Universitat Politècnica de Catalunya

Department of Computer Science

Cluster Evaluation on Weighted Networks

Martí Renedo Mirambell

Supervisor: Argimiro Arratia Quesada

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computing

2023

Abstract

This thesis presents a systematic approach to validate the results of clustering methods on weighted networks, particularly for the cases where the existence of a community structure is unknown. Including edge weights has many applications in network science, as there are many situations in which the strength of the connections between nodes is an essential property that describes the network. This evaluation of clustering methods comprises a set of criteria for assessing their significance and stability.

First, a well-established set of community scoring functions, which already existed for unweighted graphs, has been extended to the case where the edges have associated weights. There is consideration given to how in some cases many possible weighted extensions to the same function can be defined, and each of them can suit different types of weighted networks. Additionally, methods to randomize graphs but maintaining the original graph's degree distribution have been defined in order to use these random graphs as baseline networks. This randomization together with the weighted community scoring functions are then used to evaluate cluster significance, since the random networks built from the original network with our methods provide reference values for each scoring function that will allow to actually determine whether a given cluster score for the original graph is better than a comparable graph with the same degree distribution but no community structure.

As for the evaluation of stability, we define non parametric bootstrap methods with perturbations for weighted graphs where vertices are resampled multiple times, and the perturbations are applied to the edge weights. This, together with some fundamental similarity metrics for set partitions derived from information theory and combinatorics, constitutes our criteria for clustering stability. These criteria are based on the essential idea that meaningful clusters should capture an inherent structure in the data and not be overly sensitive to small or local variations, or the particularities of the clustering algorithm.

A more in-depth study of the characteristics of cluster scoring functions and their potential bias towards clusters of a certain size has also been performed. This would render some of these functions unsuitable to compare results of clustering algorithms when the size of the partition differs considerably. For this analysis, we introduce parametrized multi-level ground truth models based on the stochastic block model and on preferential attachment that can showcase how the functions respond to varying the strength of each level of clusters in a hierarchical structure. Additionally, a scoring function that doesn't suffer from this kind of bias is proposed: the density ratio.

This thesis also contributes with an efficient implementation of Newman's Reduced Mutual Information, a measure to compare set partitions based on information theory. Here it is used as a tool to compare network partitions, which is particularly useful for the evaluation of cluster stability, but it can have applications beyond the field of network clustering. Our algorithm uses an hybrid approach that combines analytical approximation with a Markov Chain Monte Carlo method for a good balance between accuracy and efficiency.

Also an indispensable part of this thesis is the associated software that we developed, which includes the implementation of all the methods discussed in it. It all has all been included in our new R package *clustAnalytics*, which is already available at the CRAN repository. This package is designed to work together with *igraph*, the main R package dedicated to graphs, to make it easy and straightforward for other researchers to use. There are many situations in which these tools can be useful: from the study and observation of new datasets, to the evaluation and benchmarking of clustering algorithms. Additionally, parts of the package like the implementation of Newman's Reduced Mutual Information can also be applied to compare partitions of all kind of sets, not only networks.

Abstract (català)

Aquesta tesi presenta un mètode sistemàtic per validar els resultats obtinguts per mètodes de clústering a xarxes amb pesos, especialment per als casos en què es desconeix l'existència d'una estructura de comunitats. La inclusió de pesos a les arestes té diverses aplicacions a l'estudi de xarxes, ja que hi ha moltes situacions on la força de les connexions entre nodes és una propietat essencial que descriu la xarxa. Aquesta avaluació de mètodes de clústering inclou una sèrie de criteris per quantificar la seva significació i estabilitat.

En primer lloc, hem estès un conjunt de funcions per avaluar comunitats, que ja existien per a grafs sense pesos, al cas on les arestes tenen pesos associats. S'ha tingut en compte com en alguns casos es poden definir diverses extensions de la mateixa funció, cada una per a diferents tipus de xarxes amb pesos. A més a més, s'han definit mètodes per aleatoritzar grafs mantenint la seqüència de graus original, per fer servir aquests grafs aleatoris de referència. Aquest procés, juntament amb les funcions definides, permet avaluar la significació dels clústers, ja que els grafs aleatoris donen valors de referència que serveixen per determinar si la puntuació del graf original és millor que la d'un de comparable però que no tingui una estructura de comunitats.

Pel que fa a l'avaluació de l'estabilitat, definim mètodes de bootstrap no paramètric amb pertorbacions per a grafs amb pesos on els vèrtexs es remostregen diverses vegades, i s'apliquen pertorbacions al pes de les arestes. Conjuntament amb mesures de similitud per a particions de conjunts basades en teoria de la informació i combinatòria, formen els criteris per avaluar l'estabilitat dels clústers. Aquests criteris es basen en la idea que els clústers rellevants haurien de captar l'estructura de les dades, però no ser excessivament sensibles a petites variacions locals o a les particularitats dels algoritmes de clústering.

També es fa un estudi més a fons de les característiques de les funcions avaluadores dels clústers i el seu possible biaix cap a clústers d'una certa mida. Això podria fer que algunes d'aquestes funcions fossin inadequades per comparar resultats d'algoritmes de clústering en cas que la mida de les particions fos prou diferent. Per aquesta anàlisi, introduïm models parametritzats de comunitats multinivell basats en el model de blocs estocàstics i en el model de connexió preferencial, que mostren com les funcions responen quan la força relativa dels diversos nivells de clusters varia. A més a més, proposem una funció avaluadora que no té aquesta mena de biaix, la ràtio de densitats.

Aquesta tesi també aporta una implementació eficient de la informació mútua reduïda ("reduced mutual information") de Newman, una mesura per comparar particions de conjunts basada en teoria de la informació. Aquí es fa servir per comparar particions de xarxes, que és especialment útil per mesurar l'estabilitat dels clusters, però pot tenir aplicacions més enllà del clústering de xarxes. El nostre algoritme té un funcionament híbrid que combina una aproximació analítica amb un mètode de Monte Carlo de cadena de Markov per trobar un bon equilibri entre exactitud i eficiència.

Una altra part essencial d'aquesta tesi és el software associat que s'ha desenvolupat, que inclou les implementacions de tots els mètodes que s'hi discuteixen. S'ha compilat tot al nou paquet *clustAnalytics*, que ja és al repositori de CRAN. Aquest paquet està fet per funcionar conjuntament amb *igraph*, el principal paquet d'R dedicat als grafs, per fer-lo fàcil i accessible d'utilitzar a altres investigadors. Hi ha moltes situacions on aquestes eines poden ser útils: des de l'estudi i observació de nous conjunts de dades, a l'avaluació d'algoritmes de clústering. A més a més, algunes parts del paquet, com ara la implementació de la informació mútua reduïda de Newman també es poden fer servir per comparar particions de tota mena de conjunts, no només de xarxes.

Acknowledgements

First and foremost, I want to thank my supervisor Argimiro Arratia Quesada for his guidance in making this thesis. It would have not been possible without his constant help, support, and inspiration, and I am very grateful for it.

I also want to show my appreciation to the MACDA research group for funding the development of the *clustAnalytics* R package, which contains all the software used to perform all of the included experiments.[1]

Lastly, but just as importantly, I must also thank my family for always being there for me during this period. Their help and encouragement has been absolutely essential. And while not entirely appreciated at the time, their creativity in finding subtle ways to ask when this work would finally be over was arguably quite remarkable.

[1]Management and Analysis of Complex Data. Funded by MINECO. TIN2017-89244-R.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cluster detection is a very active topic with applications in many research fields: from computer science to biology or sociology. On a very fundamental level, what we refer as clustering is the aggregation of data into groups that share some characteristic, like proximity or similarity. There are many types of data that one can try to group into clusters, but we focus on clustering networks.

The first example that comes to mind when talking about the study of networks is the Internet, both on the physical side (the network of connected computers) and the more abstract World Wide Web: the network of web pages and the links between each other. Another very popular application of network science is the study of the academic world itself, with citation [55, 72] and co-autorship networks [51]. Of the latter category, perhaps the most popular one is the Erdös collaboration graph [11] and the Erdös numbers derived from it. The Erdös number of a researcher is defined as the distance between them and the celebrated mathematician Paul Erdös in the network formed by joining researchers with an edge when they have co-authored an article. It was initially intended as an homage, but has since motivated numerous studies on how authors collaborate.

There have also been numerous studies of networks modeling interactions between people and group dynamics. One of the most ubiquitous examples is the Zachary's karate club graph [95]: this network describes the members of a university karate club and their interactions. A conflict between the instructor and the club administrator caused the group to split in two, and it was observed that the results of applying community detection methods to the data corresponded almost exactly to the groups that formed after the split. This is one of the first notable instances of successful community detection in social networks, and the karate club graph has since become one of the most popular benchmarks for new community detection algorithms. Even before that, multiple articles already studied networks of social interactions, as surveyed by Mitchell in 1974 [61], though with significant limitations in terms of data collection at scale. But in the 21th century, the rise of massive online platforms, such as Facebook or Twitter have allowed the study of social networks of an unprecedented size [93, 81].

Another important application of network science is transportation networks [12, 40], a field in which a deeper understanding is important for optimizing logistics. Even the traveling salesman problem [32], one of the most well known NP-complete problems, shows a very direct and intuitive practical application of the study of graphs and networks.

The terms network and graph are used almost interchangeably in the literature, and as pointed out by Barabási in [9], the difference between them is actually very subtle. Network is generally used to refer to the real world systems that are studied, while on the other hand, graph is the mathematical object that represents and models them. It is also worth noting that despite the relationship between the two subjects, there is very little overlap in the research done on the fields of graph theory and network science [46].

A graph is a structure consisting of a pair $G = (V, E)$, where V is a set of elements called vertices, and E is a set of pairs of vertices, which we call edges. When talking about networks, vertices and edges are more often referred as nodes and links, respectively.

Different types of graphs can be considered depending on the specific definition. If multiple edges between the same pair of vertices[1] are allowed to exist (that is, $E(G)$ is a multiset), the graph is generally called multigraph. If pairs of the same vertex are allowed in $E(G)$ (multisets of size two instead of sets), then the graph can contain loops: edges that connect a vertex to itself. Most of the graphs considered in this thesis, as well as the related methods, don't allow loops or parallel edges, unless it is previously specified.

If the set $E(V)$ of edges, or pairs of vertices, is replaced in the definition by a set of ordered pairs, then we obtain a directed graph. In this case, edges have orientation, and two edges can exist that join the same pair of vertices, one for each direction.

Additionally, a numerical value or weight can be assigned to each edge, which will result in a weighted graph. Weights can be integer or real values, depending on the characteristics of the network that is being modeled. These type of graphs have many applications in network science, as there is a variety of useful interpretations that one can assign to the edges: from distances between nodes in transportation networks, to the number or frequency of interactions between individuals in a social network, or even time series correlations in financial networks. Most of the commonly studied weighted networks have non-negative weights, and the methods for community evaluation that we will propose are intended for this case. While they could be used with negative weights, and most definitions would allow it, one would first have to decide what exactly constitutes a meaningful cluster in such a network and how to treat these negative edges, which is beyond the scope of this work.

It is almost always appropriate in practical applications to consider an edge of weight 0 to be equivalent to the lack of edge between two nodes. It generally matches the intuition of what it means to have or not have a weighted edge in most networks, and it is also a convenient and natural assumption to make to extend unweighted cluster scoring functions to the weighted case (see chapter 2).

[1]These are often referred as parallel edges

1.1 Community structure in networks

It is worth noting that there is not a single universal definition of what constitutes a community structure in networks. However, there are some fundamental properties that are often accepted as characteristics of good communities. A community should have a high internal or intra-cluster connectivity, that is, a high edge density for the induced subgraph. And consequently, a community should be reasonably separated from the rest of the network and have a low amount of edges connecting to it (also referred as low inter-cluster or external connectivity), though this might not be a requirement in cases where overlapping communities are allowed [92]. Additionally, one can attempt to quantify and define some notions of similarity between vertices, and group similar vertices together while keeping those that are dissimilar apart.

These ideas translate well to weighted graphs: it is possible to simply apply the same concepts while prioritizing heavy edges over light or weak ones. In particular the weighted edge density can be naturally defined as the sum of edge weight over the number of pairs of vertices[2], and this also extends to the intra-cluster and inter-cluster densities.

There are many ways to group nodes of a network into clusters, but we will focus on partitions[3]. A partition of a graph G is a set of parts such that every part is a non empty set containing elements of $V(G)$, and every element belongs to exactly one part. This disallows overlapping communities (with one or multiple nodes belonging to more than one community), as well as having nodes not belonging to any community. Nodes that are very poorly connected to the rest of the network can simply form their own single node clusters, though. Even though they are less common, there are algorithms that produce overlapping clusters, such as the clique percolation algorithm [24], or the hierarchical algorithm proposed in [53].

Very often graphs are described using matrices, most of the time using the adjacency matrix. The adjacency matrix A of a graph $G = (V, E)$ is defined as the $n \times n$ matrix (where $n = |V|$) such that the element A_{ij} is 1 if there is an edge between vertices i and j, and 0 otherwise. On graphs where self edges aren't allowed, all elements of the diagonal will then be zero. Undirected graphs will have symmetrical adjacency matrices, while on directed graphs the matrix indicates the direction of the vertices, and will not be symmetric in general. Other matrices that are frequently used in the context of community detection are the Laplacian, and the transfer matrix. The Laplacian is defined as $L = D - A$, where D is the diagonal matrix in which the i-th element of the diagonal is the degree of vertex i.

[2]Ordered pairs if considering directed graphs.

[3]This excludes the case of clustering with overlapping communities. However, while this thesis doesn't cover them, it would be possible to use at least some of its methods on overlapping communities with only minor modifications.

1.2 Clustering algorithms for networks

1.2.1 Graph partitioning

This approach consists on fixing the desired number of clusters and their size, and then determining the way to distribute vertices in them to minimize the number of inter-community edges (called the *cut size*). It is a requirement for these methods to work to have specified cluster size and number, because otherwise the optimal solutions would be the single-cluster trivial one with cut size zero (in the case of no number restriction), or it could often form size one clusters with the lowest degree vertices and group the rest of the network together (if the number of clusters is fixed but not their size). This is however a big limitation, because in most practical applications the number of clusters that best describes the structure of a given network won't be known beforehand.

Note that graph partitioning is the terminology that is commonly used in the literature to describe this category of clustering methods, because they are based on finding or approximating the optimal cuts to minimize inter-cluster connectivity. However, any grouping of a graph's vertex into non-overlapping communities is a partition from the mathematical point of view, so the other categories of algorithms introduced in this chapter are technically producing partitions as well, regardless of the method used to obtain them.

Even for the case with just two communities of equal size, the problem (called the *minimum bisection* problem) is NP-hard [37], so heuristic algorithms such as the Kernighan-Lin algorithm [49] have been frequently used to reasonably approximate a solution.

A popular approach to the problem have been the spectral methods based on the eigenvectors of the Laplacian matrix, which are later mentioned in the spectral methods category. Other methods such as the one by Sanders and Schulz [83] based on the max-flow min-cut theorem [33] or by Galinier et al. [36] based on Tabu search were recently discussed in the survey by Schulz et al. [16].

1.2.2 Hierarchical clustering

This family of methods attempts to capture the multi-level community structure of networks, and produces a family of partitions each of which is a refinement of the previous one. Fundamentally, there are two ways to approach the problem, agglomeratively (starting with single-vertex communities and merging them based on similarity), or divisively (iteratively spliting the network by removing edges that connect disimilar vertices), but historically the agglomerattive way has been more common. It starts with a similarity measure, such as the cosine similarity, the correlations between corresponding rows in the adjacency matrix, or the Euclidean distance [65]. Then, all vertices start forming a singleton cluster, and the two with highest similarity are joined. At each step, the similarities of the newly formed cluster with the rest of the network have to be recalculated, to then again join the two clusters with the highest similarity. The process is repeated until all vertices are joined together into a single group. Finally, a cut must be made at some level of the tree to select the resulting partition, though this last step can be somewhat arbitrary.

There are several ways to define the similarity between clusters from any given measure of similarity between individual vertices. In the *single-linkage* method, the most similar pair of vertices (one of each group) is taken, while in the *complete-linkage*, the similarity between the least similar pair is used. The *average-linkage* clustering method takes the average over all pairs instead. The *single-linkage* version has a time complexity of $\mathcal{O}(n^2)$, while both the *complete* and *average-linkage* methods have a time complexity of $\mathcal{O}(n^2 \log n)$ [57]. In both of these cases it is assumed that the similarity between pairs of vertices can be computed in constant time, if not, the unavoidable initial step of computing the similarity between all pairs will have complexity $\mathcal{O}(sn^2)$ (where $\mathcal{O}(s)$ is the complexity of each similarity computation) and will be the bottleneck in the *single-linkage* method, and possibly in the other two as well.

One of the challenges of the hierarchical clustering algorithms is that the multi-level structure is given by the nature of the method, but it is not trivial to determine which of the levels represent meaningful clusters in the network. In fact, they don't detect whether the network actually has a hierarchical structure at all [34].

1.2.3 Modularity-based clustering

This family of clustering algorithms is based around optimizing Newman-Girvan's modularity [64] as the objective function (see section 2.2 for the definition). Essentially, the modularity uses a null model to compute the expected number of edges between vertices, and takes high values when the number of edges between vertices that share a community is higher than their expected values. Finding the global maximum for the modularity across all the possible partitions has been shown to be a NP-complete problem [14], so it is not possible to do it in most practical applications. However, many alternative approaches have been proposed to find good approximations.

The first algorithm for modularity optimization was Newman's method [62], a greedy agglomerative clustering algorithm that starts on a version of the graph with no edges, placing each vertex on its own community. Then, edges are added iteratively , starting with those that produce the highest increase in modularity, joining the respective communities whenever an edge is added. The proper data structures needed to implement the algorithm efficiently (in particular, to quickly compute the modularity gains on very sparse matrices) were described later in [18].

A further improvement on greedy modularity optimization techniques is the Louvain algorithm [13], perhaps the most widely used algorithm of this class. It starts with each vertex on it's own cluster, and iterates over the vertices to find possible moves to other communities that produce improvements in modularity (and performs the best possible move). This process is repeated until no more modularity improvements can be done, and then it's repeated on a community level: treating the communities as single vertices and trying to merge them using the same procedure to find further optimizations.

Despite its popularity, the Louvain algorithm has been shown to sometimes produce instances of communities that are poorly connected internally, and in some cases even disconnected communities can appear. The Leiden algorithm [86] was proposed to address this issue, and its main difference is the introduction of an additional step that refines the partitions. In addition to solving the problem of poorly connected communities and as a result finding better modularity values, it has also been shown to result in a faster running time as well.

Additionally, the modularity has also been shown to fail to identify small clusters in large networks, what is described as the resolution limit [35]. To attempt to mitigate the issue, it is possible to parametrize it to favor either smaller or larger communities [75] [4], but that still doesn't solve it entirely. Lancichinetti and Fortunato [52] later showed that tuning the modularity to detect smaller communities will split larger ones, while doing the opposite will merge the smaller ones. Therefore, it does not seem possible to avoid both biases simultaneously

Other related methods include the spinglass algorithm[75], which uses simulated annealing to optimize the Potts Hamiltonian, a function analogous to the parametrized modularity.

1.2.4 Spectral methods

This family of methods uses the eigenvalues of matrices that encode the graph for cluster detection. There is some overlap with the other categories.

Many partitional methods are based on the Laplacian matrix of the graph, and particularly in the properties relating its second eigenvalue to the vertex and edge connectivity of the graph studied by Fiedler [31]. Pothen et al. [71] used this approach on their spectral partitioning algorithm that can be used to compute both vertex and edge separators of the graph. The main idea behind it is to encode the graph partitions as vectors with certain constraints, and then try to find partitions whose associated vector is as close as possible to the second eigenvector of the Laplacian matrix.

Newman later proposed a different approach [63], that instead of the Laplacian, uses the modularity matrix, which is defined by substracting the expected number of edges (according to an appropriate null model) to each element of the adjacency matrix. Then, the desired partition will be the one whose associated vector is closest to being parallel to the leading eigenvector (the eigenvector corresponding to the eigenvalue of highest absolute value, or leading eigenvalue). After that, a refinement step is performed by having single vertices switch communities when that produces an increase in modularity. The most basic implementation of this method simply divides the network into two communities, but by applying it hierarchically to each of the parts, one can keep subdividing it. To find the best possible partition, the global increase in modularity is computed at each subdivision, and the divisions are performed recursively until the modularity doesn't improve anymore.

Alternatively, instead of using a single eigenvector to split the network into two, and then proceeding recursively to further split the resulting communities, some methods use multiple eigenvectors simultaneously. Ng, Jordan and Weiss [48] proposed a method that uses the first k eigenvectors of the Laplacian matrix to obtain n points of $\mathbb{R}^k$ that are then clustered using the k-means clustering algorithm. Each of these n points is associated to a vertex of the graph, so the clusters found on the points of $\mathbb{R}^k$ induce clusters on the graph as well. Other methods that also use the eigenvectors of the Laplacian to embed the vertices of the graph in an metric space have later appeared, such as the one by Donetti and Muñoz [27], who use hierarchical clustering on the metric space while simultaneously computing the modularity on the graph, which is then used to choose the best partition out of the dendrogram.

1.2.5 Other methods

A different approach to community detection is based on random walks. Zhou [97] introduced a distance between vertices defined as the average number of steps that a random walker has to take to go from one vertex to the other. Then, the local attractor of a vertex is defined as its closest neighbor, while the global attractor is defined as its closest vertex (note a very well connected vertex can be the closest to another without being its neighbor, using this distance). Local and global communities are defined around local and global attractors, which results in a two-level hierarchical community detection (global communities can contain a single or several local communities, and Zhou found the two structures to coincide when the network is small and to be different when it is larger and has thousands of nodes). The complexity of this algorithm is $\mathcal{O}(n^3)$.

The Walktrap algorithm by Latapy and Pons [70] also uses random walks, but with a slightly different hierarchical agglomerative approach. To determine which communities are to be joined at each iteration of the algorithm, they define a distance based on the probability of random paths of a given length starting in a vertex to end in another. Using an efficient method for computing these distances, the complexity of the algorithm is $\mathcal{O}(n^2 d)$ where d is the depth of the dendrogram, resulting in an expected running time of $\mathcal{O}(n^2 \log n)$ in most cases, which allows its use on reasonably large networks.

Another way to use random walks for community detection is provided by the Infomap algorithm by Rosvall and Bergstorm [82]. It is based on information theory and on finding encodings that can efficiently describe random walks. It exploits the fact that a community structure in a network allows the description of a path using less information, as nodes within the same community are more likely to appear consecutively in the path. Ultimately this is expressed in a map equation that is optimized to find the desired clustering.

A very fast algorithm that can be used on networks of massive scale is the label propagation method introduced by Raghavan et al. [73]. It works by giving labels to vertices, and iteratively having them adopt the label that is most frequent among their neighbors. The method converges when all vertices have the same label as most of their neighbours and no more switches can be performed. Ties between labels (especially present at the first iterations, when every vertex is assigned a unique label) are resolved at random, which makes the algorithm non-deterministic, though the results of successive executions on the same graph have been observed to be similar in most cases. The complexity of each iteration is linear on the number of edges m, and the required number of iterations has been observed to grow very slowly compared to the size of the network. A recent survey by Garza and Schaeffer [38] discusses multiple potential improvements to the algorithm that have been proposed over the years to overcome some issues such as the instability of the results in some instances, to extend it to weighted or bipartite networks, or to allow overlapping of communities.

1.3 Cluster Assessment

A common issue with most clustering algorithms is that they tend to partition networks into multiple clusters even when the network has a nearly homogeneous structure. Then, when studying a network and trying to determine whether there is a prevalent community structure in it, it is necessary to validate and evaluate the partitions that result from applying one or several clustering algorithms to it. This can be done with scoring functions based on the notions of what constitutes a community, such as strong internal connectivity, weak external connectivity, and especially combinations of both. Yang and Leskovec [94] analyzed a collection of such functions and studied how they performed on real networks with known ground truth community structures.

However, there aren't general values of this functions that determine at which point any given score indicates the presence of actual communities, so they alone aren't enough to validate the results of clustering algorithms (they can, however, be used to compare different algorithms when used on the same network). To make these measures more useful in applications where the existence of a ground truth community structure is unknown, we propose to use a randomization algorithm that rewires the edges of the network without modifying its degree sequence. This effectively removes the community structure of a network, and gives us benchmark values of the scoring functions to which we can then compare the original scores to. While this rewiring algorithm was already well known for the unweighted case [60], we provide an extension to the weighted case, with two variations to accommodate different types of weighted networks.

While the cluster scoring functions are used to evaluate the significance of clusters, another important part of the evaluation process is stability. If a given algorithm is successfully detecting the community structure of a network , a reasonable expectation would be that small variations on the network (such as the addition or removal of just a few vertices or edges) don't result in large variations in the results. Hennig [42] proposed the use of nonparametric bootstrap methods, with and without perturbations, to evaluate cluster stability in Euclidean datasets. We bring this method to the study of clusters in networks, with the possibility of adding perturbation to the edge weights if the network is weighted. To measure the resulting deviations from the original clusters, we use measures from both information theory and combinatorics, such as the Variation of Information [59], the Reduced Mutual Information [66], and the adjusted Rand Index [45].

1.4 Thesis structure

Here we will briefly describe the structure of the thesis and how the contents are organized. Chapter 2 defines the cluster scoring functions, and also introduces some basic definitions and notation that will be used throughout the thesis. These scoring functions quantify to what extend a partition of a network satisfies some of the properties that are expected of clusters. On the other hand, chapter 3 introduces functions that compare how two different partitions of the same network differ from each other.

Chapters 4 and 5 study the significance and stability of clusters. Each includes a description of the corresponding methods, and their application on a collection of real life and synthetic networks.

Chapter 6 proposes two models with multi-level community structures, one based on the stochastic block model, and another using preferential attachment. These models include parameters to adjust the relative strength of each level of clusters, and are then used in the next chapter to study the potential bias of community scoring functions based on size.

Chapter 7 contains a more in depth analysis of the cluster scoring functions, and in particular studies how some of them might present biases related to the size of the clusters. It also includes the definition of the density ratio, a new cluster scoring function that doesn't suffer from this issue.

Chapter 8 is dedicated to discussing the implementation of the algorithms used in the thesis, particularly those in which achieving an efficient outcome is not trivial. This includes the computation of the Reduced Mutual Information (RMI) using an hybrid method (combining Monte Carlo sampling with an analytical approximation), as well as the graph randomization algorithms and the weighted version of the transitivity and clustering coefficient.

Finally, chapter 9 describes the clustAnalytics software package in detail, and includes many examples of its use. This is intended as a guide to researchers who might be interested in using the package, and provides indications on how to use it depending on the characteristics of the networks to be studied.

Contributions

Articles produced:

- Clustering assessment in weighted networks. *PeerJ Computer Science*, 2021 [77]. Contributions are:
  - Weighted version of the switching model for the network's edges, with two versions to suit different types of weighted networks
  - Extension of the cluster scoring functions to the weighted case.
  - Method for studying the cluster significance that combines the switching model and the scoring functions.

  - Method for studying the cluster stability by using bootstrap on the vertices.

- Identifying bias in cluster quality metrics [78].
  - Method to study bias based on size on the cluster scoring functions with the multi-level networks where the cluster strength of each level is parametrized.
  - Multi-level preferential attachment model which is also scale-free.
  - Introduction of the density ratio, a cluster scoring function with no bias relative to the size of the clusters.

- Towards an Efficient Algorithm for Computing the Reduced Mutual Information. Short paper, *24th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2022)* [80].
  - An efficient algorithm for estimating the number of contingency tables for a given pair of column and row sums, which combines an analytical approximation for the dense part of the table with a Markov chain Monte Carlo method for the rest, where the analytical approximation is inaccurate. This is the challenging step for the computation of the Reduced Mutual Information.

