by a random walker starting at node i, going to j for the first time, and going back to i. Counter to the average first passage time, the average commute time is symmetric. It is defined as [Fouss et al., 2016, Saerens et al., 2004],

$$n(i, j) = m(i, j) + m(j, i) \tag{5.5}$$

Once again, there also exists an easy way to compute this measure using the Laplacian pseudo inverse [Fouss et al., 2016],

$$n(i, j) = vol(G)(l^+_{ii} + l^+_{jj} - 2l^+_{ij}) \tag{5.6}$$

If we look back at section 5.1, we can see that the average commute time respects all four criteria to qualify as a *distance* measure. Let us now define the commute time distance matrix [Fouss et al., 2016],

$$\Delta_{CT} = vol(G)(\mathrm{diag}(\mathbf{L}^+)\mathbf{e}^{\mathrm{T}} + \mathbf{e}(\mathrm{diag}(\mathbf{L}^+))^{\mathrm{T}} - 2\mathbf{L}^+) \tag{5.7}$$

and the matrix elements are defined as [Fouss et al., 2016],

$$[\Delta_{CT}]_{ij} = n(i, j) = vol(G)(\mathbf{e}_i - \mathbf{e}_j)^{\mathrm{T}}\mathbf{L}^+(\mathbf{e}_i - \mathbf{e}_j) \tag{5.8}$$

We will see later that we need our distance metric to be *Euclidean* to make a valid kernel. We can obtain the Euclidean commute time distance by taking the commute time distance square root since $\mathbf{L}^+$ is defined as semidefinite positive. Its mathematical expression is the following [Fouss et al., 2016],

$$\Delta_{ECT} = \sqrt{\Delta_{CT}} = \sqrt{vol(G)}(\mathrm{diag}(\mathbf{L}^+)\mathbf{e}^{\mathrm{T}} + \mathbf{e}(\mathrm{diag}(\mathbf{L}^+))^{\mathrm{T}} - 2\mathbf{L}^+)^{\frac{1}{2}} \tag{5.9}$$

One major setback from this metric, is that it tends to get lost whenever the graph is too big. The commute time distance only considers the local density of the two nodes. It completely ignores the bigger picture. In other words, it becomes utterly useless whenever applied to a large graph [Fouss et al., 2016].

5.2.2 Logarithm forest distance

Chebotarev introduced a new class of distance in his two articles of 2011, [Chebotarev, 2011a, Chebotarev, 2011b], the distances are based upon the matrix forest theorem [Chebotarev, 2011a, Chebotarev, 2011b, Fouss et al., 2016]. Chebotarev begins by constructing a positive semi-define similarity matrix[1],

$$\mathbf{K} = (\mathbf{I} + \alpha\mathbf{L})^{-1} \quad \text{with} \quad \alpha > 0 \tag{5.10}$$

To institute his new family of distance, Chebotarev define a new *similarity* matrix $\mathbf{S}$ computed as [Fouss et al., 2016],

$$\mathbf{S} = \begin{cases} (\alpha - 1)\log_\alpha(\mathbf{K}) & \text{when} \quad \alpha \neq 1 \\ \ln(\mathbf{K}) & \text{when} \quad \alpha = 1 \end{cases} \tag{5.11}$$

The logarithm forest distance is defined as follow [Chebotarev, 2011a, Fouss et al., 2016],

$$\Delta_{LF}^{(2)} = \text{diag}(\mathbf{S})\mathbf{e}^{\mathrm{T}} + \mathbf{e}(\text{diag}(\mathbf{S}))^{\mathrm{T}} - 2\mathbf{S} \tag{5.12}$$

Readers can find proof that this metric is indeed a valid distance metric in [Chebotarev, 2011a]. Moreover, the logarithm forest distance becomes the commute time distance whenever α converge towards ∞ [Fouss et al., 2016].

5.2.3 Free energy distance

In this section, we introduce the last of our distance measures, the *free energy* distance. The free energy uses the *bag-of-hitting-paths* probability. The *bag-of-paths* is a rich framework that has many applications. In our case, we will use it to compute a distance, *e.g.* the free energy [Fouss et al., 2016]. The framework is based upon the *probability* of drawing a path from a "bag-of-paths" that starts at a vertex i and end at a vertex j [Fouss et al., 2016, Françoisse et al., 2017].

Before introducing the free energy distance, we would like to report that the free energy distance has proven to be one of the best performing distance in [Courtain, 2017]. It is for this reason that its kernel won't only be injected into the KKM algorithm. It will also be injected into the ARSC with the different transformations presented in the previous chapter.

[1] In the chapter about kernel, we will see that this is a required characteristic of a distance measure to build a valid kernel.

The free energy distance between a node i to a node j is define in equation 5.13 and 5.14.

$$[\Delta^{\phi}_{FE}]_{ij} = \begin{cases} \frac{\phi(i,j)+\phi(j,i)}{2} & \text{if i} \neq j \\ 0 & \text{if i} = \text{j} \end{cases} \tag{5.13}$$

with,

$$\phi(i,j) = -\frac{1}{\beta}\log z^{h}_{ij} = -\frac{1}{\beta}\log\frac{z_{ij}}{z_{jj}} \tag{5.14}$$

Where β is the inverse temperature, $\beta > 0$. z^{h}_{ij} is the bag-of-hitting-paths probability between i and j. And z_{ij} is the bag-of-paths probability from vertex i to the ending node j. We invite the readers to look at [Courtain, 2017] and [Françoisse et al., 2017] for a detailed development of the bag-of-hitting-paths probability computation.

The free energy is a valid distance metric with similar property to the logarithm forest distance. For example, if the β parameter converges towards ∞, the free energy distance becomes the shortest path distance; And like the logarithm forest distance, if β converges toward 0^{+}, we end up with the commute time distance [Françoisse et al., 2017, Fouss et al., 2016].

Now that we have demonstrated the three distances metrics used in this thesis let us now move to the next chapter. We will introduce the concept of kernels and develop kernels from our three distances metrics.

Chapter 6

Kernels

In this chapter, we are introducing the mathematical object known as kernels. We will first lay out the theory behind this concept. Then we will derive kernels from the distance measures that we introduced in the previous chapter. Kernels are affinity/repulsion matrices that we inject in the KKM and the ARSC algorithm.

6.1 Basic notions

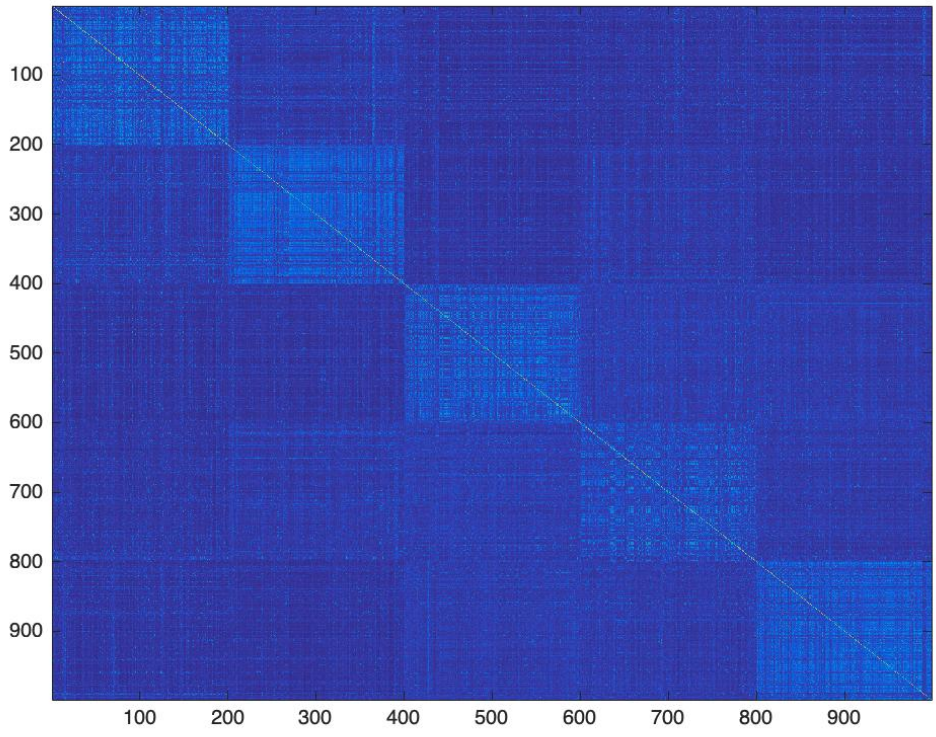


Figure 6.1: Sigmoid commute time kernel on the news_5cl_1 dataset.

A kernel is a function that maps the similarity of two objects, vertices. The similarity is the inner product of the vector representation of those two objects. Those vectors are called feature vector in the embedding space [Fouss et al., 2016, Shawe-Taylor et al., 2004]. In our case, we derive two types of kernels. The first one is derived through an MDS (multidimensional scaling); it is a means of visualizing the similarity of individual cases of a dataset [Mead, 1992]. The second type of kernel is derived through the Gaussian kernel method.

Figure 6.1 is the heat-map representation of the sigmoid commute time kernel[1] over one of the datasets[2]. Using a heat-map to visualize the SCT kernel, we can see five different clusters along the diagonal.

A kernel function must satisfy two conditions to be valid, *i.* it must be symmetric. This comes down from the inner vectors product; *ii.* it must be positive semidefinite, which in itself states symmetry [Fouss et al., 2016].

Kernels have two big advantages, *i.* it computes the similarity between objects, *i.e.* nodes, in a high dimensional space without requiring any mapping from the input space to the embedded space. *ii.* Capture a similarity between objects that cannot be naturally represented by a set of features [Fouss et al., 2016].

A kernel must comply with the property of a symmetric positive semidefinite matrix. If all eigenvalues of $\mathbf{K}$ are non-negative, then the matrix is positive semidefinite [Fouss et al., 2016, Meyer, 2000].

The matrix $\mathbf{K}$ may be only symmetric and not semidefinite positive. In that case, it is only referred to as a similarity measure. Hopefully, there is a trick to transform a symmetric matrix $\mathbf{K}$ into a symmetric semidefinite positive matrix. It is the following [Fouss et al., 2016],

$$\mathbf{K} + \alpha\mathbf{I}, \text{with } \alpha \geq |\lambda_{min}| \tag{6.1}$$

Where λ_{min} is the lowest eigenvalue, another way to do this is by computing its spectral decomposition and set the negative eigenvalues to 0 [Fouss et al., 2016].

[1] It is the kernel generated from the commute time distance introduce last chapter, the SCT (sigmoid commute time) kernel is introduced later in this chapter.

[2] See table 10.1.

6.2 Kernel computation from distance metrics

6.2.1 Kernel on a graph

The kernel matrix $\mathbf{K}$ is obtained through the following MDS [Fouss et al., 2016],

$$\mathbf{K} = -\frac{1}{2}\mathbf{H}\boldsymbol{\Delta}^{(2)}\mathbf{H} \tag{6.2}$$

With $\mathbf{H}$ as the symmetric centering matrix and $\boldsymbol{\Delta}^{(2)} = \boldsymbol{\Delta} \circ \boldsymbol{\Delta}$ is the distance matrix squared with $\circ$ an element-wise product. The $\mathbf{K}$ matrix obtained through this computation is positive semidefinite under the condition that the distance is embedded into a *Euclidean*[3] space. The distance isn't impacted if the origin of the coordinate changes. However, this is not the case for the inner product. So, we fix the origin of the coordinate to the node vector centroid to obtain one, unique, kernel matrix.

Moreover, The kernel matrix $\mathbf{K}$ procures a Euclidean distance metric Δ_{ij} for all nodes i and j, *e.g.*

$$\Delta_{ij}^2 = k_{ii} + k_{jj} - 2k_{ij} \tag{6.3}$$

The distance matrix $\boldsymbol{\Delta}$ is computed from $\mathbf{K}$ as the following,

$$\boldsymbol{\Delta}^{(2)} = \text{diag}(\mathbf{K})\mathbf{e}^{\mathrm{T}} + \mathbf{e}(\text{diag}(\mathbf{K}))^{\mathrm{T}} - 2\mathbf{K} \tag{6.4}$$

The watchful reader might recognize this formulation from 5.2.2 from when we gave the computation for the logarithm forest distance [Fouss et al., 2016].

6.2.2 Gaussian kernel

Additionally, we are going to build what is called a *Gaussian* kernel. It converts our dissimilarity, Δ_{ij}, into a similarity measure [Fouss et al., 2016, Zaki and Meira, 2014].

$$s_{ij} = \exp -\frac{(\Delta_{ij})^2}{2\sigma^2} \tag{6.5}$$

With σ an appropriate width parameter defined as,

$$\sigma = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\delta_{FE}^{\phi}(i,j)}{n(n-1)} \tag{6.6}$$

The Gaussian kernel will only be used over the *free energy*.

[3]This is why it is important for distance measures to be Euclidean.

6.3 Commute time kernel

In order to compute the *commute time kernel*, we have to use equation 6.2 to square the Euclidean commute time distance[4]. This gives us the following expression,

$$\begin{aligned}\mathbf{K} &= -\frac{vol(G)}{2}\,\mathbf{H}(\mathrm{diag}(\mathbf{L}^{+})\mathbf{e}^{\mathrm{T}} + \mathbf{e}(\mathrm{diag}(\mathbf{L}^{+}))^{\mathrm{T}} - 2\mathbf{L}^{+})\mathbf{H} \\ &= -\frac{vol(G)}{2}(-2\,\mathbf{H}\mathbf{L}^{+}\mathbf{H}) \\ &= vol(G)\mathbf{L}^{+}\end{aligned} \tag{6.7}$$

As the matrix, $\mathbf{L}^{+}$ is already centered, $\mathbf{H}\mathbf{L}^{+}\mathbf{H} = \mathbf{L}^{+}$. We have seen that $\mathbf{L}^{+}$ is already the result of an inner product of two distances embedded in a Euclidean space. The Euclidean commute time distance separates the nodes up to a scaling factor $vol(G)$. The commute time distance Kernel is then [Courtain, 2017, Fouss et al., 2016, Saerens et al., 2004],

$$\mathbf{K}_{ct} = \mathbf{L}^{+} \tag{6.8}$$

As a matter of fact, this kernel performs very poorly. To remedy this problem, we have to apply a *sigmoïd* transformation to each element of the matrix [Fouss et al., 2016],

$$[\mathbf{K}^{s}_{ct}]_{ij} = \frac{1}{1 + \exp[-\alpha\, k_{ij}/\sigma]} \tag{6.9}$$

Where α is a strictly positive parameter that needs tuning, σ is the standard deviation of the element of $\mathbf{K}_{ct}$ and k_{ij} is the element i, j of the kernel $\mathbf{K}_{ct}$.

In section 6.1, we introduce two methods to transform a none positive semidefinite matrix into one. It comes into play here, as the elementwise *sigmoïd* transformation does not guarantee a positive semidefinite matrix [Fouss et al., 2016].

$$\mathbf{K}^{s}_{sct} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \left[\begin{array}{ccccccccccc} 0.55 & 0.55 & 0.55 & 0.55 & 0.43 & -0.22 & -0.57 & -0.45 & -0.45 & -0.45 & -0.45 \\ 0.55 & 0.55 & 0.55 & 0.55 & 0.43 & -0.22 & -0.57 & -0.45 & -0.45 & -0.45 & -0.45 \\ 0.55 & 0.55 & 0.55 & 0.55 & 0.43 & -0.22 & -0.57 & -0.45 & -0.45 & -0.45 & -0.45 \\ 0.55 & 0.55 & 0.55 & 0.55 & 0.43 & -0.22 & -0.57 & -0.45 & -0.45 & -0.45 & -0.45 \\ 0.43 & 0.43 & 0.43 & 0.43 & 1 & 0.33 & -0.37 & -0.57 & -0.57 & -0.57 & -0.57 \\ -0.22 & -0.22 & -0.22 & -0.22 & 0.33 & 1.14 & 0.33 & -0.22 & -0.22 & -0.22 & -0.22 \\ -0.57 & -0.57 & -0.57 & -0.57 & -0.37 & 0.33 & 1 & 0.43 & 0.43 & 0.43 & 0.43 \\ -0.45 & -0.45 & -0.45 & -0.45 & -0.57 & -0.22 & 0.43 & 0.55 & 0.55 & 0.55 & 0.55 \\ -0.45 & -0.45 & -0.45 & -0.45 & -0.57 & -0.22 & 0.43 & 0.55 & 0.55 & 0.55 & 0.55 \\ -0.45 & -0.45 & -0.45 & -0.45 & -0.57 & -0.22 & 0.43 & 0.55 & 0.55 & 0.55 & 0.55 \\ -0.45 & -0.45 & -0.45 & -0.45 & -0.57 & -0.22 & 0.43 & 0.55 & 0.55 & 0.55 & 0.55 \end{array}\right] \end{array} \tag{6.10}$$

The matrix 6.10 is the sigmoïd commute time kernel matrix with $\alpha = 50$ for graph

[4]See equation 5.10.

3.1 in chapter 3. To verify that this matrix is positive semi-definite, we simply have to check its eigenvalues, $\lambda = [0, 0, 0, 0, 0, 0, 0, 0.1, 1.7, 5.0] \geq 0$.

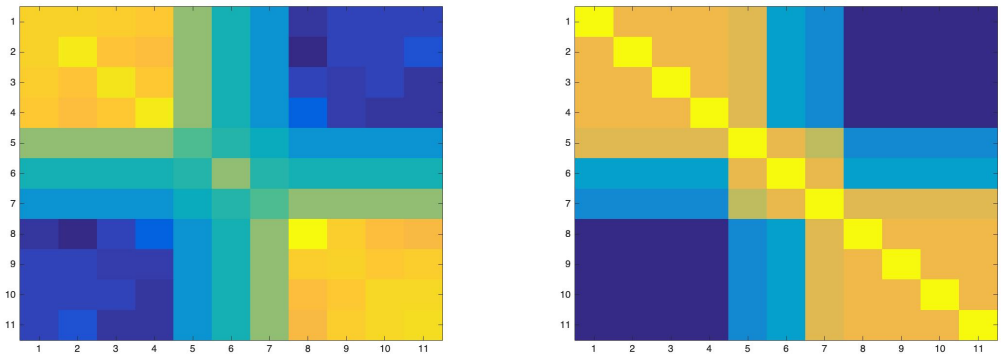
6.4 Log forest and free energy kernels

To obtain $\mathbf{K}_{fe}$, the free energy kernel and $\mathbf{K}_{lf}$, log forest[5] kernels, we once again use equation 6.2. Additionally, those two distances are not *Euclidean*. We have to employ one of the two tricks from section 6.1 to turn the matrices into valid positive semidefinite kernels [Fouss et al., 2016].

Furthermore, to test the free energy distance in the ARSC algorithm, we need to construct a *Gaussian* free energy kernel. It turns the free energy distance into a similarity measure used by our algorithm[6]. It is computed as followed [Fouss et al., 2016, Saerens et al., 2004],

$$\mathbf{K}_{gfe} = \exp \frac{-(\Delta^{\phi}_{FE})^2}{\sigma} \tag{6.11}$$

In figure 6.2, we have left the free energy kernel obtained with MDS and on the right, the free energy kernel obtained with the Gaussian kernel method. We can see that the two methods yield dramatically different heat-map.



(a) Free energy kernel (b) Gaussian free energy kernel

Figure 6.2: Heat-maps of two different free energy base kernels upon graph. 3.1.

Now that we have finally introduced all the measures that we will use to partition our graphs, we can move on to the graph partitioning itself. In the next chapter, we are first going to define what exactly is a community. Then we will look at how we can detect them base upon all the attraction/repulsion matrices we have

[5]The logarithm forest is already a squared distance.
[6]The Gaussian free energy kernel will also be injected into the KKM algorithm.

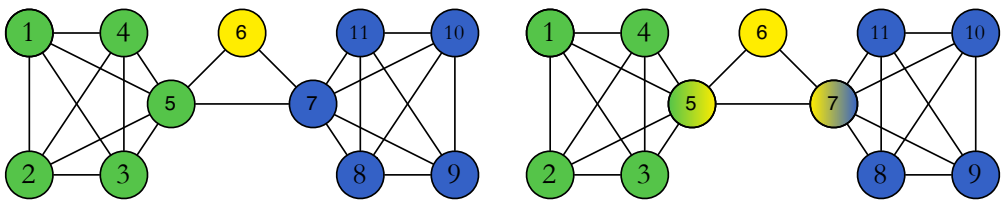
introduced in the last two chapters. We will explain step by step how the algorithms are running.

Chapter 7

K-means network clustering

We are now entering the core of this thesis, the *graph partitioning*. Now that we have seen the different maps, we can introduce the algorithms using those maps (*e.g.* attraction/repulsion matrices) to perform a community detection over the mapped graphs. Yet, we have not given any proper definition of what a community, a cluster is. This is since the concept of a community/cluster in a network is a hard concept to encapsulate mathematically [Fortunato and Hric, 2016, Fouss et al., 2016, Newman, 2010].

The consensus over what a *community* is, says that a community is a series of nodes in a group, where the number of edges inside that structure is greater than the number of edges outside of it. Communities exist due to their *structural* and *functional* similarities. In graph theory, we only have access to the *structural* similarities between nodes. This gives us very little information but a high-level view of how a network interacts [Chakraborty et al., 2017].



(a) nonoverlapping (b) overlapping

Figure 7.1: Illustration of two different kinds of community structures.

As one could expect, there is a wide variety of community types. This is another reason why communities are ill-define. In figure 7.1, we see an example of two different community types upon the same graph. Figure 7.1.a shows us disjoint or

non-overlapping communities; the vertices can only belong to one community at the time. Whereas in figure 7.1.b, we see overlapping/fuzzy communities; here, the vertex can simultaneously belong to different communities. A disjoint community graph could be a network of university friends in the real world, each belonging to different faculty. A fuzzy community graph could be the nationalities of those students, where some have double nationalities. Disjoint and fuzzy communities are not the only two types. There exist other kinds of community structures. For example, a hierarchical community structure is where the different clusters have different levels [Chakraborty et al., 2017, Fouss et al., 2016].

