Modularity in Biological Networks

Thesis
submitted in fulfillment of the requirements for the degree of
Doktor der Naturwissenschaften

by

Sharon Hüffner, M. Sc.

Supervisor: Professor Dr. Christof Schütte

Fachbereich Mathematik und Informatik
Freie Universität Berlin
Berlin, April 2014

Betreuer: Prof. Dr. Christof Schütte
Freie Universität Berlin
Fachbereich Mathematik und Informatik
Arnimallee 2–6
14195 Berlin

Gutachter: Prof. Dr. Christof Schütte
Prof. Dr. Ulrike von Luxburg (Universität Hamburg)

Tag der Disputation: 22. Oktober 2014

Acknowledgements

I am grateful to the colleagues, mentors, and friends that were a part of this long journey. My supervisor, Christof Schütte, gave me the opportunity to work in his group and guided me through an attractive field of mathematics I did not know before. Tim Conrad, Max von Kleist, and Martin Vingron provided mentorship and advice on all parts of the scientific process. Nataša Conrad, Marco Sarich, Bastian Kayser, and the Medical Bioinformatics group were a pleasure to work with. Christian Komusiewicz and other collaborators in Rolf Niedermeier's Algorithms and Complexity group at the TU Berlin made me smarter. The meticulous review of previous drafts by Victor Mireles and Nataša Conrad vastly improved this thesis. In addition to work, there was fun: I had a wonderful time with friends and colleagues from the IMPRS-CBSC and the Biocomputing group, for which I am also thankful. Finally, I would like to thank Falk Hüffner for his boundless guidance, support, and encouragement. His contributions are simply too many to recount, and I feel fortunate to have him in my life.

Berlin, April 2014 Sharon Hüffner

Contents

Introduction

Networks are used extensively to represent relationships in diverse disciplines such as social science, economics, and systems biology. The general abstraction of entities represented by nodes and the connections between entities modeled by edges can be extended in many ways: Networks can be directed or undirected. Their edges can be weighted, either reflecting a similarity between the nodes they connect, an interaction between those nodes, or our confidence in these two. Networks may contain interesting structures or motifs that characterize them (Alon, 2007). A common way of analyzing networks via such structures is to partition them into *clusters* (or *modules, communities*) where similar or interacting nodes are grouped together. This is sometimes known as *Graph Clustering*. Such a grouping can help identifying the underlying structure of the network and extract insights from it. For example, modules in a protein–protein interaction (PPI) network can correspond to protein complexes (Babu et al., 2012; Pang et al., 2008; Srihari and Leong, 2012). The function of proteins on which little data exists can be estimated by their clustering with other, more well-researched proteins. In social networks, clustering can help identify groups of individuals having common interests or common characteristics (Borgatti, Mehra, et al., 2009).

Accordingly, rich literature on clustering (also: *community detection*) exists and many methods have been developed for this task, varying in their definition of the optimal partition and in the approach taken to compute it. However, in most of these methods, the partition must be a *full partition*, meaning every node must belong to exactly one module. This constraint both limits the classes of networks that can be clustered, and the insights that can be gained from them.

In this thesis we propose a new and more flexible model for networks that is free from these limitations. Our networks have two parts: a *modular region*, which can be fully partitioned into individual modules, and a *transition region*, containing nodes that cannot be assigned to any module. The proportion between the sizes of the modular and transition regions can vary. Within the modular region, modules participate in several types of relationships among themselves and to the rest of the graph: they can be overlapping, directly connected through edges, or connected through the transition region. The transition region nodes can be further characterized: are they simply outliers? Do they serve a role in

the communication between the modules? Can we quantify their importance? These last questions were addressed, for example, by Djurdjevac, Bruckner, et al. (2011) using the framework of Transition Path Theory (Metzner, Schütte, and Vanden-Eijnden, 2009). In this thesis we focus on the quantification of this different modularity, an extension of the full-partition one.

Within this generalized notion of modularity, we would like to explore two tasks: Constructing a formal quantification of the modular content in a network using a modularity score, and developing approaches for finding good assignments. While each of these two is interesting on its own and has real-world applications, we can additionally utilize the good assignments we find in the computation of the modularity score.