- The Assessment of Clustering on Weighted Networks with R Package clustAnalytics. Short paper, *24th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2022)* [7].

- clustAnalytics: An R Package for Assessing Stability and Significance of Communities in Networks. Article under review at *The R Journal*. The package is already available at the CRAN repository [79].
  - The package includes the implementation of all the methods described in this thesis to be used in R with *igraph* graphs.
  - The software also supports directed networks, even though they haven't been studied in depth here.

- On methods to assess the significance of community structure in networks of financial time series. *ITISE 2017. 4th International Work-Conference on Time Series Analysis*, 2017 [6].
  - Study of weighted networks built from correlations between financial time series. A preliminary version of the methods for evaluating cluster significance proposed in this thesis was used to study the difference between different correlation measures used to build the networks and their effect on cluster detection.

Chapter 2

Cluster scoring functions

This chapter is dedicated to introducing scoring functions capable of evaluating clusters and quantifying how much they adhere to the notions we associate with a proper community structure in networks, such as the presence of numerous connections within clusters while still having a fair amount of separation (few connections between different clusters). Many of these functions were studied by Yang and Leskovec [94] for the unweighted case. Most of these functions extend naturally to graphs with weighted edges, and we provide the definitions. In cases such as the transitivity or clustering coefficient, there isn't a single straightforward extension, and we discuss which of them is more versatile and appropriate. These functions will later be used in chapter 4 to evaluate cluster significance on a selection of networks and algorithms.

2.1 Basic definitions.

Let $G(V, E)$ be an undirected graph of order $n = |V|$ and size $m = |E|$, where V is the set of vertices, and E the set of edges, which are pairs of vertices. In the case of a weighted graph[1] $\tilde{G}(V, \tilde{E})$, w_{uv} will denote the weight of the edge $(u, v) \in E$, and $\tilde{m} = \sum_{(u,v)\in\tilde{E}} w_{uv}$ the sum of all edge weights. Given $S \subset G$ a subset of vertices of the graph, we have $n_S = |S|$, $m_S = |\{(u, v) \in E : u \in S, v \in S\}|$, and in the weighted case $\tilde{m}_S = \sum_{(u,v)\in\tilde{E}:u,v\in S} w_{uv}$. Note that if we treat an unweighted graph as a weighted graph with weights 0 and 1 (1 if two vertices are connected by an edge, 0 otherwise), then $m = \tilde{m}$ and $m_S = \tilde{m}_S$ for all $S \subset V$. Associated to G there is its adjacency matrix $A(G) = (A_{ij})_{1\le i,j\le n}$ where $A_{ij} = 1$ if $(i, j) \in E$, 0 otherwise. We insist that $A(G)$ only take binary values 0 or 1 to indicate existence of edges, even in the case of weighted graphs.

The following definitions will also be needed later on:

- $d(u) = |\{v \in V : (u, v) \in E\}|$ is the degree of vertex u, which counts its number of incident edges.
- $\tilde{d}(u) = \sum_{v\neq u} w_{uv}$ is the natural extension of the vertex degree $d(u)$ to weighted graphs; the sum of weights of edges incident to u.

[1]For every variable or function defined over the unweighted graph, will use a "$\sim$" to denote its weighted counterpart

- $d_S(u) = |\{v \in S : (u,v) \in E\}|$ and $\tilde{d}_S(u) = \sum_{v \in S} w_{uv}$ are the (unweighted and weighted, respectively) degrees[2] restricted to the subgraph S.
- d_m and $\tilde{d}_m$ are the median values of $d(u)$, $u \in V$.[3]
- $c_S = |\{(u,v) \in E : u \in S, v \notin S\}|$ is the number of edges connecting S to the rest of the graph.
- $\tilde{c}_S = \sum_{(u,v) \in E : u \in S, v \notin S} w_{uv}$ is the natural extension of c_S to weighted graphs; the sum of weights of all edges connecting S to $G \setminus S$.

2.2 Community scoring functions

The left column in table 2.1 shows the community scoring functions for unweighted networks defined in [94]. These functions characterize some of the properties that are expected in networks with a strong community structure, with more ties between nodes in the same community than connecting them to the exterior. There are scoring functions based on internal connectivity (internal density, edges inside, average degree), external connectivity (expansion, cut ratio) or a combination of both (conductance, normalized cut, and maximum and average out degree fractions). Uparrow (respectively, downarrow) indicates the higher (resp., lower) the scoring function value the stronger the clustering.

On the right column we propose generalizations to the scoring functions which are suitable for weighted graphs while most closely resembling their unweighted counterparts. Note that for graphs which only have weights 0 and 1 (essentially unweighted graphs) each pair of functions is equivalent (any definition that didn't satisfy this wouldn't be a generalization at all).

- **Internal density, edges inside, average degree:** These definitions are easily and naturally extended by replacing the number of edges by the sum of their weights.
- **Expansion:** Average number of edges connected to the outside of the community, per node. For weighted graphs, average sum of edges connected to the outside, per node.
- **Cut Ratio:** Fraction of edges leaving the cluster, over all possible edges. The proposed generalization is reasonable because edge weights are upper bounded by 1 and therefore relate easily to the unweighted case. In more general weigthed networks, however, this could take values well over 1 while lacking many "potential" edges (as edges with higher weights would distort the measure). In general bounded networks (with bound other than 1) it would be reasonable to divide the result by the bound, which would result in the function taking values between 0 and 1 (0 with all possible edges being 0 and 1 when all possible edges reached the bound).

[2]We assume the weight function w_{uv} is defined for every pair of vertices u,v of the weighted graph, with $w_{uv} = 0$ if there is no edge between them.

[3]To prevent confusion between the function $d_S(\cdot)$ and the median value (which only depends on G) d_m we will always refer to subgraphs of G with uppercase letters.

TABLE 2.1: Community scoring functions $f(S)$ for weighted and unweighted networks.

	unweighted $f(S)$	weighted $f(S)$
↑ Internal density	$\frac{m_S}{n_S(n_S-1)/2}$	$\frac{\tilde{m}_s}{n_S(n_S-1)/2}$
↑ Edges Inside	m_S	$\tilde{m}_S$
↑ Average Degree	$\frac{2m_S}{n_S}$	$\frac{2\tilde{m}_S}{n_S}$
↓ Expansion	$\frac{c_s}{n_s}$	$\frac{\tilde{c}_s}{n_s}$
↓ Cut Ratio	$\frac{c_s}{n_s(n-n_s)}$	$\frac{\tilde{c}_s}{n_s(n-n_s)}$
↓ Conductance	$\frac{c_s}{2m_s+c_s}$	$\frac{\tilde{c}_s}{2\tilde{m}_s+\tilde{c}_s}$
↓ Normalized Cut	$\frac{c_s}{2m_s+c_s}+\frac{c_s}{2(m-m_s)+c_s}$	$\frac{\tilde{c}_s}{2\tilde{m}_s+\tilde{c}_s}+\frac{\tilde{c}_s}{2(\tilde{m}-\tilde{m}_s)+\tilde{c}_s}$
↓ Maximum ODF	$\max_{u\in S}\frac{\lvert\{(u,v)\in E: v\notin S\}\rvert}{d(u)}$	$\max_{u\in S}\frac{\sum_{v\notin S} w_{uv}}{\tilde{d}(u)}$
↓ Average ODF	$\frac{1}{n_s}\sum_{u\in S}\frac{\lvert\{(u,v)\in E: v\notin S\}\rvert}{d(u)}$	$\frac{1}{n_s}\sum_{u\in S}\frac{\sum_{v\notin S} w_{uv}}{\tilde{d}(u)}$

- **Conductance and normalized cut:** Again, these definitions are easily extended using the methods described above.
- **Maximum and average Out Degree Fraction:** Maximum and average fractions of edges leaving the cluster over the degree of the node. Again, in the weighted case the number of edges is replaced by the sum of edge weights.

Some of the introduced functions (internal density, edges inside, average degree, clustering coefficient) take higher values the stronger the clusterings are, while the others (expansion, cut ratio, conductance, normalized cut, out degree fraction) do the opposite.

Clustering coefficient.

Another possible scoring function for communities is the clustering coefficient or transitivity: the fraction of closed triplets over the number of connected triplets of vertices. A high internal clustering coefficient (computed on the graph induced by the vertices of a community) matches the intuition of a well connected and cohesive community inside a network, but its generalization to weighted networks is not trivial.

There have been several attempts to come up with a definition of the clustering coefficient for weighted networks. One is proposed in [10] and is given by $c_i = \frac{1}{\tilde{d}(i)(d(i)-1)}\sum_{j,h}\frac{w_{ij}+w_{ih}}{2}A_{ij}A_{jh}A_{ih}$. Note that this gives a local (*i.e.* defined for each vertex) clustering coefficient.

While this may work well on some weighted networks, in the case of complete networks, such as those built from correlation of time series, we obtain

$$\begin{aligned} c_i &= \frac{1}{\tilde{d}(i)(d(i)-1)} \sum_{j,h} \frac{w_{ij}+w_{ih}}{2} \\ &= \frac{\sum_{jh} w_{ij} + \sum_{jh} w_{ih}}{\tilde{d}(i)(n-2)\cdot 2} = \frac{(n-2)\sum_j w_{ij} + (n-2)\sum_h w_{ih}}{\tilde{d}(i)(n-2)\cdot 2} \\ &= \frac{(n-2)\tilde{d}(i) + (n-2)\tilde{d}(i)}{\tilde{d}(i)(n-2)\cdot 2} = 1, \end{aligned}$$

which doesn't give any information about the network.

An alternative was proposed in [58] with complete weighted networks in mind (with weights in the interval $[0,1]$), which makes it more adequate for our case.

- For $t \in [0,1]$ let A_t be the adjacency matrix with elements $A_{ij}^t = 1$ if $w_{ij} \geq t$ and 0 otherwise.
- Let C_t the clustering coefficient of the graph defined by A_t.
- The resulting weighted clustering coefficient is defined as

$$\tilde{C} = \int_0^1 C_t \, dt \tag{2.1}$$

Since C_t can only take as many different values as the number of different edge weights in the network, the integral is actually a finite sum. However, computing C_t (which is not computationally trivial) potentially as many as $n(n-1)$ times would be very costly for large values of n , so this function has been implemented by approximating the integral dividing the interval [0,1] into `n_step` parts (where `n_step`[4] is much smaller than $n(n-1)$).

For networks where the weights are either not bounded or bounded into a different interval than $[0,1]$, the most natural approach is to simply take

$$\tilde{C} = \frac{1}{\bar{w}} \int_0^{\bar{w}} C_t dt, \tag{2.2}$$

where $\bar{w}$ can be either the upper bound or, in the case of networks with no natural bound, the maximum edge weight.

It is a desirable property that the output of scoring functions remain invariant under uniform scaling, that is, if we multiply all edge weights by a constant $\phi > 0$, as the community structure of the network would be the same. This holds for all of the measures of the third group, which combine the notions of internal and external connectivity.

This means that these scores will be less biased in favour of networks with high overall weight (for the internal connectivity based scores) or low overall weight

[4]In this case we set `n_step`=100. This gives a reasonable resolution while keeping computations fast.

(for the external connectivity ones). It is particularly interesting for networks with weights that are not naturally upper bounded by one, and facilitates comparisons between networks with completely different weight distributions. When we compare each network's scores to those of a randomized counterpart generated by the switching model, though, the total weight is kept constant, so even scores without this property could still give valuable information.

Let $\tilde{G}_\phi(V, \tilde{E}_\phi)$ be the weigthed graph obtained by multiplying all edge weights in $\tilde{E}$ by ϕ. In this case, $n_{S_\phi} = n_S$, $m_{S_\phi} = \phi m_S$, $c_{S_\phi} = \phi c_S$, $d_{S_\phi}(u) = \phi d_S(u)$. This means that the internal density, edges inside, average degree, expansion and cut ratio behave linearly (with respect to their edge weights). Conductance, normalized cut and maximum and average out degree fractions, on the other hand, remain constant under these transformations. Since the notion of community structure is generally considered in relation to the rest of the network (a subset of vertices belong to the same community because they are more connected among themselves than to vertices outside of the community), it seems reasonable to consider that the same partitions on two graphs whose weights are the same up to a multiplicative positive constant factor have the same scores. This makes the scores in the third group, the only ones for which this property holds, more adequate in principle.

For the chosen definition of clustering coefficient this property also holds, as all terms in the integral in equation (2.2) behave linearly (the proof is immediate with a change of variables), and that linear factor cancels out.

Modularity

As for the modularity [64] of a partition $\mathcal{P}$, it is defined as:

$$Q = \frac{1}{2\tilde{m}} \sum_{ij} \left[w_{ij} - \frac{\tilde{d}(i)\tilde{d}(j)}{2\tilde{m}} \right] \delta_{\mathcal{P}}(i,j), \tag{2.3}$$

where $\delta_{\mathcal{P}}(i,j)$ takes value 1 if i and j belong to the same community and 0 otherwise.

Then, by multiplying the edges by a constant $\phi > 0$, we get the graph $\tilde{G}_\phi(V, \tilde{E}_\phi)$ of modularity:

$$\begin{aligned} Q_\phi &= \frac{1}{2\tilde{m}\phi} \sum_{ij} \left[\phi w_{ij} - \frac{\phi^2 \tilde{d}(i)\tilde{d}(j)}{2\tilde{m}\phi} \right] \delta_{\mathcal{P}}(i,j) \\ &= \frac{1}{2\tilde{m}\phi} \phi \sum_{ij} \left[w_{ij} - \frac{\tilde{d}(i)\tilde{d}(j)}{2\tilde{m}} \right] \delta_{\mathcal{P}}(i,j) = Q, \end{aligned} \tag{2.4}$$

which means that modularity is also invariant under uniform scaling.

Chapter 3

Cluster Similarity Measures

When working with community detection in networks it is important to be able to quantify the similarity between different partitions of the same network. Even when simply observing and comparing clustering algorithms, cluster similarity measures will help identify whether their outputs resemble each other. But it is when evaluating cluster stability that such tools become essential. We will want to determine if the clusters detected by an algorithm are robust and remain mostly intact under small perturbations of the network, and that requires comparing the initial results to those of the perturbed networks.

We will focus on two approaches: one based on information theory, which includes the well known mutual information measure, and another from combinatorics, the Rand index, which is constructed by counting the amount of agreements and disagreements in whether each pair of vertices belong or not to the same community. All of these measures are constructed from the contingency table of the labelings.

Consider a set of n elements and two labelings or partitions, one labeled by integers $r = 1, \ldots, R$ and the other labeled by integers $s = 1, \ldots, S$, which we will call $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_R\}$ and $\mathcal{P}' = \{\mathcal{P}'_1, \ldots, \mathcal{P}'_S\}$. Define a_r as the number of elements with label r in the first partition, b_s the number of elements with label s in the second partition, and c_{rs} be the number of elements with label r in the first partition and label s in the second. Formally,

$$\begin{aligned} a_r &= |\mathcal{P}_r| = \sum_{s=1}^{S} c_{rs} \\ b_s &= |\mathcal{P}'_s| = \sum_{r=1}^{R} c_{rs} \\ c_{rs} &= |\mathcal{P}_r \cap \mathcal{P}'_s|, \end{aligned}$$

and table 3.1 shows the contingency table.

3.1 Rand index

The Rand index and the different measures derived from it [45] are based on the idea of counting pairs of elements that are classified similarly and dissimilarly across the two partitions $\mathcal{P}$ and $\mathcal{P}'$. There are four types of pairs of elements:

	$\mathcal{P}'_1$	$\mathcal{P}'_2$	...	$\mathcal{P}'_S$	sum
$\mathcal{P}_1$	c_{11}	c_{12}	...	c_{1S}	a_1
$\mathcal{P}_2$	c_{21}	c_{22}		c_{2S}	a_2
	...			...	
$\mathcal{P}_R$	c_{R1}	c_{R2}	...	c_{RS}	a_r
sum	b_1	b_2	...	b_S	$n = \sum c_{ij}$

TABLE 3.1: Contingency table of partitions $\mathcal{P}$ and $\mathcal{P}'$, with labelings r and s.

- **type I**: elements are in the same class both in $\mathcal{P}$ and $\mathcal{P}'$
- **type II**: elements are in different classes both in $\mathcal{P}$ and $\mathcal{P}'$
- **type III**: elements are in different classes in $\mathcal{P}$ and in the same class in $\mathcal{P}'$.
- **type IV**: elements are in the same class in $\mathcal{P}$ and different classes in $\mathcal{P}'$.

Then, similar partitions would have many pairs of elements of types I and II (agreements) and few of type III and IV (disagreements). The Rand index is defined as the ratio of agreements over the total number of pairs of elements.

Using the terms of the contingency table (table 3.1), the Rand index is given by

$$\mathrm{RI}(r,s) = \binom{n}{2} + 2\sum_r s\binom{n_{rs}}{2} - [\sum_{r=1}^{R}\binom{a_r}{2} + \sum_{s=1}^{S}\binom{b_s}{2}] \tag{3.1}$$

An adjusted form of the Rand index [45] introduces a correction to account for all the pairings that match on both partitions because of random chance. The Adjusted Rand Index (ARI) is defined as:

$$\mathrm{ARI}(r;s) = \frac{\text{Index} - \text{Expected Index}}{\text{Maximum Index} - \text{Expected Index}}, \tag{3.2}$$

which in terms of the contingency table (table 3.1) can be expressed as:

$$\mathrm{ARI}(r;s) = \frac{\sum_{rs}\binom{c_{rs}}{2} - \sum_r\binom{a_r}{2}\sum_s\binom{b_s}{2}/\binom{n}{2}}{\frac{1}{2}[\sum_r\binom{a_r}{2} + \sum_s\binom{b_s}{2}] - \sum_r\binom{a_r}{2}\sum_s\binom{b_s}{2}/\binom{n}{2}} \tag{3.3}$$

Both the standard and adjusted versions of the Rand index are upper bounded by 1 (which will be achieved when both partitions are identical), but while the standard index is lower bounded by 0, the adjusted index can take negative values on pairs of partitions with a standard index that is lower than the expected value. And naturally, when the standard index matches the expected value, the adjusted index will be 0.

3.2 Information based measures

This family of metrics derives from Shannon's entropy and mutual information. While different bases for the logarithms can be used depending on the application, in the context of cluster analysis base 2 is generally used, which results in the unit of entropy being the bit.

Define the probabilities $P(r)$ and $P(s)$ of an object chosen uniformly at random having label r or s respectively, and the probability $P(r,s)$ that it has both labels r and s. That is:

$$P(r) = \frac{a_r}{n}, \qquad P(s) = \frac{b_s}{n}, \qquad P(r,s) = \frac{c_{rs}}{n} \tag{3.4}$$

Definition 3.2.1 *The entropy of a partition $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_R\}$ of a set is given by:*

$$\mathcal{H}(r) = -\sum_{r=1}^{R} P(r) \log\left(P(r)\right). \tag{3.5}$$

The entropy of a partition measures the level of uncertainty in the membership of elements. It is always non-negative, and will take value 0 only when there is only a single cluster (and therefore no uncertainty). It follows that the maximum entropy will correspond to the partition into n single element clusters.

Definition 3.2.2 *The mutual information is defined as:*

$$I(r;s) = \sum_{r=1}^{R}\sum_{s=1}^{S} P(r,s) \log\left(\frac{P(r,s)}{P(r)P(s)}\right). \tag{3.6}$$

Intuitively, the mutual information measures how much knowing the membership of an element of the set in partition $\mathcal{P}$ reduces the uncertainty of its membership in $\mathcal{P}'$. This is consistent with the fact that the mutual information is bounded between zero and the individual partition entropies :

$$0 \leq I(r;s) \leq \min\{\mathcal{H}(r), \mathcal{H}(s)\}, \tag{3.7}$$

and the right side equality holds if and only if one of the partitions is a refinement of the other. As for the mutual information being zero, it happens if and only if $P(r,s) = P(r)P(s)$ for all labels r and s, which is the case where knowing the label of an element in one partition doesn't give any information about its label in the other.

The mutual information can be normalized by dividing it by the average of the entropies (or equivalently, the mutual information values obtained by comparing each partition with itself).

Definition 3.2.3 *The normalized mutual information is defined as:*

$$NMI(r;s) = \frac{I(r;s)}{\frac{1}{2}(\mathcal{H}(r) + \mathcal{H}(s))}. \tag{3.8}$$

This makes it easier to interpret results when comparing results across networks of different sizes and with vastly different numbers of clusters, as the range of potential values of the standard mutual information grows as the number of clusters increases. Danon et al. [23] showed the use of the normalized mutual information in comparing clusters, and it has been widely used since.

3.2.1 Variation of Information

The variation of information between two clusterings, a criterion introduced in [59], is defined as follows.

Definition 3.2.4 *The variation of information of partitions $\mathcal{P}$ and $\mathcal{P}'$ is given by:*

$$VI(r;s) = \mathcal{H}(r) + \mathcal{H}(s) - 2I(r;s). \tag{3.9}$$

As a consequence of equation 3.7, the variation of information will be 0 if and only if the partitions are equal (up to permutations of indices of the parts), and will get bigger the more the partitions differ. It also satisfies the triangle inequality, so it is a metric in the space of partitions of any given set.

The VI is a bounded metric, and one of these bounds, dependent directly on the size n of the set, is $\log n$. For instance, the value of the VI reaches this bound when one partition separates all vertices into singleton clusters, while the other has them all in a single cluster. Another bound is based on the number of clusters of the partition, and is $2\log(\max(R, S))$.

Both these bounds can be used to normalize the VI and obtain a distance that ranges from 0 to 1. As argued in [59], normalizing by $\log n$ is appropriate as long as only the distances between partitions of the same dataset are compared. Alternatively, bounding by the maximum number of clusters, requires first setting a bound $K*$ on the number of clusters of all partitions that are going to be compared. While this might comparisons between distances obtained in different datasets that share a common bound in the number of clusters, this would be a limitation in the context of this thesis. We want to be able to evaluate the clusters obtained by any algorithm, and that means we won't be able to set a bound K^* other than the trivial value $K^* \leq n$. Therefore, normalizing by $\log n$ will be the most appropriate choice for tasks such as the evaluation of cluster stability using bootstrap resampling techniques described in chapter 5.

3.2.2 Adjusted Mutual Information

Another approach at an information based measure based on the mutual information is the adjusted mutual information [88]. It corrects the high values of mutual information due to chance by computing its expectation when the clusters are random with a fixed number of elements per cluster, similarly to how it's done for the Adjusted Rand Index (see equation 3.2). The Adjusted Mutual Information is defined as:

$$AMI(r;s) = \frac{I(r;s) - E\{I(M)|_{a,b}\}}{\max\{\mathcal{H}(r), \mathcal{H}(s)\} - E\{I(M)|_{a,b}\}}, \tag{3.10}$$

where $E\{I(M)|_{a,b}\}$ is the expected mutual information of a pair of clusterings with fixed number of clusters and cluster sizes, represented by a contingency table M of fixed row and column sums row and column sums $a = \{a_r\}$, $b = \{b_s\}$ (see table 3.1). $E\{I(M)|_{a,b}\}$ is given by the formula

$$E\{I(M)|_{a,b}\} = \sum_{i=1}^{R}\sum_{j=1}^{S}\sum_{c_{ij}=(a_i+b_j-n)^+}^{\min(a_i,b_j)}\left[\frac{c_{ij}}{n}\log(\frac{n\cdot c_{ij}}{a_i b_j})\frac{1}{n!c_{ij}!(a_i-c_{ij})!}\right. \\ \left.\frac{a_i!b_j!(n-a_i)!(n-b_j)!}{(b_j-c_{ij})!(n-a_i-b_j+c_{ij})!}\right], \tag{3.11}$$

where the notation $(a_i + b_j - n)^+$ denotes $\max(0, a_i + b_j - n)$.

As it happens with the adjusted Rand index, the adjusted mutual information will take negative values when the mutual information is lower than the expected value for partitions with the same cluster sizes. When the two partitions are identical, the AMI will be 1, and when the MI coincides with the expected value, the AMI will be 0.

3.2.3 Reduced Mutual Information

More recently, Newman et al. [66] proposed the *Reduced Mutual Information* (RMI), an improved version of the mutual information which corrects the high values given to quite dissimilar partitions in some cases. For instance, if one of the partition is the trivial one splitting the network into n clusters of one element each, the standard mutual information will always take the maximal value, even if the other is completely different. More generally, any partitions will always have maximal mutual information with all of their filtrations. This is crucial when comparing clustering algorithms, as some algorithms will output trivial partitions into single-element clusters when they fail to find a clustering structure. Therefore, it would not be possible to reliably measure the stability of these clustering methods with the standard mutual information.

Given r and s two labelings of a set of n elements, the Reduced Mutual Information is defined as:

$$\mathrm{RMI}(r;s) = I(r;s) - \frac{1}{n}\log\Omega(a,b). \tag{3.12}$$

where $\Omega(a,b)$ is a integer equal to the number $R \times S$ non-negative integer matrices with row sums $a = \{a_r\}$ and column sums $b = \{b_s\}$.

Note that, contrary to the standard mutual information, the RMI can take negative values. As described in [66], this happens only when exploiting the knowledge of one of the partitions doesn't help encoding the other more efficiently, so it will only happen with very dissimilar partitions.

The reduced mutual information can also be defined in a normalized form, in the same way the standard mutual information is, by dividing it by the average of the values of the reduced mutual information of labelings a and b with themselves:

$$\begin{aligned} \mathrm{NRMI}(r;s) &= \frac{\mathrm{RMI}(r;s)}{\frac{1}{2}[\mathrm{RMI}(r;r)+\mathrm{RMI}(s;s)]} \\ &= \frac{I(r;s)-\frac{1}{n}\log\Omega(a,b)}{\frac{1}{2}[H(r)+H(s)-\frac{1}{n}(\log\Omega(a,a)+\log\Omega(b,b))]}. \end{aligned} \tag{3.13}$$

We will use this normalized form to be able to compare more easily the results of networks with different number of nodes, as well as to compare them to other similarity measures.

3.3 An example of cluster similarity measures

We will use a very simple example to show how the different similarity measures defined in this chapter behave. It is built on a set of only four elements, defining four different partitions into clusters as seen in figure 3.1: C_1, C_2, C_3 and C_4.

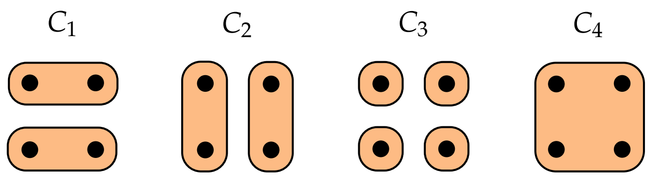

FIGURE 3.1: Four different partitions of the same set of four elements.

The entropies of these partitions are:

$$H(C_1) = H(C_2) = 1, \qquad H(C_3) = 2, \qquad H(C_4) = 0. \tag{3.14}$$

As for the mutual information, we have $I(C_1, C_2) = 0$, because the random variables associated with these clusterings are completely independent. The mutual information of any clustering with C_4 is also zero, as it contains a single cluster. Since C_3 contains only singleton clusters, $I(C_i, C_3) = H(C_i)$ for any clustering C_i, as C_3 will always be a refinement of C_i. For instance, $I(C_1, C_3) = 1$. We won't compare C_2 with C_3 and C_4, because relabeling C_1 to become C_2 doesn't change their neighbors in C_3 and C_4 so all measures evaluated for (C_1, C_3) and (C_1, C_4) are the same as when evaluated for (C_2, C_3) and (C_2, C_4), respectively.