There isn't only one type of community that opens the door for a wide range of different algorithms with different clustering methods. Algorithms are usually made to work specifically with one type of community. In this thesis, we study two algorithms. The first is the KKM (kernel k-means), which returns non-overlapping community membership. The second is the ARSC (Attraction Repulsion Soft Clustering) that returns a fuzzy community membership. However, we will only work on datasets containing non-overlapping communities. Hence, the fuzzy community membership returned by the ARSC algorithm will be turned into a disjoint membership [Courtain, 2017, Chakraborty et al., 2017, Fouss et al., 2016, Fortunato and Hric, 2016].

In this chapter, we focus on kernel-based algorithms. We first introduce *kernel k-means*. Thereafter, we will introduce the *fuzzy k-means* to explain the intuition behind the ARSC better. The latter will be introduced in the next chapter.

7.1 Kernel k-means

The KKM algorithm we are laying out in this section is a prototype-based version of the algorithm. The cluster prototype vectors are going to be set in the sample vector space. This vector space is a Euclidean space with dimensionality equal to the number of nodes n, where each coordinate corresponds to a node of the graph [Courtain, 2017, Fouss et al., 2016].

This algorithm's first step is already covered as it corresponds to the computation of a valid kernel $\mathbf{K}$. The KKM has been shown to converge even if the input kernel is not valid. More information can be found in [Fouss et al., 2016].

The KKM algorithm is an iterative optimization problem, where the total *within-cluster inertia* needs to be minimized. However, the within-cluster inertia always

decrees whenever the number of clusters increases. The optimal number of clusters is usually not known beforehand. This leads us to one of the major drawbacks of the KKM algorithm. Several clusters c need to be fixed to solve this problem[1] [Fouss et al., 2016].

A *community detection* algorithm should be able to run without any beforehand information about the number of clusters it is looking for and those clusters' size. If one of those two pieces of information is provided beforehand to the algorithm, it is defined as a *graph partitioning* algorithm, making it less interesting. Some information that we might want to uncover during our analysis must be provided to our algorithm for it to work [Newman, 2010].

This problem can be overcome by running the algorithm with different initialization. We input a different number of clusters c at each iteration. However, we are not doing this in this thesis. We want to test the performance of the algorithm whenever it has the optimal number of clusters.

The total within-cluster inertia is the following [Fouss et al., 2016],

$$J(\mathbf{g}_1, ..., \mathbf{g}_c) = \sum_{k=1}^{c} \sum_{i \in C_k} ||\mathbf{x}_i - \mathbf{g}_k||^2 \tag{7.1}$$

Where the first sum is taken over the c clusters, the second sum is over the vertex i belonging to cluster k, $i \in C_k$. Furthermore, $\mathbf{g}_k$ is the prototype vector, *i.e* the center of gravity of the cluster k. Finally, $||\mathbf{x}_i - \mathbf{g}_k||$ is the Euclidean distance between the vertex i and the cluster k it is assigned to. The vector nodes are preserved in a Euclidean space. This preserve the distance between nodes [Fouss et al., 2016]

Let's define $\mathbf{X}$, the $n \times p$ matrix, containing the n transposed vector nodes as its rows and p as the number of features in the embedded space. We can now define the prototype vector, in the sample space, with the kernel trick [Fouss et al., 2016],

$$\mathbf{g}_k = \mathbf{X}^{\mathrm{T}} \mathbf{h}_k \tag{7.2}$$

With $\mathbf{h}_k$, the prototype vector of the cluster k defined in the n-dimensional vector space, *i.e.* the $\mathbf{h}_k$ vector, simply contains a series of 0 and 1 specifying if the i node is in the k cluster. This transformation characterizes the vector $\mathbf{g}_k$ as a linear combination of $\mathbf{x}_i$. This define the within-cluster inertia as a function of the kernel matrix $\mathbf{K}$, we can now avoid computing $\mathbf{g}_k$ and $\mathbf{x}_i$. Let us now rewrite equation 7.1

[1] There exists other solutions to this problem [Fouss et al., 2016], but in our case, we resolve it this way.

using $\mathbf{h}_k$ [Fouss et al., 2016],

$$J(\mathbf{h}_1, ..., \mathbf{h}_m) = \sum_{k=1}^{c} \sum_{i \in C_k} (\mathbf{x}_i - \mathbf{g}_k)^{\mathrm{T}} (\mathbf{x}_i - \mathbf{g}_k) = \sum_{k=1}^{m} \sum_{i_k} (\mathbf{e}_i - \mathbf{h}_k)^{\mathrm{T}} \mathbf{K} (\mathbf{e}_i - \mathbf{h}_k) \quad (7.3)$$

With $\mathbf{K}$ corresponding to the kernel matrix, $\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$ [Fouss et al., 2016].

Let's now fix a variable u_{ik} as a binary membership indicator, *i.e.* $u_{ik} = 1$ if $i \in k$ else $u_{ik} = 0$. Equation 7.3 becomes [Fouss et al., 2016],

$$J(\mathbf{h}_1, ..., \mathbf{h}_m) = \sum_{i=1}^{n} \sum_{k=1}^{c} u_{ik} (\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i) \quad (7.4)$$

Now that we have expressed the total within-cluster inertia J, using only the kernel matrix $\mathbf{K}$, we can start working on the optimization problem. It is set in two stages; the first is the node vector's *reallocation* while maintaining the prototype vector fixed. The second is the *recomputation* of the prototype vectors $\mathbf{h}_k$, while we keep the cluster labels of the nodes fixed [Fouss et al., 2016],

During the reallocation step, we optimize the binary membership u_{ik}, independently for each node i, by allocating the node to the cluster for which $(\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i)$ is minimal. It is mathematically defined as [Fouss et al., 2016],

$$k_i^* = \underset{k \in 1,...,c}{\operatorname{argmin}} [(\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i)] \quad (7.5)$$

With k_i^* containing the label of the optimal cluster for node i. Once the optimal cluster is found, we can respecify the element of the ith row of the membership function, as $u_{ik_i^*} = 1$ if $k = k_i^*$ and $u_{ik_i^*} = 0$ else wise [Fouss et al., 2016].

Now that the first step is completed, we can move on to the prototype vectors' recomputation. The gradient of J is computed with respect to $\mathbf{h}_k$ in equation 7.4 to zero [Fouss et al., 2016],

$$\mathbf{K}\mathbf{h}_k = \mathbf{K} \frac{1}{n_k} \sum_{i \in C_k} \mathbf{e}_i \quad (7.6)$$

Where n_k is the number of vertices belonging to cluster k. The solution to this equation appears quite trivial, it is solved as follow,

$$\mathbf{h}_k = \frac{1}{n_k} \sum_{i \in C_k} \mathbf{e}_i \quad (7.7)$$

Hence, $[\mathbf{h}_k]_i = \frac{1}{n_k}$ if $i \in k$, else $[\mathbf{h}_k]_i = 0$. The two steps described above will iterate until the J criterion converge, *i.e.* until the *total within-cluster inertia* becomes stationary [Courtain, 2017, Fouss et al., 2016].

Before moving onto the fuzzy k-means algorithm, we invite you to look at what happens step by step in a kernel k-means algorithm with the following frame.

A simple Kernel k-means algorithm

Input:

- $\mathbf{K}$: a valid positive semidefinit kernel matrix of size $n \times n$ generated from a graph G.
- c: the desired number of clusters.

Output:

- $\mathbf{U}$: the membership matrix of size $n \times c$ containing the membership of each nodes i to cluster u, u_{ik}

Functioning

1. **Initialization**: randomly generate c different prototype nodes with indices $q_1, q_2, ..., q_c$ and set $\mathbf{h}_k = \mathbf{e}_{qk}$, with $k = 1, ..., c$ and $\mathbf{e}_i$ is a $n \times 1$ column vector.
2. $\mathbf{U} \leftarrow \mathbf{Zeros}(n \times c)$
3. **Repeat until** the allocation of the nodes does not change any more:
   (a) **for** $i = 1$ to n:
      i. $k* \leftarrow \underset{k \in 1,...,c}{\operatorname{argmin}} (\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i)$ • Reallocation step
      ii. $U_{ik*} \leftarrow 1$ • Store the cluster allocation
   (b) **for** $k = 1$ to c:
      i. $n_k \leftarrow \sum_{i=1}^{n} u_{ik}$ • Compute the number of nodes in each clusters
      ii. $h_k \leftarrow \frac{\mathbf{col}_k(\mathbf{U})}{n_k}$ • recomputation of the prototype
4. **return U**

7.2 Fuzzy algorithm

The *fuzzy* community concept is quite recent, even though in the real-world fuzzy communities are not rare if anything, they might even be the norm [Chakraborty et al., 2017]. As there has been very little groundwork on the subject, a team at *UCLouvain* came up with their own algorithm to deal with fuzzy communities. In 2017, Saerens M. and Courtain S. came up with their own soft modularity[2] algorithm.

This section introduces the *fuzzy k-means* algorithm. It will help us understand the *UCLouvain* soft modularity algorithm presented in the next chapter.

7.2.1 Fuzzy k-means

The *fuzzy k-means* algorithm is a regularize fuzzy version of the KKM algorithm [Li and Mukaidono, 1999, Miyamoto et al., 2008]. Let's look at the optimisation problem this algorithm presents us [Sadaaki and Masao, 1997, Courtain, 2017],

$$\begin{aligned} &\text{minimize} \sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik} d(x_i, g_k) - T\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}\log u_{ik} \\ &\text{subject to} \sum_{k=1}^{c} u_{ik} = 1 \end{aligned} \tag{7.8}$$

Where g_k is the center of the cluster k and $\mathbf{X} = x_1, ..., x_n$ are the n nodes to be clustered into c different clusters. $d(x_i, g_k)$ is a dissimilarity measure between the node and the center of the cluster, *i.e.* x_i and g_k, it is equal to $||x_i - g_k||_2^2$ where $||.||_2$ is a Euclidean norm. Our aim with this algorithm is to make a fuzzy clustering, therefore the variable u_{ik} is no more binary but continuous, it takes a value between 0 and 1. Moreover, the condition in equation 7.8 states that the sum over all the c clusters for each node i must be equal to 1. u_{ik} has become the probabilities of i belonging to k. The constraint is called the probabilistic partition, it guarantees that u_{ik} is a valid probability[3].

The first part of equation 7.8 is the objective function of our optimization problem, it is a difference between two terms. The first double sum, $\sum_{i=1}^{n}\sum_{k=1}^{c} d(x_i, g_k)$, is a new form of the total within-cluster inertia seen in equation 7.1; And the second

[2]The soft modularity is a fuzzy version of the classic modularity.

[3]It should also be specify if we want to be thorough that $u_{ik} \geq 0$. However, it is not necessary as the logarithm in the second part of the term does not allow for any negative values [Sadaaki and Masao, 1997, Mizutani and Miyamoto, 2005]

sum, $T\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}\log u_{ik}$, is the total entropy, with T a positive regularized parameter. The second term is the one that grants the "fuzzyfication" to the k-means [Sadaaki and Masao, 1997, Mizutani and Miyamoto, 2005].

To solve equation 7.8, we use a Lagrangian function $\mathcal{L}$ as [Sadaaki and Masao, 1997, Mizutani and Miyamoto, 2005],

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}d(x_i, g_k) - T\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}\log u_{ik} + \lambda(\sum_{k=1}^{c} u_{ik} - 1) \tag{7.9}$$

Where λ is the Lagrangian multiplier.

The stationary point at which the Lagrangian function gives the optimal solution for u_{ik}^* equate to the following,

$$u_{ik}^* = \frac{\exp[-\beta\, d(x_i, g_k)]}{\sum_{i=1}^{n}(\exp[-\beta\, d(x_i, g_k)])} \tag{7.10}$$

With the optimal solution for g_k^*,

$$g_k^* = \frac{\sum_{i=1}^{n} u_{ik}^* x_i}{\sum_{i=1}^{n} u_{ik}^*} \tag{7.11}$$

As we do not use the fuzzy k-means algorithm, we do not lay out a step-by-step program of its functioning as the KKM. We introduce the fuzzy k-means, so we could demonstrate the intuition behind the ARSC algorithm that is introduced in the next chapter.

In the next chapter, we will first introduce attraction/repulsion clustering. Then we will look at the current work found in the literature. Finally, we will mathematically develop the ARSC algorithm.

Chapter 8

Attraction repulsion network clustering

As seen in the previous chapter, kernel k-means algorithms aim at reducing within cluster-inertia. The approach we will take in this chapter is fundamentally different. We will *maximize* within-cluster affinity and *minimize* between-cluster affinities. This sounds like an obvious and trivial way to perform graph partitioning. Despite that, solving this optimization problem is super hard. Usually, only approximate solutions are chased, and therefore, most algorithms developed to solve that task are based upon some greedy procedure [Courtain et al., 2020, Bagon and Galun, 2011, Beier et al., 2014].

The solution, for optimization problems, usually has a finite number of clusters. It makes solving the problem more interesting as the number of clusters sought doesn't have to be provided beforehand. Solving this clustering problem is called **correlation clustering**, it sought to find a clustering solution to attraction/repulsion data [Courtain et al., 2020, Bagon and Galun, 2011, Beier et al., 2014].

The *ARSC* algorithm was inspired from the *modularity maximization* algorithm introduced in [Lehmann and Hansen, 2007, Rossi and Villa-Vialaneix, 2010]. As we have seen in chapter 4, the modularity matrix is an attraction/repulsion matrix. The team at *UClouvain* derived the modularity maximisation algorithm in [Lehmann and Hansen, 2007, Rossi and Villa-Vialaneix, 2010] to operate upon any attraction/repulsion matrix. The ARCS algorithm task is to cluster attraction/repulsion data when the number of clusters is provided beforehand.

We stated earlier the capacity of correlation clustering to return a finite number of clusters. However, in developing the ARSC algorithm, the team at *UClouvain* decided to remove this feature. The intention here is that in many practical situa-

tions, a limited number of clusters is desire. To illustrate, a marketing department prefers having a few big clusters than a thousand different clusters [Courtain et al., 2020].

Despite not finding the number of clusters, the ARSC algorithm still has some interesting properties: *i.* it is relatively short and easy to implement, *ii.* it is a fuzzy clustering method, so it provides a fuzzy membership, *iii.* scales on large, sparse graph and finally *iv.* it is competitive with state-of-the-art algorithms [Courtain et al., 2020].

8.1 Related works

Let's mention variational approximation/inference, [Neal and Hinton, 1998, Saul et al., 1996, Saul and Jordan, 1996], a powerful and quite recent technique capable of tackling hard optimization and parameter estimation problem by standard maximum likelihood techniques like expectation-maximization [Courtain et al., 2020, Bishop, 2006]. Another technique capable of resolving the clustering problem that was previously introduced is called deterministic annealing [Anderson and Peterson, 1987, Peterson and Söderberg, 1989, Rose et al., 1990].

In [Hofmann and Buhmann, 1997], Hofmann and Buhmann investigate a pairwise data clustering technique with deterministic annealing using an index closely related to the modularity measure. Zhang *et al.* investigated fuzzy modularity in [Zhang et al., 2007], they fixed the k clusters and the corresponding fuzzy partitioning $\mathbf{U}_k$ by maximizing their generalized modularity. Zhang *et al.* used a spectral mapping to embed their graph into a Euclidean space. They could then use a *Fuzzy c-means* to cluster.

Lehmann *et al.* and Rossi *et al.*, [Lehmann and Hansen, 2007, Rossi and Villa-Vialaneix, 2010], developed new algorithms that maximized modularity based upon the mean-field approximation to detect communities. Their paper is the main source of inspiration for the ARSC algorithm. Their work was used to make the ARSC as generic as possible.

In [Jafarov et al., 2020], the authors present an approximation algorithm for correlation clustering with asymmetric classification errors. They argue that their model is more capable of capturing the subtleties of real-world instances.

Graph neural networks are a type of neural network[1] that directly operate onto the graph structure. Graph neural networks have proven themself on many graph analyses tasks. However, they are still lagging regarding graph clustering. In [Tsitsulin et al., 2020], the authors created a *deep modularity network*, an unsupervised pooling method inspired by the modularity measure.

In [Kishore et al., 2020], they tested a new modularity metric for spatially embedded networks. In other words, they tried a new modularity metric to cluster networks that have a spatial distance between nodes, *e.g.* a network of people dispatched across a city. They developed their algorithm intending to make better partitioning of granular materials[2] to better understand their properties. Their new modularity takes into account the spatial distance between two atoms.

With this short overview of the literature, let's now get into the mathematical details.

8.2 Attraction repulsion soft clustering algorithm

In designing their soft modularity, Saerens M., Gues G., and Courtain S. were inspired by three main concepts. The first one is the *modularity*[3] introduce in chapter 4, the second is the *entropy* of the fuzzy k-means, and the third one is the *free energy* optimization [Courtain, 2017]. They defined the objective function to be minimized, as the free energy, as follow [Courtain, 2017, Courtain et al., 2020],

$$\phi(U) = -\frac{1}{2}\sum_{i,j=1}^{n}\sum_{k=1}^{m} u_{ik}u_{jk}q_{ij} + T\sum_{i=1}^{n}\sum_{k=1}^{m} u_{ik}\,\log u_{ik} \tag{8.1}$$

With T a temperature parameter to tune. $\mathbf{U}$, with $[\mathbf{U}]_{ij} = u_{ij}$ as a possible configuration of the system and finally, q_{ij} is the element of the injected attraction/repulsion matrix $\mathbf{Q}$ presented in chapter 4. The *fuzzy k-mean* equation directly inspired this equation. It is also subject to the the probabilistic partition constrain, *i.e.* $\sum_{k=1}^{c} u_{ik} = 1$ for every $u_{ik} = 1$ [Courtain, 2017, Courtain et al., 2020, Devooght and Bersini, 2017].

As the solution for the fuzzy k-means, we can take the partial derivative of the Lagrangian function and set the result to 0. We then find [Courtain, 2017, Courtain et al., 2020],

[1] Neural networks are computing systems vaguely inspired by the biological neural networks that constitute animal brains [Yung-Yao et al., 2019]

[2] Materials composed of small clusters of atoms such as metallic glasses.

[3] this is the concept we are going to modify to test alternate algorithm.

$$\mathcal{L} = -\frac{1}{2}\sum_{i,j=1}^{n}\sum_{k=1}^{c} u_{ik}u_{jk}q_{ij} + T\sum_{i=1}^{n}\sum_{k=1}^{c} u_{ik}\ \log u_{ik} + \sum_{i=1}^{n}\lambda_i(\sum_{k=1}^{m} u_{ik} - 1) \tag{8.2}$$

With λ_i as the Lagrange multiplier for every i. The similarity matrices are symmetrical, *i.e.* $q_{ij} = q_{ji}$. The derivative of the Lagrangian function becomes [Courtain, 2017, Courtain et al., 2020],

$$-\sum_{j=1}^{n} u_{jk}q_{jk} + T\left(\log(u_{ik}) + 1\right) + \lambda_i = 0 \tag{8.3}$$

By isolating the $\log(u_{ik})$ and computing λ_i from the constraints, we can write the fixed-point equation as [Courtain, 2017, Courtain et al., 2020],

$$u_{ik} = \frac{\exp[\beta\sum_{j=1}^{n} q_{ij}u_{jk}]}{\sum_{l=1}^{c}(\exp[-\beta\sum_{j'=1}^{n} q_{ij'}u_{j'l}])} \tag{8.4}$$

With $\beta = \frac{1}{T}$. Like the fuzzy k-means, the optimal cluster membership is obtained by iterating this equation until convergence.

The next page frame contains a step by step the development of the ARSC.

Now that we have introduced the two algorithms to plug the various attraction/repulsion matrices, we will explain the experiments. However, we first need to see how we will determine which algorithm is the best performing. The next chapter will detail precisely how we are going to assess the algorithm performance.

Attraction Repulsion Soft Clustering algorithm

Input:

- A connected weighted undirected graph G containing n nodes.
- The $n \times n$ adjacency matrix A associated with G.
- c: the desired number of clusters.
- The $n \times n$ attraction/repulsion matrix Q associated with G.
- The inverse temperature parameter β.

Output:

- $\mathbf{U}$: the membership matrix of size $n \times c$ containing the membership of each node i to cluster k, u_{ik}

Functioning

1. **Initialization**:
   (a) Initialize each row of $\mathbf{U}^{(0)}$ with an uniform membership distribution plus a small perturbation (*e.g.* 30%);
   (b) Set iteration counter $l = 0$
   (c) $\phi(\mathbf{U}^{(0)}) = 0$
   (d) Normalize $\mathbf{U}^{(0)} = (\text{Diag}(\mathbf{U}^{(0)}\mathbf{e}))^{-1}\mathbf{U}^{(0)}$
2. **Repeat until** $|\phi(\mathbf{U}^{(l+1)}) - \phi(\mathbf{U}^{(l)})| < \epsilon$:
   (a) Compute $\mathbf{U}^{(l+1)} = \exp[\beta\mathbf{Q}\mathbf{U}^{(l)}]$
   (b) Normalize $\mathbf{U}^{(l+1)} = (\text{Diag}(\mathbf{U}^{(l+1)}\mathbf{e}))^{-1}\mathbf{U}^{(l+1)}$
   (c) Compute $\phi(\mathbf{U}^{l+1})$
   (d) Increment l
3. **return** $\mathbf{U}^{(l+1)}$

Chapter 9

Algorithms efficiency

Once our algorithms have performed their clustering assignation, how do we assess their performances? The algorithms aim to assign the correct cluster to each node. We need quality measures that can quantify the performance of the algorithms. We will use the nodes, true class available with the datasets to compute performance indexes. In this chapter, we will introduce two quality measures to assess the clustering quality of each algorithm. Moreover, we will introduce three statistical tests that tell us which algorithms performed significantly better than the others.

9.1 Performance indices

We will first introduce the *normalized mutual information* measure that was established in [Ana and Jain, 2003]. We will then introduce the *adjusted rand index*, first introduced by W. Rand in [Rand, 1971] and later adjusted by L. Hubert in [Hubert and Arabie, 1985].