Table 3.2 contains the values of cluster comparison measures defined in this chapter. Keep in mind that they all measure similarity (so higher values are better), except for the VI and NVI, which are distances and therefore lower values represent more similar clusters. We can quickly observe that there are substantial differences between these functions, though these effects are amplified by the characteristics of our examples, in which the differences between clusters are proportionately rather large. The shortcomings of the mutual information for measuring similarity in the context of community structures are quite apparent, though, as it gives a high value

	MI	NMI	AMI	VI	NVI	RMI	NRMI	RI	ARI
(C_1, C_2)	0	0	-1/2	2	1	-0.40	-0.66	1/3	-1/2
(C_1, C_3)	1	2/3	0	1	1/2	0.35	0.49	2/3	0
(C_1, C_4)	0	0	0	1	1/2	0	0	1/3	0
(C_3, C_4)	0	0	0	2	1	0	0	0	0

TABLE 3.2: Values of the similarity measures for the example clusterings. In this order, the measures are: Mutual Information(MI), Normalized Mutual Information (NMI), Variation of Information (VI), Normalized Variation of Information (NVI), Reduced Mutual Information (RMI), Normalized Reduced Mutual Information (NRMI), Rand Index (RI) and Adjusted Rand Index (ARI).

to the similarity between C_1 and C_3, even though C_3 doesn't actually give any insight on the communities in the network. As for the VI, it is very intuitive in the distances it assigns to each pair of clusters, which is supported by the fact that it is an actual metric (and turning the set of possible partitions of a given network into a metric space has a lot of potential when comparing algorithms or evaluating stability). As for the RMI, in this example the correction with respect to the standard MI is large, and it heavily penalizes both cases in which the mutual information is not already zero.

The effect of the correction for chance in the ARI with respect to the RI is also very notable. By its construction, the score becomes zero in the extreme cases where one of the partitions is either a single cluster, or n singleton clusters. This is very interesting in the context of community detection algorithms, because neither of those partitions (or partitions very close to them) give much information about the structure of a network. Notably, the AMI has the same values as the ARI in this example, makes it very adequate for the purposes of this thesis. The coincidence is not unexpected given that both use the same method for correcting the for chance, though it must be noted that this doesn't hold true in general, and that the ARI and AMI differ in other examples and in the experiments performed in chapter 5.

Chapter 4

Cluster Significance

This chapter is dedicated to the study of cluster significance, and includes both a description of the proposed methods, and experiments on a collection of networks to test them. We consider clusters produced by a clustering algorithm to be *significant* if there are strong connections within each cluster, and weaker connections (or fewer edges) between different clusters. This notion can be quantified and formalized by applying several community scoring functions (which we introduced in chapter 2), that gauge either the intra-cluster or inter-cluster density. Then, it can be determined that the partition of a network into clusters is significant if it obtains better scores than those for a comparable network with uniformly distributed edges.

In this chapter, we will introduce a general methodology that can be applied to any combination of graph (either weighted or unweighted, and directed or undirected) and clustering algorithm that partitions it into non overlapping clusters. It combines the use of cluster scoring functions with the generation of a randomized graph to serve as null model and put the value of each score into context. This model, defined in section 4.1.1), generates a new random graph with the same degree distribution of the original, but with the edges *rewired*, which will dissolve any existing community structure that might be present.

We then apply this methodology to a series of datasets and clustering algorithms (see 4.2 section) and evaluate the results. The discussion of the results can be found in section 4.3. The examples have been selected to be representative of different ways in which weighted edges can be used to describe a network: from the well known examples of the karate club or the Enron email networks, where edge weights count interactions, to a financial network of exchange rates where there is an edge between each pair of nodes with the correlation between the corresponding time series as its weight.

Also note that while this work focuses on weighted networks, the methodology is also valid for the unweighted case. In fact, using the unweighted version of both the edge rewiring algorithm and the scoring functions is equivalent to simply setting all the weights to one and using the methodology for the weighted case. As for directed graphs (weighted or not), this case hasn't been covered here, though both the definitions of the scoring funcions and the rewiring algorithm would suport it. The implementation of the software (see chapter 9 for details on the *clustAnalytics* software) is compatible with these graphs.

4.1 Methods

Given a partition of the set of vertices of a graph into communities produced by a clustering algorithm, the first step in evaluating their significance is computing the scoring functions described in chapter 2. If our goal is simply to compare the algorithms to each other, all on that same network, this could suffice, and we will generally have a good overview of which ones perform best[1]. However, we generally don't have a reference for what values of those functions identify good clusters on any given network, so whenever we encounter a new network and want to determine whether a significant community structure is present, this is simply not enough. We could approach it with a varied collection of clustering algorithms, and some might perform better than others, but that won't tell us if the best performing algorithms are detecting actual network communities or if they simply are better than the others at obtaining high scores out of statistical noise.

This is why we use a method to obtain randomized graphs that are similar as possible to those we are studying, but that contain no community structure whatsoever. The method is described in section 4.1.1, and consists of taking the original graph, and successively rewire the edges in a way that keeps the original degree distribution (as well as weighted degree distribution in the case of weighted graphs, and in and out degrees if the graph is directed). Then, the scores that a clustering algorithm obtained on the original network can be compared to those of this new randomized network: if they are similar we will know that there is no community structure (or at the very least, the algorithm is not capable of detecting it), but if they are significantly better, then that means that the communities in the network are actually significant.

With an efficient implementation of this randomization algorithm (see chapter 8.1), it is possible to generate a good amount of randomized graphs, and then study each score of the original graph within the distribution of scores of the random graphs. This allows us to determine whether each of the properties associated with a community structure that are captured by the scoring functions is stronger that those of the random graphs with statistical significance.

4.1.1 Randomized graph

Unweighted case

The algorithm is described in [60, 74] and uses a Markov chain where in every step two edges are selected at random and interchanged. It produces a graph with the same weighted degree sequence as the original, but otherwise as independent from it as possible. Each step of this algorithm involves randomly selecting two edges ac and bd and replacing them with the new edges ad and bc (provided they did not exist already). This leaves the degrees of each vertex a, b, c and d unchanged while shuffling the edges of the graph. Whenever the selection of ac and bd would produce self-edges (*i.e.* if $a = d$ or $b = c$) or multiple edges (*i.e.* if edges ad or bc already exist), that step is skipped and the graph is left unchanged.

[1]Note that some of the scoring functions can have limitations when the number of parts of the partitions (and therefore their size) is very different. This is studied in depth in chapter 7.

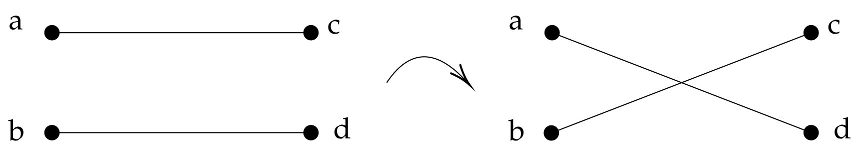

FIGURE 4.1: Rewiring of the edges for a step of the switching algorithm.

The extension of the method to directed graphs is completely straightforward. Two directed edges *ac* and *bd* (with the orientations from *a* to *c* and from *b* to *d*, respectively) are sampled, and they are rewired to *ad* and *bd* as well. In this case, both the in and out degrees of all vertices are kept constant, as expected.

Weighted case

One way to adapt this algorithm to our weighted graphs (more specifically, complete weighted graphs, with weights in $[0, 1]$) is, given vertices a, b, c and d, transfer a certain weight $\bar{w}$ from w_{ac} to w_{ad}, and from w_{bd} to w_{bc}[2]. We will select only sets of vertices $\{a, b, c, d\}$ such that $w_{ac} > w_{ad}$ and $w_{bd} > w_{bc}$, that is, we will be transferring weight from "heavy" edges to "weak" edges. If edges *ad* or *bc* are not in the network already, they will be created with weight $\bar{w}$. Note that here we consider two vertices with no edge between them equivalent to them having an edge of weight 0 (so whenever the weight of an edge is reduced to zero, that is effectively removing the edge). Naturally, if we assign weight 1 to all edges of an unweighted graph and set $\bar{w} = 1$, then we obtain the original unweighted version of the algorithm.

For any value of $\bar{w}$, the weighted degree of the vertices remains constant, but if it is not chosen carefully there could be undesirable side effects.

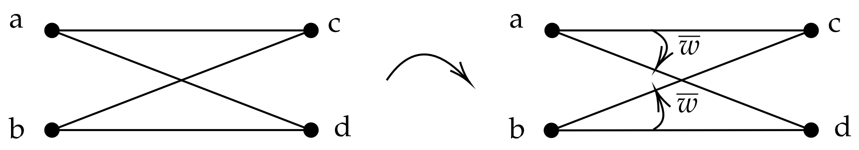

FIGURE 4.2: Rewiring of the edges for a step of the switching algorithm, in the weighted case.

Just as in the unweighted case, the method also extends naturally to weighted directed graphs without requiring any modifications.

Selection of $\bar{w}$

We distinguish between two types of weighted networks: those with an upper bound on the possible values of their edge weights given by the nature of the data (usually 1, such as in the Forex correlation network –see section 4.2.2 below), and those without (such as social networks where edge weights count the number of interactions between nodes). Networks which may include negative weights have not been studied here, so 0 will be a lower bound in all cases.

[2]Recall w_{ij} refers to the weight of the edge between vertices i and j

However, in the case of networks which are upper and lower bounded, this results in a very large number of edge weights attaining the bounds, which might be undesirable (particularly networks like the Forex network, in which very few edges, if any, have weights 0 or 1) and give new randomized graphs that look nothing like the original data.

The value of $\bar{w}$ that most closely translates the essence of the switching method for unweighted graphs would perhaps be the maximum that still keeps all edges within their set bounds. This method seems particularly suited to sparse graphs with no upper bound, because it eliminates (by reducing its weight to zero) at least an edge per iteration. Other methods without this property could dramatically increase the edge density of the graph, constantly adding edges by transferring weight to them, while rarely removing them.

However, in the case of very dense graphs such as the Forex correlation network (or any other graph similarly constructed from a correlation measure), this method results in a large number of edge weights attaining the bounds (and in the case of the lower bound 0, removing the edge), which can reduce this density dramatically.

As an alternative, to produce a new set of edges with a similar distribution to those of the original network, we can impose the sample variance (i.e. $\frac{1}{n-1}\sum_{i,j=1}^{n}(w_{ij} - m)^2$, where m is the mean) to remain constant after applying the transformation, and find the appropriate value of $\bar{w}$. The variance remains constant if and only if the following equality holds:

$$\begin{aligned} & (w_{ac} - m)^2 + (w_{bd} - m)^2 + (w_{ad} - m)^2 + (w_{bc} - m)^2 \\ = \quad & (w_{ac} - \bar{w} - m)^2 + (w_{bd} - \bar{w} - m)^2 + \\ & \quad (w_{ad} + \bar{w} - m)^2 + (w_{bc} + \bar{w} - m)^2 \\ \iff \quad & 2\bar{w}^2 + \bar{w}(-w_{ac} - w_{bd} + w_{ad} + w_{bc}) = 0. \end{aligned} \tag{4.1}$$

The solutions to this equation are $\bar{w} = 0$ (which is trivial and corresponds to not applying any transformation to the edge weights) and $\bar{w} = \frac{w_{ac}+w_{bd}-w_{ad}-w_{bc}}{2}$.

While this alternative can result in some weights falling outside of the bounds, in the networks we studied it is very rare, so it is enough to discard these few steps to obtain the desired results.

Figure 4.3 shows how the graph size decreases as the algorithm iterates with the maximum weight method, which also produces a dramatic increase in the variance. The constant variance method on the other hand does not remove any edges and the the size stays constant (as well as the variance, which is constant by definition, so the their corresponding lines coincide at 1).

However, applying the constant variance method on networks that are sparsely connected (such as most reasonably big social networks) results in a big increase in the graph size, to the point of actually becoming complete weighted graphs (see figure 4.4). Meanwhile, the maximum weight method does not significantly alter the size of the graph.

Therefore, we will use the constant variance method only for very densely connected networks, such as correlation networks, which are in fact complete weighted graphs. For sparse networks, the maximum weight method will be the preferred choice.

Note that if all edge weights are either 0 or 1, in both cases this algorithm is equivalent to the original switching algorithm for discrete graphs, as in every step the transferred weight will be one if the switch can be made without creating double edges, or zero otherwise (which corresponds to the case in which the switch cannot be made).

Number of iterations

To determine the number Tm of iterations for the algorithm to sufficiently "shuffle" the network (where m is the size of the graph, and T a parameter we select), we study the *variation of information* [59] of the resulting clustering compared to the initial one. In this case using the Louvain algorithm, though other clustering algorithms could be used instead. In chapter 3 we discuss variation of information, and other clustering similarity metrics that we use in this work, and in section 4.2.1 we detail all the clustering algorithms that we put to test.

Figures 4.3 and 4.4 show a plateau where the variation of information stops increasing after around $T = 1$ (which corresponds to one iteration per edge of the initial graph). This is consistent with the results for the original algorithm in [60] for unweighted graphs, and we can also select $T = 100$ as a value that is by far high enough to obtain a sufficiently mixed graph.

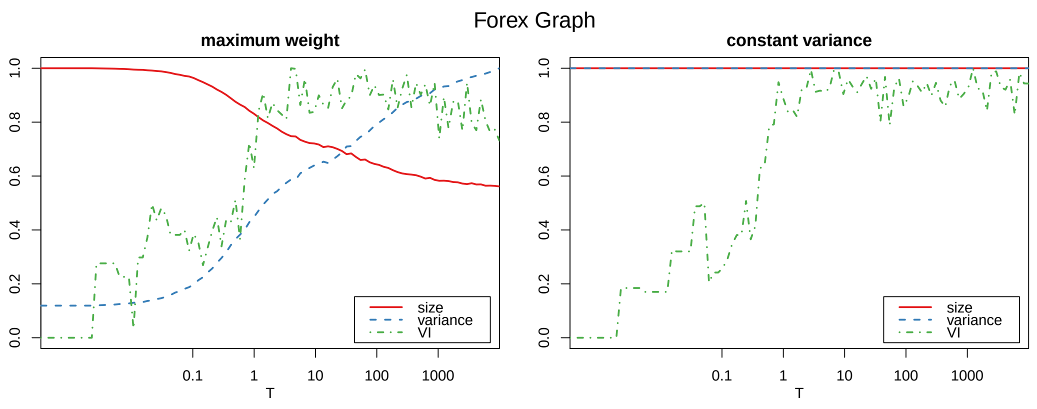

FIGURE 4.3: Normalized size, variance and variation of information for the Louvain clustering after applying the proposed algorithm on the Forex graph. Horizontal axis is on logarithmic scale.

4.2 Experimental Design

The experiments for the detection of cluster significance are performed on a selection of five weighted networks, which are each clustered with five different clustering algorithms. These are meant as examples, and as a way to show how to apply this methodology to any other network. With this approach, the only decision that must be made is whether to use the *maximum weight* or the *constant variance* versions of the rewiring algorithm. As discussed previously, only in the case of very

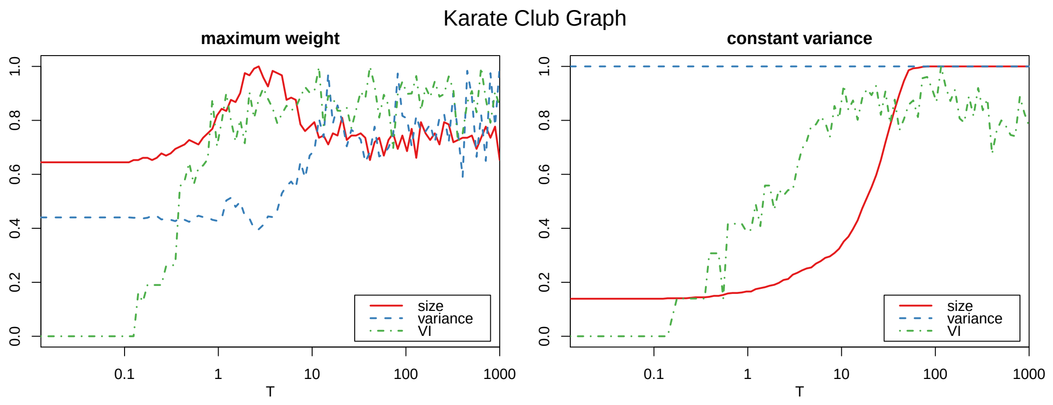

FIGURE 4.4: Normalized size, variance and variation of information for the Louvain clustering after applying the proposed algorithm on the karate club graph. Horizontal axis is on logarithmic scale.

dense or even complete neworks (such as networks built from correlation measures, like our Forex network) is the *constant variance* method recommended, while on the more common sparsely connected networks the *maximum weight* method is the default approach.

4.2.1 Clustering algorithms

In any application, the choice of the clustering algorithm will be hugely dependent on the characteristics of the dataset, as well as its size. The methods proposed here, though, can be applied to evaluate any combination of weighted graph and clustering algorithm.

We have selected five well known state-of-the-art clustering algorithms based on different approaches, and all suitable for weighted graphs. A more in-depth overview of the different types and categories of clustering algorithms is provided in section 1.2 of the introduction. They will be applied to all of the networks to then evaluate the results. The chosen algorithms are the following:

1. Louvain method [13], a multi-level greedy algorithm for modularity optimization. We use the original algorithm, without tweaking the resolution parameter (*i.e.* with resolution $\gamma = 1$).
2. Leading eigenvector method [63], based on spectral optimization of modularity.
3. Label propagation [73], a fast algorithm in which nodes are iteratively assigned to the communities most frequent in their neighbors.
4. Walktrap [70], based on random walks.
5. Spin-glass [75], tries to find communities in graphs via a spin-glass model and simulated annealing.

4.2.2 Data

The following five datasets of varying sizes and with different characteristics are used to showcase the cluster evaluation methods:

- **Zachary's karate club:** Social network of a university karate club [95]. The vertices are its 34 members, and the edge weights are the number of interactions between each pair of them. In this case, we have a "ground truth" clustering, which corresponds to the split of the club after a conflict, resulting into two clusters.
- **Forex network:** Network built from correlations between time series of exchange rate returns [76]. It was built from the 13 most traded currencies and with data of January 2009. It is a complete graph of 78 edges (corresponding to pairs of currencies) and has edge weights bounded between 0 and 1.
- **News on Corporations network:** In this network, a list of relevant companies are the nodes, while the weighted edges between them are set by the amount of times they have appeared together in news stories over a certain period of time (in this instance, on 2019-03-13). It has 899 nodes and 13469 edges.
- **Social network:** A Facebook-like social network for students from the University of California, Irvine [67]. It has 1899 nodes (students) and 20296 edges, weighted by the number of characters of the messages sent between users.
- **Enron emails:** a network composed of email communications among Enron employees [50]. The version of the dataset used here is available in the *igraphdata* R package [21], and consists of a multigraph with 184 vertices (users) and 125,409 edges, corresponding to emails between users. We convert it to a an undirected weighted graph by using as weights for the edges the number of edges in the multigraph (*i.e.* the number of emails between the corresponding users).

4.2.3 Synthetic Ground Truth Models

Another way of comparing and assessing the fit of a clustering algorithm is to compare it to a ground truth community structure if there is one, which is seldom known in reality. Alternatively one can synthetically generate a graph with a ground truth community structure. This will allow us to verify that the results of the algorithm match the expected outcome. For the particular case of time series correlation networks one can generate the time series using a suitable model that imposes a community structure with respect to correlations, such as the Vector Autoregressive (VAR) model construction in [5], and then compute the values of the edges accordingly.

A common benchmark for clustering algorithm evaluation is the family of graphs with a pre-determined community structure generated by the *l-partition model* [20, 39, 34]. It is essentially a block-based extension of the Erdös-Renyi model, with l blocks of g vertices, and with probabilities p_{in} and p_{out} of having edges within the same block and between different blocks respectively.

A more general approach is the stochastic block model (SBM) [43, 91], which uses a probability matrix P (which has to be symmetric in the undirected case) to determine probabilities of edges between blocks. P_{ij} will be the probability of having an edge between any given pair of vertices belonging to blocks i and j respectively. Then, having higher values in the diagonal than in the rest of the matrix will produce strongly connected communities. Note that subgraph induced by each community is in itself an Erdös-Renyi graph (with $p = P_{ii}$ for the community i). This model also allows having blocks of different sizes. While this model can itself be used for community detection by trying to fit it to any given graph [54], here we will simply use it as a tool to generate graphs of a predetermined community structure.

To obtain a weighted SBM (WSBM) graph, we propose a variation of the model which produces multigraphs, which can then be easily converted into weighted graphs by setting all edge weights as their corresponding edge count. In this case, probability matrix of the original SBM will be treated as the matrix of expectations between edges of each pair of blocks. Then, we simply add edges one by one with the appropriate probability (the same at each step) that will allow each weight expectation to match its defined value. By definition, the probability of the edge added at step k to join vertices i and j is given by

$$P(e_k = (i, j)) = \frac{E_{ij}}{\#\text{steps}}, \tag{4.2}$$

where E_{ij} is the expected number of edges between them given by the expectation matrix. The sum of these probabilities for all vertices must add up to one, which gives

$$\#\text{steps} = \frac{1}{2} \sum (|C_i||C_j|E_{i,j}). \tag{4.3}$$

Note that the $\frac{1}{2}$ factor is added because we are using undirected graphs, and we do not want to count edges (i, j) and (j, i) twice.

This process produces a binomially distributed weight for each edge, though these distributions are not independent, so independently sampling each edge weight from the appropriate binomial distribution would not be equivalent.

We will use a graph sampled from this model with block sizes $(40, 25, 25, 10)$, with a parametrized expectation matrix:

$$\begin{pmatrix} 0.03\lambda & 0.01 & 0.01 & 0.03 \\ 0.01 & 0.02\lambda & 0.05 & 0.02 \\ 0.01 & 0.05 & 0.02\lambda & 0.01 \\ 0.03 & 0.02 & 0.01 & 0.03\lambda \end{pmatrix}. \tag{4.4}$$

With $\lambda = 1$ the network will be quite uniform. But as λ increases, the high values in the diagonal compared to the rest of the matrix will result in a very strong community structure, which should be detected by the clustering algorithms. (see figure 4.5).

There are other possible extensions of the stochastic block model to weighted networks such as [1], which can have edges sampled from any exponential family distribution. While our approach produces Bernoulli distributed edges (which can be approximated by a Poisson distribution in most cases), the edge distributions obtained in [1] are not independent from each other, so the results are not exactly equivalent. For instance, in our case the total network weight is fixed and will not vary between samples.

4.3 Results and Discussion

To test for cluster significance of the results of a given clustering algorithm, we apply the scoring functions defined in chapter 2 to the clustering produced on the original graph and on randomized versions obtained by the method described in section 4.1.1. It should be expected that whenever the communities found by an algorithm on the original graph are significant, they will receive better scores than those found by the same algorithm on a graph with no actual community structure.

The results of computing these scores on the clustering obtained by the algorithms on each of the networks can be seen on tables 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 (recall that $\uparrow$ identifies scores for which higher is best, and $\downarrow$ means lower is best). For each combination of scoring function and algorithm, we represent its value on the original network, its mean across multiple samples of its randomized switching model, and the percentile rank of the original score in the distribution of randomized graph scores. The percentile rank is obtained after sorting the scores according to their type (in ascending order when lower is better, and descending when higher is better), so a value of 0 means that a given value is the best within the distribution. This percentile rank value serves as a statistical test of significance for each of the scores: a score is significant if its value is more extreme than most of the distribution.

It is important to note that some of the scores greatly depend on the number of clusters, and cannot adequately compare partitions in which that number differs. For instance, internal density can easily be high on small communities, while it

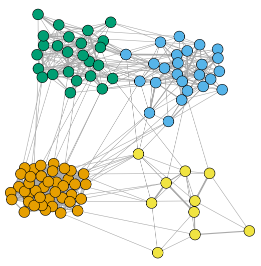

FIGURE 4.5: Plot of a sample of the stochastic block model graph with $\lambda = 15$. Vertex colors identify the blocks, which should correspond to the communities found by good clustering algorithms.

will generally take lower values on bigger ones, even when they are very well connected. This can result in networks with no apparent community structure having high overall internal density scores just because they are partitioned into many small clusters.

In comparison, scores that combine both internal and external connectivity (conductance, normalized cut, out degree fractions), clustering coefficient, and modularity suffer less from this effect and seem more adequate in most circumstances. These also happen to be the scores that are invariant under the multiplication of the weights by positive constants (see section 2.1).

We suggest focusing on the relative scores (the score of the actual network over the mean of the randomized ones) to simplify the process of interpreting the results, especially when trying to compare graphs of different nature. With relative scores, anything that differs significantly from 1 will suggest that the clustering is strong. For instance, in figure 4.6 we have the modularity of the stochastic block model for each algorithm, and for different values of the parameter λ (which will give increasingly stronger clusters). While the algorithms find results closer to the ground truth the bigger λ is, only the relative scores give us that insight. However, when comparing several clustering methods on the same network (and not simply trying to determine if a single given method produces significant results), absolute scores are more meaningful to determine which one is best.

For the weighted stochastic block model graph, the clustering algorithms get results closer to the ground truth clustering the bigger the λ parameter is, as one would expect, and for $\lambda > 30$ the results perfectly match the ground truth clustering outcome in almost all cases (a bit earlier for the Louvain, Walktrap and spin-glass cases). The relative scores match these results, and get better as λ increases as well (figure 4.6). Note that in figure 4.6, there are some jumps for the relative modularity in the spin-glass case, which are caused by the instability of this algorithm (see chapter 5). This effect is no longer present when the structure of the network is stronger ($\lambda > 8$).

In table 4.1, corresponding to $\lambda = 15$, we can see how for the Louvain algorithm, the scores are more extreme (lower when lower is better, higher when higher is better) than those of the randomized network in almost all circumstances. In the case of the leading eigenvector algorithm the scores are slightly worse, but almost all of them still fall within statistical significance (if we consider p-values < 0.05). In both cases, the only metric that is better in the random network is internal density, due to the smaller size of the detected clusters (which is why by itself internal density is not a reliable metric, as even in a network with very poor community structure it will be high for certain partitions into very small clusters that arise by chance). For both the label propagation and the Walktrap algorithms, the real network scores are not as close to the edge of the distribution of random scores, but they are still much better than the mean in all meaningful cases (the only exceptions are the internal density and edges inside, which are hugely dependent on cluster size and are therefore inadequate to compare partitions with a different number of clusters).

In the case of the karate club network (table 4.2), the label propagation algorithm gets the closest results to the ground truth clustering, and this is reflected in most scores being better than those of other methods. This does not apply to the modularity though, which is always higher for the Louvain and spin-glass, which produce identical clusters (this is to be expected, because Louvain is a method based on modularity optimization).

On the Forex graph (table 4.3), we can see that both the leading eigenvector and Walktrap algorithms produce almost identical results splitting the network into two clusters, while the spin-glass algorithm splits it into three and Louvain into four. The scores which are based on external connectivity give better results to the Walktrap and leading eigenvector, while the spin-glass partition has slightly better clustering coefficient and better modularity (with Louvain having very similar values in those two scores).

It is also important to disregard the results of the scoring functions whenever the algorithms fail to distinguish any communities and either groups the whole network together or separates each element into its own cluster (such as the label propagation algorithm on the Forex network, seen in table 4.3). In this case, the scores which are based on external connectivity will be optimum, as the cut c_s of the partition is 0, but that of course does not give any information at all. In addition, the normalized cut and conductance could be not well defined in this case, as it is possible to have a division by 0 for some of the clusters.

As for the news on corporations graph (table 4.4), the results and in particular the number of clusters vary greatly between algorithms (from 82 clusters for the Walktrap to only 2 for the label propagation). While the label propagation algorithm scores well on some measures due to successfully splitting to very weakly connected components of the network, others such as the clustering coefficient or internal density are very low. Louvain and spin-glass have very similar scores across most measures and seem to be the best, though leading eigenvector does have better conductance and normalized cut. In this case the high variation in number of clusters across algorithms that still score highly could suggest that there is not a single predominant community structure in the network.

In the Enron graph (table 4.5) Louvain also produces the best results for most scores, particularly in conductance and normalized cut, and it significantly surpasses all other algorithms while having larger clusters, with the only exception of label propagation, which partitions the network into much smaller clusters. The spin-glass algorithm stands out as having by far the worse results across all scores, even though its number of clusters (12) is the same as in leading eigenvector and similar to Louvain.

For the social network (table 4.6), the Louvain, leading eigenvector and label propagation algorithms produce the same number of clusters (with spin-glass being also very close), which allows an unbiased comparison of scores. In this case, leading eigenvector has better results for almost all scores, except for clustering coefficient and modularity, for which Louvain is again the best algorithm. This huge disparity may be explained by the fact that modularity compares edge weights to a null model that considers the degrees of their incident vertices, and does not only discriminate between internal and external edges (as most of the scoring functions do).

Overall the Louvain algorithm seems to be the best at finding significant clusters, performing consistently well on a variety of weighted networks of very different nature. It is worth noting though that there are some limitations to it (and all modularity based methods in general) in terms of resolution limit [35] that can appear when there are small communities in large networks, though there are methods to address it, such as the use of a resolution parameter [4].

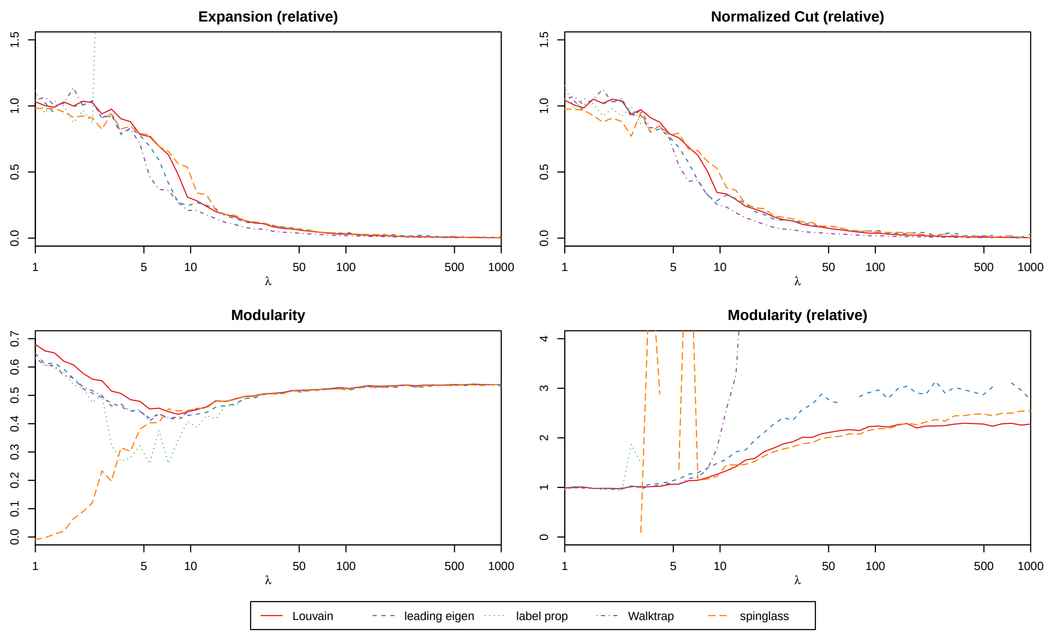

FIGURE 4.6: Scores of the weighted stochastic block model as a function of the parameter λ, for each of the algorithms

	Louvain	leading eigen	label prop	Walktrap	spin-glass	ground truth
↑ internal density	0.39|0.479(0.98)	0.384|0.362(0.36)	0.311|0.131(0)	0.39|0.33(0.17)	0.46|0.531(0.89)	3.90e-01
↑ edges inside	197|42.6(0)	194|61(0)	252|640(1)	196|22.4(0)	95.7|40.9(0)	1.96e+02
↑ av degree	11.5|5.77(0)	11.3|5.62(0)	11.7|12.8(1)	11.4|2.66(0)	8.92|5.85(0)	1.15e+01
↑ FOMD	0.38|0.0805(0)	0.38|0.0939(0)	0.38|0.45(1)	0.38|0.0359(0)	0.25|0.084(0)	3.80e-01
↓ expansion	0.67|3.51(0)	0.73|3.59(0)	0.53|0.16(1)	0.68|5.07(0)	1.94|3.47(0)	6.70e-01
↓ cut ratio	0.00931|0.041(0)	0.00997|0.0444(0)	0.00783|0.0408(0)	0.00945|0.0581(0)	0.0245|0.0401(0)	9.31e-03
↓ conductance	0.0785|0.381(0)	0.089|0.398(0)	0.0621|0.0125(1)	0.08|0.654(0)	0.159|0.375(0)	7.86e-02
↓ norm cut	0.101|0.42(0)	0.112|0.452(0)	0.0839|0.148(0)	0.103|0.699(0)	0.195|0.412(0)	1.01e-01
↓ max ODF	0.164|0.528(0)	0.215|0.552(0)	0.203|0.00875(1)	0.222|0.673(0)	0.266|0.521(0)	1.66e-01
↓ average ODF	0.142|0.544(0)	0.157|0.563(0)	0.113|0.0259(1)	0.144|0.762(0)	0.272|0.536(0)	1.41e-01
↓ flake ODF	0.72|0.996(0)	0.72|0.994(0)	0.62|0.17(1)	0.7|0.975(0)	0.9|0.995(0)	7.10e-01
↑ clustering coef	0.312|0.011(0)	0.31|0.00904(0)	0.303|0.00563(0)	0.312|0.00402(0)	0.352|0.0109(0)	3.12e-01
↑ modularity	0.454|0.307(0)	0.447|0.242(0)	0.411|0.000267(0)	0.453|0.082(0)	0.446|0.321(0)	4.54e-01
n_communities	4|7.8(0)	4|6.84(0.02)	3|1.01(1)	4|21.8(0)	5|8.51(0)	4.00e+00
VIdist_to_GT	0.122|4.45(0)	0.299|4.17(0)	0.361|1.86(0)	0.122|4.35(0)	0.521|4.52(0)	0.00e+00

TABLE 4.1: Values of scoring functions for the weighted SBM graph generated with $\lambda = 15$ (left), compared to those of their averages for 100 randomized samples (right), and its percentile rank (parentheses).

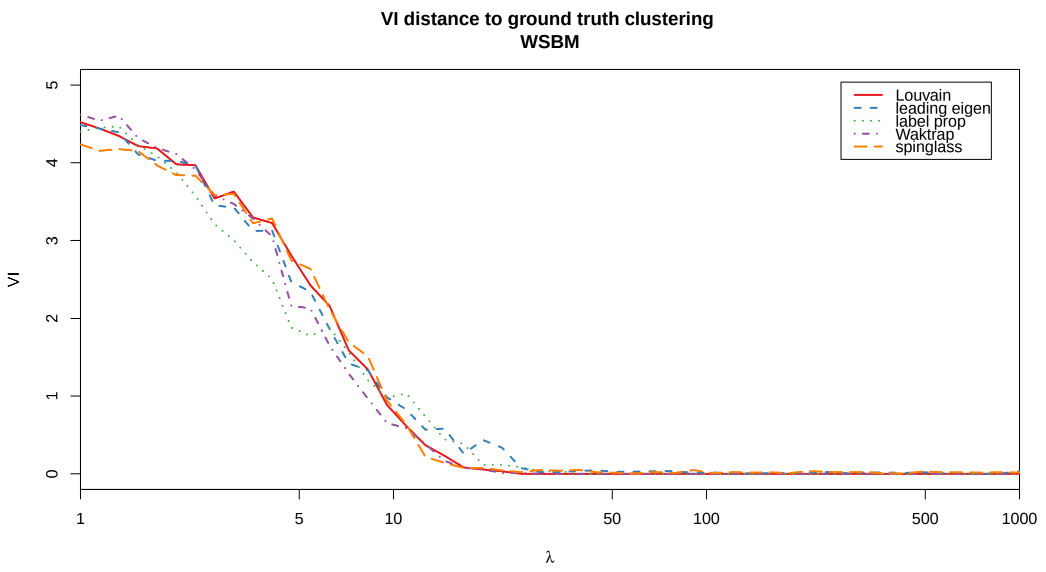

FIGURE 4.7: VI distance between the ground truth clustering and the result of each of the algorithms for the weighted stochastic block model (WSBM), as a function of the parameter λ.

	Louvain	leading eigen	label prop	Walktrap	spin-glass	ground truth
↑ internal density	1.29|1.6(0.87)	1.32|1.24(0.31)	1.07|1.17(0.6)	1.32|1.5(0.58)	1.29|1.7(0.94)	0.7574
↑ edges inside	50.4|30.5(0.01)	47.4|37.5(0.18)	83.4|106(0.43)	51.8|41.9(0.25)	50.4|28.6(0)	103.0000
↑ av degree	10.1|8.26(0)	9.94|7.98(0)	11.6|10.4(0.28)	10.1|8.44(0.1)	10.1|8.21(0)	12.1176
↑ FOMD	0.412|0.29(0.01)	0.412|0.273(0.01)	0.471|0.38(0.29)	0.382|0.299(0.18)	0.412|0.288(0.02)	0.4412
↓ expansion	1.74|2.67(0)	1.82|2.81(0)	0.971|1.99(0.0886)	1.74|2.57(0.08)	1.74|2.69(0)	0.7353
↓ cut ratio	0.0716|0.0994(0)	0.0739|0.11(0)	0.0469|0.0994(0.0253)	0.0727|0.101(0.06)	0.0716|0.0987(0)	0.0433
↓ conductance	0.149|0.257(0)	0.179|0.293(0.02)	0.0784|0.18(0.114)	0.148|0.273(0.05)	0.149|0.264(0)	0.0573
↓ norm cut	0.194|0.302(0)	0.223|0.349(0)	0.119|0.25(0)	0.194|0.328(0.01)	0.194|0.307(0)	0.1038
↓ max ODF	0.216|0.394(0)	0.246|0.396(0.04)	0.0441|0.274(0.0506)	0.211|0.382(0.06)	0.216|0.399(0.01)	0.0833
↓ average ODF	0.183|0.389(0)	0.215|0.442(0)	0.114|0.292(0.0759)	0.185|0.391(0.01)	0.183|0.397(0)	0.0871
↓ flake ODF	0.559|0.946(0)	0.588|0.941(0)	0.5|0.827(0.0759)	0.559|0.921(0.01)	0.559|0.95(0)	0.3824
↑ clustering coef	0.245|0.031(0)	0.241|0.0226(0)	0.243|0.0248(0)	0.247|0.029(0)	0.245|0.0303(0)	0.2205
↑ modularity	0.445|0.373(0.01)	0.437|0.309(0)	0.435|0.241(0)	0.44|0.307(0)	0.445|0.376(0.01)	0.3914
n_communities	4|5.23(0)	5|5.44(0.26)	3|3.55(0.36)	4|6.16(0.03)	4|5.74(0)	2.0000
VIdist_to_GT	1.2|3.18(0)	1.34|3(0)	0.747|2.23(0)	1.17|3.1(0)	1.2|3.27(0)	0.0000

TABLE 4.2: Values of scoring functions for the karate club graph (left), compared to those of their averages for 100 randomized samples (right), and its percentile rank (parentheses).

	Louvain	leading eigen	label prop	Walktrap	spin-glass
↑ internal density	0.657|0.579(0)	0.587|0.548(0)	0.52|0.52(1)	0.586|0.408(0)	0.628|0.573(0)
↑ edges inside	186|117(0.06)	434|333(0)	1562|1562(1)	434|687(0.55)	253|149(0)
↑ av degree	14.4|11(0.04)	22.3|18.3(0)	40|40(1)	22.2|22(0.49)	17|12.5(0)
↑ FOMD	0|0(1)	0|0(1)	0.5|0.5(1)	0|0.135(1)	0|0(1)
↓ expansion	12.8|14.5(0.04)	8.89|10.9(0)	-|-(-)	8.9|12.3(0.301)	11.5|13.8(0)
↓ cut ratio	0.232|0.251(0)	0.229|0.25(0)	-|-(-)	0.229|0.254(0)	0.229|0.25(0)
↓ conductance	0.492|0.575(0.06)	0.285|0.381(0)	-|-(-)	0.286|0.52(0.247)	0.415|0.526(0)
↓ norm cut	0.585|0.665(0.04)	0.419|0.51(0)	-|-(-)	0.419|0.622(0.205)	0.522|0.626(0)
↓ max ODF	0.643|0.727(0.03)	0.472|0.544(0.04)	-|-(-)	0.443|0.619(0.301)	0.567|0.688(0)
↓ average ODF	0.649|0.726(0.05)	0.445|0.543(0)	-|-(-)	0.445|0.617(0.301)	0.58|0.687(0)
↓ flake ODF	1|1(0)	1|1(0)	-|-(-)	1|1(0)	1|1(0)
↑ clustering coef	0.87|0.585(0)	0.863|0.562(0)	0.812|0.539(0)	0.864|0.421(0)	0.884|0.583(0)
↑ modularity	0.0576|0.0158(0)	0.0558|0.0136(0)	-1.6e-14|-3.73e-16(1)	0.0553|0.000591(0)	0.06|0.0187(0)
n_communities	4|4.21(0.06)	2|2.45(0)	1|1(0)	2|19.3(0.27)	3|3.6(0)

TABLE 4.3: Values of scoring functions for the Forex graph (left), compared to those of their averages for 100 randomized samples (right), and its percentile rank (parentheses). "-" indicates the degenerate cases for which a given score is not defined.

	Louvain	leading eigen	label prop	Walktrap	spin-glass
↑ internal density	0.235|0.183(0)	0.204|0.0997(0)	0.0574|0.0539(0)	0.444|0.244(0)	0.249|0.185(0)
↑ edges inside	5631|3779(0)	9051|5935(0)	21692|21771(1)	7123|2088(0)	5755|3409(0)
↑ av degree	35.7|24.9(0)	39|29(0)	48.4|48.4(1)	39.4|21.1(0)	36|24.9(0)
↑ FOMD	0.38|0.31(0)	0.392|0.35(0)	0.493|0.493(1)	0.384|0.244(0)	0.378|0.302(0)
↓ expansion	6.38|11.8(0)	4.69|9.74(0)	0.00334|-(-)	4.49|13.7(0)	6.2|11.8(0)
↓ cut ratio	0.0108|0.0219(0)	0.0108|0.0272(0)	0.000558|-(-)	0.00791|0.0205(0)	0.0106|0.02(0)
↓ conductance	0.205|0.391(0)	0.154|0.271(0)	0.000702|-(-)	0.22|0.57(0)	0.202|0.412(0)
↓ norm cut	0.259|0.472(0)	0.219|0.384(0)	0.111|-(-)	0.27|0.636(0)	0.257|0.487(0)
↓ max ODF	0.229|0.509(0)	0.178|0.481(0)	0.00111|-(-)	0.277|0.7(0)	0.199|0.534(0)
↓ average ODF	0.241|0.518(0)	0.185|0.425(0)	0.00113|-(-)	0.266|0.701(0)	0.238|0.54(0)
↓ flake ODF	0.759|0.999(0)	0.605|0.999(0)	0.00445|-(-)	0.727|1(0)	0.747|0.999(0)
↑ clustering coef	0.296|0.233(0)	0.152|0.221(1)	0.0437|0.217(1)	0.222|0.187(0)	0.289|0.226(0)
↑ modularity	0.373|0.149(0)	0.334|0.122(0)	0.000276|-1.13e-14(0)	0.342|0.0973(0)	0.372|0.152(0)
n_communities	14|8.1(1)	10|2.3(1)	2|1(1)	82|76.7(0.6)	16|12.4(1)

TABLE 4.4: Values of scoring functions for the News on Corporations graph (left), compared to those of their averages for 100 randomized samples (right), and its percentile rank (parentheses).

	Louvain	leading eigen	label prop	Walktrap	spin-glass
↑ internal density	109|43(0)	57.3|24.4(0.02)	158|8.93(0)	65|7.23(0)	26.2|39.5(1)
↑ edges inside	14828|10482(0.11)	11351|23434(0.67)	7350|111195(1)	10451|121642(1)	6612|4502(0.01)
↑ av degree	1047|583(0)	963|650(0.06)	902|1255(1)	1001|1322(1)	440|482(0.93)
↑ FOMD	0.44|0.26(0)	0.391|0.267(0.06)	0.386|0.488(1)	0.413|0.5(1)	0.163|0.195(0.98)
↓ expansion	113|370(0)	155|357(0)	186|36.4(1)	136|-(-)	417|420(0.35)
↓ cut ratio	0.787|2.58(0)	1|3.05(0)	1.13|17(0)	0.851|-(-)	2.66|2.55(0.91)
↓ conductance	0.113|0.414(0)	0.162|0.407(0)	0.224|0.0167(1)	0.162|-(-)	0.564|0.534(0.9)
↓ norm cut	0.133|0.471(0)	0.182|0.481(0)	0.24|0.15(1)	0.177|-(-)	0.611|0.566(1)
↓ max ODF	0.248|0.608(0)	0.303|0.569(0.0426)	0.32|0.00558(1)	0.29|-(-)	0.824|0.695(1)
↓ average ODF	0.207|0.6(0)	0.308|0.578(0)	0.353|0.0277(1)	0.296|-(-)	0.713|0.705(0.63)
↓ flake ODF	0.935|0.998(0)	0.957|0.997(0)	0.967|0.423(1)	0.962|-(-)	0.989|0.999(0)
↑ clustering coef	0.0837|0.194(1)	0.0902|0.19(1)	0.123|0.19(1)	0.112|0.191(1)	0.0351|0.195(1)
↑ modularity	0.673|0.27(0)	0.615|0.202(0)	0.615|0.1(0)	0.639|0(0)	0.226|0.251(0.93)
n_communities	10|6.22(1)	12|9.54(0.74)	31|1.94(1)	19|1(1)	12|12.8(0.27)

TABLE 4.5: Values of scoring functions for the Enron graph (left), compared to those of their averages for 100 randomized samples (right), and its percentile rank (parentheses).

	Louvain	leading eigen	label prop	Walktrap	spin-glass
↑ internal density	45.3|3.17(0)	15.8|1.22(0)	98|1.19(0)	1.54|0(0)	4.37|5.48(1)
↑ edges inside	255578|86181(0)	1076946|223193(0)	5690431|1230459(0)	896171|0(0)	52171|113572(1)
↑ av degree	3680|342(0)	3639|499(0)	6572|1298(0)	1663|0(0)	577|337(0)
↑ FOMD	0.404|0.217(0)	0.375|0.287(0.1)	0.497|0.5(1)	0.23|0(0)	0.131|0.222(1)
↓ expansion	1594|478(1)	1615|400(1)	148|0.337(1)	2603|649(1)	3146|481(1)
↓ cut ratio	0.906|0.354(1)	1.27|0.374(1)	0.856|0.0422(1)	1.83|0.342(1)	1.81|0.401(1)
↓ conductance	0.307|0.54(0)	0.31|0.434(0.1)	0.0191|0.00021(1)	0.594|1(0)	0.858|0.49(1)
↓ norm cut	0.324|0.6(0)	0.378|0.535(0.1)	0.127|-(0.667)	0.658|1(0)	0.894|0.561(1)
↓ max ODF	0.0311|0.626(0)	0.0371|0.553(0)	0.00862|0.00189(0.9)	0.432|1(0)	0.857|0.58(1)
↓ average ODF	0.334|0.668(0)	0.478|0.563(0.1)	0.187|0.000483(1)	0.636|1(0)	0.924|0.822(1)
↓ flake ODF	0.767|0.996(0)	0.763|0.993(0)	0.209|0.0839(0.7)	0.864|0.999(0)	0.974|0.996(0)
↑ clustering coef	0.00417|0.147(1)	0.00127|0.147(1)	0.00179|0.15(1)	0.00031|0(0)	0.0038|0.143(1)
↑ modularity	0.461|0.122(0)	0.277|0.0676(0)	0.0778|0.00077(0)	0.0502|-0.00268(0)	-0.00289|0.119(1)
n_communities	26|13.7(1)	26|7.7(1)	26|3.3(1)	822|1899(0)	24|23.4(0.4)

TABLE 4.6: Values of scoring functions for the social network graph (left), compared to those their averages for 10 randomized samples (right), and its percentile rank (parentheses)

Chapter 5

Cluster Stability

Evaluating stability is an important part in the validation process of community detection methods. If the ultimate goal is to find the underlying community structure of a network (in case there is one), the process must be robust and not be too sensible to small variations in the data. Cluster stability has been studied more widely for algorithms that work on Euclidean data (as opposed to networks, weighted or not). For instance, [90] uses both resampling and adding noise to generate perturbed versions of the data. [42] introduces bootstrap resampling (with and without perturbation) to evaluate cluster stability. Also for Euclidean data, [87] introduce a systematic approach for cluster evaluation that combines cluster quality criteria with similarity and dissimilarity metrics between partitions, and searches for correlations between them. Our approach consists of a bootstrap technique with perturbations adapted to clustering on networks, that resembles what Hennig does for Euclidean data. That is, the set of vertices is resampled multiple times, and the clustering algorithms are applied to the resulting induced networks. In this case, the perturbations are applied to the edge weights after resampling the vertices, but the standard bootstrap method without perturbation can be used on all networks, weighted or not.

To compare how the clusters of the resampled networks differ from the originals, we use four measures, which are defined and discussed in more detail in chapter 3. The adjusted Rand index [45] is a similarity measure that counts the rate of pairs of vertices that are in agreement on both partitions, corrected for chance. Additionally, we use measures derived from information theory to compare partitions such as the recently introduced Reduced Mutual Information [66] as well as the Adjusted Mutual Information [88], which correct some of the issues with the original mutual information or its normalized version [23]. For example, giving maximal scores when one of the partitions is trivial, which in our case would mean that failed algorithms that split most of the network into single vertex clusters would be considered very stable. Other attempts at providing adjusted versions of the mutual information include [26], [89] and [96]. The other information theory measure we employ for the sake of comparison and control is the Variation of Information (VI) [59]. The VI is a distance measure (as opposed to a similarity measure, like the Rand index and mutual information) that actually satisfies the properties of a proper metric.

The proposed cluster evaluation method is tested on the same networks used in chapter 4 to evaluate significance.

5.1 Methods

The evaluation of cluster stability is performed by resampling the network using bootstrap with perturbation, and applying the clustering algorithms to all these instances. The similarity metrics (Variation of Information, Reduced Mutual Information, and Adjusted Rand Index) are then used to quantify how much the resulting partitions differ from the initial partition for the original network. Values indicating a very high similarity (high RMI and ARI, low VI) will mean that the clusters are stable.

5.1.1 Bootstrap with perturbation

Non-parametric bootstrap, with and without perturbation or "jittering", has been used to study the stability of clusters of euclidean data sets [42]. For graphs, bootstrap resampling can be done on the set of vertices, and then build the resampled graph with the edges that the original graph induces on them (*i.e.* two resampled vertices will be joined by an edge if and only if they were adjacent in the original graph, with the same weight in the case of weighted graphs). As for adding noise to avoid duplicate elements, it can be added to the edge weights. We suggest generating that noise from a normal distribution truncated to stay within the bounds of the edge weights of each graph (which means it can be truncated on one or both sides depending on the graph).

Then, to deal with copies of the same vertex on the resampled graph, it seems necessary to add heavy edges between them to reflect the idea that a vertex and its copy should be similar and well connected between each other. Not doing so would incentivize the clustering methods to separate them in different clusters, because they generally try to separate poorly connected vertices. We can distinguish two cases:

- Graphs with edge weights built from correlations or other similar graphs which by their nature have a specific upper bound on the edge weights (usually 1): We assign the value of the upper bound to the edge weight. After applying the perturbation, this will result in a weight which will be close to that upper bound.
- Other weighted graphs, where no particular upper bound to the edge weights is known: To assign these edges very high weights (to reflect the similarity that duplicate vertices should have in the resampled network) within the context of the network, one option is to sample values from the highest weights (*e.g.* the top 5%) of the original edge set.

5.2 Results and Discussion

Using the non-parametric bootstrap method described in section 5.1.1, we resample the networks 999 times ($R = 999$), apply clustering algorithms to them, and compare them to their original clustering with the metrics from chapter 3. Stable clusterings are expected to persist through the process, giving small mean values of the variation of information, and high (close to 1) values of the normalized reduced mutual information and the Rand index. The results of the same method applied to the randomized versions of each network (see section 4.1.1) are also included, to have reference values for the stability of networks where there is no community structure. If the values of the clustering similarity measures, for the original and randomized networks, happened to be close together, that would suggest that the chosen algorithm produces a very unstable clustering on the network.

We observe in table 5.1 that for the stochastic block model example graph, all algorithms except for spin-glass produce very stable clusters, which is consistent with the fact that we chose parameters to give it a very strong community structure. Meanwhile, clustering algorithms applied to the Zachary and Forex networks (tables 5.2 and 5.3) produce clusters which are not as stable, but still much better than their baseline randomized counterparts. Note that the stability values for the label propagation algorithm in the Forex network (table 5.3) should be ignored, as in that instance the output is a single cluster (see table 4.3) which does not give any information. It is clear that while it works on less dense networks, the label propagation algorithm is not useful for complete weighted networks and it fails to give results that are at all meaningful.

On the news on corporations graph (table 5.4) spin-glass is again the most unstable algorithm, with results for the RMI and ARI (which are both close to 0) that suggest that the clusters of the original network and all the resampled ones are completely unrelated. In this case the label propagation algorithm is the most stable, while the rest of the algorithms are not as good. This might be in part explained by the fact that its clusters are much bigger than in other networks, which allows them to remain strongly connected after small perturbations.

Finally, we observe that algorithms on the Enron graph (table 5.5) produce the most unstable clusters out of all that were tested, which would suggest that the network does not have a single prevalent clustering structure that can be consistently detected, at least in the weighed graph configuration that we tested.