9.1.1 Normalized mutual information measure

The *NMI* is the quotient of mutual information divided by the arithmetic average of the entropies of Ω, and C. Ω is the set of clusters returned by the algorithms, and C, the set of natural classes. Its mathematical expression is the following [Fortunato and Hric, 2016, Ana and Jain, 2003],

$$NMI(\Omega, C) = \frac{2\, I(\Omega, C)}{H(\Omega) + H(C)} \tag{9.1}$$

With the mutual information $I(\Omega, C)$,

$$I(\Omega, C) = \sum_k \sum_j P(w_k \cap c_j) \log \frac{P(w_k \cap c_j)}{P(w_k)P(c_j)} \tag{9.2}$$

With $P(w_k \cap c_j)$, $P(w_k)$ and $P(c_j)$ are respectively the probability of a node being at the intersection of c_j and w_k, being in the cluster w_k and being in the class c_j. The mutual information measures the amount of information by which our knowledge of the true class membership increases when given the real clusters membership. This measure reaches its minimum 0 whenever the cluster memberships are randomly distributed across the true classes. It reaches its maximum whenever the clustering algorithm manages to predict exactly the true classes or if the number of clusters is the same as the number of nodes [Manning et al., 2008].

To give penalization to large cardinals, and avoid to be bias by a high number of clusters, the mutual information, $I(\Omega, C)$, is normalized by the entropy, $[H(\Omega)+H(C)]/2$ [Manning et al., 2008],

$$H(\Omega) = -\sum_k P(w_k)\log(P(w_k)) \quad \text{and} \quad H(C) = -\sum_j P(c_j)\log(P(c_j)) \tag{9.3}$$

Thanks to this normalization, the NMI metric is bound between $[0, 1]$, and it is no more biased by large cardinalities [Manning et al., 2008].

9.1.2 Adjusted rand index measure

To evaluate an algorithm's accuracy, the ARI criterion looks at each $n(n-1)/2$ pair of nodes and checks whether they are classified in the same cluster in the two partitions Ω and C. There are four different possible instances for each pair of nodes; if the pair are in the same cluster for both partitions, they are qualified as True Positive TP. The converse is the true negative TN; the two nodes are in different communities for both partitions. Then there is the false-positive FP case where the pair is in the same cluster in C but in different clusters in Ω. Finally, the false-negative FN where the pair is in the same cluster in Ω but in different clusters in C. We can now define the rank index as the ratio of the pairs correctly classified in both partition over the total numbers of pairs, it mathematical expression is the following [Fortunato and Hric, 2016, Manning et al., 2008],

$$RI(\Omega, C) = \frac{TP + TN}{TP + TN + FP + FN} \tag{9.4}$$

However, this index has a fatal flaw, it too gets close to its maximum of 1, whenever the number of clusters is large. To overcome this problem, the adjusted rank index ARI was introduced [Hubert and Arabie, 1985],

$$ARI(\Omega, C) = \frac{\binom{n}{2}(TP + TN) - [(TP + FP)(TP + FN) + (TN + FP)(TN + FN)]}{\binom{n}{2}^2 - [(TP + FP)(TP + FN) + (TN + FP)(TN + FN)]} \tag{9.5}$$

9.2 Performance comparison

To statistically compare the performance indices, we will follow the same process used in [Sommer et al., 2016]. This process performs *three* different tests, a *Friedman* test, which determines if one algorithm performed significantly differently. If it is the case, we can go on with a *Nemenyi* test. It performs multiple comparisons to determine if one, or several algorithms, performed significantly differently. Finally, a *Wilcoxon* signed-rank test two-by-two is performed to have a one-on-one comparison of two algorithm performances. We accept a level of trust, $\alpha = 0.05$.

9.2.1 Friedman test

The *Friedman* test introduced by Friedman in [Friedman, 1937], creates a ranking r_j^i of the j algorithm in descending order based upon their performance, *e.g.* *ARI*, on every i datasets. Hereafter, the average rank of every algorithm will be computed as follow [Demšar, 2006],

$$\bar{R}_j = \frac{1}{N} \sum_i r_i^j \tag{9.6}$$

With N the number of datasets and r_i^j the rank of the algorithm j on dataset i.

Under the null hypothesis, the Friedman test stats that all average rank, R_j, are equal for a level of trust α. The Friedman test is defined as follow [Demšar, 2006],

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{9.7}$$

With k, the number of algorithms.

The Friedman test is distributed as a χ_F^2 with $k - 1$ degrees of freedom when $k > 5$ and $N > 10$ [Demšar, 2006].

9.2.2 Nemenyi test

In the instance where, the null hypothesis is rejected for the Friedman test, we can proceed with a *Nemenyi* test. *Nemenyi* test introduced by P. Nemenyi in [Nemenyi, 1963], states that two algorithms are significantly different, whenever their average ranks R_j differ by at least the *Critical Difference* define as [Courtain, 2017, Demšar, 2006],

$$\mathbf{CD} = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{9.8}$$

With the critical q_α base onto the *Studentized* range statistic divided by $\sqrt{2}$ [Demšar, 2006]. This test compares all the methods based on the investigated datasets.

9.2.3 Wilcoxon signed-rank test

The *Wilcoxon* signed-rank test introduce by F. Wilcoxon in [Wilcoxon, 1992], ranks the difference, d, in performance of two algorithms for the N datasets and compare the ranks for the *positive* and *negative* differences [Demšar, 2006].

Let us define d_i as the difference between the performance of two algorithms on the i'th dataset. The differences are ranked base upon their absolute value. R^+ is the sum of the rank for which the second algorithm outperformed the first and R^- the opposite. If there are ranks equal to zero, they are shared evenly between the sums; in the case there are an odd number of them one is ignored [Demšar, 2006].

$$R^+ = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \quad R^- = \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{9.9}$$

The statics can be defined as [Demšar, 2006],

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{9.10}$$

With T equals to the smallest sum, $T = min(R^+, R^-)$ [Demšar, 2006].

Now that we have a strong protocol to assess the best performing algorithm, we can move onto the next chapter. In the next chapter, we will briefly introduce the datasets and the experiment methodology.

Chapter 10

Experimental methodology

This chapter will describe the experiment conduct to test our algorithm's performance to answer the research questions. First, let's present the datasets onto which we performed our graph partitioning experiment. We will then introduce how *clustering* is performed.

10.1 Datasets

We have 6 different databases. From those 6 databases, we have 17 datasets on which we perform the clustering experiment. See table 10.1.

The dolphin database comes from a 2003 research paper [Lusseau et al., 2003, Lusseau, 2003] about dolphin communities in New Zealand. The researcher found two big communities that can be subdivided into 2 additional communities. This is why there are 2 dolphin datasets.

The football dataset was introduced in [Girvan and Newman, 2002]. It contains the match of 115 American football teams during the season 2000. Each node is a football team, and each edge a match between two teams. Every football team belongs to one of the 12 conferences. Every conference contains 8 to 12 teams. Games between members of the same conference are more frequent. The clustering will aim at finding those 12 conferences.

The LFR datasets are artificially generated by the *Lancichinetti-Fortunato-Radicchi* model [Lancichinetti et al., 2008, Lancichinetti and Fortunato, 2009]. The generated binary graph mimics the property of real world-graphs. We have generated three of these artificial graphs. Each contains 600 vertices. The first one has 3 classes, the others each have 6 classes.

Databases	Datasets	Number of vertices	Number of classes
Dolphin	dolphin_2	62	2
	dolphin_4	62	4
Football	foot	115	12
LFR	LFR_1	600	3
	LFR_2	600	6
	LFR_3	600	6
Newsgroup	news_2cl_1	400	2
	news_2cl_2	400	2
	news_2cl_3	400	2
	news_3cl_1	600	3
	news_3cl_2	600	3
	news_3cl_3	600	3
	news_5cl_1	1000	5
	news_5cl_2	1000	5
	news_5cl_3	1000	5
Political books	polbook	105	3
Zachary	zachary	34	2

Table 10.1: Overview of each dataset, its number of nodes and number of true classes.

The newsgroup database comes from a collection of 20.000 unclassified documents from 20 different newsgroups or topics [Lang, 1995, Yen et al., 2009]. From this database, we extracted 9 different graphs, each containing different topics. Each topic contains 200 documents, nodes. The first 3 graphs have 400 nodes; they all have two different topics, clusters two be uncover. The next 3 graphs have 600 nodes, and therefore 3 topics, communities. Finally, the last 3 have 1000 nodes, 5 topics. The adjacency matrices of those graphs are weighted, with the weight representing the documents' number of shared words.

This dataset is a collection of 105 political books. There are classed in 3 communities, depending on their political orientation: conservative, liberal, and neutral. An edge is placed between two books if they are often sold together on Amazon [Xu et al., 2007]

This dataset represents the relationship of 34 members of an American university karate club in the 1970s. The graph has two clusters, each member belonging to one [Zachary, 1977].

10.2 Algorithm abbreviations

Before moving on to the experiment, let's first warp up all the different attraction/repulsion matrices and algorithms we experiment with. We are using two clustering technique, the *KKM*, and the *ARSC*. We are injecting different attraction/repulsion and kernel matrices to perform the clustering in those two algorithms. We are testing seven similarity measures, and three distance measures are turned into kernels. Moreover, we are carrying out transformations upon those matrices to test if we can yield better performance. In total, we are testing 32 attraction/repulsion matrices injected into two different algorithms. Table 10.2 gives the abbreviation used[1].

Abbreviation	**Meaning**
AR	Attraction Repulsion Soft Clustering
KKM	Kernel k-means
D	Doubly Centered
DN	Doubly Centered Normalized
Mod	Modularity Transformation
AA	Adamic-Adar Similarity
A	Adjacency or direct Similarity
CN	Common Neighbor Similarity
FE	Free Energy Kernel
GFE	Gaussian Free Energy Kernel
JS	Jaccard Similarity
LHNS	Leich-Holms-Newman Similarity
COS	Cosine Similarity
SS	Sørsen Similarity
SCT	Sigmoid Commute Times
LF	Log Forests

Table 10.2: Table of Abbreviations.

In the various tables and graphics, there are abbreviations such as COSMod AR. This refers to the ARSC algorithm injected with the cosine similarity under a modularity transformation.

10.3 Experiments

Two things are going on with the experiment, there is, of course, the clustering, but there are also parameters that need tuning. We are first going over the parameter tuning. We will then look at the clustering itself.

[1]The abbreviation for the ARSC algorithm is shortened to AR because the ARSC abbreviation took too much space on tables and figures.

10.3.1 Parameter tuning

To fine-tune the parameter, we perform the clustering with different parameter initialization. We will only keep the result that yields the highest performance base on the *modularity*[2]. Let see which parameter needs tuning.

First, for the *ARSC*, we have to tune a temperature parameter β. We also have a parameter α for the LF and the SCT kernel. Finally, we have a parameter θ to be tuned for the FE and GFE.

The KKM algorithm does not require any parameter tuning. However, when we are using a kernel, *e.g.* free energy kernel, a parameter,*e.g.* θ, has to be tuned. When experimenting with the GFE kernel in the *ARSC* algorithm, there are two parameters to tune, β & θ. The detail of each value tried for the tuning of the different parameters is available in table 10.3.

	Parameters	Values
ARSC	β	0.001 0.01 0.1 1 10
SCT	α	10 20 30 40 50
LF	α	0.001 0.01 0.1 1 10
FE	θ	0.001 0.01 0.1 1 10

Table 10.3: Algorithm and kernels and their parameters tuning values.

10.3.2 Clustering

The clustering takes place for every parameter. We cluster every time we loop over a parameter value. Once the computer has finished looping over all the parameters, it will only keep the result of the best parameter. The selection of the best parameter is based upon the modularity measure. We keep the results with the highest modularity. Let's look at how the clustering takes place first for the *ARSC* and then for the *KKM*.

Attraction repulsion soft clustering

We start by clustering 30 times in a row, each time with a different initialization of the matrix $\mathbf{U}$, we set a perturbation in the initialization of the matrix. The perturbation follows a *uniform* distribution between -1 and $+1$ times 0.1. Once the algorithm has performed the clustering and returned the matrix $\mathbf{U}$ containing the fuzzy membership, we will have to *crisp* the membership. Crisping the membership

[2]Here we are not talking about the modularity matrix, but the modularity index that gives a flavor of how well a graph is partitioned.

means that we assign each node i a cluster k for which u_{ik} was the largest, *i.e* each node is assigned to its most probable cluster. We have to crisp the fuzzy membership returned by the *ARSC* to compute the quality measures, including the modularity. If we do not crisp the membership, it is impossible to compute those measures.

Out of those 30 different runs, we keep the trial, which has the largest *modularity*. We repeat this process 30 more times for each parameter value. We then compute the average *modularity* and the average of the different quality measures obtained from those 30 trials. The optimal parameter value is the one that hands back the highest average modularity. In the end, we only store the average modularity, NMI, and ARI with their standard deviation of the runs with the best parameter. This is all we need to perform the different tests to determine which algorithm did best. The subsequent table illustrates the different levels of loops.

Clustering with ARSC

Start timer t

- Loop over the parameter $\beta = [0.001, 0.01, 0.1, 1, 10]$:
  1. Repeat 30 times:
     (a) Run 30 times the clustering algorithm and only keep the results of the run that got the highest modularity
  2. Once we have our 30 best run results we can compute the mean of the different efficiency indexes.

Stop timer t

The timer is used to monitor how long it takes for the algorithm to run over one dataset, t_d is the time it takes to run over dataset d. We divide the time t_d it took to run over one dataset by the number of parameters tuned[3], $\frac{t_d}{5}$ (there are 5 parameters). We then sum the average time taken for each dataset, $\sum_{d=1}^{17} \frac{t_d}{5}$.

[3]We divide by the number of parameters tuned as we do not want it to influence the algorithms' time.

Kernel k-means

The same method is applied for the KKM algorithm, except that the membership $\mathbf{U}$ return non-overlapping membership. So, we do not need to crunch the membership. As for the ARSC, we only keep at the end the average modularity, NMI, and ARI with their standard deviation of the runs with the best parameter.

It is now time to move on to the results yield by the experiment. In the next chapter, we will review the result and try answering the questions outline at the beginning of this thesis.

Chapter 11

Results and discussions

In this chapter, we will analyze and sum up the key findings from our experiment. We will start by looking at the results yield from clustering with the ARSC. We will directly look at the statistical test results to find the best version of the ARSC. We will then move on to look at the KKM result. Finally, we will test the best results from the ARSC and the KKM. We will also look at the time the best algorithm took to perform the clustering. At the end of this chapter, we will be able to answer the three questions outlined at the beginning of this thesis,

- Which attraction/repulsion matrix achieves the most competitive results with the *attraction repulsion soft clustering* algorithm?
- Which attraction/repulsion matrix achieves the most competitive results with the *kernel k-means* algorithm?
- Which of the two algorithms, ARSC and KKM, is the most competitive when plugged with attraction/repulsion matrices? And are they competing against well-established algorithms?

As stated before, all tests are performed with a level of confidence of 95%, *i.e* $\alpha \leq 0.05$.

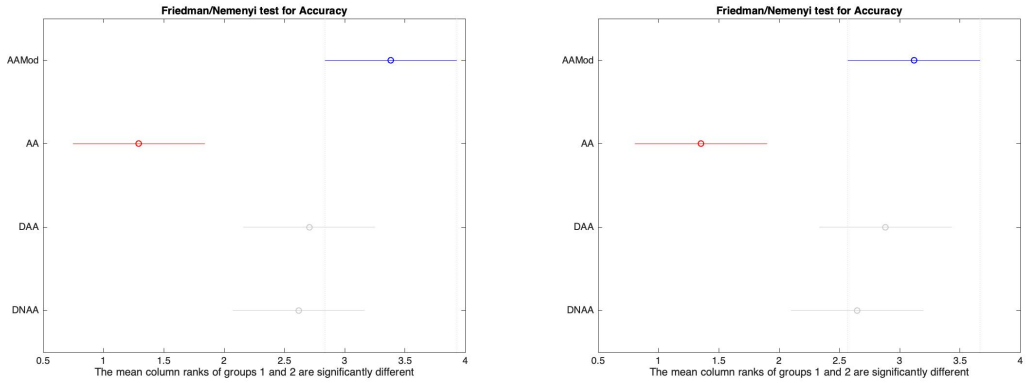
11.1 Attraction repulsion soft clustering results

We will look at the results yield by our variations over the affinity/repulsion matrix for the ARSC algorithm. The average NMI and ARI scores for each algorithm variation on the ARSC are available in Appendix **A**.

We will first compare the results by similarities. We will select the best transformations for all the similarity measures and test them against one another. Once this is done, we will only keep the best ones.

Adamic-Adar Index

For both the ARI and NMI criterion, the p-value of the Friedman test is 0.001. This means that we can reject H0. There is a significant difference in effectiveness between the ARSC injected with different attraction/repulsion matrices. The *Friedman/Nemenyi* tests figure 11.1, return a graphic onto which we can interpret the results. There are two figures because we run the test over the ARI and the NMI criterion. We can see in figure 11.1 that there is a significant difference in performance between the untransformed AA similarity and the rests. Moreover, the Friedman test starts by ranking the algorithm by performance. We can see this ranking on the graph. The best algorithm is placed furthest to the right. *AAMod* is rank as the best performing.



(a) ARI (b) NMI

Figure 11.1: Friedman/Nemenyi tests, over the Adamic-Adar index and its transformations, results for the ARSC algorithm.

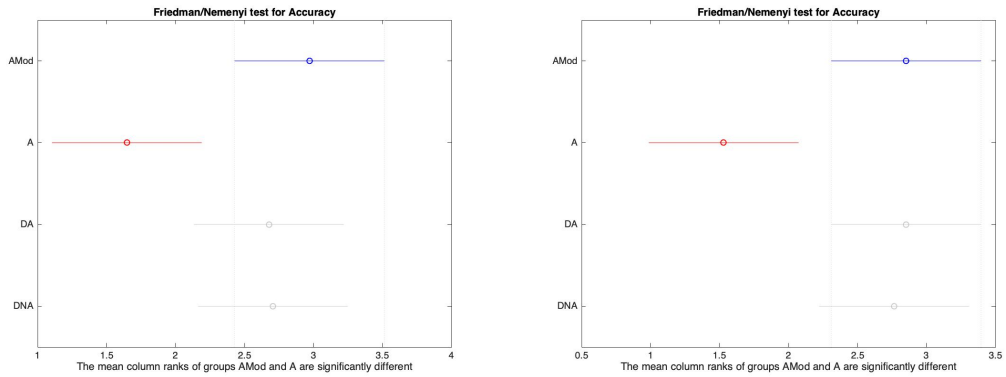
The *Wilcoxon* test returns a *p-value* that tells us if the two algorithms performed significantly differently. We organized the results in tables for easy reading. Looking at the results of the Wilcoxon tests in table 11.1, we see that the AA matrix makes the ARSC algorithm perform significantly differently, for a level $\alpha = 5\%$. From figure 11.1, we see that it performed significantly less well by looking at the ranking. There are no significant differences between the ARSC injected with the other matrices, except for AAMod and DNAA under the ARI criterion. Eventually, for the AA similarities, we keep AAMod as the matrix that made ARSC perform best. Its performance will be tested against the other best attraction/repulsion matrices injected in the ARSC algorithm.

		AAMod	AA	DAA	DNAA
AAMod	ARI	1.0000	**0.0005**	0.0637	**0.0419**
	NMI	1.0000	**0.0007**	0.1205	0.1040
AA	ARI	**0.0005**	1.0000	**0.0007**	**0.0007**
	NMI	**0.0007**	1.0000	**0.0007**	**0.0007**
DAA	ARI	0.0637	**0.0007**	1.0000	0.9658
	NMI	0.1205	**0.0007**	1.0000	0.7002
DNAA	ARI	**0.0419**	**0.0007**	0.9658	1.0000
	NMI	0.1040	**0.0007**	0.7002	1.0000

Table 11.1: Results of the Wilcoxon tests for the Adamic-Adar index matrix and its transformations injected into the ARSC algorithm.

Adjacency matrix

The Friedman test results returned 0.0096 and 0.0028 for the ARI and NMI, respectively. We reject H0; there is an effectiveness disparity between the ARSC injected with the different adjacency based attraction/repulsion matrices. Figure 11.2 shows us that the modularity transformation is ranked first, though it is not significantly better than the others except for the adjacency matrix. Tables detailing every algorithm ARI and NMI score can be found in appendix A.1. Looking at section A.1.2 in the appendix, we see that A performed poorly compared to the rests.



(a) ARI (b) NMI

Figure 11.2: Friedman/Nemenyi tests, over the adjacency matrix and its transformations, results for the ARSC algorithm.

Looking at the Wilcoxon tests results, in table 11.2, we see that the raw Adjacency matrix performed significantly worse than the other transformations. Nevertheless, there is no significant difference between the other matrices. The *modularity* transformation, AMod, is the one selected for the comparison between the other matrices injected in the ARSC algorithms, as it is ranked as the best in the Fried-

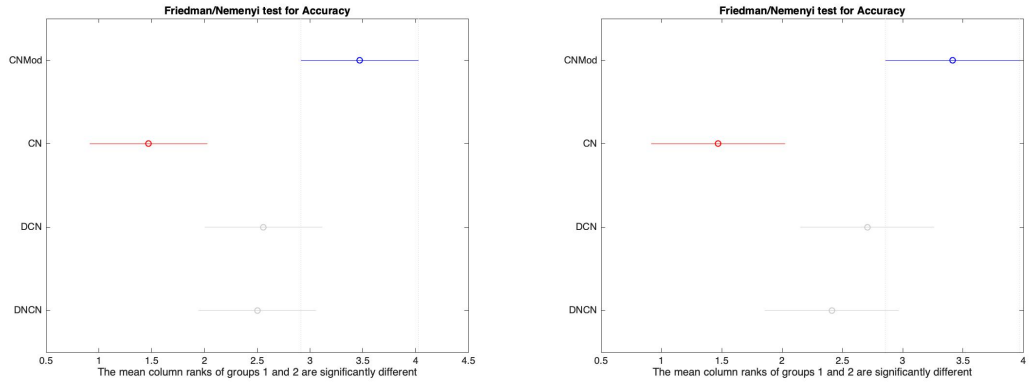
man/Nemenyi test.