As a general remark on stability observed from all resulting experiments is that the spin-glass algorithm is the most unstable across the networks we tested, which are a diverse representation of different kinds of weighted networks.

	original					randomized				
	VI	AMI	RMI	ARI	#clust	VI	AMI	RMI	ARI	#clust
Louvain	0.19	0.72	0.73	0.75	5.00	0.65	0.31	0.20	0.24	7.26
leading ev	0.19	0.71	0.72	0.76	4.67	0.72		0.09	0.17	7.03
label prop	0.22	0.58	0.67	0.77	5.48	0.26	0.00	0.00	0.18	4.40
Walktrap	0.14	0.78	0.80	0.81	4.99	0.70	0.29	0.07	0.13	9.60
spinglass	0.39	0.49	0.45	0.53	5.74	0.76	0.21	0.08	0.19	7.82

TABLE 5.1: Mean values of the metrics after bootstrapping with $R = 999$, for both the WSBM graph and its randomized counterpart, for all tested clustering algorithms

	original					randomized				
	VI	AMI	RMI	ARI	#clust	VI	AMI	RMI	ARI	#clust
Louvain	0.27	0.62	0.66	0.67	5.57	0.49	0.49	0.39	0.42	5.49
leading ev	0.28	0.60	0.64	0.72	5.66	0.55	0.41	0.30	0.39	5.79
label prop	0.26	0.51	0.63	0.71	5.01	0.50	0.16	0.36	0.33	4.86
Walktrap	0.27	0.62	0.68	0.71	5.94	0.42	0.39	0.49	0.47	6.57
spinglass	0.62	0.29	0.20	0.38	6.34	0.62	0.28	0.22	0.35	6.04

TABLE 5.2: Mean values of the metrics after bootstrapping with $R = 999$, for both the Zachary graph and its randomized counterpart, for all tested clustering algorithms

	original					randomized				
	VI	AMI	RMI	ARI	#clust	VI	AMI	RMI	ARI	#clust
Louvain	0.32	0.52	0.52	0.55	3.13	0.66	0.17	0.04	0.15	3.34
leading ev	0.25	0.48	0.47	0.62	2.34	0.40	0.14	0.10	0.19	2.18
label prop	0.00				1.00	0.00			1.00	1.00
Walktrap	0.24	0.49	0.49	0.62	2.41	0.61	0.00	0.00	0.00	31.60
spinglass	0.28	0.57	0.54	0.60	3.07	0.63	0.13	0.08	0.18	3.10

TABLE 5.3: Mean values of the metrics after bootstrapping with $R = 999$, for both the Forex graph and its randomized counterpart, for all tested clustering algorithms

	original					randomized				
	VI	AMI	RMI	ARI	#clust	VI	AMI	RMI	ARI	#clust
Louvain	0.34	0.45	0.42	0.59	27.51	0.70	0.06	0.07	0.08	23.47
leading ev	0.27	0.40	0.40	0.75	27.50	0.32	0.15	0.24	0.24	23.40
label prop	0.08	0.56	0.79	0.86	21.16	0.03	0.00	-0.00	0.00	16.62
Walktrap	0.30	0.55	0.42	0.50	85.00	0.18	0.21	0.06	0.07	91.70
spinglass	0.63	0.09	-0.12	0.13	30.65	0.94	0.00	-0.14	0.00	26.01

TABLE 5.4: Mean values of the metrics after bootstrapping with $R = 999$, for both the News graph and its randomized counterpart, for all tested clustering algorithms

	original					randomized				
	VI	AMI	RMI	ARI	#clust	VI	AMI	RMI	ARI	#clust
Louvain	0.30	0.64	0.60	0.54	10.34	0.63	0.20	0.09	0.12	5.48
leading ev	0.42	0.53	0.46	0.40	12.78	0.50	0.21	0.14	0.20	5.96
label prop	0.34	0.60	0.54	0.48	16.36	0.02	0.21	0.22	0.48	1.44
Walktrap	0.32	0.65	0.58	0.51	17.08	1.36	0.15	0.00	0.03	170.93
spinglass	0.93	0.04	-0.20	0.03	12.63	0.74	0.17	0.08	0.13	9.36

TABLE 5.5: Mean values of the metrics after bootstrapping with $R = 999$, for both the Enron graph and its randomized counterpart, for all tested clustering algorithms

Chapter 6

Network models with multi level community structures

6.1 Introduction

In this chapter we introduce network models with a hierarchical community structure where the relative strength of each level of cluster is parametrized. That is, not only the density of edges is different between vertices sharing the same cluster, but there can also be sub-clusters of higher density within a cluster, and the relation between this density can be controlled with a parameter. This allows to generate graphs where the sub-clusters are emphasized, or on the contrary, others where these lower level clusters cannot be distinguished from the rest of the larger clusters where they are contained. These models will be used in chapter 7 to determine whether cluster scoring functions are biased towards the lower or upper levels of clusters within the hierarchical structure.

We propose two approaches to obtain this multi-level structure. The first one is based on the Stochastic Block Model (SBM), which samples potential edges uniformly with a given probability for each level of clusters. Alternatively, we present a model based on preferential attachment with the same multi-level structure, but with scale-free degree distributions. It uses the same principles as the original Barabási-Albert model, but with the addition of the community structure.

6.2 Related work

Hierarchical or multi-level stochastic block models, have been mostly used for community detection by trying to fit them to any given graph ([54] and [68] are good examples of this technique). Here we use the SBM and multilevel SBM for the purpose of generating networks with predetermined community structure. Of utmost importance is the Barabási-Albert preferential attachment model with community structure construction of [41], which motivates our own construction of a multilevel block model with preferential attachment to produce networks with community structure and degree distribution ruled by a power law. The proof of this latter fact is a novel contribution of this thesis.

There is scarce literature in the use of SBM and multilevel SBM, let alone the preferential attachment model for constructing synthetic ground truth communities in networks as benchmarks for testing clustering algorithms. A notable exception is [69], that uses SBM in this sense to explore how metadata relate to the structure of the network when the metadata only correlate weakly with the identified communities. This thesis contributes to the literature of clustering assessment through probabilistic generative models of communities in networks.

6.3 Stochastic Block Model (SBM)

A potential benchmark for clustering algorithm evaluation is the family of graphs with a pre-determined community structure generated by the planted l-partition model [20, 39, 34]. It is essentially a block-based extension of the well known Erdös-Renyi model, with l blocks of g vertices, and with probabilities p_{in} and p_{out} of having edges within the same block and between different blocks respectively.

The generalization of this idea is the stochastic block model, which allows blocks to have different sizes, as well as setting distinct edge probabilities for edges between each pair of blocks, and for internal edges within each block [43, 91]. These probabilities are commonly expressed in matrix form, with probability matrix P where P_{ij} is the probability of having an edge between each pair of vertices from blocks i and j. Then, graphs generated by probability matrices where the values in the diagonal are larger than the rest, will have very strong and significant clusters, while more uniform matrices will produce similarly uniform (and therefore poorly clustered) graphs.

Note that this matrix is symmetric for undirected graphs. Of course, to use this model for the undirected case, the only modification needed is to separately sample the edges in both directions for all pairs of vertices. If the probability matrix is not symmetric, then the density of edges joining any two communities may be different in each direction.

Also note that this is a model for unweighted graphs. It would be possible to derive a weighted version by sampling edge weights from a discrete or continuous probability distribution, and replacing the matrix P with an appropriately chosen matrix (or matrices) of parameters. For example, the Poisson distribution could be used, and then, for each pair v_i, v_j of vertices, one would take a sample with parameter λ_{ij} taken from a matrix of λ parameters, and that would give the edge weight (a zero would mean no edge). That matrix would be structured just like the probability matrix of the standard stochastic block model, but in this case each element would be the expected average edge weight (counting the lack of edge as a zero) between the corresponding two communities. However, we will focus on the unweighted model for simplicity.

6.4 Multi-level stochastic block model

To generate networks with multi-level community structures, we propose a variation of the stochastic block model with two levels of communities. A more general model with an arbitrary number of levels (where the hierarchical structure of the network can be any tree) was proposed by Cohen-Addad and Kanade [19], but for the purposes of this thesis, a simpler model with few parameters that can be easily described is more convenient. This two-level variation of the stochastic block model is defined as follows:

- C_1,..., C_n are the first level of communities.
- each community C_i is split into $C_{i_1}, ..., C_{i_{m_i}}$ sub-communities.
- d_1 is the edge probability within sub-communities.
- d_2 is the edge probability within communities (but with different sub-communities).
- d_3 is the edge probability outside communities.

Consequently, the model takes as parameters the upper and lower level block size lists, as well as the edge probabilities d_1, d_2, d_3. Note that the lower level partition $\mathcal{P}_l = \{C_{i_j}\}$ is a refinement of the upper level partition $\mathcal{P}_u = \{C_i\}$. A representation of this multi-level stochastic block model is shown in figure 6.1, where the upper level is the light coloured region whilst the lower level is the darker region, and the edge probabilities are clearly identified.

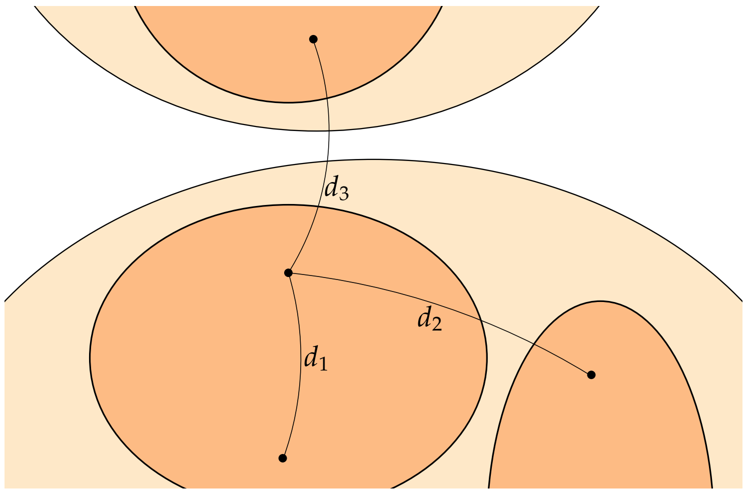

FIGURE 6.1: Multi-level stochastic block model.

The resulting model can itself be expressed as a standard (single-level) stochastic block model, using the lower-level communities as blocks, and with probability matrix as seen in figure 6.2, which means it is actually a particular case of the standard stochastic block model. This idea can be extended to define stochastic block models with a hierarchical or multi-level structure of any number of levels, but in our experiments we have used 2 levels for the sake of simplicity.

By varying the relation between d_1 and d_2, we can give different strength to the multi-level community structure. If d_2 is close to or equal to d_1, the lower level of communities will not be distinguishable and will merge into the upper level. If d_1 is instead much larger, then the smaller communities will become more visible. And similarly, if d_2 is not significantly larger than d_3, the upper level structure will be weak.

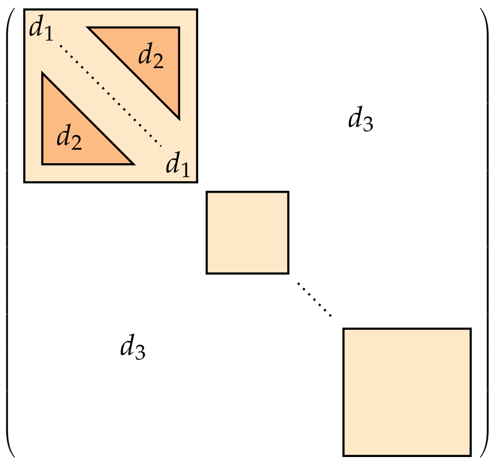

FIGURE 6.2: Probability matrix of the multi-level stochastic block model.

We won't consider cases where $d_1 < d_2$ and $d_2 < d_3$, as the resulting structure would be closer to a k-partite graph (with k being the number of blocks on the corresponding level) than to a community structure (*i.e.* blocks would be more connected to each other than to themselves).

6.4.1 Preferential attachment model

An alternative way to generate benchmark graphs that resemble real networks is the Barabási-Albert preferential attachment model [8]. In this model, new vertices are added successively, and at each addition, a fixed number m of new edges are added connecting the new vertex to the rest of the network, with probabilities of linking to each of the existing vertices proportional to their current degree.

An extension to this model to include communities has been explored in [41, 47], which consists on basing the preferential attachment not only on the degree of the vertices, but also on a fitness matrix that depends on their labels (*i.e.* which block each of the vertices belongs to). This construction can be seen as a weighted preferential attachment, with weights being the affinities between vertex labels. We propose a variation of this preferential attachment block model where new half edges are first randomly assigned a community (with probabilities given by the affinity matrix), from which we then sample the vertex with standard preferential attachment.

The model consist of a sequence of graphs $\{G_t = (V_t, E_t) : t \geq t_0\}$, where $V_t = [t]$ (hence there are t vertices) and $|E_t| = mt$, no self-loops, with the possibility of having parallel edges, and communities $C = (C_1, ..., C_r)$, with distribution determined by the following parameters:

- $m \geq 1$: fixed number of edges added connecting each new vertex to the graph.
- $r \geq 1$: fixed number of communities.

- β: $r \times r$ fitness matrix that determines the probability of edges connecting each pair of communities.
- $p = (p_{c_1}, p_{c_2}, ..., p_{c_r})$: vertex community membership probability distribution.
- $t_0 \geq 1$: initial time (which is also the order of the G_0 graph).
- G_{t_0}: initial graph

Given graph G_t and community memberships $c = (c_1, ..., c_t)$, the graph G_{t+1} is generated as follows: A new vertex $t + 1$ is added with community membership sampled from the probability vector p, and with m half-edges attached. To obtain the other ends of these half edges, m communities are sampled with replacement with probabilities weighted by $\beta_{c_t,1}, ..., \beta_{c_t,r}$, and for each of them, a vertex is sampled within the community with preferential attachment (that is, with probabilities proportional to the degree of each vertex). The initial graph G_{t_0} has to be chosen carefully as we will explain below, and further we will show that the resulting network has a scale-free degree distribution.

6.4.2 Generating the initial graph

The initial graph G_{t_0} is crucial for computing G_t in discrete time, because early vertices have a higher probability to end up with a high degree than later ones due to the nature of the preferential attachment model. To avoid bias with respect to any of the communities, we assign the initial vertices with the same vector of probabilities $p = (p_{c_1}, ..., p_{c_{|\mathcal{P}|}})$ used later when adding new vertices. Then, we sample $t_0 \times m$ edges between them with probabilities proportional to the fitness matrix. In order for G_{t_0} to be able to have enough edges without parallel or self edges, we need $t_0 - 1 > 2m$ (if we have the equality, G_{t_0} will be an t_0-clique). It is suggested to use $t_0 = 5m$ to produce a graph that is not too close to a clique, because G_{t_0} is generated with community structure according to a fitness matrix, but the closer it is to a clique, the less this structure matters (at least if we sample without replacement to avoid parallel edges).

6.4.3 Degree distribution

Now the goal is to analytically obtain the expected degree distribution of the model. Recall that for the original Barabási-Albert model it is done as follows (shown in [2]). Let k_i be the expectation of the degree of node i. Then,

$$\frac{dk_i}{dt} = m \frac{k_i}{\sum_{j=1}^{N-1} k_j} \tag{6.1}$$

The sum in the denominator is known, since the total amount of edges at a certain point is fixed by the model, so $\sum_{j=1}^{N-1} k_j = 2mt - m$, and thus $\frac{dk_i}{dt} = \frac{k_i}{2t-1}$. For large t the -1 term can be neglected, and hence we get $\frac{dk_i}{k_i} = \frac{dt}{2t}$. Integrating and taking exponents we get $Ck_i = t^{\frac{1}{2}}$, for some constant C. By definition, $k_i(t_i) = m$, and hence $C = \frac{t_i^{\frac{1}{2}}}{m}$. Substituting this value into the last equation, we obtain the desired power law:

$$k_i(t) = m\left(\frac{t}{t_i}\right)^{\frac{1}{2}} \tag{6.2}$$

For our block model the argument goes as follows. We will assume k_i belongs to the first community to simplify the notation (by symmetry the same principles work on all communities). The rate at which the degree grows is given by

$$\frac{dk_i}{dt} = \left(\sum_{j=1}^{r} p_j\beta_{j1}\right)\frac{k_i}{\sum_{j\in C_1} k_j}m, \tag{6.3}$$

where the denominator is the expected sum of degrees in community 1:

$$E\left[\sum_{j\in C_1} k_j\right]_t = tm\left(p_1 + \sum_{j=1}^{r} p_j\beta_{j1}\right). \tag{6.4}$$

Then, equation (6.3) becomes

$$\frac{dk_i}{dt} = \left(\sum_{j=1}^{r} p_j\beta_{j1}\right)\frac{k_i}{t(p_1 + \sum_{j=1}^{r} p_j\beta_{j1})} = A\frac{k_i}{t}, \tag{6.5}$$

where $A = (\sum_{j=1}^{r} p_j\beta_{j1})/(p_1 + \sum_{j=1}^{r} p_j\beta_{j1})$ is a constant that depends only on the parameters of the model (p_1, ..., p_r and β). Then, we integrate the equation:

$$\int \frac{dk_i}{k_i} = \int A\frac{dt}{t} \tag{6.6}$$

to obtain that $k_i = Ct^A$, for some constant C. Now, using the fact that $k_i(t_i) = m$, we obtain the value of the constant as $C = \frac{m}{t_i^A}$, which results in k_i being given by

$$k_i = m\left(\frac{t}{t_i}\right)^A. \tag{6.7}$$

By an argument similar to one in [2, Sec. VII.B] we use equation (6.7) to write the probability that a node has degree $k_i(t)$ smaller than k as

$$P(k_i(t) < k) = P\left(t_i > \frac{tm^{1/A}}{k^{1/A}}\right) = 1 - \frac{tm^{1/A}}{(t+m_0)k^{1/A}}. \tag{6.8}$$

The last equality obtained from assuming that nodes are added to the network at equal time intervals, and in consequence $P(t_i) = 1/(t+m_0)$. Using equation (6.8) one can readily conclude that the degree distribution of the community $P(k) = \partial P(k_i(t) < k)/\partial k$ is asymptotically given by: $P(k) \sim 2m^{1/A}k^{-\gamma}$, where $\gamma = 1/A + 1$.

Chapter 7

Identifying bias in cluster quality metrics

7.1 Introduction

Determining how meaningful the results of cluster analysis are can often be difficult, as well as choosing which clustering algorithm better suits a particular network. In many cases, various clustering algorithms can give substantially different results when applied to the same network. This is not only due to the limitations or particularities of the algorithms, but also to some networks possibly having multiple coexisting community structures.

This chapter is motivated by the results of the cluster significance evaluation performed on chapter 4. There, we use many community scoring functions that evaluate one or multiple of the characteristics that we identify with strong and significant clusters, such as high internal connectivity and low external connectivity. However, when evaluating the results of clustering algorithms, it is commonly observed that some algorithms tend to group the network vertices into larger clusters, while others seem to favor smaller ones. And some of the scoring functions are observed to also favor partitions of a certain size, even to the point of giving them better scores than the ground truth partition in some cases where it is present and known. Then, if we are to use these functions to study the results of clustering algorithms on networks for which we don't have any prior kowledge of any community structure, it is important to know if they can potentially be biased towards communities of a certain size.

Our goal is to study how existing cluster or community quality metrics behave when comparing several partitions of the same network, and determine whether they properly show which of them better match the properties of a good clustering, or if they are biased in favour of either finer or coarser partitions. Knowing this is essential, for there could be cases in which a cluster quality metric simply scores better than another because it tends to find smaller or larger clusters, with less regard for other properties, and not because it is better at revealing the structure of the network.

We selected a few popular cluster quality metrics, both local, which assign a score to evaluate each individual cluster (*e.g.*, conductance, expansion, cut ratio, and others), and global, which evaluate the partition as a whole (*e.g.*, modularity). We also propose a new metric, the density ratio, which combines some of the ideas behind other metrics, in an attempt to improve over some of their limitations, and more particularly, to avoid bias caused by the number of clusters of the partitions.

Our analysis is split in two parts. In the first part, we use the stochastic block models [43, 91] to generate networks of predetermined community structures, and we then compute the correlation of each metric to the size and number of communities. On the second part, we define networks with a two level hierarchical community structure (with the lower partition being a refinement of the upper partition), where one additional parameter controls the strength of one level respect to the other. Then, by varying this parameter and evaluating the quality metrics on both levels, we can see if certain metrics are biased towards finer or coarser partitions. These networks with multi-level community structure are implemented on two different models that have been defined in chapter 6: a stochastic block model and a preferential attachment model which results in networks with a scale-free degree distribution.

7.1.1 Related work

We briefly survey some of the studies in cluster quality metrics and analyses of their performance which are relevant to this work. A milestone is the work by Yang and Leskovec [94], where they analysed and classified many popular cluster scoring functions based on combinations of the notions of internal and external connectivity. More recently, Emmons et al. in [30] study the performance of three quality metrics (modularity, conductance and coverage), as well as that of various clustering algorithms by applying them to several well known benchmark graphs. For the case of modularity, it was already shown in [35] to have a resolution limit below which small and strong communities are merged together, even when that goes against the intuition of what a proper clustering should be. This limit depends on the total amount of edges in the network, in such a way that it is more pronounced the larger the network and the smaller the community. Almeida et al. [3] do a descriptive comparison of the behaviour of five cluster quality metrics (modularity, silhouette, conductance, coverage, and performance) for four different clustering algorithms applied over different real networks, to conclude that none of those quality metrics represents the characteristics of a well-formed cluster with a good degree of precision. Chakraborty et al. [17] survey several popular metrics, comparing their application to networks with ground truth communities to the results of a selection of clustering algorithms, though the potential of bias relative to cluster size is not addressed.

7.2 Methods

7.2.1 Cluster quality metrics

We study two different kinds of quality metrics: cluster-wise (or local), which evaluate each cluster separately, and global, which give a score to the entire network. Additionally, for each local metric, we also consider the global metric obtained by computing its weighted mean, with weights being the corresponding sizes of the clusters.

We consider a collection of local metrics (or community scoring functions) introduced in [94] which combine the notions of strong internal and weak external connectivity that are expected of good clusters. Definitions of these local clustering quality metrics, which we already defined in chapter 2, are summarised in table 7.1. While many of them are too focused on a single property to be able to give a general overview on their own (like internal density, average degree, cut ratio ...), they all capture properties that are considered desirable in a proper clustering (which essentially come down to a combination of strong internal and weak external connectivity). The ones that actually combine both internal and external connectivity are the conductance and normalized cut (which happen to be highly correlated, as seen in [94]), so we will mostly focus our analysis on those.

As for global metrics, we consider modularity [64] and coverage [30]. Additionally, we propose another metric, which we named *density ratio*, and is defined as $1 - \frac{\text{external density}}{\text{internal density}}$. It is based on some of the metrics in table 7.1, but defined globally over the whole partition (see table 7.2). It takes values on $(-\infty, 1]$, with 1 representing the strongest partition, with only internal connectivity, while poor clusterings with similar internal and external connectivity have values around 0. Only clusterings with higher external than internal connectivity (so worse on average than clustering randomly) will have negative values, and if we keep decreasing the internal connectivity, the density ratio will tend to $-\infty$ as the internal density approaches 0. The density ratio can be computed on linear time over the number of edges. A local version of this metric is defined in table 7.1.

7.3 Cluster metrics analysis

7.3.1 Standard SBM network

Using stochastic block models—particularly the l-partition model, given the use of the same fixed probabilities for all blocks (cf. sec. 6.3)—we generate a collection of networks with predetermined clusters of varying sizes. The networks are generated as follows: The number of vertices n is fixed, and then, each vertex is assigned to a community with probability $p_{c_1}, p_{c_2}, ..., p_{c_{|\mathcal{P}|}}$ (such that $\sum P_i = 1$). Then, this set of probabilities is what will determine the expected sizes of the clusters. Finally, once each vertex is assigned to a community, the edges are generated using the stochastic block model with probabilities $p_{in} = 0.1$ and $p_{out} = 0.001$ (which control the probability of intra and inter community edges, respectively).

To generate $p_{c_1}, p_{c_2}, ..., p_{c_{|\mathcal{P}|}}$ we sample $x_1, ..., x_{\mathcal{P}}$ from a power law distribution with $\beta = 1.5$, and then use the probabilities $p_i = \frac{x_i}{\sum_j x_j}$. For this experiment, we have used networks of 300 nodes, with a number of clusters ranging from 5 to 25. For each number of clusters, 1000 samples have been generated.

Since both the internal and external densities remain constant across all clusters, a strong correlation of a quality metric with cluster size could suggest that it is biased. We can then study the correlation between each quality metric like cluster

↑ Internal density	$\frac{m_S}{n_S(n_S-1)/2}$
↑ Edges inside	m_S
↑ Fraction over median degree	$\frac{\lvert\{u:u\in S,\lvert\{(u,v)_v\in S\}\rvert>k_{med}\}\rvert}{n_S}$
↑ Triangle participation ratio	$\frac{\lvert\{u:u\in S,\{(v,w):v,w\in S,(u,v)\in E,(u,w)\in E,(v,w)\in E\}\neq\emptyset\}\rvert}{n_S}$
↑ Average degree	$\frac{2m_S}{n_S}$
↓ Expansion	$\frac{c_s}{n_s}$
↓ Cut ratio	$\frac{c_s}{n_s(n-n_s)}$
↓ Conductance	$\frac{c_s}{2m_s+c_s}$
↓ Normalized cut	$\frac{c_s}{2m_s+c_s}+\frac{c_s}{2(m-m_s)+c_s}$
↓ Maximum ODF	$\max_{u\in S}\frac{\lvert\{(u,v)\in E:v\notin S\}\rvert}{k_u}$
↓ Average ODF	$\frac{1}{n_s}\sum_{u\in S}\frac{\lvert\{(u,v)\in E:v\notin S\}\rvert}{k_u}$
↑ Local density ratio	$1-\frac{c_s/(n_S(n-n_S))}{m_S/(n_S(n_S-1))}$

TABLE 7.1: Local scoring functions of a community S of the graph $G=(V,E)$. Arrows indicate whether the score takes higher values when the cluster is stronger and lower values when it is weaker (↑), or vice versa (↓).

↑ Modularity	$\frac{1}{2m}\sum_{u,v}(A_{uv}-\frac{k_uk_v}{2m})\delta_{\mathcal{P}}(u,v)$
↑ Coverage	$\frac{\sum_{u,v}A_{uv}\delta_P(u,v)}{\sum_{u,v}A_{uv}}$
↑ Global density ratio	$1-\frac{\lvert\{(u,v)\in E:\delta_{\mathcal{P}}(u,v)=0\}\rvert/\lvert\{u,v\in V:\delta_{\mathcal{P}}(u,v)=0\}\rvert}{\lvert\{(u,v)\in E:\delta_{\mathcal{P}}(u,v)=1\}\rvert/\lvert\{u,v\in V:\delta_{\mathcal{P}}(u,v)=1\}\rvert}$

TABLE 7.2: Global scoring functions of a partition $\mathcal{P}$ of the graph $G=(V,E)$.

size (for cluster-wise scores), mean cluster size, and number of clusters relative to graph size (for global scores). Results are shown in table 7.3.

	global		local
	#clusters	size (mean)	size
internal density	0.0830	-0.0610	-0.0606
edges inside	-0.6587	0.7620	0.9027
FOMD	-0.6308	0.5951	0.8126
expansion	0.6855	-0.7388	-0.3388
cut ratio	0.0095	-0.0113	0.0010
conductance	0.9718	-0.9432	-0.8164
norm cut	0.9682	-0.9369	-0.7674
max ODF	0.9154	-0.9320	-0.8005
average ODF	0.9671	-0.9390	-0.8016
density ratio	-0.3079	0.2648	0.0913
modularity	-0.3169	0.2096	-
coverage	-0.9234	0.8778	-
global density ratio	-0.0035	0.0069	-

TABLE 7.3: Pearson Correlation table for both global and local scores with respect to size on the SBM. For the local scores, the first two rows are weighted means, giving a score for the whole network. Note that the last three rows correspond to global scores, so there is no value given for the correlation with local cluster size.

Note that by the properties of our model, some of the correlations are to be expected and are a direct consequence of their definitions. This is the case of the cut ratio, internal density or density ratio, which remain nearly constant because they are determined by the values of p_{in} and p_{out}, which are constant across all networks. We must remark the very high correlations of both conductance and normalized cut with the number of clusters. For these two metrics, lower scores indicate better clusterings, so they overwhelmingly favour coarse partitions into few large clusters. We also observe a similarly strong correlation in the case of the coverage metric, but in this case it is a trivial consequence of its definition (coarser partitions will always have equal or better coverage, with the degenerate partition into a single cluster achieving full coverage). In contrast, the modularity only shows very weak correlations to either the number of clusters or their average size.

In the case of the density ratio, we observe that the global version has no correlation to mean size or number of clusters, and that the local version has no correlation with individual cluster size. There is a weak correlation of the weighted mean of the local density ratio with the global properties of the network, which we attribute to the higher likelihood of outliers whenever there is a high number of clusters. Since the score has no lower bound and an upper bound of 1, outliers can have a very small values of the local density ratios, with a great effect on the mean. That is why we suggest using the global density ratio and not the weighted mean of local density ratios when evaluating a network clustering as a whole, and only using the local version when studying and comparing individual clusters.

7.3.2 Multi-level SBM

We use the multi-level stochastic block model to identify whether a certain metric favours either finer or coarser partitions. Given values of d_1 and d_2, we can define a parameter $0 \leq \lambda \leq 1$ that will set the strength of the lower level of clustering (if $\lambda = 1$ the upper level dominates, if $\lambda = 0$, the lower level dominates), and then set $d_2 = d_3 + \lambda(d_1 - d_3)$. Upper level representing coarser partitions than those in lower level, being the latter a refinement of the former.

To evaluate each metric on different configurations of the multi-level community structure, we set a benchmark graph with 4 communities of 50 vertices on the upper level, each of which splits into two 25 vertex communities on the lower level. We then generate samples across the whole range $[0, 1]$ of values of λ, with $d_1 = 0.2$ and $d_3 = 0.01$.

Figure 7.1 shows that all scores fail to capture the multi-level nature of the clustering except for modularity and density ratio. The point at which the scores of the lower and upper levels cross on the plot gives us the value of λ for which both clusterings are considered equally good by the metric. Then, for values of λ smaller than the one attained at the crossing point, the lower level structure is considered preferable, while for larger values, it's the upper level. Then, the value of λ at which we find this tipping point characterises the propensity of a metric to favor finer or coarser partitions. In this case, since this occurs for a smaller value of λ on the modularity (about 0.15) than the density ratio (a bit over 0.20), we can conclude that the former favors coarser partitions than the later.

As for the rest of the metrics, they always prioritize one level of clustering over the other across all the range of λ (that is, the score lines don't cross). Even the conductance and normalized cut, which take into account both internal and external connectivity, fail to give a better score to the lower level when λ is zero. Note that when $\lambda = 0$, our model on the upper level is equivalent to an *l-partition* model of 8 blocks which have been clustered arbitrarily joined in pairs, and that still gets a similar or better score than the correct finer partition into the 8 ground truth clusters.

7.3.3 Multi-level preferential attachment

Here we describe how to set the parameters of the preferential attachment block model defined in section 6.4.1 to obtain graphs with a multi-level or hierarchical community structure. This is given by the selection of the vector p and the matrix β. Let $p_{l_1} = (p_1, ..., p_r)$ be the vector of probabilities for the upper level, where p_i is the probability of membership to community C_i (see figure 7.2). Then, similarly to section 7.3.2, we want to define a lower level structure that can vary according to a parameter λ, which will determine which level dominates. We will define $C_{i,1}, ..., C_{i,s_i}$

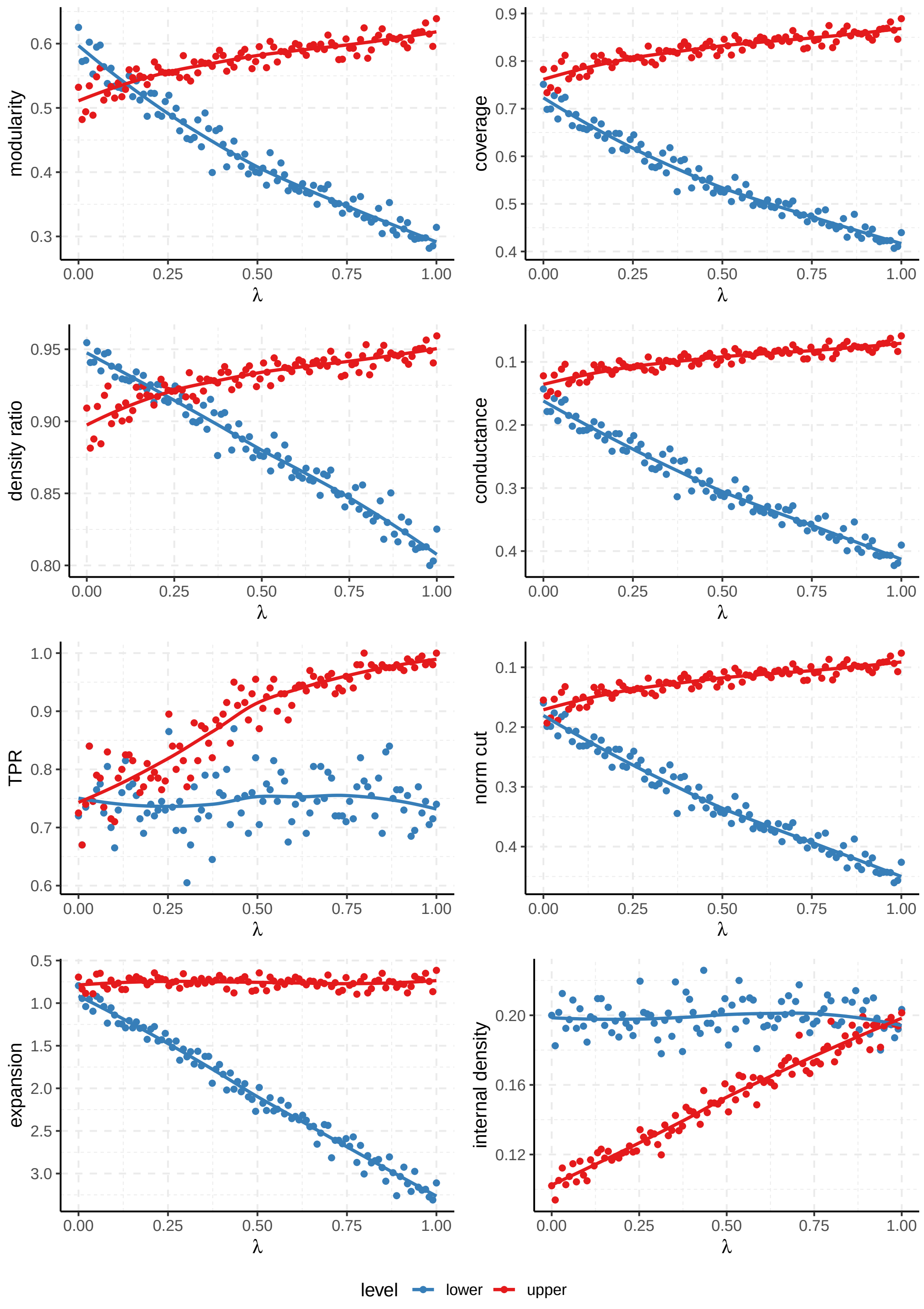

FIGURE 7.1: Values of the quality metrics on each level of clustering on the Multilevel SBM. λ controls the strength of the multi-level community structure. The y axis has been inverted for the scores where lower values are better (conductance, normalized cut and expansion).

as the lower level sub-communities of the higher level community C_i. Then, to sample the membership of vertices when generating a multi-level preferential attachment graph, we need a vector of probabilities $p_{l_2} = (p_{1,1}, ..., p_{1,s_1}, ..., p_{r,1}, ..., p_{r,s_r})$, where $p_{i,j}$ is the probability of membership to $C_{i,j}$, and such that $\sum_{j=1}^{s_i} p_{i,j} = p_i$ for all $i \in \{1, ..., r\}$, and $\sum_{i=1}^{r} p_i = 1$.

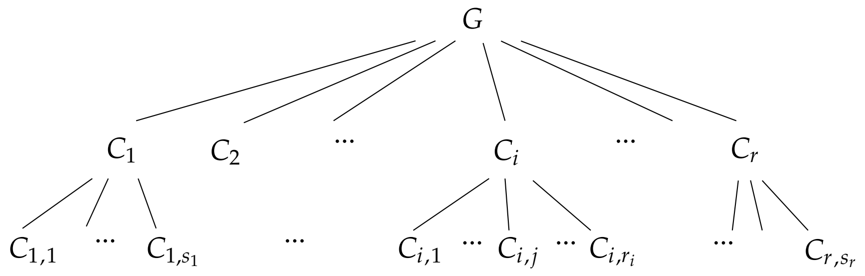

FIGURE 7.2: Diagram of the multi-level preferential attachment graph with community structure following the previously defined notation.

Then, to sample the membership of each vertex on both levels, we simply need to sample its lower level membership according to the probability weights in p_{l_2}, which will also induce its upper level membership. As for the affinity matrix (see figure 7.3), it will have the same block structure as the probability matrix of the multi-level stochastic block model in section 7.3.2.

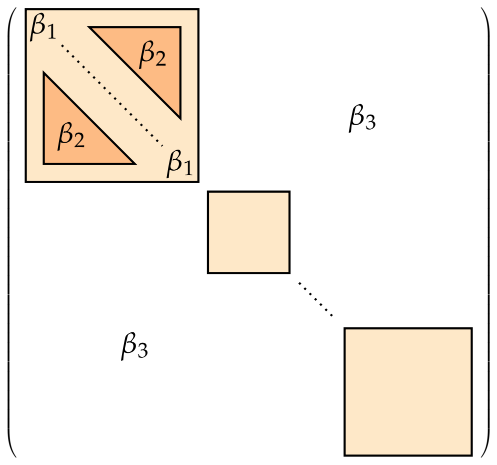

FIGURE 7.3: Affinity matrix of the multi-level block model with preferential attachment.

Again, we use the parameter $0 \leq \lambda \leq 1$, which controls the values of β_2 as follows:

$$\beta_2 = \beta_3 + \lambda(\beta_1 - \beta_3), \tag{7.1}$$

and then, following the preferential attachment model we sample edges attached to a new vertex i with probability distribution $(\deg(1)\beta_{1,i}, ..., \deg(i-1)\beta_{(i-1),i})$. For our experiments, we set $\beta_1 = 0.2$, $\beta_3 = 0.01$, and generate samples of λ across the whole $[0, 1]$ interval, just as with the multi-level SBM in section 7.3.2. m has been set at 4. The results are shown in figure 7.4.

The results are consistent with those of the multi-level SBM, and again only the modularity and density ratio prioritize either partition depending on the strength of the parameters (the rest always favour the same level of partition). This is seen when the plot lines for the lower and upper level partitions cross, and the value of λ at which the lines cross tells us at which degree of relative strength both levels of clustering receive the same score. Ultimately, this value of λ characterizes to what extent any given score favours fine or coarse partitions.

7.4 Final remarks

We have observed that most of the considered community metrics are heavily biased with respect to cluster size. While this does not mean that they are useless for cluster quality evaluation, it makes them inadequate for a simplistic approach based on either of them individually. They do however characterize properties that are expected of good clusters, and can complement other methods on a more qualitative analysis. And considering that there isn't a single universal definition of what constitutes a good clustering, being able to evaluate each of these properties separately can be valuable. Also note that these metrics can be particularly useful when comparing partitions of the same number of elements, because in that case the potential of bias related to cluster size is not a concern.

The results of the tests performed on both multi-level models are similar and show that both the modularity and our newly introduced density ratio are capable of evaluating multi-level community structures successfully. When compared among them, though, the modularity favors slightly coarser partitions. Therefore, there are grounds for further analysis of the density ratio metric in future work, such as evaluating it both on well known benchmark graphs and real world networks. It would also be particularly interesting to see how it fares in circumstances were the modularity has limitations, such as when there are strong clusters below its resolution limit.

Additionally, the methods we propose for studying network metrics using multi-level models gave valuable insight and can be of use in to study any new metrics that might be introduced in the future. We strongly suggest the use of metrics that can appropriately detect clusters at different scales when comparing the results of clustering algorithms, because as we have shown, otherwise cluster size has too much influence on the result.

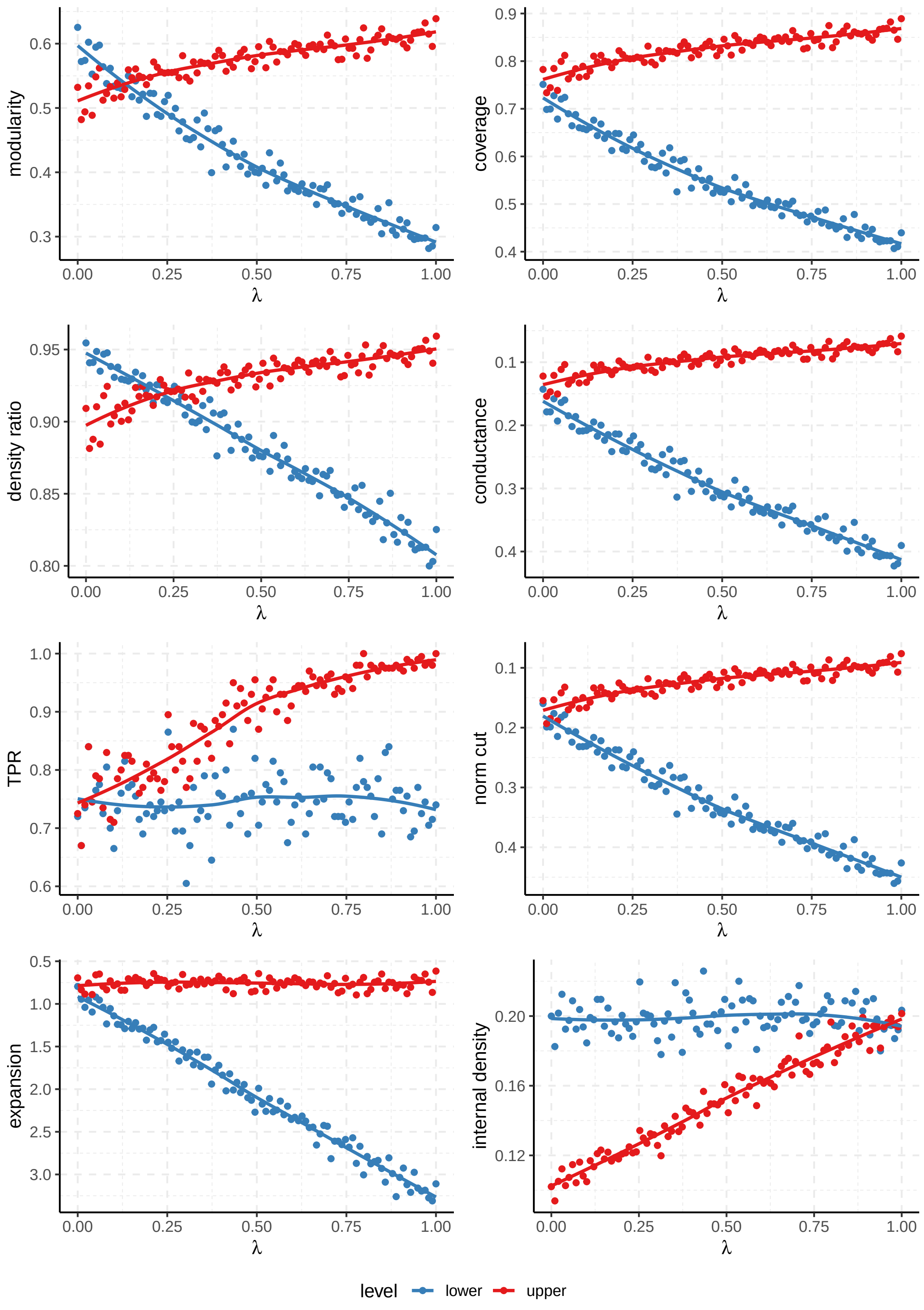

FIGURE 7.4: Values of the quality metrics on each level of clustering for the multi-level preferential attachment model. λ controls the relative strength of the multi-level community structure. The y axis has been inverted for the scores where lower values are better (conductance, normalized cut and expansion).

Chapter 8

Software Development

This chapter is dedicated to the discussion of how the methods described in the thesis have been implemented. We will focus particularly on those where finding an efficient algorithm is not completely trivial, and where a naive approach would be incredibly slow and severely limit the size of networks that could be evaluated.

In the evaluation of cluster significance, the two bottlenecks are the graph rewiring algorithm (which ideally needs to run enough iterations to guarantee that the network has been sufficiently shuffled), and extension of the transitivity and clustering coefficient for weighted networks. As for cluster stability, the computation of the Reduced Mutual Information is a big challenge, because it requires estimating the number of contingency tables with fixed margins (which in practice cannot be computed exactly as it is too costly).

8.1 Graph rewiring algorithm

Here we will discuss the implementation of the algorithm described in section 4.1.1. We only treat the weighted version of the algorithm, because it can be used on unweighted graphs by giving edges weight 1, and in that case it is equivalent to the unweighted version. Since ommiting the weights doesn't simplify the problem substantially, and the efficiency would only be improved by a small constant, the *clustAnalytics* package doesn't include an implementation for unweighted graphs, and internally treats them as weighted graphs of weight 1 for all edges.

Any fast implementation of the algorithm will need a data structure with the following properties:

- Checking whether two vertices are adjacent is fast.
- Checking the weight of a known edge is fast.
- It is possible to sample edges uniformly.
- Adding or removing edges to the graph, as well as modifying their weight, is also fast.

The first two properties can be easily satisfied by using a map containing all edge information, where each key is a pair of vertex indices (as integers), and the corresponding value stores the edge weight. However, it is not trivial to sample elements uniformly out of a search tree or hash table. For this, an additional data structure

class Graph		
int n **int** m **unordered_map<pair<int,int> >**, CantorHash> edge_map **SVector** sampling_vector		
	average	worst case
get_weight()	$\mathcal{O}(1)$	$\mathcal{O}(m)$
set_weight()	$\mathcal{O}(1)$	$\mathcal{O}(m)$
sample_pair_edges()	$\mathcal{O}(1)$	$\mathcal{O}(1)$

TABLE 8.1: Graph class structure and complexity of its member functions

will be needed, which we have called *sampling vector* (in the *clustAnalytics* package, it is defined as the class *SVector*).

Table 8.1 contains a summary with the most relevant members and methods of the resulting `Graph` class that we define. For simplicity, additional variables and methods that don't affect the computational complexity of the algorithm (such as those related to the edge weight bounds, or controlling whether the graph is directed) haven't been included. In the case of directed and undirected graphs, for example, the only particularity is that the edges are treated as unordered pairs of vertices for the former and ordered pairs for the latter.

Algorithm 1 Graph class methods

```
 1: function GET_WEIGHT($a, b$)
 2:     $w \leftarrow$ edge_map$[(a, b)]$
 3:     return$(w)$
 4: end function
 5: function SET_WEIGHT($a, b, w$)
 6:     if $w = 0$ then
 7:         delete_edge$(a, b)$
 8:     end if
 9:     edge_map$(a, b) \leftarrow w$
10:     sampling_vector.insert$((a, b))$
11: end function
12: function SAMPLE_PAIR_EDGES
13:     $(a, c) \leftarrow$ sampling_vector.sample_element()
14:     $(b, d) \leftarrow$ sampling_vector.sample_element()
15:     while $(a, c) = (b, d)$ do
16:         $(b, d) \leftarrow$ sampling_vector.sample_element()
17:     end while
18:     return$((a, c), (b, d))$
19: end function
```

8.1.1 Hash function for edge indices

For storing our edges in hash tables we need to provide an adequate hash function. In the case of our particular implementation done in C++, the `unordered_map` container in the standard library doesn't have a default hash function for the `std::pair<int,int>` type, so we have to provide our own. The integers in this case are always going to be non negative, because they correspond to the indices of the edges (0-based for C++). We also want the function to work for both the directed and undirected cases, so inverting the order of the elements of the pair should not cause colisions. In other words, a perfect hash function in this case would be any injective function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

A natural choice in this case is the Cantor pairing, possibly the most well known bijection between $\mathbb{N}^2$ and $\mathbb{N}$. It is a very intuitive pairing that sorts the elements of $\mathbb{N}^2$ by navigating its grid diagonally as shown in figure 8.1. It is defined by the following expression:

$$C(v_1, v_2) = \frac{1}{2}(v_1 + v_2 - 2)(v_1 + v_2 - 1) + v_1. \tag{8.1}$$

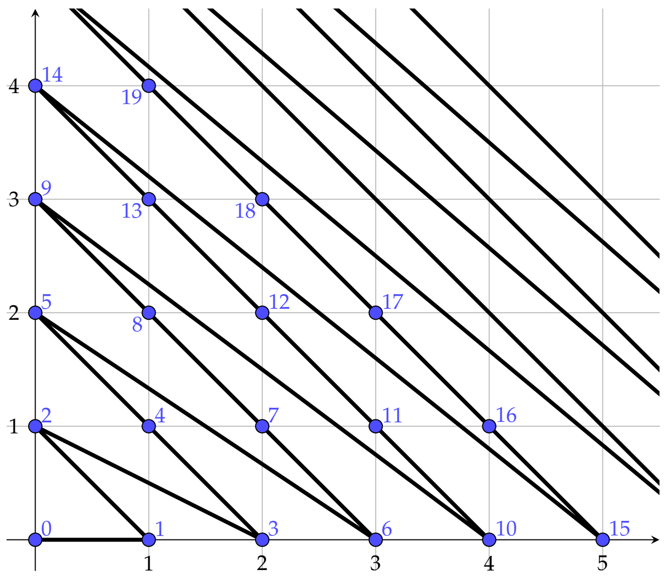

FIGURE 8.1: Cantor pairing function, which assigns a single natural number to each pair of natural numbers.

8.1.2 Sampling vector

This data structure combines a vector v of pairs of integers (each corresponding to the indices of the incident vertices of an edge, resulting with a vector of length m) and an unordered map M with all elements of v as keys, and their position whithin v as values. As we insert or remove edges to the graph, the order of the elements of v is irrelevant as long as they are kept indexed in M. Then, insertions, deletions, and uniform sampling can all be done in constant time on average.

- **uniform sampling:** We can simply sample elements uniformly from vector v.

class SVector		
vector<pair<int,int> > v **unordered_map<pair<int,int> >**, CantorHash> edge_map **int** len		
	average	worst case
insert()	$\mathcal{O}(1)$	$\mathcal{O}(len)$
remove()	$\mathcal{O}(1)$	$\mathcal{O}(len)$
sample_element()	$\mathcal{O}(1)$	$\mathcal{O}(1)$

TABLE 8.2: Sampling vector class structure and complexity of its member functions

- **insertion** Insert the element at the end of v, and insert it also to M with its index. This is $\mathcal{O}(m)$ in the worst case, but $\mathcal{O}(1)$ on average.
- **deletion** The order of elements in v doesn't matter, so to delete it from v, we simply swap it with the last element, and then delete the last position of the vector. The value in M of what used to be the last element of v needs to be updated with its new position, and the deleted element is also deleted from M. Again, the complexity is given by the insertions and deletions into M, resulting in $\mathcal{O}(m)$ in the worst case, but $\mathcal{O}(1)$ on average.

Algorithm 2 SVector class methods

```
function INSERT($a, b$)
    if $(a, b) \notin$ edge_map then
        $v$.push_back($(a, b)$)
        edge_map[$(a, v)$] $\leftarrow$ $v$.size() $- 1$
        $len \leftarrow len + 1$
    end if
end function
function REMOVE($a, b$)
    if $(a, b) \notin$ edge_map then
        $p_a \leftarrow$ edge_map[$(a, b)$]
        edge_map.erase($(a, b)$)
        last $\leftarrow$ $v$.back()
        if $a \neq$ last then
            $v[p_a] \leftarrow$ last
            edge_map[last] $\leftarrow p_a$
        end if
        $v$.pop_back()
        len $\leftarrow$ len $- 1$
    end if
end function
```

8.1.3 Overview of the algorithm

Algorithm 3 describes in pseudo-code a single step of the randomization process: that is, the selection of two edges, and the transfer of a certain amount of weight from them to the edges that share their end points as described previously. While the amount t of transferred weight can vary depending on the parameters of the algorithm, here it is described for the ”max weight” version, with no upper bound (so the maximum amount of weight is trensferred that still keeps all edge weights non negative). The resulting implementation of the complete randomization algorithm is described in algorithm 4.

Algorithm 3 Randomization step

```
1: $\{(a,c),(b,d)\} \leftarrow g$.sample_pair_edges()
2: if $a = d$ or $b = c$ then
3:     return
4: end if
5: for $e \in \{(a,c),(b,d),(a,d),(b,c)\}$ do                ▷ read edge weights
6:     $w_e = g$.get_weight($e$)
7: end for
8: $t \leftarrow \min(w_{AC}, w_{BD})$
9: $w_{AC} \leftarrow w_{AC} - t$
10: $w_{AD} \leftarrow w_{AD} + t$
11: $w_{BD} \leftarrow w_{BD} - t$
12: $w_{BC} \leftarrow w_{BC} + t$
13: for $e \in \{(a,c),(b,d),(a,d),(b,c)\}$ do               ▷ update edge weights
14:     $g$.set_weight($e, w_e$)
15: end for
```

Algorithm 4 Randomization algorithm

```
1: function RANDOMIZE($EdgeList$)
2:     $g \leftarrow$ Graph()                                ▷ Initialize empty graph
3:     for $(e, w) \in EdgeList$ do
4:         $g$.set_weight($e, w$)           ▷ fill the Graph class object with the data
5:     end for
6:     for $i \in [Tm]$ do   ▷ T is the parameter that controls the number of iterations
7:         randomization_step($g$)
8:     end for
9:     return($g$)
10: end function
```

8.2 Weighted transitivity and clustering coefficient

Let Γ be the number of connected triplets in the graph and γ the number of closed triplets (*i.e.* 3 times the number of triangles). As before $\Gamma(t)$ and $\gamma(t)$ are their respective values when only edges with weight greater or equal than t are considered. Then, the clustering coefficient or transitivity is defined as:

$$\tilde{C} = \frac{1}{\bar{w}} \int_{t \geq 0} \frac{\gamma(t)}{\Gamma(t)}\, dt, \tag{8.2}$$

This is an integral of a step function that takes a finite number of values (bounded by the number of different edge weights) which we will compute as follows:

1. Construct a hash table of all edges with their corresponding weights to be able to search if there is an edge between any two edges (and obtain its weight) in constant time. Complexity: $\mathcal{O}(m)$
2. Construct a hash table for each vertex containing all its neighbors. Can be done by iterating once over the edges and updating the corresponding tables at each step. This will be used to iterate over the connected triplets incident to each vertex. Complexity: $\mathcal{O}(m)$
3. Construct a sorted list containing the edge weights at which either a connected triplet or a triangle appears (*i.e.* the maximum edge weight of that triangle or triplet), and an associated variable for each indicating whether it corresponds to a triangle or a triplet. For this, we iterate over the connected triplets using the hash tables from step 2, and for each, we check if it also forms a triangle by checking the hash table from step 1 (which allows each iteration to be done in constant amortized time). This step has complexity $\mathcal{O}(\Gamma \log \Gamma)$, as the list has $\Gamma + \gamma$ elements, and $(\Gamma + \gamma) \in \mathcal{O}(\Gamma)$.
4. We iterate the list from step 3 and compute the cumulative sums of connected triplets and closed connected triplets (which correspond to $\gamma(t)$ and $\Gamma(t)$ for increasing values of t in the list). This gives us all values of $\frac{\gamma(t)}{\Gamma(t)}$, from which we compute the integral (equation 8.2). This involves $\mathcal{O}(\Gamma)$ steps of constant complexity.

Therefore, the overall complexity of the algorithm is $\mathcal{O}(m + \Gamma \log \Gamma)$. Because Γ is bounded by m^2, we can also express the complexity only in terms of m (which will then be $\mathcal{O}(m^2)$), but that bound is not tight in most graphs.

8.3 Counting of contingency tables

This section will be dedicated to the computation or approximation of the number of contingency tables with fixed row and column sums, which is a necessary step for the computation of Reduced Mutual Information (RMI), described in section 3.2.3. This is a #P-complete problem (it was proven in [29], with a proof that we reproduce below in slightly more detail in theorem 8.3.1), so we don't have any algorithm to perform the exact computation in polynomial time, which rules it out for even moderately sized networks. When introducing the RMI [66], Newman et al. suggest using analytical approximations, but they have important limitations. Particularly, they don't give accurate results when row and column sums contain numerous small elements (instead, the approximation is accurate when the contingency tables are very dense). On the other hand, it is possible to use a Markov chain Monte Carlo method, as described in section 8.3.1 , but it is much slower to compute.

The idea behind our approach is to separate the part of the table for which the analytical formula is accurate, and use that to then obtain the result with fewer steps of the Monte Carlo method.

Theorem 8.3.1 *Determining the number of contingency tables with given row and column sums is a #P-complete problem.*

Proof: It is clear that the problem belongs to #P, because for any guess consisting on $R \times S$ integers, one simply has to compute the row and column sums (which is done in linear time), and the number of guesses that satisfy the constraints is the number of contingency tables.