		MOD	A	DA	DNA
MOD	ARI	1.0000	**0.0016**	0.3258	0.4543
	NMI	1.0000	**0.0023**	0.9515	0.7615
A	ARI	**0.0016**	1.0000	**0.0019**	**0.0031**
	NMI	**0.0023**	1.0000	**0.0014**	**0.0014**
DA	ARI	0.3258	**0.0019**	1.0000	0.5703
	NMI	0.9515	**0.0014**	1.0000	0.4961
DNA	ARI	0.4543	**0.0031**	0.5703	1.0000
	NMI	0.7615	**0.0014**	0.4961	1.0000

Table 11.2: Results of the Wilcoxon tests for the adjacency matrix and its transformations injected into the ARSC algorithm.

Common neighbors similarity

We reject H0 for the Friedman tests; 0.00008 and 0.0001 are the p-values for the ARI and NMI, respectively. The Friedman/Nemenyi test results, in Figure 11.3, exhibit that the modularity transformation is ranked as the best and is significantly better than the raw common neighbors' similarity matrix.



(a) ARI (b) NMI

Figure 11.3: Friedman/Nemenyi tests, over the common neighbors similarity and its transformations, results for the ARSC algorithm.

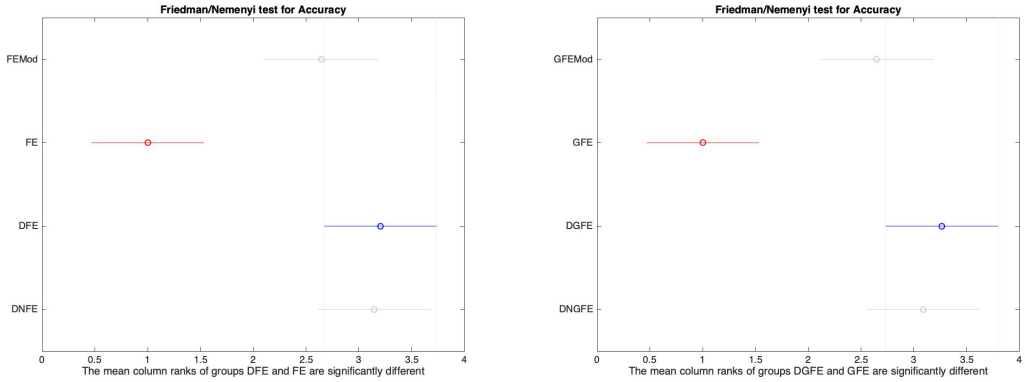
In table 11.3, we see that the modularity transformation is significantly more effective in the ARSC algorithm than any other transformations. CNMod is then selected as the best one to compare against the other matrices injected in the ARSC algorithm.

		CNMOD	CN	DCN	DNCN
CNMOD	ARI	1.0000	**0.0007**	**0.0097**	**0.0174**
	NMI	1.0000	**0.0007**	**0.0200**	**0.0262**
CN	ARI	**0.0007**	1.0000	**0.0008**	**0.0008**
	NMI	**0.0007**	1.0000	**0.0008**	**0.0008**
DCN	ARI	**0.0097**	**0.0008**	1.0000	0.7910
	NMI	**0.0200**	**0.0008**	1.0000	0.5693
DNCN	ARI	**0.0174**	**0.0008**	0.7910	1.0000
	NMI	**0.0262**	**0.0008**	0.5693	1.0000

Table 11.3: Results of the Wilcoxon tests for the common neighbors similarity matrix and its transformations injected into the ARSC algorithm.

Gaussian free energy

For both the ARI and NMI criterion, the p-values of the Friedman tests are of the order of 4×10^{-8}, we can safely assume that there is a significant efficiency gap with at least one algorithm. From figure 11.4, we conclude that the FE matrix makes a significantly less competitive ARSC than the other matrices. Moreover, we see that DFE is rank first. Thus it performs best.



(a) ARI (b) NMI

Figure 11.4: Friedman/Nemenyi tests, over the Gaussian free energy and its transformations, results for the ARSC algorithm.

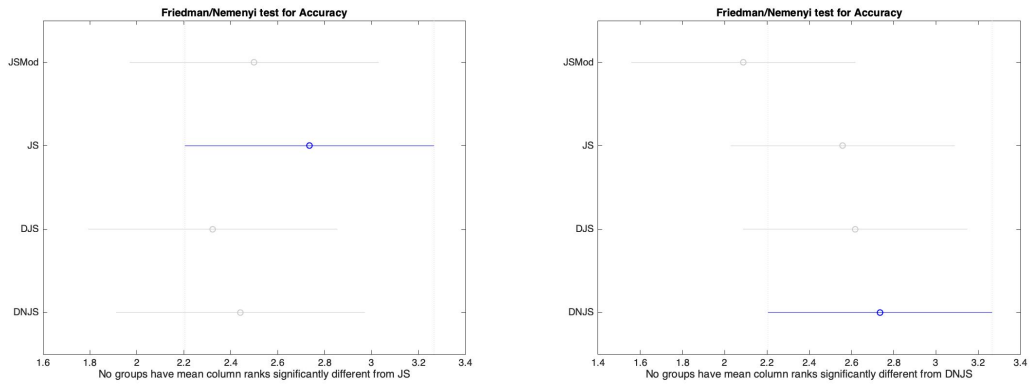
Furthermore, the Wilcoxon tests, in table 11.4, confirm the results as we reject the *null* hypothesis between FE and the other transformations. We also see that the Modularity transformation over the Gaussian kernel is significantly less efficient than the DFE and DNFE. As DFE is rank highest by the Friedman/Nemenyi, it is selected as best and will be compared against the other matrices injected in the ARSC algorithm.

		GFEMod	GFE	DGFE	DNGFE
GFEMod	ARI	1.0000	**0.0003**	**0.0245**	**0.0166**
	NMI	1.0000	**0.0003**	**0.0203**	**0.0107**
GFE	ARI	**0.0003**	1.0000	**0.0003**	**0.0003**
	NMI	**0.0003**	1.0000	**0.0003**	**0.0003**
DGFE	ARI	**0.0245**	**0.0003**	1.0000	1.0000
	NMI	**0.0203**	**0.0003**	1.0000	0.6250
DNGFE	ARI	**0.0166**	**0.0003**	1.0000	1.0000
	NMI	**0.0107**	**0.0003**	0.6250	1.0000

Table 11.4: Results of the Wilcoxon tests for the Gaussian free energy kernel and its transformations injected into the ARSC algorithm.

Jaccard similarity

For the Jaccard similarity, we fail to reject H0 with the Friedman test. The p-values are 0.7882 and 0.4173 for the ARI and NMI, respectively. If we follow the test protocol laid out in chapter 9, we should not perform Nemenyi tests as the Friedman tests tell us that we are not going to find any significant difference between the attraction/repulsion matrices. However, the Nemenyi test is still performed so that we can have figure 11.5. As expected, in figure 11.5, the Friedman/Nemenyi test does not conclude that any of the different Jaccard similarity matrices makes the ARSC algorithm perform differently. Also, the best algorithm changes whether we look at the ARI or the NMI criterion.



(a) ARI (b) NMI

Figure 11.5: Friedman/Nemenyi tests, over the Jaccard similarity and its transformations, results for the ARSC algorithm.

On top of that, the Wilcoxon tests, table 11.5, do not provide us with much more information. The only significant information we get is that under the NMI criterion, JSMod is significantly less effective than DJS with $p = 0.04$. As we don't have

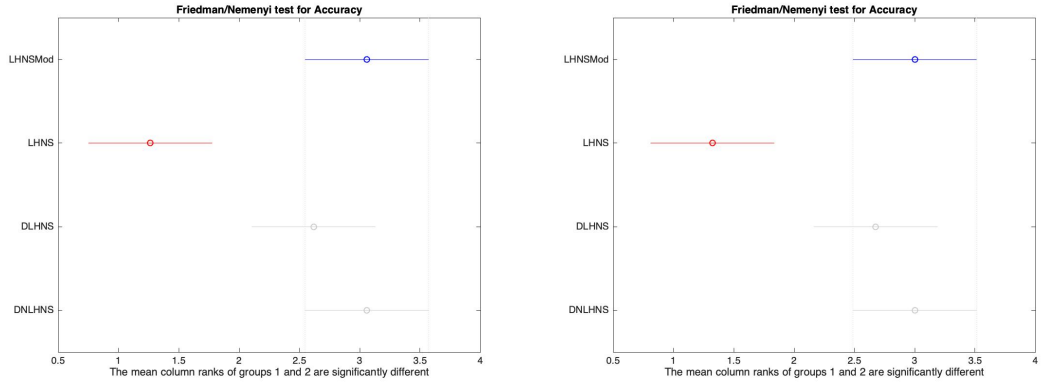
any big differences between the different Jaccard attraction/repulsion matrices, we arbitrarily select JS to be tested against the other attraction/repulsion matrices.

		JSMod	JS	DJS	DNJS
JSMod	ARI	1.0000	0.4691	1.0000	0.5830
	NMI	1.0000	0.4691	**0.0419**	0.1353
JS	ARI	0.4691	1.0000	0.4691	0.7174
	NMI	0.4691	1.0000	0.1627	0.2775
DJS	ARI	1.0000	0.4691	1.0000	1.0000
	NMI	**0.0419**	0.1627	1.0000	1.0000
DNJS	ARI	0.5830	0.7174	1.0000	1.0000
	NMI	0.1205	0.2775	1.0000	1.0000

Table 11.5: Results of the Wilcoxon tests for the Jaccard similarity and its transformations injected into the ARSC algorithm.

Leich-Holmes-Newman similarity

The Friedman tests p-values for the ARI and NMI criterion are 5×10^{-6} and 2×10^{-5}, respectively. Figure 11.6 looks familiar. We once again see the modularity transformation ranked highest, with DNLHN, and performing significantly better than LHNS according to the Friedman/Nemenyi tests.



(a) ARI (b) NMI

Figure 11.6: Friedman/Nemenyi tests, over the Leich-Holmes-Newman similarity and its transformations, results for the ARSC algorithm.

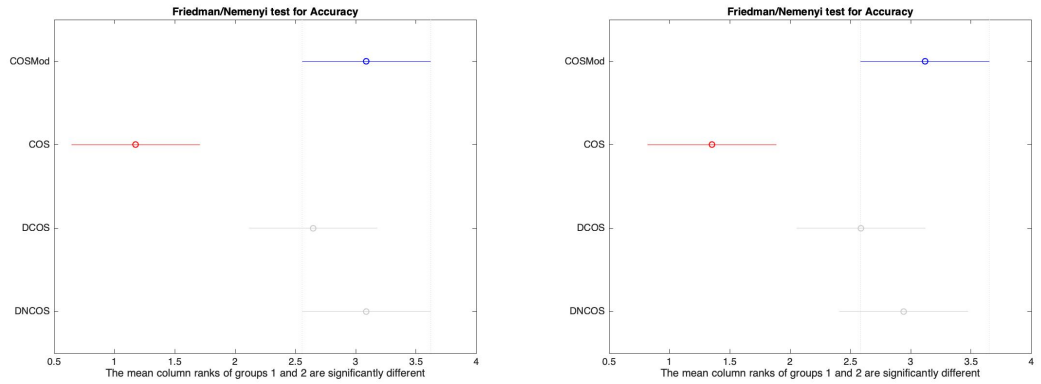
The Wilcoxon tests, table 11.6, confirm that LHNS makes the ARSC algorithm significantly less effective than the other attraction/repulsion matrices. LHNSMod is selected to be compared with the other attraction/repulsion matrices injected in the ARSC algorithm.

		LHNSMod	LHNS	DLHNS	DNLHNS
LHNSMod	ARI	1.0000	**0.0005**	0.0781	0.5693
	NMI	1.0000	**0.0006**	0.0781	0.6772
LLHNS	ARI	**0.0005**	1.0000	**0.0005**	**0.0005**
	NMI	**0.0006**	1.0000	**0.0006**	**0.0008**
DLHNS	ARI	0.0781	**0.0005**	1.0000	**0.0244**
	NMI	0.0781	**0.0006**	1.0000	**0.0322**
DNLHNS	ARI	0.5693	**0.0005**	**0.0244**	1.0000
	NMI	0.6772	**0.0008**	**0.0322**	1.0000

Table 11.6: Results of the Wilcoxon tests for the Leich-Holme-Newman Similarity matrix and its transformations injected into the ARSC algorithm.

Cosine similarity

The Friedman test H0 can be rejected with its p-values, $\alpha = 7 \times 10^{-5}$ for the ARI measure, and $\alpha = 2 \times 10^{-6}$ for the NMI measure. We find the same sight as the previous algorithm, looking at the Friedman/Nemenyi figure 11.7. *COS* makes the ARSC algorithm less effective than the other matrices.



(a) ARI (b) NMI

Figure 11.7: Friedman/Nemenyi tests over the cosine similarity and its transformations results for the ARSC algorithm.

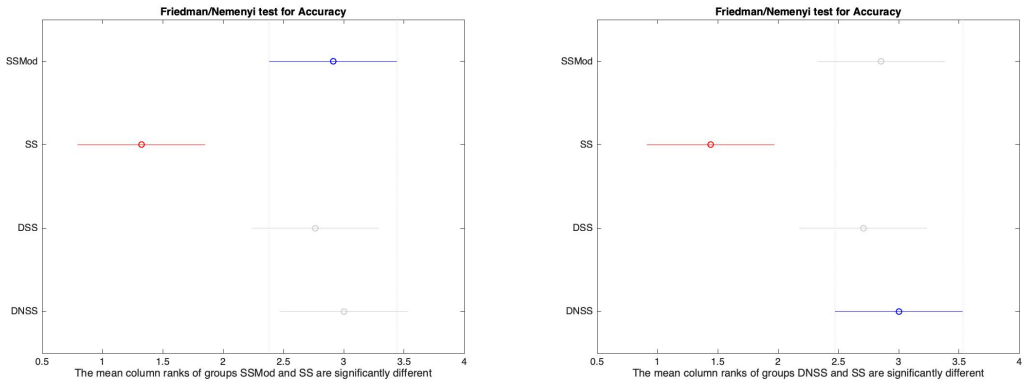
The Wilcoxon tests, table 11.2, confirm the results of the Friedman/Nemenyi, COS makes ARSC perform significantly less well. COSMod is chosen to be compared with the other attraction/repulsion matrices injected in the ARSC algorithm, as it is rank as the most efficient in figure 11.7.

		COSMod	COS	DCOS	DNCOS
COSMod	ARI	1.0000	**0.0005**	0.2166	0.2734
	NMI	1.0000	**0.0007**	0.1726	0.3054
COS	ARI	**0.0005**	1.0000	**0.0005**	**0.0005**
	NMI	**0.0007**	1.0000	**0.0006**	**0.0006**
DCOS	ARI	0.2166	**0.0005**	1.0000	**0.0313**
	NMI	0.1726	**0.0006**	1.0000	**0.0313**
DNCOS	ARI	0.2734	**0.0005**	**0.0313**	1.0000
	NMI	0.3054	**0.0006**	**0.0313**	1.0000

Table 11.7: Results of the Wilcoxon tests for the cosine similarity matrix and its transformations injected into the ARSC algorithm.

Sørsen similarity

For the ARI criterion, the p-value of the Friedman test is 6×10^{-5}, and for the NMI criterion, it is 0.0004. Depending on whether you look at the ARI or NMI results, the Friedman/Nemenyi tests, in table 11.8, rank DNSS or SSMod as the best. There is a significant difference between the performance provided by SS and the other matrices. There is no significant difference between SSMod, DNSS, and DSS.



(a) ARI (b) NMI

Figure 11.8: Friedman/Nemenyi tests over the Sørsen similarity and its transformations results for the ARSC algorithm.

Additionally, the Wilcoxon test, table 11.8, gives us the same results as the Friedman/Nemenyi tests. So, we arbitrarily choose SSMod as the best.

		SSMod	SS	DSS	DNSS
SSMod	ARI	1.0000	**0.0008**	0.3575	0.8077
	NMI	1.0000	**0.0008**	0.7609	0.5016
SS	ARI	**0.0008**	1.0000	**0.0008**	**0.0006**
	NMI	**0.0008**	1.0000	**0.0009**	**0.0008**
DSS	ARI	0.3575	**0.0008**	1.0000	0.2500
	NMI	0.7609	**0.0009**	1.0000	0.3008
DNSS	ARI	0.8077	**0.0006**	0.2500	1.0000
	NMI	0.5016	**0.0008**	0.3008	1.0000

Table 11.8: Results of the Wilcoxon tests for the Sørsen similarity matrix and its transformations injected into the ARSC algorithm.

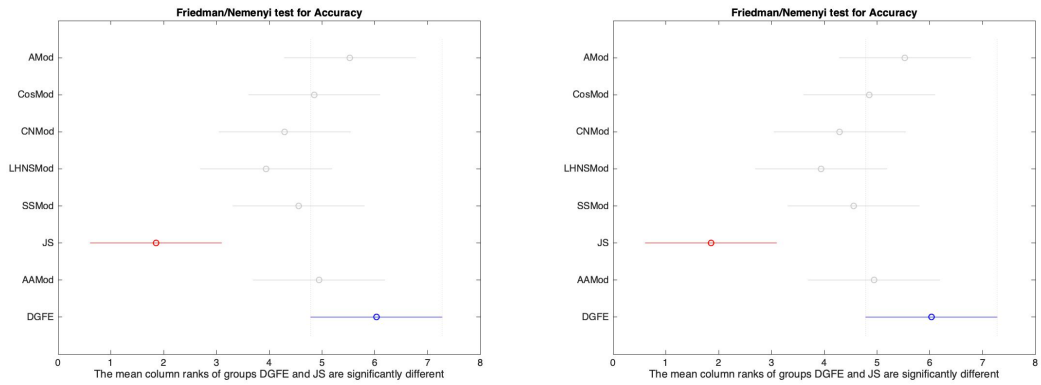
11.1.1 Best matrices for the attraction repulsion soft clustering

Before proceeding to the tests' results between the best attraction/repulsion matrices injected in the ARSC algorithm, we would like to point out some quick observations.

The modularity transformation is the one that yields the best results most of the time. Similarity matrices are not very effective. They always have the worst effectiveness of the lot. This is probably because they are just *attraction* matrices. Centered matrices are almost as effective as centered and normalize matrices. However, centered and normalized matrices are almost always rank higher than a centered matrix. In most cases, it is impossible to tell them apart from the modularity transformation.

Let's begin with the Friedman test results, 3×10^{-5} and 2×10^{-5} are the p-values for the ARI and NMI. We can reject H0. Let's now take a look at figure 11.9, which contains the results for the Friedman/Nemenyi tests on the best performing version of the tested attraction/repulsion matrices.

The first observation we can make is that JS matrix makes the ARSC algorithm perform significantly worse than SMod, AMod, AAMod, and DGFE. It is safe to assume that JS is not a good attraction/repulsion matrix to run ARSC. DGFE performed best, but did it do significantly better than the rest? The Friedman/Nemenyi tests say that it does not.



(a) ARI (b) NMI

Figure 11.9: Friedman/Nemenyi tests over the best attraction/repulsion matrices results for the ARSC algorithm.

Looking at the Wilcoxon tests results, in table 11.9, gives us a sharper image. We can see that every matrix make the ARSC algorithm performs significantly better than the JS matrix. SMod and DGFE performed significantly better than LHNSMod, which was rank second-worst according to the Friedman/Nemenyi tests.

		AMod	COSMod	CNMod	LHNSMod	SSMod	JS	AAMod	DGFE
AMod	ARI	1.0000	0.9811	0.3011	0.2097	0.5228	**0.0007**	0.8926	0.4080
	NMI	1.0000	0.7226	0.2343	0.2274	0.1930	**0.0008**	0.6355	0.6417
COSMod	ARI	0.9811	1.0000	0.2553	**0.0437**	0.1937	**0.0008**	0.5862	0.1961
	NMI	0.7226	1.0000	0.1788	**0.0200**	0.1726	**0.0008**	0.3088	0.1788
CNMod	ARI	0.3011	0.2553	1.0000	0.1788	0.7564	**0.0008**	0.0942	0.1189
	NMI	0.2343	0.1788	1.0000	0.1089	0.7564	**0.0008**	0.1099	0.1353
LHNSMod	ARI	0.2097	**0.0437**	0.1788	1.0000	0.1477	**0.0005**	0.1961	**0.0437**
	NMI	0.2274	**0.0200**	0.1089	1.0000	0.0879	**0.0006**	0.1089	**0.0200**
SSMod	ARI	0.5228	0.1937	0.7564	0.1477	1.0000	**0.0008**	0.9434	0.1961
	NMI	0.1930	0.1726	0.7564	0.0879	1.0000	**0.0008**	0.9434	0.1788
JS	ARI	**0.0007**	**0.0008**	**0.0008**	**0.0005**	**0.0008**	1.0000	**0.0007**	**0.0006**
	NMI	**0.0008**	**0.0008**	**0.0008**	**0.0006**	**0.0008**	1.0000	**0.0008**	**0.0008**
AAMod	ARI	0.8926	0.5862	0.0942	0.1961	0.9434	**0.0007**	1.0000	0.1337
	NMI	0.6355	0.3088	0.1099	0.1089	0.9434	**0.0008**	1.0000	0.1477
DGFE	ARI	0.4080	0.1961	0.1189	**0.0437**	0.1961	**0.0006**	0.1337	1.0000
	NMI	0.6417	0.1788	0.1353	**0.0200**	0.1788	**0.0008**	0.1477	1.0000

Table 11.9: Results of the Wilcoxon tests over the best ARSC algorithms.

COSMod and DGFE are the only two matrices that make the ARSC algorithm significantly better for both JS and LHNSMod.

In table 11.10, we see how much time it takes for the ARSC algorithm, with the

different attraction/repulsion matrices, to perform a loop over each dataset once. We see some large time differences between the algorithms. JS manages to make the ARSC algorithm not very effective and also makes it slower.

Algorithm	Time
AMod	10:29
COSMod	01:32
CNMod	02:10
LHNSMod	12:10
SSMod	02:40
JS	18:17
AAMod	02:55
DFE	02:43

Table 11.10: The time it takes in minutes for the AR algorithm with various attraction/repulsion matrices to run once over every dataset.