Now, we will see that the case with $2 \times n$ dimensions is #P-hard using polynomial-time reductions, and therefore the problem is #P-hard and #P-complete.

We start with a known #P-hard problem (see [28]): given positive integers a_1,a_2,..., a_{n-1},b, computing the $(n-1)$-dimensional volume of the polytope

$$\sum_{j=1}^{n-1} a_j y_j \leq b, \qquad 0 \leq y_j \leq 1 \quad \forall j \in [n-1]. \tag{8.3}$$

Then, it is also #P-hard to compute the $(n-1)$-dimensional volume of the polytope

$$\sum_{j=1}^{n} a_j y_j = b, \qquad 0 \leq y_j \leq 1 \quad \forall j \in [n], \tag{8.4}$$

where $a_n = b$. This is because the polytope 8.3 is the orthogonal projection of the polytope 8.4 to the hyperplane $y_n = 0$, so the volume of the former is the volume of the latter times $\cos(\alpha)$, where α is the angle between the two hyperplanes.

Now, we perform the change of variables $x_{1j} = a_j y_j$, $x_{2j} = a_j(1 - y_j)$ for all $j \in [n]$. Now, in terms of $x_i j$ the polytope 8.4 can be expressed as

$$\begin{aligned} P(r,s) = \{x \in \mathbb{R}^{2\times n} : & \sum_{j=1}^{n} x_{1j} = b \\ & \sum_{j=1}^{n} x_{2j} = \sum_{j=1}^{n-1} a_j \\ & x_{1j} + x_{2j} = a_j \quad \forall j \in [n] \\ & x_i j \geq 0 \quad \forall i,j \qquad\qquad \}. \end{aligned} \tag{8.5}$$

It is clear that each point of integer coordinates inside $P(r,s)$ corresponds to a $2 \times n$ contingency table with row and column sums

$$r = (b, \sum_{j=1}^{n-1} a_j), \qquad s = (a_1, ..., a_{n-1}, b) \tag{8.6}$$

respectively. Note that if all a_j and b are integers, $P(r,s)$ is a polytope with integer vertices.

Now consider $L(P,t) = L_{n-1}(P)t^{n-1} + L_{n-2}(P)t^{n-2} + ... + L_0(P)$ the Ehrhart polynomial of $P(r,s)$, which has degree $n-1$ and corresponds to the number of integer points inside the polytope $P(tr,ts)$. This polynomial also has the property that its coefficient for the t^{n-1} term corresponds to the $(n-1)$-dimensional volume of $P(u,v)$ (by proposition 4.6.13 in [85]). Then, to prove that counting contingency tables is #P-hard, we need to see that if we can determine the number of contingency tables for any r, s, we can find the value of this coefficient in polynomial time (in the size of r, s).

We take $t = 1, ..., n$, and by definition, we know the values of $L(P,1)$, ..., $L(P,n)$. Then, by solving the system of n equations and n variables

$$\begin{cases} L(P,1) = L_{n-1}(P) + L_{n-2}(P) + ... + L_0(P) \\ . \\ . \\ . \\ L(P,n) = L_{n-1}(P)n^{n-1} + L_{n-2}(P)n^{n-2} + ... + L_0(P), \end{cases} \tag{8.7}$$

which is determined (because the vectors $(t^{n-1}, t^{n-2}, ..., t^0)$ for $t = 1, ..., n$ are linearly independent) and can be done in linear time, we obtain $L_{n-1}(P)$, the volume of P. □

8.3.1 Analytical approximation

The following approximation works in cases where the numbers of clusters R and S are relatively small relative to the total number of elements, resulting in very populated clusters. Let a and b vectors of lengths R and S respectively be the margins of the contingency table, and $\Omega(a,b)$ the its corresponding number of contingency tables. Also, define:

$$w = \frac{n}{n + \frac{1}{2}RS}, \tag{8.8}$$

$$x_r = \frac{1-w}{R} + \frac{wa_r}{n}, \quad y_s = \frac{1-w}{S} + \frac{wb_s}{n}, \tag{8.9}$$

$$\mu = \frac{R+1}{R\sum_s y_s^2} - \frac{1}{R}, \quad v = \frac{S+1}{S\sum_r x_r^2} - \frac{1}{S}. \tag{8.10}$$

Then:

$$\begin{aligned}\log \Omega(a,b) \simeq &(R-1)(S-1)\log(n+\frac{1}{2}RS)+\frac{1}{2}(R+v-2)\sum_s \log y_s \\ &+\frac{1}{2}(S+\mu-2)\sum_r \log x_r+\frac{1}{2}\log\frac{\Gamma(\mu R)\Gamma(vS)}{[\Gamma(v)\Gamma(R)]^S[\Gamma(\mu)\Gamma(S)]^R}.\end{aligned} \tag{8.11}$$

However, this approximation can become highly inaccurate when the conditions aren't met. This can easily happen, for example, when a relatively high number of vertices are left isolated forming their own clusters, even if the rest of the clusters are large.

8.3.2 Monte Carlo approximation

An alternative approach is to use a Monte Carlo method to estimate the number of contingency tables by successively iterating over the set of solutions using an approppriately defined Markov chain. The method, introduced by Diaconis and Gangolli [25], uses a nested chain of subsets $\Sigma_{ab} \supset H_1 \supset H_2 \supset ... \supset H_t$. Then, Monte Carlo sampling is used to estimate each ratio $|H_i|/|H_{i+1}|$, which will allow the estimation of the whole set by just being able to enumerate H_t, which will be small (more specifically, it will contain a single element).

Random walk

First let's define a random walk on the set Σ_{ab} of matrices with row sums a and column sums' b. Let $M \in \Sigma_{ab}$. A pair of rows i_1, i_2 and columns j_1, j_2 is selected randomly. Then, $M' \in \Sigma_{ab}$ is obtained by adding 1 to the (i_1, j_1), (i_2, j_2) elements and substracting 1 to the (i_1, j_2), (i_2, j_1) elements, or viceversa, each of the two possibilities with probability $\frac{1}{2}$. In other words, the table is modified in one of the following cross patterns:

$$\begin{matrix} + & - \\ - & + \end{matrix} \qquad \begin{matrix} - & + \\ + & - \end{matrix} \tag{8.12}$$

Whenever a substraction would make an element become negative, the table remains invariant. This gives a connected, symmetric, aperiodic Markov chain on Σ_{ab}.

Subset chain

Let $M \in \Sigma_{ab}$. Then, define $[\Sigma_{ab}|M;(k,l)]$ the subset of Σ_{ab} containing only tables that match M in all positions strictly preceding (k,l) in the lexocographic order. Then, if (k',l') succeeds (k,l), then $[\Sigma_{ab}|M;(k',l')] \subseteq [\Sigma_{ab}|M;(k,l)]$. This gives a chain of subsets $\Sigma_{ab} = [\Sigma_{ab}|M;(1,1)] \subseteq ... \subseteq [\Sigma_{ab}|M;(r,s)]$.

Theorem 8.3.2 *The random walk on $[\Sigma_{ab}|M;(k,l)]$ is ergodic and has uniform stationary distribution for all $M \in \Sigma_{ab}$.*

Proof: Proof in [25]. □

8.3.3 Hybrid analytical Monte Carlo approximation

In this section, we are going to redefine the subset chain of the Markov Monte Carlo method to reduce its length by estimating the size of the biggest subset we can analitically. First of all, we want to concentrate all the denser comunities on one corner of the matrix, so before starting, a and b will be sorted to be in ascending order. Then, we will divide the matrix into four blocks Q_1, Q_2, Q_3, Q_4 such that $(\Sigma_{Q_4})_{ab}$ can be estimated analytically.

Of course, it is not possible to extend this estimation directly using the method described in section 8.3.1 because not all elements of $Q1$, Q_2 and Q_3 precede those of Q_4 unless Q_4 has only one row.

Order relation

Here we will define an order in which to traverse the matrix M of $R \times S$ elements, or equivalently, a total order relation on the set $[R] \times [S]$. Let $\prec$, and $\preceq$ denote the lexicographical order (the strict and non-strict versions respectively), and $p \in [R] \times [S]$ the element at the lower right corner of Q_1. Then, we define the strict order relation $\sqsubset$ as follows:

$$x \sqsubset y \iff x \prec y \qquad \text{if } x, y \in Q_1 \cup Q_2 \tag{8.13}$$

$$x \sqsubset y \qquad \text{if } x \in (Q_1 \cup Q_2), y \in (Q_3 \cup Q_4) \tag{8.14}$$

$$(x_1, x_2) \sqsubset (y_1, y_2) \iff \qquad \text{if } x_1, y_1 > p_1 \tag{8.15}$$

$$\iff x_2 < y_2 \text{ or} (x_2 = y_2 \text{ and } x_1 < y_1) \tag{8.16}$$

In other words, $\sqsubset$ puts the elements of Q_1 and Q_2 first in lexicographical order, and then those of Q_3 and Q_4 in a variation of the lexicographical order that goes from left to right and top to bottom in that order. That puts all elements of Q_4 after any element of Q_1, Q_2, and Q_3. We will denote $\sqsubseteq$ the non-strict version of the strict order relation $\sqsubset$.

Hybrid algorithm

Then, with the order relation $\sqsubset$, we can define $[\Sigma_{ab}|M; (k,l)]_{\sqsubset}$ as the subset of Σ_{ab} containing tables that match M in all positions strictly preceding (k,l) in the $\sqsubset$ order. Then, to obtain a random walk on $[\Sigma_{ab}|M; (k,l)]_{\sqsubset}$, we just need to uniformly select a pair of rows $i_1 < i_2 \leq R$ and columns $j_1 < j_2 < S$ such that $(k,l) \sqsubseteq (i_1, j_1)$. Then, only elements that succeed (k,l) in the $\sqsubseteq$ order will be modified by the random walk.

Corollary 8.3.3 *of theorem 8.3.2. The random walk on $[\Sigma_{ab}|M; (k,l)]_{\sqsubset}$ is ergodic and has uniform stationary distribution for all $M \in \Sigma_{ab}$.*

Proof: We have to show that there is a path between any pair of elements $X, Y \in [\Sigma_{ab}|M; (k,l)]_{\sqsubset}$ using only steps of the walk.

Let (i,j) be the first coordenate in which X and Y differ. Then, there are two cases:

- $(i,j) \in (Q_1 \cup Q_2)$. In this case, $[\Sigma_{ab}|M;(k,l)]_{\sqsubset} = [\Sigma_{ab}|M;(k,l)]_{\prec}$, so by the proof of theorem 8.3.2, there is a path between X and Y.
- $(i,j) \in (Q_3 \cup Q_4)$. In this case, take the $Q_3 \cup Q_4$ submatrix, transpose it, and again apply the proof of the previous theorem. The path on the transposed matrices can be transposed as well respecting the restrictions to become a path between the lower submatrices of X and Y that doesn't modify elements preceding (i,j). Then, the path on the reduced matrices induces a path on $[\Sigma_{ab}|M;(k,l)]_{\sqsubset}$ because all elements outside of this submatrix are fixed by definition.

□

Then, the resulting algorithm can be described as follows:

- Rearrange the rows and columns of M so that their sums are in ascending order.
- Determine $p = (p_1, p_2)$ the position of the upper left corner of Q_4. This is the cutoff point between the small and large communities, here we are using the first row and column with size > 1.
- Estimate the values $|H_1|/|H_2|, |H_2|/|H_3|, \ldots, |H_{q-1}|/|H_q|$, where $H_q = [\Sigma_{ab}|M;(p_1,p_2)]_{\sqsubset}$, with the Markov chain Monte Carlo method.
- Approximate H_q with the analytical formula described in section 8.3.1.
- Multiply the chain of fractions from the previous steps to obtain $H_1 = \Sigma_{ab}$.

8.3.4 Experiments and discussion

To test the standard Markov chain Monte Carlo and the hybrid algorithms, we use two vectors to set the margins of the tables, and execute both. The chosen vectors are:

$$\begin{aligned} a &= (20,10,10,5,5,1,1,1,1,1,1,1,1,1,1) \\ b &= (10,10,5,5,5,5,5,1,1,1,1,1), \end{aligned} \tag{8.17}$$

which correspond to a network of 50 nodes. Even in such a relatively small network, exact counting algorithms are not practical. As for using the analytical approximation alone, the results are not meaningful because of the presence of a few isolated vertices, which makes the contingency tables less dense.

For the hybrid method, the matrix is split such that Q_4 sub-matrix is formed by the rows and columns with sum greater than 1. The computation took 24.2 seconds, almost twice as fast as the standard Markov chain Monte Carlo method (41.32 seconds). The term $\frac{1}{n}\log\Omega(a,b)$ estimated with each method differs by less than 0.01, so there is not a significant loss of accuracy when the method is used for the computation of the Reduced Mutual Information. In comparison, using only the analytical formula on the whole matrix produces an estimation that is off by over 0.3, which is clearly too inaccurate to obtain any meaningful estimation of the Reduced mutual Information.

If we instead study a case with fewer single element labels:

$$\begin{aligned} a &= (25, 25, 15, 10, 4, 1) \\ b &= (25, 20, 15, 9, 8, 8, 1, 1, 1), \end{aligned} \tag{8.18}$$

the difference is much more apparent with the hybrid method taking 2.98 seconds compared to 37.41 of the standard Monte Carlo.

It is worth noting that the implementation of the Markov chain uses a naive sampling method that doesn't take advantage of the sparsity of the matrix in some areas. When the chosen elements that have to be decreased by one are already 0, the matrix remains invariant for that step of the chain. Then, when the matrix is very sparse and most of the steps are going to be invariant, it is possible to optimize the process by simply simulating the number of invariant steps before the matrix changes with a geometric distribution, and then sampling only from the rows and columns which will result in a step that modifies the matrix. This optimization would be a lot more beneficial on the sparser parts of the matrix (Q_1, Q_2, Q_3) and much less on the Q_4 sub-matrix, which would benefit the hybrid method more than the standard Monte Carlo method.

Conclusions

The proposed hybrid algorithm offers a more efficient alternative to the Markov chain Monte Carlo method to compute the Reduced Mutual Information (RMI) in cases where the analytical approximation is not accurate enough. The advantages of this hybrid approach are particularly apparent when using the RMI for cluster analysis, as it is not uncommon to have a few single element or small clusters in a partition mostly consisting of larger clusters.

There is room for future work in determining the optimal point at which to split the contingency table into the parts that will be estimated analytically or with the Markov chain Monte Carlo method, to find the best possible trade-off between speed and accuracy. This, together with some potential optimizations discussed in the article should make this algorithm capable of dealing with larger datasets.

The implementation of the RMI measure presented here is available as part of the `clustAnalytics` R package [79], with the goal of providing a readily available tool for cluster analysis on networks.

Chapter 9

The clustAnalytics R package

9.1 Introduction

In this final chapter we introduce the R package *clustAnalytics*, which contains the implementations of all the methods used to perform the experiments in this thesis. It is mainly intended to be a useful toolbox for other researchers working on clustering of networks (particularly the main functions dedicated to the evaluation of cluster significance and stability), but some of its functionalities, like the implementations of the weighted rewiring method or the reduced mutual information could be used in other applications as well. While most of our experiments are focused on weighted networks, care has been taken to make sure that the package can cover as many use cases as possible.

clustAnalytics handles weighted networks, as well as unweighted, and it also supports both directed and undirected edges (again, for both the weighted and unweighted cases). Some useful functions used internally, like a version of the apply function for the communities in a graph, have been made available as part of the package as well.

In the following sections we introduce some examples of usage and highlight the principal tasks resolved by *clustAnalytics*.

9.2 Background concepts

Community detection on networks, which are represented by graphs, is a very active topic of research with many applications. The *igraph* [22] package contains a collection of popular algorithms for this task, such as the Louvain [13], walktrap [70] or label propagation [73] algorithms.

Evaluating the significance of the community structure of a network is no simple task, because there is not a single definition of what a significant community is. However, there is some agreement in the literature (see the survey by Fortunato [34]) in that communities should have high internal connectivity (presence of edges connecting nodes in the community) while being well separated from each other. These notion can be quantified and formalized by applying several community scoring functions (also known as quality functions in [34]), that gauge either the intra-cluster or inter-cluster density. *clustAnalytics* implements the most relevant, or representative, community scoring functions following the taxonomy of these quality measures done by Yang and Leskovec [94] and the further discussion on how to adapt them to weighted networks in [77]. The definitions of these functions can be found in chapter 2.

However, to evaluate the significance of clusters on a given network, one needs reference values of the scoring functions to determine whether they are actually higher than those of a comparable network with no community structure. For this, we use a method described in chapter 4 that rewires edges (or transfers some of their weight, in the case of weighted networks) while keeping the degree distribution constant. Then, it can be determined that the partition of a network contains significant clusters if it obtains sufficiently better scores than those for a comparable network with uniformly distributed edges.

On the other hand, stability measures how much the partition of a network into communities remains unchanged under small perturbations. In the case of weighted networks, these could include the addition and removal of vertices, as well as the perturbation of edge weights. This is consistent with the idea that meaningful clusters should capture an inherent structure in the data and not be overly sensitive to small or local variations, or the particularities of the clustering algorithm. To measure the variation that such perturbations present in the clusters, there are multiple similarity metrics available. We have selected to include the Variation of Information [59], the Reduced Mutual Information [66], and the Rand Index (both in its original and adjusted forms) [45]. The first two are based on information theory, while the second one counts agreements and disagreements in the membership of pairs of elements. Then, it is possible to evaluate the network using resampling methods such as nonparametric bootstrap, as described for clustering on Euclidean data by Hennig [42], and later for networks as proposed here in chapter 5, and quantify the deviations from the initial partition with the similarity measures.

9.3 The *clustAnalytics* package

The *clustAnalytics* package (current version 0.5.0) contains 23 functions for assessing clustering significance and stability, and other useful utilites. These are listed in Table 9.1 grouped by category. It also contains some other 14 auxiliary functions to support package management and provide useful baseline graphs with communities. Check these in the reference manual. In what follows we detail the usage of the main functions.

9.3.1 Cluster significance

The scoring functions are formally defined in chapter 2, and were selected and programmed based on the analysis of appropriate scoring functions for unweighted graphs made in [94]. They will take into account the weights of the edges if the graph is weighted. They take as arguments the graph as an *igraph* object, and a membership vector: a vector of the same length as the graph order for which each element is an integer that indicates the cluster that its corresponding vertex belongs to.

significance	scoring functions	`scoring_functions(g, com, type , weighted, w_max)`
		`average_degree, average_odf, conductance, coverage, cut_ratio, density_ratio, edges_inside, expansion, FOMD, internal_density, max_odf, normalized_cut, weighted_clustering_coefficient, weighted_transitivity`
	graph rewiring	`rewireCpp(g, Q=100, weight_sel="const_var", lower_bound=0, upper_bound=NULL)`
	evaluation	`evaluate_significance(g, alg_list, gt_clustering, w_max)` `evaluate_significance_r(g, alg_list, gt_clustering, Q=100, lower_bound=0, w_max=NULL, table_style = "default")`
stability		`boot_alg_list(g, alg_list, R=999, return_data=FALSE, type="global")` `reduced_mutual_information(c1, c2, base=2, normalized=FALSE, method="approximation2")`
other functions		`apply_subgraphs(g, com, f, ...)` `barabasi_albert_blocks(m, p, B, t_max, G0=NULL, t0=NULL, G0_labels=NULL, type="block_first", sample_with_replacement=FALSE)` `sort_matrix(M)`

TABLE 9.1: *clustAnalytics* list of functions split by category.

A general call to all the scoring functions is made with `scoring_functions()`, which computes all the scores and returns a dataframe containing a row for each community (if `type = "local"`) and a column for each score, or alternatively (if `type = "global"`) returns a single row with the weighted average scores. Additionally, an individual function is available for each of the scores, as listed in Table 9.1. The package includes efficient implementations of the clustering coefficient and transitivity for weighted networks introduced by McAssey and Mijma [58].

The main functions for significance evaluation are
`evaluate_significance_r()` and `evaluate_significance()`.

The first one takes an *igraph* graph and a list of clustering algorithms, and computes the scoring functions of the resulting communities, both on the original graph and on rewired versions of it for comparison. The second version does the same while skipping the rewired graphs. By default the clustering algorithms used by these functions are Louvain, label propagation and Walktrap, but they can take any list of clustering algorithms for *igraph* graphs. Both functions allow for comparison against ground-truth in case this is known.

The edge rewiring method (including its versions for weighted networks) is available separately as `rewireCpp`. This differs from the *igraph* function `rewire`, in that it is capable of rewiring weighted as well as directed graphs while keeping the weighted degrees constant.

Rewiring algorithm

The function `rewireCpp` provided by the package is an implementation of the switching algorithm that rewires edges while keeping the degree distribution constant described in [60, 74] (conceived originally for unweighted graphs). The function has been extended to work with weighted and/or directed graphs.

The directed version works very similarly to the undirected one. In the unweighted case, at each step of the algorithm, two directed edges AC and BD are selected randomly, and replaced with the new edges AD, BC (as in the original algorithm, any steps that would produce self-edges or multi-edges are skipped). For vertices A and B, we add and remove 1 to the out-degree, so it remains constant (as well as the in-degree, since no incoming edges are modified). Analogously, we add and remove one incoming edge to both the C and D vertices, so their in-degrees remain constant as well.

We do the same for the directed weighted case, extending the undirected unweighted algorithm. This time, when edges AC and BD are selected, there is a transfer of a certain amount $\bar{w}$ of weight from both AC and BD to AD and BC. This means that the only effects on the in and out-degrees are adding and removing $\bar{w}$ to out-degrees of vertices A and B, and the same to out-degrees of vertices C and D, which means that again they all remain constant.

If the graph is directed, the `rewireCpp` function automatically detects it and internally runs the implementation for directed graphs, so there is no need to specify direction as a parameter. The following example is a food network (where edges indicate predator-prey relationships) from the *igraphdata* package:

```
> data(foodwebs, package="igraphdata")
> rewired_ChesLower <- rewireCpp(foodwebs$ChesLower,
                                 weight_sel = "max_weight")
```

In the weighted case, the rewiring algorithm transfers a certain amount w of weight from some edges to others. The package provides two settings, which we will choose according to what type of weighted graph we are working with:

- **Complete graphs with a fixed upper bound:** These graphs have an edge between every pair of vertices, which will usually be the result of applying some function to each pair. For example, networks resulting from computing correlations of time series (where each series corresponds to a vertex, and the edge weights are the correlations between series) fall into this category.
- **More sparse graphs with weights that are non-negative but not necessarily upper bounded:** This describes most commonly found weighted graphs, where the weights quantify some characteristic of the edges. Multigraphs also fit here, if we reinterpret them as weighted graphs where the edge weight is the number of parallel edges between each pair of vertices.

Of the first type, we show an example built from correlations of currency exchange time series (from [77]). In this network (`g_forex` included in the package) vertices are pairs of exchange rates, and the edge weights are the correlations of their corresponding time series, scaled to the interval $[0, 1]$. In this case, the appropriate setting is the one that keeps the variance of the edge weights constant.

```
> data(g_forex, package="clustAnalytics")
> rewireCpp(g=g_forex, weight_sel="const_var",
            lower_bound=0, upper_bound=1)
```

As for the second type, this includes most of the well known examples of weighted graphs, such as Zachary's karate club graph:

```
> data(karate, package="igraphdata")
> rewired_karate <- rewireCpp(karate, weight_sel="max_weight")
```

The number of iterations, which is computed as $Q \cdot \#edges$ can be controlled with the parameter Q, but we recommend leaving it on the default value ($Q = 100$), which has been shown to provide more than enough shuffling, while still being very fast (see section 4.1.1 for details).

9.3.2 Cluster stability

As for the study of cluster stability, the function used to perform the evaluation is `boot_alg_list()`. This performs a bootstrap resampling (i.e. uniform sampling of the vertices with replacement) of the input graph, applies a given list of clustering algorithms, and measures the variation of the communities obtained in the resampled graphs with respect to the original communities. In more detail, for each input graph and a list of clustering algorithms, the set of vertices in the input graph is resampled (`R` times), the induced graph is obtained by taking the new set of vertices with the induced edges from the original graph (two vertices are joined with an edge on the resampled graph if they were on the original graph), and the clustering algorithms are applied to it. Then, the resulting clusterings (in each of the resampled graphs) are compared to the clustering of the original graph using several metrics: the variation of information (`vi.dist` from package *mcclust*), normalized reduced mutual information (NRMI) and both adjusted and regular Rand index (`rand.index` from package *fossil* and `adjustedRandIndex` from package *mclust*). If `return_data` is set to `TRUE`, the output is a list of objects of class `boot` (from package *boot*); otherwise, returns a table with the mean distances from the clusters in the original graph to the resampled ones, for each of the algorithms.

The Reduced Mutual Information is provided as a separate function:
`reduced_mutual_information()`.
This is an implementation of Newman's Reduced Mutual Information (RMI) [66], a version of the mutual information that is corrected for chance. The exact computation of this metric is not reasonable for even moderately sized graphs, so it must be approximated. We provide two analytical methods for this approximation, and also a hybrid approach that combines a Markov Chain Monte Carlo method with the analytical approximation.

```
> data(karate, package="igraphdata")
> c1 <- membership(cluster_louvain(karate))
> c2 <- V(karate)$Faction
> reduced_mutual_information(c1, c2, method="approximation2")
[1] 0.5135699
```

Just as with the standard mutual information, the RMI can be normalized as well:

```
> reduced_mutual_information(c1, c2, method="approximation2",
                             normalized=TRUE)
[1] 0.6621045
```

9.3.3 Graph generators and other utilities

In the analysis of clustering algorithms it is useful to generate controlled examples of networks with communities. The package *igraph* provides the function `sample_sbm` which builds random graphs with communities from the stochastic block model, and hence these networks have binomial degree distribution.

We provide in *clustAnalytics* the `barabasi_albert_blocks()` function, which produces scale-free graphs using extended versions of the Barabási-Albert model that include a community structure. This function generates the graph by iteratively adding vertices to an initial graph and joining them to the existing vertices using preferential attachment (existing higher degree vertices are more likely to receive new edges). Additionally, vertices are assigned labels indicating community membership, and the probability of one vertex connecting to another is affected by their community memberships according to a fitness matrix B (if a new vertex belongs to community i, the probability of connecting to a vertex of community j is proportional to B_{ij}).

The parameters that need to be set are m the number of new edges per step, the vector p of label probabilities, the fitness matrix B (with the same dimensions as the length of p), and t_max the final graph order. The initial graph $G0$ can be set manually, but if not, an appropriate graph will be generated with m edges per vertex, labels sampled from p, and edge probabilities proportional to B.

There are two variants of the model. If `type="Hajek"`, new edges are connected with preferential attachment to any existing vertex but using the appropriate values of B as weights (see [41]). If `type="block_first"`, new edges are connected first to a community with probability proportional to the values of B, and then a vertex is chosen within that community with regular preferential attachment. In this case, the resulting degree distribution is scale-free (see section 6.4.1) for a proof of this fact).

This is a simple example with just two communities and a graph of order 100 and size 400:

```
> B <- matrix(c(1, 0.2, 0.2, 1), ncol=2)
> G <- barabasi_albert_blocks(m=4, p=c(0.5, 0.5), B=B,
                              t_max=100, type="Hajek",
                              sample_with_replacement = FALSE)
> plot(G, vertex.color=(V(G)$label), vertex.label=NA, vertex.size=10)
```

Finally, it is worth mentioning the `apply_subgraphs()` function, which is used internally in the package, but has also been made available to the user because it can

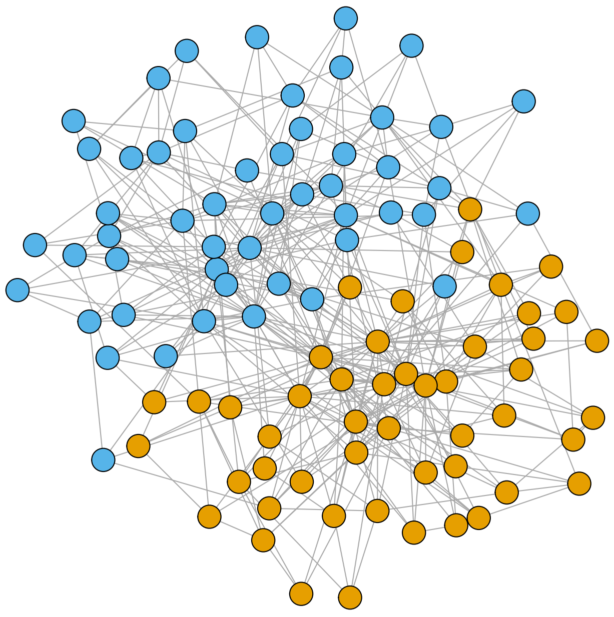

FIGURE 9.1: Example of the `barabasi_albert_communities` function with the community labels as vertex colors.

be very convenient. It simply calls a function `f` on each of the communities of a graph (treated as its own *igraph* object), acting as a wrapper for the `vapply` function. The communities are given as a membership vector `com`. For a very simple example, we call it to obtain the order of each of the factions of the karate club graph:

```
> apply_subgraphs(g=karate, com=V(karate)$Faction, f=gorder)
[1] 16 18
```

9.4 An introductory example

As a toy example we consider the Zachary's karate club graph [95]. First to showcase the graph randomization procedure `rewireCpp`, we apply it to the Zachary's karate club graph with the default settings (positive weights with no upper bound, which suits this graph):

```
> library(clustAnalytics)
> data(karate, package="igraphdata")
> rewired_karate <- rewireCpp(karate, weight_sel = "max_weight")
> par(mfrow=c(1,2), mai=c(0,0.1,0.3,0.1))
> plot(karate, main="karate")
> plot(rewired_karate, main="rewired_karate")
```

The resulting plots are shown in Figure 9.2.

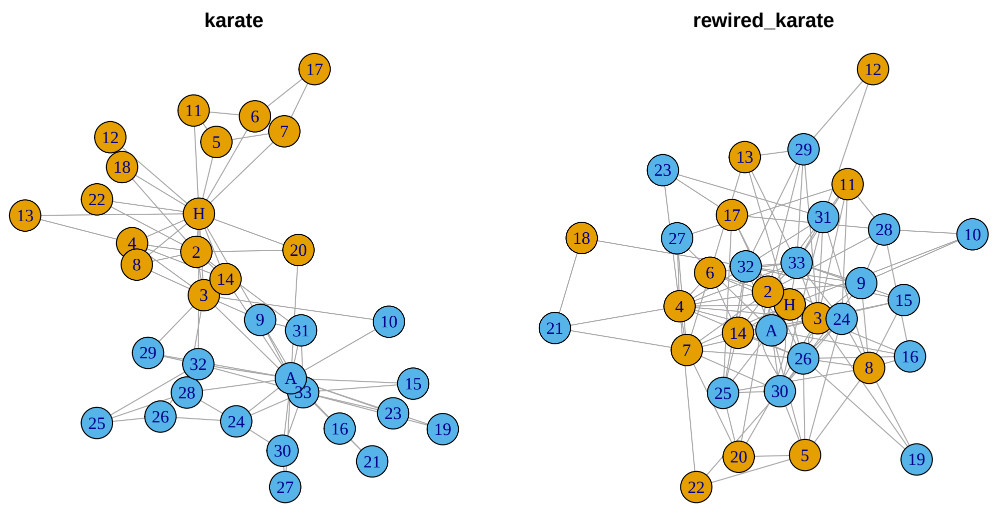

FIGURE 9.2: Karate club graph before and after the edge randomization process. Colors represent the faction of each participant, the *ground truth* clustering in this network.

Now we continue with an analysis of significance and stability of some known clustering algorithms on the Zachary's karate club graph.

9.4.1 Evaluating cluster Significance

The function `evaluate_significance` takes the graph and a list of clustering functions as arguments. If the graph has a known *ground truth* community structure (such as the factions in the karate club), we can set `ground_truth=TRUE` and set `gt_clustering` as the membership vector to evaluate it and compare it to the results of the clustering algorithms. In our `karate` graph the ground truth is available with `V(karate)$Faction`.

```
> evaluate_significance(karate, ground_truth=TRUE,
+                         alg_list=list(Louvain=cluster_louvain,
+                                       "label prop"= cluster_label_prop,
+                                       walktrap=cluster_walktrap),
+                         gt_clustering=V(karate)$Faction)
                         Louvain  label prop    walktrap ground truth
size                  9.58823529 10.52941176 10.00000000  17.05882353
internal density      1.29491979  1.29766214  1.32254902   0.76885813
edges inside         50.35294118 59.17647059 51.82352941 104.82352941
av degree             5.05882353  5.29411765  5.05882353   6.14705882
FOMD                  0.26470588  0.29411765  0.26470588   0.41176471
expansion             3.47058824  3.00000000  3.47058824   1.29411765
cut ratio             0.14311885  0.12786548  0.14540629   0.07638889
conductance           0.25696234  0.22578022  0.25484480   0.09518717
```

```
norm cut              0.37937069  0.34206059  0.38131607   0.19090909
max ODF               0.43576990  0.42077566  0.51172969   0.38911607
average ODF           0.18336040  0.17884402  0.18493603   0.07498851
flake ODF             0.05882353  0.02941176  0.08823529   0.00000000
density ratio         0.87751142  0.88955342  0.86846364   0.90017702
modularity            0.41978961  0.41510519  0.41116042   0.37146614
graph_order          34.00000000 34.00000000 34.00000000  34.00000000
n_clusters            4.00000000  4.00000000  4.00000000   2.00000000
mean_cluster_size     8.50000000  8.50000000  8.50000000  17.00000000
coverage              0.74458874  0.77922078  0.74458874   0.90476190
global density ratio  0.75864388  0.76992481  0.74273256   0.80043860
VIdist_to_GT          0.90782167  0.82624391  0.87293838   0.00000000
```

If a *ground truth* clustering has been provided, the row `VIdist_to_GT` indicates the variation of information distance [59] between that and each of the partitions. In this case the label propagation algorithm obtains the partition closest to the ground truth, while the Louvain algorithm is the furthest.

With the function `evaluate_significance_r` we compute the scoring functions as above, and we compare the results to those of a distribution of randomized graphs obtained with the rewiring method. The parameters of the rewiring method can be selected as shown in Table 9.1, in this case we specify `weight_sel="max_weight"`, but we could also set an upper bound if appropriate to the graph. The resulting (default) table is shown below. This is a table with three columns per algorithm: the original one, the mean of the corresponding rewired scores and its percentile rank within the distribution of rewired scores. If parameter `table_style = "string"`, the function instead returns a table with a column per algorithm where each element is of the format "original | rewired(percentile)".

```
> evaluate_significance_r(karate,
+                        alg_list=list(Lv=cluster_louvain,
+                                      "WT"= cluster_walktrap),
+                        weight_sel="max_weight", n_reps=100)
                         Lv     WT  Lv_r  WT_r Lv_perc WT_perc
size                  9.588 10.000  8.04  8.22    0.89    0.81
internal density      1.295  1.323  1.40  1.43    0.45    0.50
edges inside         50.353 51.824 28.72 37.86    0.99    0.83
av degree             5.059  5.059  3.68  3.85    1.00    0.98
FOMD                  0.265  0.265  0.17  0.19    0.99    0.86
expansion             3.471  3.471  6.24  5.88    0.00    0.02
cut ratio             0.143  0.145  0.24  0.23    0.00    0.00
conductance           0.257  0.255  0.48  0.48    0.00    0.00
norm cut              0.379  0.381  0.68  0.70    0.00    0.00
max ODF               0.436  0.512  0.64  0.66    0.00    0.04
average ODF           0.183  0.185  0.45  0.47    0.00    0.00
flake ODF             0.059  0.088  0.35  0.41    0.00    0.00
density ratio         0.878  0.868  0.75  0.76    1.00    1.00
modularity            0.420  0.411  0.18  0.16    1.00    1.00
clustering coef       0.541  0.614  0.46  0.44    0.73    0.82
```

```
graph_order           34.000 34.000 34.00 34.00    0.00    0.00
n_clusters             4.000  4.000  4.89  6.58    0.01    0.05
mean_cluster_size      8.500  8.500  7.09  5.77    0.74    0.84
coverage               0.745  0.745  0.54  0.57    1.00    0.98
global density ratio  0.759  0.743  0.54  0.59    1.00    0.99
```

9.4.2 Applying scoring functions

If it is the case that we already have some explicit community partition, but not the algorithm that produced it, we can assess its significance by applying the scoring functions directly to the network and the partition. To apply all scoring functions at once use `scoring_functions` with either `type` local or global:

```
> scoring_functions(karate, V(karate)$Faction, type="local")
  size internal density edges inside av degree      FOMD expansion
1   16        0.8250000           99  6.187500 0.5000000  1.375000
2   18        0.7189542          110  6.111111 0.3333333  1.222222
   cut ratio conductance  norm cut   max ODF average ODF flake ODF
1 0.07638889  0.10000000 0.1909091 0.3636364  0.05651941         0
2 0.07638889  0.09090909 0.1909091 0.4117647  0.09140548         0
  density ratio modularity
1     0.9074074         NA
2     0.8937500         NA

> scoring_functions(karate, V(karate)$Faction, type="global")
         size internal density edges inside av degree      FOMD
[1,] 17.05882        0.7688581     104.8235  6.147059 0.4117647
     expansion  cut ratio conductance  norm cut   max ODF average ODF
[1,]  1.294118 0.07638889  0.09518717 0.1909091 0.3891161  0.07498851
     flake ODF density ratio modularity graph_order n_clusters
[1,]         0      0.900177  0.3714661          34          2
     mean_cluster_size  coverage global density ratio
[1,]                17 0.9047619            0.8004386
```

Alternatively, apply the scoring functions individually. Each is called with the graph and the membership vector as arguments, and return a vector with the scores for each community:

```
> cut_ratio(karate, V(karate)$Faction)
[1] 0.07638889 0.07638889

> conductance(karate, V(karate)$Faction)
[1] 0.10000000 0.09090909
```

A case in point are the clustering coefficient and transitivity. As they can be applied to weighted graphs in general and not only to their partition into communities, they are simply called with the graph as the only argument:

```
> weighted_clustering_coefficient(karate)
[1] 0.8127164
```

To be able to obtain the result for every community in the graph, we provide the function `apply_subgraphs`; which given a graph, a membership vector and a scalar function, applies the function to every community and returns the vector of results. In this case it works as follows:

```
> apply_subgraphs(karate, V(karate)$Faction,
                  weighted_clustering_coefficient)
[1] 0.9514233 0.7783815
```

9.4.3 Evaluating cluster Stability

Here we perform a nonparametric bootstrap to the karate club graph and the same selection of algorithms. For each instance, the set of vertices is resampled, the induced graph is obtained by taking the new set of vertices with the induced edges from the original graph, and the clustering algorithms are applied. Then, these results are compared to the induced original clusterings using the metrics mentioned above: the variation of information (VI), the Adjusted Mutual Information (AMI), the normalized reduced mutual information (NRMI), and both adjusted and regular Rand index (Rand and adRand).

```
> boot_alg_list(g=karate, return_data=FALSE, R=99,
+              alg_list=list(Louvain=cluster_louvain,
+                            "label prop"= cluster_label_prop,
+                            walktrap=cluster_walktrap))
             Louvain label prop  walktrap
VI         0.2657555  0.3623330 0.2608622
AMI        0.6202236  0.4896837 0.5900336
NRMI       0.7024417  0.3415649 0.6959898
Rand       0.8584598  0.5969139 0.8609266
AdRand     0.6457648  0.2574423 0.6645099
n_clusters 5.9191919  5.1313131 6.3030303
```

Note that in this table the variation of information is a distance, so lower values indicate similar partitions, while for the AMI, NRMI, Rand, and adRand, higher values mean the partitions are more similar (1 means they are the same partition). The exact definitions of these measures can all be found in chapter 5.

9.4.4 Clustering assessment on synthetic ground truth networks

We can evaluate the significance and stability of clusters produced by a set of clustering algorithms on a network with known community synthetically created with the stochastic block model (with function `sample_sbm`) or the preferential attachment model (with `barabasi_albert_blocks`). The former produces a network with binomial degree distribution, and the latter produces networks with scale-free degree distribution.

Let us generate a graph from a stochastic block model in which we set very strong clusters: the elements in the diagonal of the matrix are much larger than the rest, so the probability of intra-cluster edges is much higher than that of inter-cluster edges.

```
> pm <- matrix (c(.3, .001, .001, .003,
                .001, .2, .005, .002,
                .001, .005, .2, .001,
                .003, .002, .001, .3), nrow=4, ncol=4)
> g_sbm <- igraph::sample_sbm(100, pref.matrix=pm,
                              block.sizes=c(25,25,25,25))
> E(g_sbm)$weight <- 1
> significance_table_sbm <- evaluate_significance(g_sbm)
> significance_table_sbm
                          Louvain   label prop     walktrap
size                 2.500000e+01 2.212000e+01 2.500000e+01
internal density     2.666667e-01 2.983333e-01 2.666667e-01
edges inside         8.000000e+01 7.240000e+01 8.000000e+01
av degree            3.200000e+00 3.100000e+00 3.200000e+00
FOMD                 4.400000e-01 4.100000e-01 4.400000e-01
expansion            2.800000e-01 4.800000e-01 2.800000e-01
cut ratio            3.733333e-03 5.918681e-03 3.733333e-03
conductance          4.479596e-02 8.784949e-02 4.479596e-02
norm cut             5.943574e-02 1.059249e-01 5.943574e-02
max ODF              2.847222e-01 3.307937e-01 2.847222e-01
average ODF          4.642605e-02 8.307287e-02 4.642605e-02
flake ODF            0.000000e+00 0.000000e+00 0.000000e+00
density ratio        9.849546e-01 9.794867e-01 9.849546e-01
modularity           6.860276e-01 6.700177e-01 6.860276e-01
graph_order          1.000000e+02 1.000000e+02 1.000000e+02
n_clusters           4.000000e+00 5.000000e+00 4.000000e+00
mean_cluster_size    2.500000e+01 2.000000e+01 2.500000e+01
coverage             9.580838e-01 9.281437e-01 9.580838e-01
global density ratio 9.720000e-01 9.580098e-01 9.720000e-01
```

We now assess the stability of the clustering algorithms on this SBM graph.

```
> b_sbm <- boot_alg_list(g=g_sbm, return_data=FALSE, R=99)
> b_sbm

              Louvain label prop  Walktrap
VI          0.1234341  0.1769217 0.1178832
AMI         0.86969891  0.7686690 0.88815033
NRMI        0.8536997  0.7841236 0.8656356
Rand        0.9411244  0.9230160 0.9472768
AdRand      0.8306925  0.7651778 0.8476909
n_clusters 6.9797980  7.7070707 7.4646465
```

We can clearly see that for all metrics, the results are much more stable, which makes sense because we created the sbm graph with very strong clusters.

9.5 *clustAnalytics* in context of related R packages

The defining characteristic of *clustAnalytics* is that it provides set of robust and efficient measures for assessing significance and stability of clustering algorithms on graphs with the convenience of working with *igraph* objects, which makes it a valuable complement to the *igraph* package [22]. A revision of the *CRAN Task View: Cluster Analysis & Finite Mixture Models* [56] shows that there are very few packages devoted to assessing quality of clusters in general, and none for *igraph* graphs as input. One could use in a limited manner some of the existing packages by converting *igraph* graphs to their adjacency matrices, but then quality evaluation follows different paradigms not quite pertaining to networks. For instance, the package *ClustAssess* [84] conceived for evaluating robustness of clustering of single-cell RNA sequences data using proportion of ambiguously clustered pairs, as well as similarity across methods and method stability using element-centric clustering comparison; *sigclust* [44] which provides a single function to assess the statistical significance of splitting a data set into two clusters; *clValid* [15] implements Dunn Index, Silhouette, Connectivity, Stability, BHI and BSI, for a statistical and biological-based validation of clustering results. None of these apply directly to *igraph* objects, and were not conceived for the analysis of clustering in social networks.

Chapter 10

Conclusions

The main goal of this thesis was to tackle the problem of evaluating and validating the results of clustering on networks, especially when working with networks for which the existence or lack thereof of a community structure is unknown. The methods proposed for the evaluation of significance and stability can therefore serve two purposes: first of all determine if the partition into clusters obtained by a given algorithm is indeed properly capturing the structure of the network, but also serves to compare different clustering algorithms with each other. These methods are intended to provide a systematic approach to cluster evaluation and to be useful to other researchers when studying all sorts of networks. With that in mind, even though the initial focus of the thesis were weighted undirected graphs, some care has been taken to ensure that the methods are also suitable for the directed as well as the unweighted case. Additionally, the wide variety of weighted graphs that can be studied has been taken into account as well, providing indications on how to treat cases such as graphs with very high density of weighted edges when necessary.

A study on the limitations of some of the functions used to score the results of clustering on networks has also been conducted, which shows varying degrees of bias of most of these functions regarding the number and size of clusters on a given network. While this doesn't invalidate their usefulness, it is something to be aware of when comparing clustering algorithms that produce very different results in terms of the number of clusters in their respective partitions. The methods and in particular the multi-level network models used for this evaluation can be useful to evaluate potential metrics that might be proposed in the future, including objective functions that are optimized by clustering algorithms themselves.

An important part of this work is all the associated code, which has been gathered in an R package available at the CRAN repository. There are many R packages dedicated to the study of networks, and we hope that this is a tool that can complement them well and fit in this ecosystem, and provides other researchers useful and accessible tools to help their research. In addition to the implementation of our own methods, the package also includes efficient implementations of works by other authors, such as one of the versions of the weighted clustering coefficient, or the Reduced Mutual Information, which to our knowledge were not available publicly. And this latter contribution also extends the potential use cases of the package outside of just clustering on networks, as measures to compare partitions can be useful for clustering in general, among other applications.

Bibliography

[1] Christopher Aicher, Abigail Z. Jacobs, and Aaron Clauset. Learning latent block structure in weighted networks. *Journal of Complex Networks*, 3(2):221–248, 06 2014.

[2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[3] Hélio Almeida, Dorgival Guedes, Wagner Meira, and Mohammed J. Zaki. Is there a best quality metric for graph clusters? In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 44–59, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] A. Arenas, A. Fernández, and S. Gómez. Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics*, 10(5):053039, may 2008.

[5] Argimiro Arratia and Alejandra Cabaña. A graphical tool for describing the temporal evolution of clusters in financial stock markets. *Computational Economics*, 41(2):213–231, 2013.

[6] Argimiro Arratia and Martí Renedo-Mirambell. On methods to assess the significance of community structure in networks of financial time series. In *Proceedings ITISE 2017. 4th International Work-Conference on Time Series Analysis.*, pages 585–596, 2017.

[7] Argimiro Arratia and Martí Renedo-Mirambell. The assessment of clustering on weighted networks with R package clustAnalytics. pages 143–146, 2022. Publisher: IOS Press.

[8] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[9] Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016.

[10] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(11):3747–3752, 2004.

[11] Vladimir Batagelj and Andrej Mrvar. Some analyses of Erdős collaboration graph. *Social Networks*, 22(2):173–186, 2000.

[12] Michael G. H. Bell and Yasunori Lida. Transportation networks. In *Transportation Network Analysis*, pages 17–40. John Wiley & Sons, Ltd.

[13] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[14] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity – NP-completeness and beyond, 2006.

[15] Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta. *clValid: Validation of Clustering Results*, 2021. R package version 0.7.

[16] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering*, volume 9220, pages 117–158. Springer International Publishing, 2016. Series Title: Lecture Notes in Computer Science.

[17] Tanmoy Chakraborty, Ayushi Dalmia, Animesh Mukherjee, and Niloy Ganguly. Metrics for community analysis: A survey. *ACM Comput. Surv.*, 50(4), 2017.

[18] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004.

[19] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Hierarchical clustering beyond the worst-case. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[20] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures & Algorithms*, 18(2):116–140, 2001.

[21] Gabor Csardi. *igraphdata: A Collection of Network Data Sets for the 'igraph' Package*, 2015. R package version 1.0.1.

[22] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[23] Leon Danon, Albert Díaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008–P09008, 2005.

[24] Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical Review Letters*, 94(16):160202, 2005.

[25] Persi Diaconis and Anil Gangolli. Rectangular arrays with fixed margins. In David Aldous, Persi Diaconis, Joel Spencer, and J. Michael Steele, editors, *Discrete Probability and Algorithms*, pages 15–41. Springer New York, 1995.

[26] Byron E. Dom. An information-theoretic external cluster-validity measure. UAI'02, page 137–145, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[27] Luca Donetti and Miguel A Muñoz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2004(10):P10012.

[28] M. E. Dyer and A. M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing*, 17(5), 1988.

[29] Martin Dyer, Ravi Kannan, and John Mount. Sampling contingency tables. *Random Structures & Algorithms*, 10(4):487–506, 1997.

[30] Scott Emmons, Stephen Kobourov, Mike Gallant, and Katy Börner. Analysis of network clustering algorithms and cluster quality metrics at scale. *PLOS ONE*, 11(7):1–18, 07 2016.

[31] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[32] Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[33] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

[34] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75 – 174, 2010.

[35] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[36] Philippe Galinier, Zied Boujbel, and Michael Coutinho Fernandes. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1):1–22, 2011.

[37] Michael R Garey. A guide to the theory of NP-completeness. *Computers and intractability*, 1979.

[38] Sara E. Garza and Satu Elisa Schaeffer. Community detection with the label propagation algorithm: A survey. *Physica A: Statistical Mechanics and its Applications*, 534:122058, 2019.

[39] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[40] R. Guimerà, S. Mossa, A. Turtschi, and L. A. N. Amaral. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proceedings of the National Academy of Sciences*, 102(22):7794–7799, 2005.

[41] Bruce Hajek and Suryanarayana Sankagiri. Community recovery in a preferential attachment graph. *IEEE Transactions on Information Theory*, 65(11):6853–6874, 2019.

[42] Christian Hennig. Cluster-wise assessment of cluster stability. *Computational Statistics & Data Analysis*, 52(1):258–271, 2007.

[43] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[44] Hanwen Huang, Yufeng Liu, and J. S. Marron. *sigclust: Statistical Significance of Clustering*, 2014. R package version 1.1.0.

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

[46] Gerardo Iñiguez, Federico Battiston, and Márton Karsai. Bridging the gap between graphs and networks. *Communications Physics*, 3(1):1–5, 2020.

[47] J. Jordan. Geometric preferential attachment in non-uniform metric spaces. *Electronic Journal of Probability*, 18(8):1–15, 2013.

[48] Michael I Jordan, Yair Weiss, and Andrew Y Ng. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14:849–856, 2002.

[49] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[50] Bryan Klimt and Yiming Yang. The Enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*, pages 217–226. Springer, 2004.

[51] Sameer Kumar. Co-authorship networks: a review of the literature. *Aslib Journal of Information Management*, 67(1):55–73, 2015.

[52] Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical Review E*, 84(6):066122, 2011.

[53] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

[54] Clement Lee and Darren J. Wilkinson. A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4(1):122, 2019.

[55] S. Lehmann, B. Lautrup, and A. D. Jackson. Citation networks in high energy physics. *Physical Review E*, 68(2):026113, 2003.

[56] Friedrich Leisch and Bettina Gruen. CRAN task view: Cluster analysis & finite mixture models, 2022.

[57] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[58] Michael P. McAssey and Fetsje Bijma. A clustering coefficient for complete weighted networks. *Network Science*, 3(2):183–195, 2015.

[59] Marina Meilă. Comparing clusterings - an information based distance. *Journal of Multivariate Analysis*, 98(5):873 – 895, 2007.

[60] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences, 2003.

[61] J. Clyde Mitchell. Social networks. *Annual Review of Anthropology*, 3:279–299, 1974. Publisher: Annual Reviews.

[62] M. E. J. Newman. Fast algorithm for detecting community structure in networks. 69(6):066133, 2004.

[63] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3), Sep 2006.

[64] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[65] M. E. J. Newman. *Networks: An Introduction*. OUP Oxford, 2010.

[66] M. E. J. Newman, George T. Cantwell, and Jean-Gabriel Young. Improved mutual information measure for clustering, classification, and community detection. *Phys. Rev. E*, 101:042304, Apr 2020.

[67] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social Networks*, 31(2):155–163, 2009.

[68] Subhadeep Paul and Yuguo Chen. Consistent community detection in multi-relational data through restricted multi-layer stochastic blockmodel. *Electronic Journal of Statistics*, 10(2):3807–3870, 2016.

[69] Leto Peel, Daniel B Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science advances*, 3(5):e1602548, 2017.

[70] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *International symposium on computer and information sciences*, pages 284–293. Springer, 2005.

[71] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, 1990.

[72] Filippo Radicchi, Santo Fortunato, and Alessandro Vespignani. Citation networks. In Andrea Scharnhorst, Katy Börner, and Peter van den Besselaar, editors, *Models of Science Dynamics*, pages 233–257. Springer Berlin Heidelberg, 2012. Series Title: Understanding Complex Systems.

[73] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3), Sep 2007.

[74] A. Ramachandra Rao, Rabindranath Jana, and Suraj Bandyopadhyay. A Markov chain Monte Carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 58(2):225–242, 1996.

[75] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1):016110, 2006.

[76] Martí Renedo-Mirambell and Argimiro Arratia. Clustering of exchange rates and their dynamics under different dependence measures. In *Proc. of the First Workshop on MIning DAta for financial applicationS (MIDAS 2016), collocated ECML-PKDD.*, pages 17–28, 2016.

[77] Martí Renedo-Mirambell and Argimiro Arratia. Clustering assessment in weighted networks. *PeerJ Computer Science*, 2021.

[78] Martí Renedo-Mirambell and Argimiro Arratia. Identifying bias in cluster quality metrics. Technical report, arXiv:2112.06287, 2021.

[79] Martí Renedo-Mirambell. *clustAnalytics: Cluster Evaluation on Graphs*, 2022. R package version 0.5.2.

[80] Martí Renedo-Mirambell and Argimiro Arratia. Towards and efficient algorithm for computing the reduced mutual information. pages 168–171, 2022. Publisher: IOS Press.

[81] Daniel M. Romero, Brendan Meeder, and Jon Kleinberg. Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter. In *Proceedings of the 20th international conference on World wide web - WWW '11*, page 695. ACM Press, 2011.

[82] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[83] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, volume 6942, pages 469–480. Springer Berlin Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[84] Arash Shahsavari, Andi Munteanu, and Irina Mohorianu. *ClustAssess: Tools for Assessing Clustering*, 2022. R package version 0.3.0.

[85] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Second edition, 2012.

[86] V. A. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.

[87] Lucas Vendramin, Ricardo J. G. B. Campello, and Eduardo R. Hruschka. Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 3(4):209–235, 2010.

[88] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1073–1080. Association for Computing Machinery.

[89] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010.

[90] Ulrike von Luxburg. Clustering stability: An overview. *Found. Trends Mach. Learn.*, 2(3):235–274, 2010.

[91] Yuchung J Wang and George Y Wong. Stochastic blockmodels for directed graphs. *Journal of the American Statistical Association*, 82(397):8–19, 1987.

[92] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *2012 IEEE 12th International Conference on Data Mining*, pages 1170–1175. ISSN: 2374-8486.

[93] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In *Proceedings of the fourth ACM international conference on Web search and data mining - WSDM '11*, page 177. ACM Press, 2011.

[94] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

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

[96] Pan Zhang. Evaluating accuracy of community detection using the relative normalized mutual information. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(11):P11006, nov 2015.

[97] Haijun Zhou. Network landscape from a brownian particle's perspective. *Physical Review E*, 67(4):041908, 2003.