The most competitive version of the algorithm uses the DGFE matrix. In second place comes the one using AMod, and in the third spot, the using COSMod and AAMod, they are almost even. Looking at the algorithm's time to run over the 17 databases, we see that AMod takes 3 times more time than the other 3 algorithms. It should be noted that the DGFE requires double tuning and might take more time to run if we do not know the right parameters.

To sum up, we have declared the DGFE, AMod, and COSMod to be the attraction/repulsion matrix that makes the most competitive versions of the ARSC. The ARSC, injected with those attraction/repulsion matrices, will be tested against the KKM, injected with the matrices that yield the best results, and the 3 MDS kernel developed in 6. We did not keep AAMod as an arbitrary choice.

11.2 The kernel k-means results

In this section, to avoid too much redundancy[1], we make a swift analysis of the various similarities and their effectiveness. We will give more detailed analyses for the final comparison between the best results obtained from each similarity measure injected into the KKM algorithm.

[1] Also avoid going over the 80 pages threshold.

Algorithm	ARI	NMI
AA	$\mathbf{3 \times 10^{-7}}$	$\mathbf{3 \times 10^{-7}}$
A	**0.0013**	**0.0013**
COS	0.1807	0.0724
CN	$\mathbf{1 \times 10^{-6}}$	$\mathbf{1 \times 10^{-6}}$
JS	**0.0126**	0.0736
LHNS	0.5630	0.4919
SS	0.0854	0.0903
GFE	$\mathbf{2.5 \times 10^{-6}}$	$\mathbf{6.5 \times 10^{-6}}$

Table 11.11: Friedman tests p-values for the comparison in the effectiveness of each similarity with its transformations.

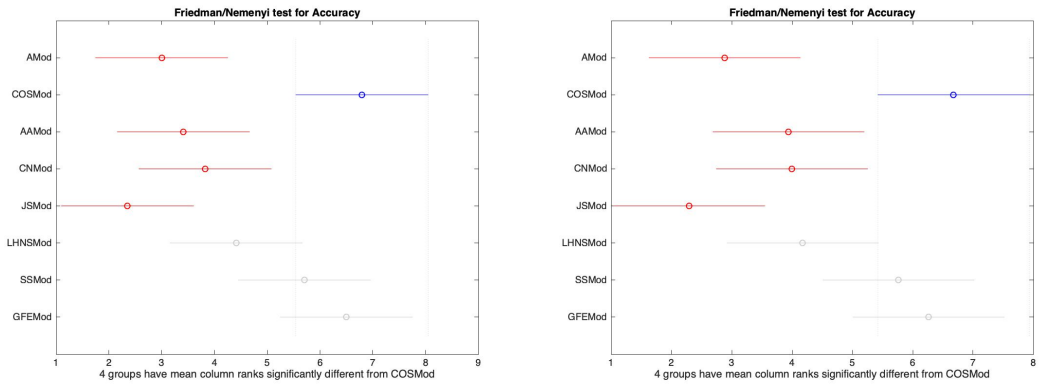
Thanks to table 11.11, we can reject the null hypothesis of the Friedman tests for the KKM algorithms injected with AA, A, CN, and GFE attraction/repulsion matrices. We can also reject H0 for the Jaccard similarity and its transformations but only under the ARI criterion. Looking at the Friedman/Nemenyi figures for those algorithms in Appendix A.3[2], we see that the modularity transformation yield, most of the times, the best results.

Centering and normalized centering did not return outstanding results, and for some algorithms normalizing the centered matrix had absolutely no additional effects. A prime example of that can be found in the Wilcoxon test results for the adjacency A.36, we see that the p-value between DA and DNA for both the ARI and the NMI is equal to 1. In other words, the results are very close.

For measures such as the LHNS, where no versions of the measure were significantly better, we arbitrarily picket the modularity version as it ranked first.

Let us now look analyze the performance of the best attraction/repulsion matrices against one another. The p-values of the Friedman tests strongly indicate that the effectiveness of the KKM algorithm varies depending on the attraction/repulsion matrices injected, $\alpha = 3.5 \times 10^{-8}$ for the NMI and $\alpha = 8.5 \times 10^{-9}$ for the ARI. In figure 11.10, we note that COSMod performed significantly better than the rest, except for SSMod, GFEMod, and LHNSMod. The worst performance came from, once again, the algorithm injected with a Jaccard similarity base matrix.

[2]See p.95



(a) ARI (b) NMI

Figure 11.10: Friedman/Nemenyi tests over the best attraction/repulsion matrices results for KKM algorithm.

Moving on to the Wilcoxon tests in table 11.12, we see that COSMod makes the KKM perform significantly better than all the other attraction/repulsion matrices tested, except for GFEMod. Therefore, COSMod is the only algorithm of the KKM variations compared with the other algorithms.

		AMod	COSMod	AAMod	CNMod	JSMod	LHNSMod	SSMod	GFEMod
AMod	ARI	1.0000	**0.0007**	0.2274	0.1488	**0.0352**	**0.0014**	**0.0006**	**0.0019**
	NMI	1.0000	**0.0006**	0.0552	**0.0395**	**0.0191**	**0.0129**	**0.0007**	**0.0016**
COSMod	ARI	**0.0007**	1.0000	**0.0004**	**0.0008**	**0.0004**	**0.0008**	**0.0032**	0.0787
	NMI	**0.0006**	1.0000	**0.0016**	**0.0016**	**0.0004**	**0.0007**	**0.0038**	0.1961
AAMod	ARI	0.2274	**0.0004**	1.0000	0.7197	**0.0097**	**0.0245**	**0.0005**	**0.0032**
	NMI	0.0552	**0.0016**	1.0000	0.9341	**0.0045**	0.2097	**0.0019**	**0.0072**
CNMod	ARI	0.1488	**0.0008**	0.7197	1.0000	**0.0084**	**0.0684**	**0.0008**	**0.0061**
	NMI	**0.0395**	**0.0016**	0.9341	1.0000	**0.0052**	**0.2461**	**0.0019**	**0.0061**
JSMod	ARI	**0.0352**	**0.0004**	**0.0097**	**0.0084**	1.0000	**0.0168**	**0.0001**	**0.0023**
	NMI	**0.0191**	**0.0004**	**0.0045**	**0.0052**	1.0000	**0.0148**	**0.0001**	**0.0019**
LHNSMod	ARI	**0.0014**	**0.0008**	**0.0245**	0.0684	**0.0168**	1.0000	0.1773	**0.0042**
	NMI	**0.0129**	**0.0007**	0.2097	0.2461	**0.0148**	1.0000	**0.0352**	**0.0031**
SSMod	ARI	**0.0006**	**0.0032**	**0.0005**	**0.0008**	**0.0001**	0.1773	1.0000	0.0557
	NMI	**0.0007**	**0.0038**	**0.0019**	**0.0019**	**0.0001**	**0.0352**	1.0000	0.0787
GFEMod	ARI	**0.0019**	0.0787	**0.0032**	**0.0061**	**0.0023**	**0.0042**	0.0557	1.0000
	NMI	**0.0016**	0.1961	**0.0072**	**0.0061**	**0.0019**	**0.0031**	0.0787	1.0000

Table 11.12: Results of the Wilcoxon tests over the best of the KKM algorithm.

When looking at the computation time, in table 11.13, we see that the KKM algorithm runs more or less at the same speed regardless of the attraction/repulsion matrix injected. On top of that, the KKM is running faster than most of its ARSC counterparts.

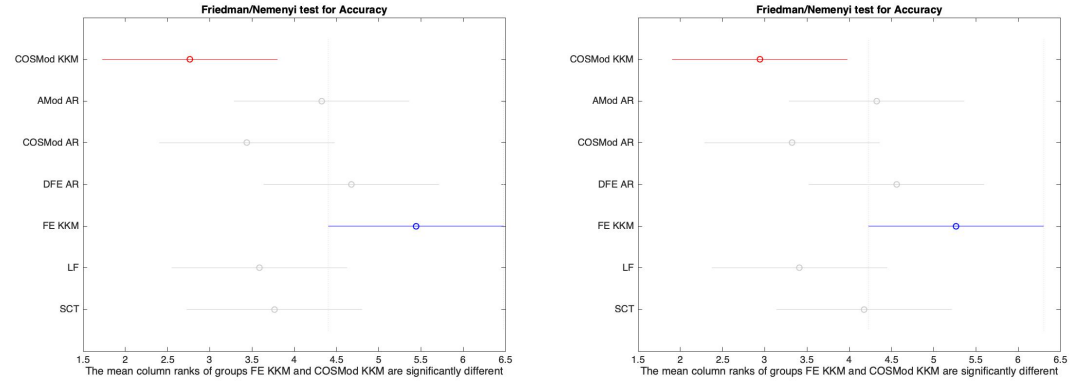
Algorithm	Time
AMod	01:08
COSMod	01:11
AAMod	01:35
CNMod	01:09
JSMod	00:56
LHNSMod	01:06
SSMod	01:09
GFE	01:03

Table 11.13: The time it takes in minutes for the KKM algorithm with various attraction/repulsion matrices to run once over every dataset.

11.3 Final results

Now that we have analyzed and selected the best attraction/repulsion matrices to inject into the ARSC and the KKM, we will test them against one another and establish kernels for the KKM algorithm.

The Friedman tests p-values are 0.0042 and 0.0137 for the ARI and NMI, respectively. We can reject the null hypothesis. The effectiveness of the algorithms varies depending on the attraction/repulsion matrices injected. The Friedman/Nemenyi test results, available in figure 11.11, shows us that the mean column rank of COSMod KKM is significantly different than FE KKM for the NMI and ARI index. In the first position, comes FE KKM, in second DFE AR, and AMod AR takes the third spot.



(a) ARI (b) NMI

Figure 11.11: Friedman/Nemenyi test over the final selection results.

The Wilcoxon test results show us that COSMod KKM performed significantly less well compared with FE KKM, with a strong p-value of 0.0002 for the ARI criterion. It is also significantly less effective than SCT KKM under the ARI measure and LF

KKM under both criteria. Finally, FE KKM is significantly better than AMod AR under the NMI index.

		COSMod KKM	AMod AR	COSMod AR	DFE AR	FE KKM	SCT KKM	LF KKM
COSMod KKM	ARI	1.0000	0.3591	0.0787	0.1205	**0.0002**	**0.0245**	**0.0203**
	NMI	1.0000	0.4212	0.1961	0.1514	**0.0031**	0.2412	**0.0134**
AMod AR	ARI	0.3591	1.0000	0.9811	0.4080	**0.0302**	0.7615	0.7615
	NMI	0.4212	1.0000	0.7226	0.6417	0.0730	0.5614	0.9780
COSMod AR	ARI	0.0787	0.9811	1.0000	0.1961	0.0787	0.6791	0.6791
	NMI	0.1961	0.7226	1.0000	0.1788	0.0557	0.5014	0.2343
DFE AR	ARI	0.1205	0.4080	0.1961	1.0000	0.4543	0.3894	0.5830
	NMI	0.1514	0.6417	0.1788	1.0000	0.2769	0.4212	1.0000

Table 11.14: Wilcoxon two-by-two test between final selection of algorithms.

The ARSC is definitely a little bit slower when looking at table 11.15. AMod AR is definitely the slowest to run.

Algorithm	Time
COSMod KKM	01:11
DFE AR	02:43
AMod AR	10:29
COSMod AR	01:32
FE KKM	01:04
LF KKM	01:01
SCT KM	01:13

Table 11.15: The time it takes in minutes for the algorithms injected with various attrac-tion/repulsion matrices to run once over every dataset.

To sum up, none of our attraction/repulsion matrices managed to make the ARSC or the KKM algorithms more effective than the KKM injected with well-established kernels, especially FE KKM. Injecting the various attraction/repulsion matrices into the KKM algorithm proved very ineffective, except for COSMod KKM. This algorithm might be slightly less efficient but interesting because it is straightforward to implement and does not require any tuning.

Regarding AMod AR, COSMod AR, and GFE AR, they all perform very well and can hardly be told apart from one another. However, GFE AR takes a lot of time to run if we need to fine-tune its *two* parameters, and AMod AR is quite slow.

Now that we have analyzed the results, we will sum up our findings and answer the questions at the beginning of this thesis in the next chapter.

Chapter 12

Conclusion

We are now at the end of this thesis; we will now explicitly answer the questions presented at the beginning. We will then look at some limitations and possibilities for future works.

12.1 Findings

Let's start by answer the first question,

- Which attraction/repulsion matrix achieves the most competitive results with the *attraction repulsion soft clustering* algorithm?

The first question has no simple answer. In the previous chapter, we have selected three matrices that make the ARSC algorithm work best. We cannot tell one apart with a significant difference from the other two. So, the answer to this first question is that *DFE*, *AMod*, and *COSMod* are the matrices that make the ARSC algorithm more efficient.

We have seen that the best transformation is the modularity transformation. It yields the best results almost every time, except for the Jaccard similarity and the Gaussian free energy. One additional note on the transformations is that a raw similarity matrix does not yield good results.

- Which attraction/repulsion matrix achieves the most competitive results with the *kernel k-means* algorithm?

For the *second* question, we have two winners, they are the cosine similarity, with a modularity transformation, and the Gaussian free energy kernel, also with a modularity transformation.

- Which of the two algorithms, ARSC and KKM, is the most competitive when plugged with attraction/repulsion matrices? And are they competing against well-established algorithms?

The ARSC algorithm is more competitive when injected with and attraction/repulsion matrix than the KKM algorithm. Unfortunately, we cannot say that the ARSC is more competitive than the KKM algorithm injected with a well-established kernel. However, we can say that it is competitive against it. Our COSMod AR and AMod AR yield results that are very close to it. Furthermore, the ARSC algorithm, unlike the KKM algorithm, can *scale* on a big graph.

DGFE AR gave us our best results with the ARSC algorithm. However, it did not significantly outperform the other best.

The COSMod KKM did not perform significantly as well as the FE KKM, but it might be interesting to use as it runs very fast and does not perform too badly. It still competes with the LF KKM and the SCT KKM.

12.2 Limitations

We are now going to present the limitations this thesis faced. So, here is a non-exhaustive list of the limitations and possibilities of improvements for this research:

- We tested our algorithms onto a very limited number of datasets. In total, we had only 17 different datasets, just enough to make acceptable statistical tests. The variety of the datasets is another issue; 9 of our datasets are from the Newsgroup datasets. The 17 datasets all represent social network datasets. Finally, all our datasets are relatively small, as the biggest one only contains 1000 nodes. To overcome those limitations, the solution would be to find new datasets to run the algorithm. However, graph and network datasets are hard to come by.

- We only tested the Gaussian free energy kernel; we did not test the Gaussian kernels using the commute time distance and the logarithm forest distance. This could be a future track for exploration; it might yield a more efficient ARSC.

- The number of parameters used to fine-tune the algorithms are limited. A bigger range of parameter tuning might lead to better results.

- We compared the algorithm using only two measures, *e.g.* NMI and ARI. We could have also compared the algorithm based on the soft modularity measures or other fancy modularity such as the *adaptive scale modularity* [Chakraborty et al., 2017]. The number of statistical tests could also be expanded.
- We only used local similarity measures. We could extend to global similarity measures that exploit the global structure of an undirected graph like the *Katz index* [Fouss et al., 2016].

12.3 Future works

Every research generates more questions than answers. Here, we will still don't know how well the ARSC algorithm would scale on a big graph. We suggest testing the ARSC onto a large graph in future research. Another direction for future research would be to compare how good a fuzzy algorithm the ARSC algorithm is. We could test it against fuzzy algorithms, *e.g. fuzzy c-means* [Li and Mukaidono, 1999, Miyamoto et al., 2008], on a fuzzy network. Finally, we could try different modularities to select our algorithm's best run than the Newman modularity.

12.4 Skills

Deciding to write my thesis on the clustering algorithm's subject came from a curiosity that emerged after the Cambridge analytic scandal [Carole and Emma, 2018]. If algorithms could have such a huge influence on our lives, I wanted to know-how. Now that I have written this thesis, I am happy to have a deeper knowledge of the subject. Although network clustering is a very complex field, and I only scratched the surface.

Getting into network theory was very tedious as the learning curve is quite steep. I had to re-learn a couple of mathematical principles, especially in matrix algebra, and I had to learn many things specific to the subject. I also learned to use Matlab; it was easy to learn as I already knew python and RStudio. I made mistakes in managing my time and focus, but I have learned to manage myself better in the end.

I also greatly improved my communication skill as I learned to put the complex topic into words. Finally, I have learned first-hand the scientific procedure through the on-going research of this thesis.

Appendix A

Appendix

A.1 Appendix A: Attraction repulsion soft clustering algorithm effectiveness

A.1.1 Adamic-Adar index scores[1]

Table A.1: ARI scores of AA-AR algorithm variations.

	AAMod	AA	DAA	DNAA
dolphin_2	0.5928	0.7890	0.7480	0.7480
dolphin_4	0.8239	0.3667	0.8239	0.8239
football	0.6111	0.2486	0.6147	0.5747
LFR1	0.9900	0.0000	0.1856	0.1066
LFR2	1.0000	0.1643	1.0000	1.0000
LFR3	1.0000	0.1058	0.9566	0.9719
news_2cl_1	0.9023	0.0521	0.8928	0.8928
news_2cl_2	0.7551	0.0498	0.7728	0.7728
news_2cl_3	0.8736	0.0806	0.7754	0.7961
news_3cl_1	0.8316	0.0891	0.7607	0.7562
news_3cl_2	0.8012	0.0401	0.6554	0.6517
news_3cl_3	0.7234	0.0575	0.7201	0.7518
news_5cl_1	0.7040	0.0537	0.4873	0.4891
news_5cl_2	0.5675	0.0474	0.4180	0.5017
news_5cl_3	0.5197	0.0319	0.5163	0.5100
polbook	0.6517	0.6461	0.5919	0.5919
zachary	0.9725	0.9556	0.9882	0.9725

[1]We can see in both tables that the ARI and NMI scores are equal to 0 for the AA matrix over the LFR1 dataset. This is because the ARSC algorithm is degenerating. It is not finding the right amount of cluster.

Table A.2: NMI scores of AA-AR algorithm variations.

	AAMod	AA	DAA	DNAA
dolphin_2	0.5785	0.7386	0.7040	0.7040
dolphin_4	0.8308	0.5054	0.8308	0.8308
football	0.8110	0.5901	0.8112	0.7974
LFR1	0.9809	0.0000	0.3117	0.2162
LFR2	1.0000	0.3296	1.0000	1.0000
LFR3	1.0000	0.2065	0.9652	0.9743
news_2cl_1	0.8313	0.0519	0.8332	0.8332
news_2cl_2	0.6612	0.0536	0.6759	0.6759
news_2cl_3	0.8101	0.0762	0.7206	0.7409
news_3cl_1	0.7707	0.1015	0.7377	0.7344
news_3cl_2	0.7392	0.0558	0.6676	0.6654
news_3cl_3	0.6554	0.0731	0.6783	0.6993
news_5cl_1	0.6663	0.0964	0.5981	0.5941
news_5cl_2	0.6103	0.0790	0.5489	0.5627
news_5cl_3	0.5595	0.0671	0.5572	0.5513
polbook	0.5540	0.5778	0.5237	0.5237
zachary	0.9620	0.9549	0.9837	0.9620

A.1.2 Adjacency matrix scores

Table A.3: ARI scores of A-AR algorithm variations.

	AMod	A	DA	DNA
dolphin_2	0.5434	0.7095	0.5264	0.4510
dolphin_4	0.7728	0.5394	0.7861	0.7861
football	0.6421	0.4044	0.6416	0.6430
LFR1	0.9950	0.2642	0.9950	0.9950
LFR2	1.0000	0.4379	1.0000	1.0000
LFR3	1.0000	0.4399	0.9878	0.9878
news_2cl_1	0.8928	0.8000	0.9023	0.9023
news_2cl_2	0.7551	0.5222	0.7551	0.7769
news_2cl_3	0.8840	0.8261	0.8181	0.8181
news_3cl_1	0.8099	0.2654	0.8243	0.8195
news_3cl_2	0.8147	0.3451	0.7912	0.7944
news_3cl_3	0.8477	0.3448	0.8374	0.8208
news_5cl_1	0.7037	0.0490	0.6765	0.7247
news_5cl_2	0.5760	0.0648	0.5607	0.5420
news_5cl_3	0.5842	0.0632	0.5867	0.5551
polbook	0.6517	0.6802	0.6496	0.6496
zachary	0.8823	0.9924	0.9843	0.9843

Table A.4: NMI scores of A-AR algorithm variations;

	AMod	A	DA	DNA
dolphin_2	0.5450	0.6753	0.5341	0.4861
dolphin_4	0.8180	0.6150	0.8079	0.8079
football	0.8300	0.7260	0.8301	0.8311
LFR1	0.9904	0.3086	0.9904	0.9904
LFR2	1.0000	0.5917	1.0000	1.0000
LFR3	1.0000	0.5787	0.9846	0.9846
news_2cl_1	0.8191	0.7346	0.8320	0.8320
news_2cl_2	0.6612	0.4518	0.6612	0.6822
news_2cl_3	0.8234	0.7395	0.7691	0.7691
news_3cl_1	0.7521	0.3404	0.7781	0.7717
news_3cl_2	0.7568	0.4242	0.7500	0.7530
news_3cl_3	0.7849	0.3820	0.7874	0.7696
news_5cl_1	0.6830	0.1368	0.6783	0.7052
news_5cl_2	0.6223	0.1474	0.6209	0.5994
news_5cl_3	0.5928	0.1436	0.5959	0.5695
polbook	0.5540	0.5785	0.5757	0.5757
zachary	0.8372	0.9910	0.9783	0.9783

A.1.3 Common neighbors scores[2]

Table A.5: ARI scores of CN-AR algorithm variations.

	AMod	A	DA	DNA
dolphin_2	0.5928	0.8134	0.7228	0.7228
dolphin_4	0.8239	0.3869	0.7861	0.7436
football	0.5807	0.2975	0.5822	0.5750
LFR1	0.9868	0.0000	0.4725	0.1357
LFR2	1.0000	0.2662	1.0000	1.0000
LFR3	0.9978	0.0122	0.9529	0.9469
news_2cl_1	0.8928	0.0441	0.8553	0.8553
news_2cl_2	0.7551	0.0121	0.7713	0.7713
news_2cl_3	0.8736	0.1329	0.7127	0.7949
news_3cl_1	0.8316	0.1160	0.7106	0.7470
news_3cl_2	0.8054	0.0681	0.7182	0.6361
news_3cl_3	0.7156	0.0866	0.7032	0.7395
news_5cl_1	0.6830	0.0804	0.4912	0.4763
news_5cl_2	0.5687	0.0625	0.3781	0.4927
news_5cl_3	0.5046	0.0436	0.4973	0.5006
polbook	0.6389	0.6601	0.5644	0.5290
zachary	1.0000	0.9924	0.9922	0.9922

[2]We can see in both tables that the ARI and NMI scores are equal to 0 for the CN matrix over the LFR1 dataset. This is because the ARSC algorithm is degenerating. It is not finding the right amount of cluster.

Table A.6: NMI scores of CN-AR algorithm variations.

	CNMod	CN	DCN	DNCN
dolphin_2	0.5785	0.7597	0.6875	0.6875
dolphin_4	0.8308	0.5458	0.8079	0.7741
football	0.7984	0.6023	0.8002	0.7974
LFR1	0.9749	0.0000	0.5114	0.2412
LFR2	1.0000	0.4754	1.0000	1.0000
LFR3	0.9982	0.0203	0.9611	0.9575
news_2cl_1	0.8185	0.0564	0.7925	0.7925
news_2cl_2	0.6582	0.0179	0.6741	0.6741
news_2cl_3	0.8119	0.1525	0.6769	0.7393
news_3cl_1	0.7707	0.1520	0.7083	0.7273
news_3cl_2	0.7436	0.1206	0.6934	0.6552
news_3cl_3	0.6491	0.1355	0.6587	0.6835
news_5cl_1	0.6556	0.1630	0.6038	0.5934
news_5cl_2	0.6098	0.1257	0.5292	0.5574
news_5cl_3	0.5552	0.1013	0.5478	0.5469
polbook	0.5405	0.5782	0.4992	0.4753
zachary	1.0000	0.9910	0.9891	0.9891

A.1.4 Gaussian free energy kernel scores[3]

Table A.7: ARI scores of GFE-AR algorithm variations.

	GFEMod	GFE	DGFE	DNGFE
dolphin_2	0.7971	0.6834	0.7955	0.7955
dolphin_4	0.4624	0.0581	0.4763	0.4763
football	0.3872	0.0077	0.3879	0.3943
LFR1	0.9900	0.0000	0.9900	0.9900
LFR2	1.0000	0.0000	1.0000	1.0000
LFR3	1.0000	0.0011	0.9953	0.9953
news_2cl_1	0.8928	0.0462	0.8928	0.8928
news_2cl_2	0.7728	0.0261	0.7639	0.7639
news_2cl_3	0.8837	0.0862	0.8830	0.8830
news_3cl_1	0.8279	0.0109	0.8546	0.8543
news_3cl_2	0.8118	0.0181	0.8290	0.8290
news_3cl_3	0.8221	0.0348	0.8408	0.8408
news_5cl_1	0.6927	0.0324	0.7182	0.7127
news_5cl_2	0.5702	0.0154	0.5725	0.5735
news_5cl_3	0.6041	0.0125	0.6077	0.6074
polbook	0.6843	0.1207	0.7028	0.7028
zachary	0.9608	0.2976	1.0000	1.0000

[3]We can see in both tables that the ARI and NMI scores are equal to 0 for the GFE matrix over the LFR1 and the LFR2 datasets. This is because the ARSC algorithm is degenerating. It is not finding the right amount of cluster.

Table A.8: NMI scores of GFE-AR algorithm variations;

	GFEMod	GFE	DGFE	DNGFE
dolphin_2	0.7417	0.6510	0.7406	0.7406
dolphin_4	0.5521	0.0819	0.5805	0.5805
football	0.6932	0.0285	0.6936	0.7000
LFR1	0.9830	0.0000	0.9830	0.9830
LFR2	1.0000	0.0000	1.0000	1.0000
LFR3	1.0000	0.0065	0.9942	0.9942
news_2cl_1	0.8185	0.0504	0.8185	0.8185
news_2cl_2	0.6785	0.0296	0.6669	0.6669
news_2cl_3	0.8089	0.0972	0.8083	0.8083
news_3cl_1	0.7593	0.0162	0.7949	0.7949
news_3cl_2	0.7514	0.0263	0.7669	0.7669
news_3cl_3	0.7607	0.0452	0.7760	0.7760
news_5cl_1	0.6632	0.0525	0.6880	0.6842
news_5cl_2	0.6180	0.0296	0.6216	0.6216
news_5cl_3	0.6042	0.0251	0.6223	0.6219
polbook	0.5755	0.1124	0.5937	0.5937
zachary	0.9457	0.3231	1.0000	1.0000

A.1.5 Jaccard similarity scores

Table A.9: ARI scores of JS-AR algorithm variations.

	JSMod	JS	DJS	DNJS
dolphin_2	0.5928	0.8285	0.7378	0.7378
dolphin_4	0.7861	0.4505	0.7861	0.7861
football	0.5919	0.1669	0.5891	0.5950
LFR1	0.9457	0.0237	0.9603	0.9603
LFR2	0.9960	0.2278	0.9960	0.9960
LFR3	0.9948	0.2300	0.9907	0.9907
news_2cl_1	0.0222	0.1635	0.0404	0.0069
news_2cl_2	0.3277	0.1373	0.3539	0.1324
news_2cl_3	0.0358	0.1387	0.1676	0.2163
news_3cl_1	0.0776	0.1510	0.0665	0.0787
news_3cl_2	0.1182	0.2666	0.0438	0.1327
news_3cl_3	0.1049	0.2016	0.0581	0.0652
news_5cl_1	0.0551	0.0497	0.0479	0.0329
news_5cl_2	0.0086	0.1356	0.0090	0.0136
news_5cl_3	0.0386	0.1336	0.0365	0.0167
polbook	0.6465	0.6467	0.6109	0.6109
zachary	1.0000	1.0000	1.0000	1.0000

Table A.10: NMI scores of JS-AR algorithm variations.

	JSMod	JS	DJS	DNJS
dolphin_2	0.5785	0.7724	0.6951	0.6951
dolphin_4	0.8079	0.5661	0.8079	0.8079
football	0.8042	0.5103	0.8024	0.8049
LFR1	0.9084	0.0411	0.9286	0.9286
LFR2	0.9948	0.3743	0.9948	0.9948
LFR3	0.9941	0.3123	0.9897	0.9897
news_2cl_1	0.0196	0.1292	0.1125	0.0593
news_2cl_2	0.2674	0.1095	0.2939	0.1116
news_2cl_3	0.0391	0.1356	0.2456	0.2654
news_3cl_1	0.1253	0.1756	0.1393	0.1424
news_3cl_2	0.1207	0.2555	0.0838	0.1663
news_3cl_3	0.1021	0.2074	0.1133	0.1232
news_5cl_1	0.1180	0.0853	0.1739	0.1711
news_5cl_2	0.0333	0.1877	0.0589	0.0749
news_5cl_3	0.0869	0.1763	0.1043	0.0563
polbook	0.5590	0.5629	0.5378	0.5378
zachary	1.0000	1.0000	1.0000	1.0000

A.1.6 Leicht-Holme-Newman similarity scores

Table A.11: ARI scores of LHNS-AR algorithm variations.

	LHNSMod	LHNS	DLHNS	DNLHNS
dolphin_2	0.8117	0.8216	0.8117	0.8117
dolphin_4	0.8252	0.4785	0.7243	0.7257
football	0.4117	0.2110	0.4117	0.5502
LFR1	0.9264	0.0893	0.6949	0.7319
LFR2	0.9920	0.4159	0.9920	0.9920
LFR3	0.9890	0.1820	0.9522	0.9517
news_2cl_1	0.9023	0.5500	0.9023	0.9023
news_2cl_2	0.7205	0.4190	0.7205	0.7205
news_2cl_3	0.8563	0.7141	0.8534	0.8534
news_3cl_1	0.8369	0.3349	0.8369	0.8230
news_3cl_2	0.7890	0.3049	0.7890	0.8108
news_3cl_3	0.7489	0.3518	0.7489	0.7706
news_5cl_1	0.4997	0.0577	0.5005	0.7150
news_5cl_2	0.4151	0.0535	0.4148	0.5622
news_5cl_3	0.4323	0.0690	0.4320	0.5099
polbook	0.6762	0.6372	0.6762	0.6567
zachary	1.0000	1.0000	1.0000	1.0000

Table A.12: NMI scores of LHNS-AR algorithm variations.

	LHNSMod	LHNS	DLHNS	DNLHNS
dolphin_2	0.7534	0.7688	0.7534	0.7534
dolphin_4	0.8312	0.5611	0.7245	0.7260
football	0.6784	0.5370	0.6784	0.7891
LFR1	0.8813	0.1288	0.6469	0.6822
LFR2	0.9896	0.5927	0.9896	0.9896
LFR3	0.9905	0.2909	0.9486	0.9457
news_2cl_1	0.8313	0.5058	0.8313	0.8313
news_2cl_2	0.6152	0.3695	0.6152	0.6152
news_2cl_3	0.7720	0.6589	0.7683	0.7683
news_3cl_1	0.7701	0.3795	0.7701	0.7573
news_3cl_2	0.7224	0.3586	0.7224	0.7477
news_3cl_3	0.6844	0.3870	0.6844	0.7091
news_5cl_1	0.5263	0.1517	0.5273	0.6810
news_5cl_2	0.4951	0.1234	0.4941	0.5896
news_5cl_3	0.4744	0.1556	0.4745	0.5534
polbook	0.5675	0.5566	0.5675	0.5534
zachary	1.0000	1.0000	1.0000	1.0000

A.1.7 Cosine similarity scores

Table A.13: ARI scores of COS-AR algorithm variations.

	COSMod	COS	DCOS	DNCOS
dolphin_2	0.7493	0.8236	0.7541	0.7541
dolphin_4	0.8207	0.3884	0.8207	0.8207
football	0.5681	0.1276	0.5658	0.5790
LFR1	0.9506	0.0167	0.9800	0.9800
LFR2	0.9960	0.2112	0.9960	0.9960
LFR3	0.9948	0.2816	0.9916	0.9948
news_2cl_1	0.8833	0.3901	0.8928	0.8928
news_2cl_2	0.7906	0.5028	0.7212	0.7212
news_2cl_3	0.8549	0.5493	0.8364	0.8364
news_3cl_1	0.8455	0.0996	0.8412	0.8412
news_3cl_2	0.8198	0.0860	0.7921	0.7921
news_3cl_3	0.8226	0.1857	0.7737	0.7738
news_5cl_1	0.6996	0.0582	0.6585	0.6627
news_5cl_2	0.5802	0.0268	0.5474	0.5488
news_5cl_3	0.5583	0.0357	0.5646	0.5664
polbook	0.6335	0.6321	0.6465	0.6465
zachary	1.0000	0.9922	1.0000	1.0000

Table A.14: NMI scores of COS-AR algorithm variations.

	COSMod	COS	DCOS	DNCOS
dolphin_2	0.7051	0.7705	0.7082	0.7082
dolphin_4	0.8496	0.5394	0.8496	0.8496
football	0.7954	0.4417	0.7962	0.8020
LFR1	0.9180	0.0354	0.9618	0.9618
LFR2	0.9948	0.3474	0.9948	0.9948
LFR3	0.9941	0.4647	0.9923	0.9941
news_2cl_1	0.8062	0.4121	0.8197	0.8197
news_2cl_2	0.6944	0.4664	0.6167	0.6167
news_2cl_3	0.7747	0.5198	0.7544	0.7544
news_3cl_1	0.7849	0.1588	0.7813	0.7813
news_3cl_2	0.7601	0.1315	0.7295	0.7295
news_3cl_3	0.7569	0.2227	0.7205	0.7208
news_5cl_1	0.6689	0.1068	0.6491	0.6522
news_5cl_2	0.6219	0.0648	0.5994	0.6008
news_5cl_3	0.5916	0.0910	0.5864	0.5885
polbook	0.5423	0.5616	0.5590	0.5590
zachary	1.0000	0.9891	1.0000	1.0000

A.1.8 Sørsen similarity scores

Table A.15: ARI scores of SS-AR algorithm variations.

	SSMod	SS	DSS	DNSS
dolphin_2	0.7363	0.8194	0.5928	0.7573
dolphin_4	0.7861	0.3962	0.7861	0.7861
football	0.5681	0.1270	0.5658	0.5807
LFR1	0.9457	0.0285	0.9554	0.9554
LFR2	0.9960	0.2058	0.9960	0.9960
LFR3	0.9948	0.2682	0.9642	0.9642
news_2cl_1	0.9026	0.2941	0.8833	0.8833
news_2cl_2	0.7188	0.3427	0.7639	0.7551
news_2cl_3	0.8830	0.7821	0.8469	0.8460
news_3cl_1	0.8270	0.0451	0.8178	0.8362
news_3cl_2	0.8102	0.0253	0.7959	0.7959
news_3cl_3	0.7776	0.0804	0.7959	0.8007
news_5cl_1	0.7191	0.0552	0.6680	0.6803
news_5cl_2	0.5725	0.0403	0.5522	0.5442
news_5cl_3	0.5438	0.0269	0.5674	0.5672
polbook	0.6214	0.6435	0.6465	0.6465
zachary	1.0000	1.0000	1.0000	1.0000

Table A.16: NMI scores of SS-AR algorithm variations.

	SSMod	SS	DSS	DNSS
dolphin_2	0.6944	0.7662	0.5785	0.7104
dolphin_4	0.8079	0.5380	0.8079	0.8079
football	0.7971	0.4403	0.7962	0.8027
LFR1	0.9084	0.0432	0.9212	0.9212
LFR2	0.9948	0.4099	0.9948	0.9948
LFR3	0.9941	0.4366	0.9739	0.9739
news_2cl_1	0.8318	0.3267	0.8062	0.8062
news_2cl_2	0.6142	0.3308	0.6669	0.6582
news_2cl_3	0.8223	0.7069	0.7899	0.7882
news_3cl_1	0.7657	0.0727	0.7704	0.7889
news_3cl_2	0.7505	0.0345	0.7450	0.7450
news_3cl_3	0.7044	0.1143	0.7350	0.7398
news_5cl_1	0.6825	0.0971	0.6678	0.6761
news_5cl_2	0.6092	0.0683	0.5999	0.5928
news_5cl_3	0.5774	0.0552	0.5905	0.5931
polbook	0.5293	0.5671	0.5590	0.5590
zachary	1.0000	1.0000	1.0000	1.0000

A.2 Appendix B: Kernel k-means algorithm effectiveness

A.2.1 Adamic-Adar index scores

Table A.17: ARI scores of AA-KKM algorithm variations.

	AA	DAA	DNAA	AAMod
dolphin_2	0.6135	0.6232	0.6232	0.6471
dolphin_4	0.3520	0.3561	0.3561	0.4495
football	0.8434	0.8367	0.8366	0.8350
LFR1	0.1507	0.1507	0.1507	0.9301
LFR2	0.7203	0.7203	0.7203	0.7957
LFR3	0.6954	0.6954	0.6954	0.8459
news_2cl_1	0.6447	0.6447	0.6447	0.8526
news_2cl_2	0.4633	0.4633	0.4633	0.6732
news_2cl_3	0.4655	0.4655	0.4655	0.8642
news_3cl_1	0.3274	0.3274	0.3274	0.3715
news_3cl_2	0.2802	0.2802	0.2802	0.3928
news_3cl_3	0.3619	0.3619	0.3619	0.4428
news_5cl_1	0.2300	0.2300	0.2300	0.2382
news_5cl_2	0.1588	0.1588	0.1588	0.2179
news_5cl_3	0.2685	0.2685	0.2685	0.3244
polbook	0.6043	0.6043	0.6043	0.6217
zachary	1.0000	1.0000	1.0000	1.0000

Table A.18: ARI scores of AA-KKM algorithm variations.

	AA	DAA	DNAA	AAMod
dolphin_2	0.5825	0.5891	0.5891	0.6777
dolphin_4	0.5253	0.5266	0.5266	0.7170
football	0.9046	0.9026	0.9019	0.9003
LFR1	0.1896	0.1896	0.1896	0.9400
LFR2	0.8651	0.8651	0.8651	1.0000
LFR3	0.8111	0.8111	0.8111	1.0000
news_2cl_1	0.6321	0.6321	0.6321	0.8250
news_2cl_2	0.4741	0.4741	0.4741	0.6012
news_2cl_3	0.4932	0.4932	0.4932	0.8320
news_3cl_1	0.4705	0.4705	0.4705	0.7277
news_3cl_2	0.4459	0.4459	0.4459	0.7067
news_3cl_3	0.5072	0.5072	0.5072	0.6771
news_5cl_1	0.4500	0.4500	0.4500	0.5608
news_5cl_2	0.3758	0.3758	0.3758	0.5514
news_5cl_3	0.4304	0.4304	0.4304	0.5231
polbook	0.5224	0.5224	0.5224	0.5471
zachary	1.0000	1.0000	1.0000	1.0000

A.2.2 Adjacency matrix scores

Table A.19: ARI scores of A-KKM algorithm variations.

	AA	DAA	DNAA	AAMod
dolphin_2	0.5385	0.5763	0.5763	0.5763
dolphin_4	0.5995	0.4660	0.4660	0.5193
football	0.8773	0.8469	0.8469	0.8475
LFR1	0.3342	0.5028	0.5028	0.8106
LFR2	0.9827	0.6744	0.6744	0.8167
LFR3	0.1238	0.4453	0.4453	0.6599
news_2cl_1	0.8627	0.6012	0.6012	0.6665
news_2cl_2	0.7542	0.4299	0.4299	0.5462
news_2cl_3	0.7226	0.7639	0.7639	0.9087
news_3cl_1	0.5601	0.4445	0.4445	0.5345
news_3cl_2	0.5944	0.3676	0.3676	0.4060
news_3cl_3	0.4878	0.3827	0.3827	0.4217
news_5cl_1	0.2668	0.2230	0.2230	0.2644
news_5cl_2	0.2033	0.1840	0.1840	0.2177
news_5cl_3	0.3052	0.2258	0.2258	0.2467
polbook	0.5897	0.4993	0.4993	0.5372
zachary	0.9882	1.0000	1.0000	0.9924

Table A.20: NMI scores of A-KKM algorithm variations.

	A	DA	DNA	AMod
dolphin_2	0.5408	0.5748	0.5748	0.5523
dolphin_4	0.6967	0.5879	0.5879	0.6299
football	0.9178	0.9070	0.9070	0.9075
LFR1	0.4677	0.6248	0.6248	0.8608
LFR2	0.9915	0.8307	0.8307	0.9124
LFR3	0.4151	0.6555	0.6555	0.7871
news_2cl_1	0.7876	0.5730	0.5730	0.6209
news_2cl_2	0.6555	0.4240	0.4240	0.4864
news_2cl_3	0.6857	0.7201	0.7201	0.8521
news_3cl_1	0.6050	0.5372	0.5372	0.5953
news_3cl_2	0.6312	0.4881	0.4881	0.5176
news_3cl_3	0.5859	0.5043	0.5043	0.5212
news_5cl_1	0.4787	0.4460	0.4460	0.4589
news_5cl_2	0.4240	0.3784	0.3784	0.3987
news_5cl_3	0.4494	0.3930	0.3930	0.4108
polbook	0.5377	0.4650	0.4650	0.5009
zachary	0.9837	1.0000	1.0000	0.9910

A.2.3 Common neighbors scores

Table A.21: ARI scores of CN-KKM algorithm variations.

	CN	DCN	DNCN	CNMod
dolphin_2	0.6344	0.6152	0.6360	0.6101
dolphin_4	0.4074	0.4403	0.4444	0.4916
football	0.8229	0.8234	0.8262	0.8236
LFR1	0.3360	0.3910	0.4222	0.9703
LFR2	0.8126	0.8143	0.8154	0.8636
LFR3	0.7918	0.7985	0.7985	0.8864
news_2cl_1	0.5712	0.5712	0.5712	0.8572
news_2cl_2	0.4577	0.4577	0.4577	0.6695
news_2cl_3	0.4434	0.4434	0.4434	0.8642
news_3cl_1	0.3451	0.3451	0.3451	0.3591
news_3cl_2	0.2916	0.2916	0.2916	0.3966
news_3cl_3	0.3615	0.3615	0.3615	0.4329
news_5cl_1	0.2193	0.2193	0.2193	0.2307
news_5cl_2	0.1523	0.1523	0.1523	0.2105
news_5cl_3	0.2580	0.2580	0.2580	0.3265
polbook	0.5330	0.5136	0.5331	0.6259
zachary	1.0000	1.0000	1.0000	1.0000

Table A.22: NMI scores of CN-KKM algorithm variations.

	CN	DCN	DNCN	CNMod
dolphin_2	0.6214	0.5942	0.6112	0.5502
dolphin_4	0.5507	0.5783	0.5815	0.6077
football	0.8969	0.8952	0.8965	0.8944
LFR1	0.3946	0.4485	0.4764	0.9624
LFR2	0.9103	0.9108	0.9117	0.9394
LFR3	0.8683	0.8682	0.8682	0.9262
news_2cl_1	0.5676	0.5676	0.5676	0.7908
news_2cl_2	0.4698	0.4698	0.4698	0.5916
news_2cl_3	0.4802	0.4802	0.4802	0.8146
news_3cl_1	0.4850	0.4850	0.4850	0.5024
news_3cl_2	0.4522	0.4522	0.4522	0.5307
news_3cl_3	0.5073	0.5073	0.5073	0.5517
news_5cl_1	0.4413	0.4413	0.4413	0.4735
news_5cl_2	0.3730	0.3730	0.3730	0.4221
news_5cl_3	0.4191	0.4191	0.4191	0.4605
polbook	0.4825	0.4712	0.4788	0.5423
zachary	1.0000	1.0000	1.0000	1.0000

A.2.4 Jaccard similarity scores[4]

Table A.23: ARI scores of JS-KKM algorithm variations.

	JS	DJS	DNJS	JSMod
dolphin_2	0.6966	0.7214	0.7277	0.6511
dolphin_4	0.5383	0.5252	0.5225	0.4422
football	0.7845	0.7987	0.7839	0.8000
LFR1	0.9466	0.9501	0.9501	0.9443
LFR2	0.9969	0.9973	0.9979	0.9987
LFR3	0.9992	0.9997	0.9997	0.9998
news_2cl_1	0.0018	0.0026	0.0026	0.0055
news_2cl_2	0.0057	0.0043	0.0043	0.0437
news_2cl_3	0.0010	0.0011	0.0011	0.0082
news_3cl_1	0.0019	0.0042	0.0042	0.0330
news_3cl_2	0.0015	0.0017	0.0017	0.0022
news_3cl_3	0.0015	0.0012	0.0012	0.0357
news_5cl_1	0.0005	0.0005	0.0005	0.0011
news_5cl_2	0.0004	0.0003	0.0003	0.0080
news_5cl_3	0.0002	0.0002	0.0002	0.0004
polbook	0.6178	0.5956	0.6004	0.6107
zachary	1.0000	1.0000	1.0000	1.0000

[4]The Jaccard similarity and its transformations perform very poorly with the weighted graph, *i.e.* the newsgroup database. We need further investigations to understand why it performs so poorly.

Table A.24: NMI scores of JS-KKM algorithm variations.

	JS	DJS	DNJS	JSMod
dolphin_2	0.6400	0.6799	0.6805	0.6338
dolphin_4	0.6009	0.5888	0.5844	0.5551
football	0.8757	0.8813	0.8743	0.8793
LFR1	0.9102	0.9156	0.9156	0.9118
LFR2	0.9960	0.9965	0.9972	0.9965
LFR3	0.9990	0.9996	0.9996	0.9996
news_2cl_1	0.0425	0.0412	0.0412	0.0360
news_2cl_2	0.0680	0.0660	0.0660	0.0265
news_2cl_3	0.0443	0.0452	0.0452	0.0493
news_3cl_1	0.0684	0.0803	0.0803	0.0480
news_3cl_2	0.0530	0.0485	0.0485	0.0350
news_3cl_3	0.0597	0.0581	0.0581	0.0678
news_5cl_1	0.0637	0.0629	0.0629	0.0640
news_5cl_2	0.0544	0.0537	0.0537	0.0460
news_5cl_3	0.0462	0.0454	0.0454	0.0607
polbook	0.5393	0.5338	0.5354	0.5279
zachary	1.0000	1.0000	1.0000	0.9946

A.2.5 Leicht-Holme-Newman similarity scores

Table A.25: ARI scores of LHNS-KKM algorithm variations.

	LHNS	DLHNS	DNLHNS	LHNSMod
dolphin_2	0.6914	0.7279	0.6906	0.7101
dolphin_4	0.4871	0.4863	0.4768	0.4913
football	0.8368	0.8309	0.8391	0.8521
LFR1	0.6173	0.6515	0.6648	0.8676
LFR2	0.8860	0.9034	0.9034	0.8998
LFR3	0.7258	0.7436	0.7436	0.8192
news_2cl_1	0.8818	0.8818	0.8818	0.8792
news_2cl_2	0.6622	0.6622	0.6622	0.6543
news_2cl_3	0.8541	0.8541	0.8541	0.8547
news_3cl_1	0.6844	0.6844	0.6844	0.6845
news_3cl_2	0.6525	0.6525	0.6525	0.6519
news_3cl_3	0.6646	0.6646	0.6646	0.6560
news_5cl_1	0.4450	0.4450	0.4450	0.4451
news_5cl_2	0.4083	0.4083	0.4083	0.4039
news_5cl_3	0.4518	0.4518	0.4518	0.4538
polbook	0.6219	0.6189	0.6127	0.6204
zachary	0.8783	0.9585	0.9585	0.9403

Table A.26: NMI scores of LHNS-KKM algorithm variations.

	LHNS	DLHNS	DNLHNS	LHNSMod
dolphin_2	0.6313	0.6708	0.6335	0.6475
dolphin_4	0.5908	0.5943	0.5857	0.5993
football	0.8997	0.8990	0.9006	0.9055
LFR1	0.6338	0.6563	0.6668	0.8185
LFR2	0.9398	0.9482	0.9482	0.9481
LFR3	0.8144	0.8226	0.8226	0.8617
news_2cl_1	0.8044	0.8044	0.8044	0.8012
news_2cl_2	0.5659	0.5659	0.5659	0.5608
news_2cl_3	0.7766	0.7766	0.7766	0.7769
news_3cl_1	0.6829	0.6829	0.6829	0.6832
news_3cl_2	0.6556	0.6556	0.6556	0.6546
news_3cl_3	0.6421	0.6421	0.6421	0.6389
news_5cl_1	0.5675	0.5675	0.5675	0.5672
news_5cl_2	0.5453	0.5453	0.5453	0.5443
news_5cl_3	0.5454	0.5454	0.5454	0.5476
polbook	0.5382	0.5354	0.5331	0.5284
zachary	0.8774	0.9599	0.9599	0.9300

A.2.6 Cosine similarity scores

Table A.27: ARI scores of COS-KKM algorithm variations.

	COS	DCOS	DNCOS	COSMod
dolphin_2	0.7252	0.7369	0.7383	0.7322
dolphin_4	0.6568	0.6793	0.6873	0.6811
football	0.8351	0.8419	0.8460	0.8394
LFR1	0.9792	0.9791	0.9789	0.9669
LFR2	1.0000	0.9999	0.9999	1.0000
LFR3	1.0000	1.0000	1.0000	1.0000
news_2cl_1	0.8833	0.8830	0.8830	0.8994
news_2cl_2	0.7587	0.7587	0.7587	0.7349
news_2cl_3	0.8469	0.8469	0.8469	0.8680
news_3cl_1	0.7974	0.7974	0.7974	0.8176
news_3cl_2	0.7052	0.7052	0.7052	0.7841
news_3cl_3	0.6763	0.6761	0.6761	0.7742
news_5cl_1	0.5658	0.5658	0.5658	0.5614
news_5cl_2	0.4267	0.4267	0.4267	0.4687
news_5cl_3	0.4153	0.4153	0.4153	0.4393
polbook	0.5973	0.6042	0.6103	0.6248
zachary	1.0000	1.0000	1.0000	1.0000

Table A.28: NMI scores of S-KKM algorithm variations.

	COS	DCOS	DNCOS	COSMod
dolphin_2	0.6807	0.6921	0.6977	0.6777
dolphin_4	0.7023	0.7166	0.7232	0.7170
football	0.8988	0.9004	0.9027	0.9003
LFR1	0.9604	0.9602	0.9600	0.9400
LFR2	1.0000	0.9998	0.9998	1.0000
LFR3	1.0000	1.0000	1.0000	1.0000
news_2cl_1	0.8074	0.8070	0.8070	0.8275
news_2cl_2	0.6617	0.6617	0.6617	0.6423
news_2cl_3	0.7804	0.7804	0.7804	0.8060
news_3cl_1	0.7580	0.7580	0.7580	0.7688
news_3cl_2	0.6900	0.6900	0.6900	0.7370
news_3cl_3	0.6672	0.6670	0.6670	0.7261
news_5cl_1	0.6250	0.6250	0.6250	0.6297
news_5cl_2	0.5487	0.5487	0.5487	0.5933
news_5cl_3	0.4956	0.4956	0.4956	0.5194
polbook	0.5348	0.5359	0.5441	0.5471
zachary	1.0000	1.0000	1.0000	1.0000

A.2.7 Sørsen similarity scores

Table A.29: ARI scores of SS-KKM algorithm variations.

	SS	DSS	DNSS	SSMod
dolphin_2	0.7088	0.6904	0.7140	0.6748
dolphin_4	0.6647	0.6223	0.6283	0.5702
football	0.8377	0.8466	0.8372	0.8422
LFR1	0.9677	0.9670	0.9660	0.9556
LFR2	0.9976	0.9979	0.9980	0.9996
LFR3	0.9997	1.0000	0.9997	0.9998
news_2cl_1	0.8609	0.8609	0.8609	0.8862
news_2cl_2	0.6715	0.6715	0.6715	0.6955
news_2cl_3	0.8184	0.8184	0.8184	0.9001
news_3cl_1	0.6334	0.6334	0.6334	0.6911
news_3cl_2	0.6217	0.6217	0.6217	0.7083
news_3cl_3	0.5684	0.5684	0.5684	0.6047
news_5cl_1	0.3938	0.3938	0.3938	0.3866
news_5cl_2	0.3368	0.3368	0.3368	0.3561
news_5cl_3	0.3907	0.3907	0.3907	0.4105
polbook	0.6152	0.5861	0.5935	0.6181
zachary	1.0000	1.0000	1.0000	1.0000

Table A.30: NMI scores of SS-KKM algorithm variations.

	SS	DSS	DNSS	SSMod
dolphin_2	0.6674	0.6423	0.6632	0.6062
dolphin_4	0.7064	0.6812	0.6855	0.6376
football	0.9001	0.9047	0.8999	0.9016
LFR1	0.9414	0.9403	0.9386	0.9234
LFR2	0.9969	0.9972	0.9974	0.9995
LFR3	0.9996	1.0000	0.9996	0.9998
news_2cl_1	0.7972	0.7972	0.7972	0.8141
news_2cl_2	0.5912	0.5912	0.5912	0.6090
news_2cl_3	0.7694	0.7694	0.7694	0.8418
news_3cl_1	0.6641	0.6641	0.6641	0.6934
news_3cl_2	0.6496	0.6496	0.6496	0.6975
news_3cl_3	0.6234	0.6234	0.6234	0.6438
news_5cl_1	0.5537	0.5537	0.5537	0.5427
news_5cl_2	0.5225	0.5225	0.5225	0.5307
news_5cl_3	0.4970	0.4970	0.4970	0.5118
polbook	0.5451	0.5284	0.5281	0.5417
zachary	1.0000	1.0000	1.0000	1.0000

A.2.8 Gaussian free energy scores[5]

Table A.31: NMI scores of GFE and its variations injected into the KKM algorithm.

	GFEMod	GFE	DGFE	DNGFE
dolphin_2	0.7493	0.8236	0.7541	0.7541
dolphin_4	0.8207	0.3884	0.8207	0.8207
football	0.5681	0.1276	0.5658	0.5790
LFR1	0.9506	0.0167	0.9800	0.9800
LFR2	0.9960	0.2112	0.9960	0.9960
LFR3	0.9948	0.2816	0.9916	0.9948
news_2cl_1	0.8833	0.3901	0.8928	0.8928
news_2cl_2	0.7906	0.5028	0.7212	0.7212
news_2cl_3	0.8549	0.5493	0.8364	0.8364
news_3cl_1	0.8455	0.0996	0.8412	0.8412
news_3cl_2	0.8198	0.0860	0.7921	0.7921
news_3cl_3	0.8226	0.1857	0.7737	0.7738
news_5cl_1	0.6996	0.0582	0.6585	0.6627
news_5cl_2	0.5802	0.0268	0.5474	0.5488
news_5cl_3	0.5583	0.0357	0.5646	0.5664
polbook	0.6335	0.6321	0.6465	0.6465
zachary	1.0000	0.9922	1.0000	1.0000

[5]The GFE kernel is performing surprisingly poorly. This behavior is inconsistent with the findings of M. Saerens and Courtain S.; we need further investigations to understand why.

Table A.32: NMI scores of GFE and its variations injected in the KKM algorithm.

	GFEMod	GFE	DGFE	DNGFE
dolphin_2	0.7051	0.7705	0.7082	0.7082
dolphin_4	0.8496	0.5394	0.8496	0.8496
football	0.7954	0.4417	0.7962	0.8020
LFR1	0.9180	0.0354	0.9618	0.9618
LFR2	0.9948	0.3474	0.9948	0.9948
LFR3	0.9941	0.4647	0.9923	0.9941
news_2cl_1	0.8062	0.4121	0.8197	0.8197
news_2cl_2	0.6944	0.4664	0.6167	0.6167
news_2cl_3	0.7747	0.5198	0.7544	0.7544
news_3cl_1	0.7849	0.1588	0.7813	0.7813
news_3cl_2	0.7601	0.1315	0.7295	0.7295
news_3cl_3	0.7569	0.2227	0.7205	0.7208
news_5cl_1	0.6689	0.1068	0.6491	0.6522
news_5cl_2	0.6219	0.0648	0.5994	0.6008
news_5cl_3	0.5916	0.0910	0.5864	0.5885
polbook	0.5423	0.5616	0.5590	0.5590
zachary	1.0000	0.9891	1.0000	1.0000

A.2.9 Kernels scores

Table A.33: ARI scores of LF, SCT and FE kernels injected in the KKM algorithm.

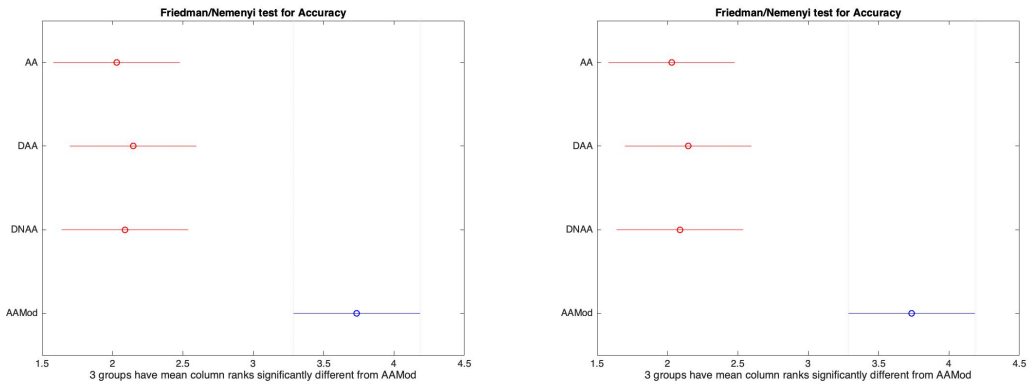
	LF	SCT	FE
dolphin_2	0.8855	0.8600	0.8600
dolphin_4	0.8008	0.7755	0.7840
football	0.8583	0.7435	0.8611
LFR1	0.9900	0.9933	0.9988
LFR2	1.0000	1.0000	1.0000
LFR3	1.0000	1.0000	1.0000
news_2cl_1	0.8887	0.8928	0.8950
news_2cl_2	0.7395	0.7369	0.7498
news_2cl_3	0.8370	0.8793	0.8833
news_3cl_1	0.8147	0.8098	0.8292
news_3cl_2	0.8027	0.8059	0.8287
news_3cl_3	0.7686	0.8285	0.8345
news_5cl_1	0.6263	0.6764	0.7097
news_5cl_2	0.5553	0.5455	0.5761
news_5cl_3	0.5763	0.5703	0.6304
polbook	0.6507	0.6686	0.6460
zachary	1.0000	1.0000	1.0000

Table A.34: NMI scores of LF, SCT and FE kernels injected in the KKM algorithm.

	LF	SCT	FE
dolphin_2	0.8339	0.8021	0.8021
dolphin_4	0.8310	0.8078	0.8163
football	0.9103	0.8561	0.9116
LFR1	0.9809	0.9875	0.9978
LFR2	1.0000	1.0000	1.0000
LFR3	1.0000	1.0000	1.0000
news_2cl_1	0.8129	0.8185	0.8215
news_2cl_2	0.6489	0.6471	0.6584
news_2cl_3	0.7491	0.8267	0.8108
news_3cl_1	0.7547	0.7572	0.7636
news_3cl_2	0.7426	0.7490	0.7705
news_3cl_3	0.6976	0.7686	0.7681
news_5cl_1	0.6464	0.6840	0.6866
news_5cl_2	0.6065	0.6168	0.6463
news_5cl_3	0.5824	0.5787	0.6229
polbook	0.5484	0.5629	0.5462
zachary	1.0000	1.0000	1.0000

A.3 Appendix C: Friedman/Nemenyi & Wilcoxon results over the kernel k-means

A.3.1 Adamic-Adar index



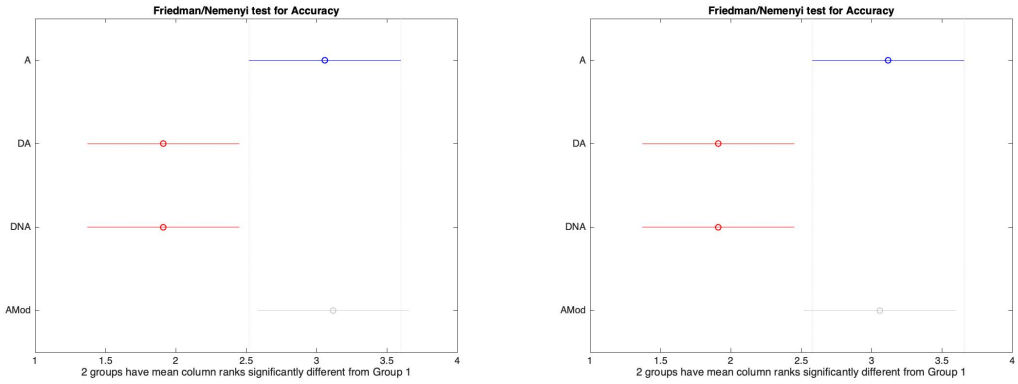
(a) ARI (b) NMI

Figure A.1: Friedman/Nemenyi test over the Adamic-Adar similarity results for the KKM algorithm.

		AA	DAA	DNAA	AAMod
AA	ARI	1.0000	0.7500	0.7500	**0.0006**
	NMI	1.0000	0.7500	0.7500	**0.0005**
DAA	ARI	0.7500	1.0000	1.0000	**0.0005**
	NMI	0.7500	1.0000	1.0000	**0.0005**
DNAA	ARI	0.7500	1.0000	1.0000	**0.0005**
	NMI	0.7500	1.0000	1.0000	**0.0005**
AAMod	ARI	**0.0006**	**0.0005**	**0.0005**	1.0000
	NMI	**0.0005**	**0.0005**	**0.0005**	1.0000

Table A.35: Results of the Wilcoxon tests for the Adamic-Adar index matrix and its transformations injected into the KKM algorithm.

A.3.2 Adjacency matrix



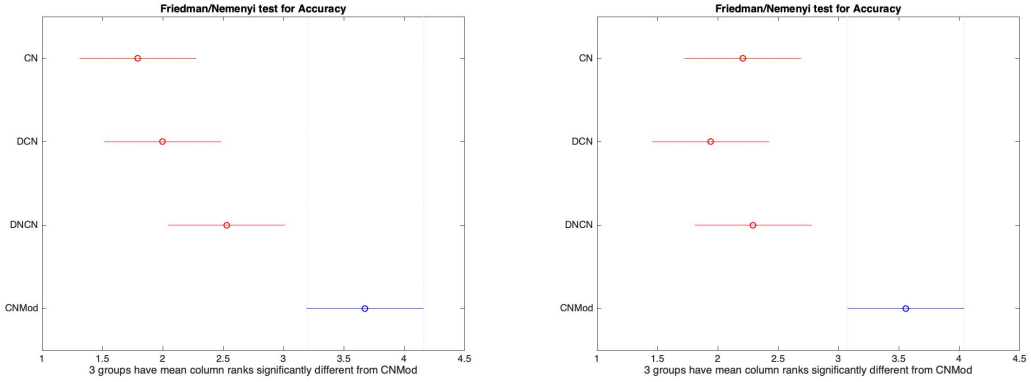
(a) ARI (b) NMI

Figure A.2: Friedman/Nemenyi test over the adjacency matrix results for the KKM algorithm.

		A	DA	DNA	AMod
A	ARI	1.0000	0.0684	0.0684	0.3318
	NMI	1.0000	0.0929	0.0929	0.2274
DA	ARI	0.0684	1.0000	1.0000	**0.0006**
	NMI	0.0929	1.0000	1.0000	**0.0014**
DNA	ARI	0.0684	1.0000	1.0000	**0.0006**
	NMI	0.0929	1.0000	1.0000	**0.0014**
AMod	ARI	0.3318	**0.0006**	**0.0006**	1.0000
	NMI	0.2274	**0.0014**	**0.0014**	1.0000

Table A.36: Results of the Wilcoxon tests for the adjacency index matrix and its transformations injected into the KKM algorithm.

A.3.3 Common neighbors



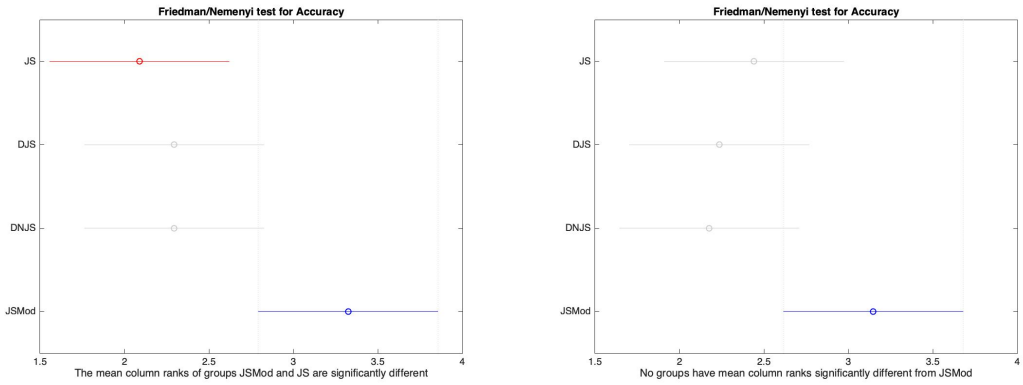
(a) ARI (b) NMI

Figure A.3: Friedman/Nemenyi test over the common neighbor similarity results for the KKM algorithm.

		CN	DCN	DNCN	CNMod
CN	ARI	1.0000	0.4688	**0.0156**	**0.0009**
	NMI	1.0000	0.9375	0.8125	**0.0038**
DCN	ARI	0.4688	1.0000	**0.0313**	**0.0006**
	NMI	0.9375	1.0000	**0.0313**	**0.0019**
DNCN	ARI	**0.0156**	**0.0313**	1.0000	**0.0011**
	NMI	0.8125	**0.0313**	1.0000	**0.0032**
CNMod	ARI	**0.0009**	**0.0006**	**0.0011**	1.0000
	NMI	**0.0038**	**0.0019**	**0.0032**	1.0000

Table A.37: Results of the Wilcoxon tests for the common neighbors index matrix and its transformations injected into the KKM algorithm.

A.3.4 Jaccard similarity



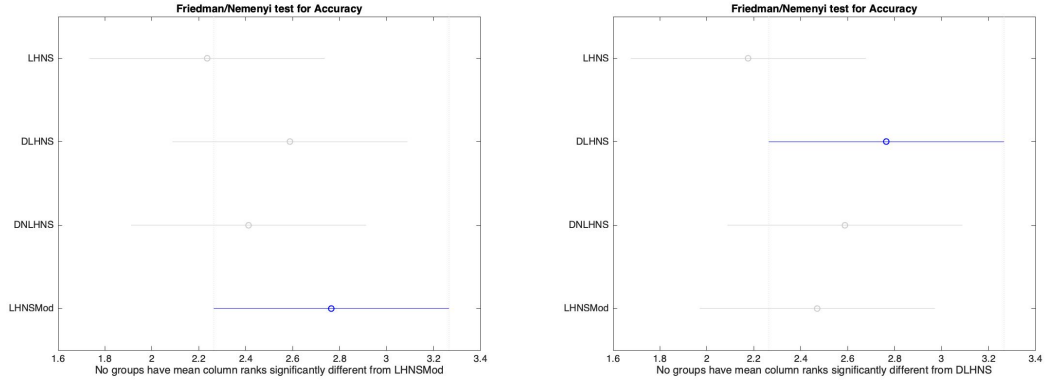
(a) ARI (b) NMI

Figure A.4: Friedman/Nemenyi test over the Jaccard similarity results for the KKM algorithm.

		JS	DJS	DNJS	JSMod
JS	ARI	1.0000	0.2775	0.5695	0.2343
	NMI	1.0000	0.7960	0.5014	0.1477
DJS	ARI	0.2775	1.0000	1.0000	0.1337
	NMI	0.7960	1.0000	0.8125	0.2343
DNJS	ARI	0.5695	1.0000	1.0000	0.1208
	NMI	0.5014	0.8125	1.0000	0.1788
JSMod	ARI	0.2343	0.1337	0.1208	1.0000
	NMI	0.1477	0.2343	0.1788	1.0000

Table A.38: Results of the Wilcoxon tests for the Jaccard index matrix and its transformations injected into the KKM algorithm.

A.3.5 Leich-Holme-Newman similarity



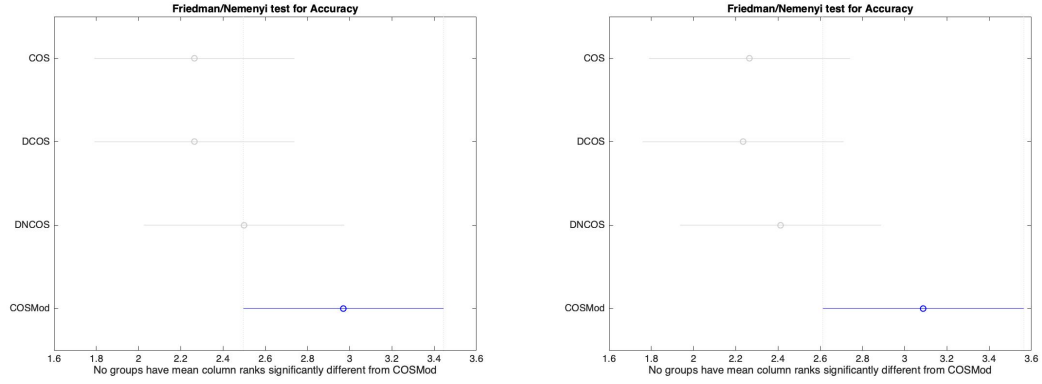
(a) ARI (b) NMI

Figure A.5: Friedman/Nemenyi test over the Leich-Holmes-Newman Similarity results for the KKM algorithm.

		LHNS	DLHNS	DNLHNS	LHNSMod
LHNS	ARI	1.0000	0.1094	0.1953	0.1488
	NMI	1.0000	0.0391	0.1484	0.1773
DLHNS	ARI	0.1094	1.0000	0.6875	0.9434
	NMI	0.0391	1.0000	0.5625	0.6874
DNLHNS	ARI	0.1953	0.6875	1.0000	0.4348
	NMI	0.1484	0.5625	1.0000	0.7226
LHNSMod	ARI	0.1488	0.9434	0.4348	1.0000
	NMI	0.1773	0.6874	0.7226	1.0000

Table A.39: Results of the Wilcoxon tests for the Leich-Holme-Newman index matrix and its transformations injected into the KKM algorithm.

A.3.6 Cosine similarity



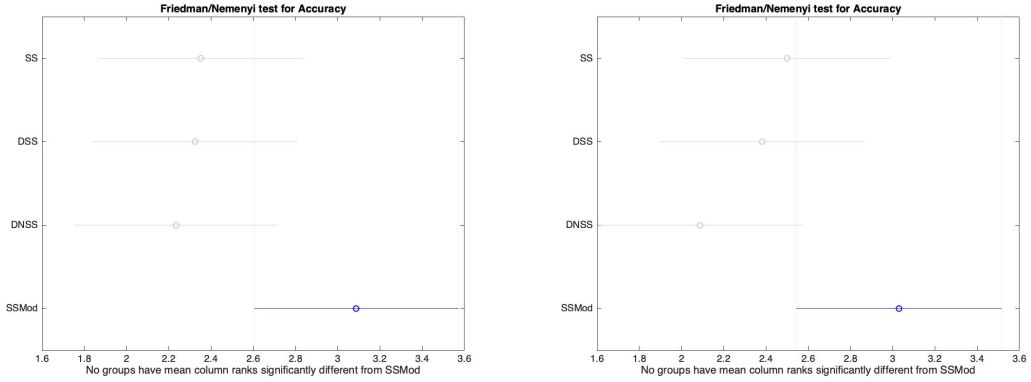
(a) ARI (b) NMI

Figure A.6: Friedman/Nemenyi tests over the cosine similarity results for the KKM algorithm.

		COS	DCOS	DNCOS	COSMod
COS	ARI	1.0000	0.3125	0.3125	**0.0134**
	NMI	1.0000	0.3125	0.3125	**0.0295**
DCOS	ARI	0.3125	1.0000	0.1250	0.0833
	NMI	0.3125	1.0000	0.1250	**0.0479**
DNCOS	ARI	0.3125	0.1250	1.0000	0.1070
	NMI	0.3125	0.1250	1.0000	0.1070
COSMod	ARI	**0.0134**	0.0833	0.1070	1.0000
	NMI	**0.0295**	**0.0479**	0.1070	1.0000

Table A.40: Results of the Wilcoxon tests for the cosine similarity matrix and its transformations injected into the KKM algorithm.

A.3.7 Sørsen similarity



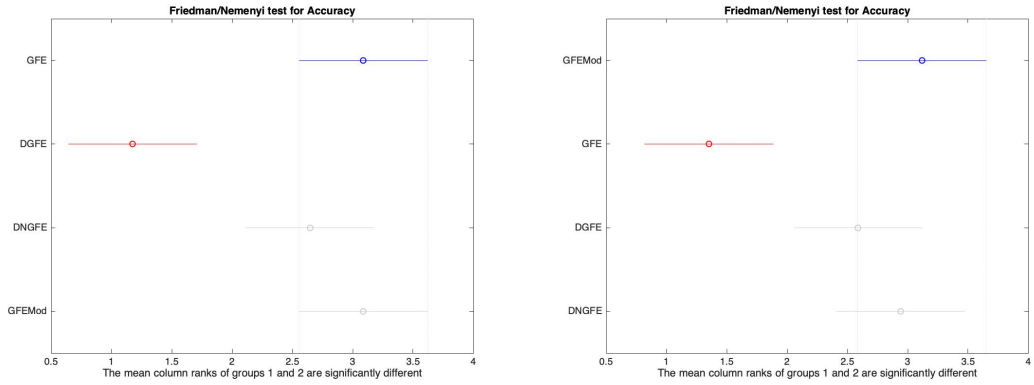
(a) ARI (b) NMI

Figure A.7: Friedman/Nemenyi tests over the Sørsen similarity results for the KKM algorithm.

		SS	DSS	DNSS	SSMod
SS	ARI	1.0000	0.2969	0.3125	0.1208
	NMI	1.0000	0.2969	0.0938	0.3259
DSS	ARI	0.2969	1.0000	0.6875	0.0627
	NMI	0.2969	1.0000	0.9375	0.2343
DNSS	ARI	0.3125	0.6875	1.0000	0.0879
	NMI	0.0938	0.9375	1.0000	0.1627
SSMod	ARI	0.1208	0.0627	0.0879	1.0000
	NMI	0.3259	0.2343	0.1627	1.0000

Table A.41: Results of the Wilcoxon tests for the Sørsen similarity matrix and its transformations injected into the KKM algorithm.

A.3.8 Gaussian free energy kernel



(a) ARI (b) NMI

Figure A.8: Friedman/Nemenyi tests over the Gaussian free energy similarity results for the KKM algorithm.

		GFEMod	GFE	DGFE	DNGFE
GFEMod	ARI	1.0000	**0.0005**	0.2166	0.2734
	NMI	1.0000	**0.0007**	0.1726	0.3054
GFE	ARI	**0.0005**	1.0000	**0.0005**	**0.0005**
	NMI	**0.0007**	1.0000	**0.0006**	**0.0006**
DGFE	ARI	0.2166	**0.0005**	1.0000	**0.0313**
	NMI	0.1726	**0.0006**	1.0000	**0.0313**
DNGFE	ARI	0.2734	**0.0005**	**0.0313**	1.0000
	NMI	0.3054	**0.0006**	**0.0313**	1.0000

Table A.42: Results of the Wilcoxon tests for the free energy kernel matrix and its transformations injected into the KKM algorithm.

Bibliography

[Adamic and Adar, 2003] Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social Networks*, 25:211—230.

[Ana and Jain, 2003] Ana, L. and Jain, A. K. (2003). Robust data clustering. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II. IEEE.

[Anderson and Peterson, 1987] Anderson, J. R. and Peterson, C. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019.

[Bagon and Galun, 2011] Bagon, S. and Galun, M. (2011). Large scale correlation clustering optimization. *arXiv preprint arXiv:1112.2903*.

[Beier et al., 2014] Beier, T., Kroeger, T., Kappes, J. H., Kothe, U., and Hamprecht, F. A. (2014). Cut, glue & cut: A fast, approximate solver for multicut partitioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 73–80.

[Berry and Linoff, 2004] Berry, M. J. and Linoff, G. S. (2004). *Data mining techniques: for marketing, sales, and customer relationship management*. John Wiley & Sons.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[Carole and Emma, 2018] Carole, C. and Emma, G.-H. (2018). 50 million facebook profiles harvested for cambridge analytica in major data breach. `https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election`. (Accessed on 10/08/2020).

[Chakraborty et al., 2017] Chakraborty, T., Dalmia, A., Mukherjee, A., and Ganguly, N. (2017). Metrics for community analysis: A survey. *ACM Computing Surveys (CSUR)*, 50(4):1–37.

[Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

[Chebotarev, 2011a] Chebotarev, P. (2011a). A class of graph-geodetic distances generalizing the shortest-path and the resistance distances. *Discrete Applied Mathematics*, 159(5):295–302.

[Chebotarev, 2011b] Chebotarev, P. (2011b). The graph bottleneck identity. *Advances in Applied Mathematics*, 47(3):403–413.

[Clauset et al., 2004] Clauset, A., Newman, M. E., and Moore, C. (2004). Finding community structure in very large networks. *Physical review E*, 70(6):066111.

[Courtain, 2017] Courtain, S. (2017). Community detection in networks by soft modularity maximization : A new approach and empirical comparisons. Master's thesis, Louvain School of Management.

[Courtain et al., 2020] Courtain, S., Guex, G., and Saerens, M. (2020). Clustering of attraction/repulsion data: a simple, scalable, algorithm.

[Courtain et al., 2019] Courtain, S., Lebichot, B., Kivimäki, I., and Saerens, M. (2019). Graph-based fraud detection with the free energy distance. In *International Conference on Complex Networks and Their Applications*, pages 40–52. Springer.

[Demšar, 2006] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30.

[Devooght and Bersini, 2017] Devooght, R. and Bersini, H. (2017). Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 13–21.

[Feld, 1981] Feld, S. L. (1981). The focused organization of social ties. *American journal of sociology*, 86(5):1015–1035.

[Fernandes et al., 2019] Fernandes, A., Gonçalves, P. C., Campos, P., and Delgado, C. (2019). Centrality and community detection: a co-marketing multilayer network. *Journal of Business & Industrial Marketing*.

[Fleming et al., 2007] Fleming, L., King III, C., and Juda, A. I. (2007). Small worlds and regional innovation. *Organization Science*, 18(6):938–954.

[Fortunato and Hric, 2016] Fortunato, S. and Hric, D. (2016). Community detection in networks: A user guide. *Physics reports*, 659:1–44.

[Fouss et al., 2016] Fouss, F., Saerens, M., and Shimbo, M. (2016). *Algorithms and Models for Network Data and Link Analysis*. Cambridge University Press, One Liberty Plaza, 20th Floor, New York, NY 10006, USA.

[Françoisse et al., 2017] Françoisse, K., Kivimäki, I., Mantrach, A., Rossi, F., and Saerens, M. (2017). A bag-of-paths framework for network data analysis. *Neural Networks*, 90:90–111.

[Friedman, 1937] Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701.

[Girvan and Newman, 2002] Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826.

[Grey, 1981] Grey, D. (1981). Multivariate analysis, by kv mardia, jt kent and jm bibby. pp 522.£ 14· 60. 1979. isbn 0 12 471252 5 (academic press). *The Mathematical Gazette*, 65(431):75–76.

[Hamdouch, 2007] Hamdouch, A. (2007). Innovation clusters and networks: A critical review of the recent literature. In *19th EAEPE conference*, pages 1–3.

[Hofmann and Buhmann, 1997] Hofmann, T. and Buhmann, J. M. (1997). Pairwise data clustering by deterministic annealing. *Ieee transactions on pattern analysis and machine intelligence*, 19(1):1–14.

[Huang et al., 2019] Huang, H., Shen, H., Meng, Z., Chang, H., and He, H. (2019). Community-based influence maximization for viral marketing. *Applied Intelligence*, 49(6):2137–2150.

[Huang et al., 2007] Huang, J.-J., Tzeng, G.-H., and Ong, C.-S. (2007). Marketing segmentation using support vector clustering. *Expert systems with applications*, 32(2):313–317.

[Hubert and Arabie, 1985] Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of classification*, 2(1):193–218.

[Izenman, 2008] Izenman, A. J. (2008). Modern multivariate statistical techniques. *Regression, classification and manifold learning*, 10:978–0.

[Jaccard, 1902] Jaccard, P. (1902). Distribution comparée de la flore alpine dans quelques régions des alpes occidentales et orientales. *Bulletin de la Murithienne*, (31):81–92.

[Jafarov et al., 2020] Jafarov, J., Kalhan, S., Makarychev, K., and Makarychev, Y. (2020). Correlation clustering with asymmetric classification errors.

[Jones et al., 2013] Jones, R., Suoranta, M., and Rowley, J. (2013). Strategic network marketing in technology smes. *Journal of Marketing Management*, 29(5-6):671–697.

[Kemeny and Snell, 1976] Kemeny, J. G. and Snell, J. L. (1976). *Markov chains*. Springer-Verlag, New York.

[Kishore et al., 2020] Kishore, R., Nussinov, Z., and Sahu, K. K. (2020). A new nature inspired modularity function adapted for unsupervised learning involving spatially embedded networks: A comparative analysis. *arXiv preprint arXiv:2007.09330*.

[Lancichinetti and Fortunato, 2009] Lancichinetti, A. and Fortunato, S. (2009). Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118.

[Lancichinetti et al., 2008] Lancichinetti, A., Fortunato, S., and Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110.

[Lang, 1995] Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pages 331–339. Elsevier.

[Lehmann and Hansen, 2007] Lehmann, S. and Hansen, L. K. (2007). Deterministic modularity optimization. *The European Physical Journal B*, 60(1):83–88.

[Leicht et al., 2006] Leicht, E. A., Holme, P., and Newman, M. E. (2006). Vertex similarity in networks. *Physical Review E*, 73(2):026120.

[Li and Mukaidono, 1999] Li, R.-P. and Mukaidono, M. (1999). Gaussian clustering method based on maximum-fuzzy-entropy interpretation. *Fuzzy Sets and systems*, 102(2):253–258.

[Lü and Zhou, 2011] Lü, L. and Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170.

[Lusseau, 2003] Lusseau, D. (2003). The emergent properties of a dolphin social network. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(suppl_2):S186–S188.

[Lusseau et al., 2003] Lusseau, D., Schneider, K., Boisseau, O. J., Haase, P., Slooten, E., and Dawson, S. M. (2003). The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405.

[Manning et al., 2008] Manning, C. D., Schütze, H., and Raghavan, P. (2008). *Introduction to information retrieval*. Cambridge university press.

[Mantrach et al., 2010] Mantrach, A., Yen, L., Callut, J., Francoise, K., Shimbo, M., and Saerens, M. (2010). The sum-over-paths covariance kernel: A novel covariance between nodes of a directed graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):1112–1126.

[Marden, 1996] Marden, J. I. (1996). *Analyzing and modeling rank data*. CRC Press.

[Mead, 1992] Mead, A. (1992). Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39.

[Meyer, 2000] Meyer, C. D. (2000). *Matrix analysis and applied linear algebra*, volume 71. Siam.

[Miyamoto et al., 2008] Miyamoto, S., Ichihashi, H., Honda, K., and Ichihashi, H. (2008). *Algorithms for fuzzy clustering*. Springer.

[Mizutani and Miyamoto, 2005] Mizutani, K. and Miyamoto, S. (2005). Possibilistic approach to kernel-based fuzzy c-means clustering with entropy regularization. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 144–155. Springer.

[Neal and Hinton, 1998] Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer.

[Nemenyi, 1963] Nemenyi, P. (1963). Distribution-free multiple comparisons.

[Newman, 2010] Newman, M. E. J. (2010). *Networks an Introduction*. Oxford University Press, Great Clarendon Street, Oxford, OX2 6DP, United Kingdom.

[Onnela et al., 2003] Onnela, J.-P., Chakraborti, A., Kaski, K., Kertesz, J., and Kanto, A. (2003). Asset trees and asset graphs in financial markets. *Physica Scripta*, 2003(T106):48.

[Onnela et al., 2004] Onnela, J.-P., Kaski, K., and Kertész, J. (2004). Clustering and information in correlation based financial networks. *The European Physical Journal B*, 38(2):353–362.

[Peterson and Söderberg, 1989] Peterson, C. and Söderberg, B. (1989). A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(01):3–22.

[Rand, 1971] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850.

[Rose et al., 1990] Rose, K., Gurewitz, E., and Fox, G. (1990). A deterministic annealing approach to clustering. *Pattern Recognition Letters*, 11(9):589–594.

[Rossi and Villa-Vialaneix, 2010] Rossi, F. and Villa-Vialaneix, N. (2010). Optimizing an organized modularity measure for topographic graph clustering: a deterministic annealing approach. *Neurocomputing*, 73(7-9):1142–1163.

[Sadaaki and Masao, 1997] Sadaaki, M. and Masao, M. (1997). Fuzzy c-means as a regularization and maximum entropy approach.

[Saerens et al., 2004] Saerens, M., Fouss, F., Yen, L., and Dupont, P. (2004). The principal components analysis of a graph, and its relationships to spectral clustering. In *European conference on machine learning*, pages 371–383. Springer.

[Salton, 1989] Salton, G. (1989). Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*, 169.

[Saul et al., 1996] Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4:61–76.

[Saul and Jordan, 1996] Saul, L. K. and Jordan, M. I. (1996). Exploiting tractable substructures in intractable networks. In *Advances in neural information processing systems*, pages 486–492.

[Seo et al., 2013] Seo, H., Kim, W., Lee, J., and Youn, B. (2013). Network-based approaches for anticancer therapy. *International journal of oncology*, 43(6):1737–1744.

[Shawe-Taylor et al., 2004] Shawe-Taylor, J., Cristianini, N., et al. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

[Simmie, 2004] Simmie, J. (2004). Innovation and clustering in the globalised international economy. *Urban studies*, 41(5-6):1095–1112.

[Sommer et al., 2016] Sommer, F., Fouss, F., and Saerens, M. (2016). Comparison of graph node distances on clustering tasks. pages 192–201.

[Sorensen, 1948] Sorensen, T. A. (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biol. Skar.*, 5:1–34.

[Tsitsulin et al., 2020] Tsitsulin, A., Palowitch, J., Perozzi, B., and Müller, E. (2020). Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*.

[Wilcoxon, 1992] Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer.

[Wittman, 2002] Wittman, T. (2002). Time-series clustering and association analysis of financial data. *University of Texas, Austin*.

[Xu et al., 2007] Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. (2007). Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833.

[Yen et al., 2009] Yen, L., Fouss, F., Decaestecker, C., Francq, P., and Saerens, M. (2009). Graph nodes clustering with the sigmoid commute-time kernel: A comparative study. *Data & Knowledge Engineering*, 68(3):338–361.

[Yung-Yao et al., 2019] Yung-Yao, C., Yu-Hsiu, L., Chia-Ching, K., Ming-Han, C., and I-Hsuan, Y. (2019). Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes. *Sensors (Basel)*.

[Zachary, 1977] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473.

[Zaki and Meira, 2014] Zaki, M. J. and Meira, W. (2014). *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press.

[Zhang et al., 2007] Zhang, S., Wang, R.-S., and Zhang, X.-S. (2007). Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490.

[Zhou et al., 2009] Zhou, T., Lü, L., and Zhang, Y.-C. (2009). Predicting missing links via local information. *The European Physical Journal B / Condensed Matter and Complex Systems*, 71:623–630.

LOUVAIN-LA-NEUVE | BRUXELLES | MONS | TOURNAI | CHARLEROI | NAMUR
Place des Doyens, 1 bte L2.01.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/lsm**