Before we define modularity, we must define more precisely what we mean by *modules*. Unfortunately, there is no single agreed-upon definition. Instead, there is intuition: modules are connected subgraphs that are densely connected within themselves and sparsely connected to the rest of the graph. Here, "densely" and "sparsely" are a matter of taste. For example, Newman and Girvan (2004) assume, when defining their modularity Q-score, that the number of edges between nodes constituting a module in the network to be examined is higher than the number of edges between those same nodes if the network was rewired, maintaining the degree sequence. This is a useful definition, but it has some drawbacks, as we will explain in Chapter 1. We stay therefore with the admittedly hazy notion of modules as dense, connected structures, and a transition region sparse in relation to the modules. In this thesis we propose several models for formulating the notion of modularity more exactly.

The main approach we introduce is one based on *random walks* and *metastability*. A random walk on a network is a succession of random steps from node to node. This is a stochastic process, a sequence of random variables X_t where X_t is the node the process can be found at at time t. It is sometimes imagined as a person, a *random walker* traveling from one network node to an adjacent network node, choosing the next node to visit according to some probability distribution. Our random walks are Markovian (Bremaud, 1999), meaning that the choice of next node to be visited depends only on the current node, and not on the nodes visited before.

We would like to say that modules are *metastable sets* of the random walk, that is, on the one hand they are attractive for the random walker, while, on the other hand, they are well-separated in the sense that transitions between them are rare: the random walker would stay in one module for a long time, then, once exiting, will spend only a short time in the transition region before entering any module.

If it is possible to find such a set of modules and a transition region in a given network, then we call it *modular*. A large part of this thesis will be dedicated to constructing a network modularity score based on this idea. In the remainder we will discuss alternative models of modularity, notably a combinatorial, graph-

theoretic approach. We introduce both a score and a way of finding good assignments, along with examples of biological applications.

In Chapter 1 we begin with real-world motivation for the study of modularity in networks. We then provide some useful terminology for graphs and matrices. We subsequently open the discussion on qualitatively defining what it means for a network to have high or low modularity. The rest of the chapter is concerned with the state-of-the-art methods of quantifying modularity, demonstrating the advantages and disadvantages of each and their fit to our notion of modularity with a modular region and a transition region.

In Chapter 2 we introduce a new random-walk-based approach to network modularity. We start by providing background on discrete and continuous random walks, and equate network modules with metastable sets of a Markov process. We introduce the I_m score as a formalization of this notion, then test its performance to establish that it does not exhibit the same problems as previously-reviewed scoring approaches. We then formally prove, using perturbation analysis, that I_m behaves according to our intuition on some simple synthetic network classes. Finally, a possible biological application for the new modularity approach is presented, as we analyze the brain networks of autistic and typically-developed children and compare the distribution of the I_m score for the networks in these two classes.

Finally, in Chapter 3 we introduce algorithms for finding *assignments*, dividing a network into a transition region and a modular region, which is itself fully partitioned into modules. Such assignments are interesting on their own, but are also needed as input for the modularity score I_m. We begin by introducing a method of based on the same principles as I_m, called MSM clustering. We then propose modifications to well-known clustering algorithms to convert them to output assignments rather than full partitions. Next we introduce a similarity score for assignments and use it to evaluate the performance of these algorithms on a class of benchmark networks, highlighting their advantages and disadvantages. The rest of the chapter is then dedicated to the introduction of a combinatorial approach to finding assignments, followed by experiments on biological protein–protein interaction networks with the goal of inferring protein complexes. The chapter ends with an outlook and open questions.

Chapter 1

About network modularity

This chapter provides some useful background for the rest of the thesis. First, we motivate the study of modules and modularity in networks, citing examples from social science and biology. Next, we present basic definitions of network-related terms and concepts that will be used throughout the work. We then begin our study of modularity by surveying previous work, organizing known modularity measures into two classes, and detailing their advantages and disadvantages.

1.1 Motivation

Networks representing real-world systems (*real networks* or *real-world networks*) have often been shown to contain modules (or: clusters, communities): dense regions containing many edges that are sparsely connected to the rest of the network. This is true both for large networks such as the Facebook network (Wilson, Gosling, and Graham, 2012), containing hundreds of millions of nodes, and for small networks representing, for example, the workplace dynamics at a sawmill, containing 25 nodes (Michael, 1997).

In the case of Facebook, the nodes represent user profiles, and edges between them represent Facebook friendships. Wilson, Gosling, and Graham (2012) offer a review of the numerous studies performed on this network. A cluster in this network could correspond to a group of friends, and by analyzing the structures of such groups it is possible for social scientists to explore issues such as race, social capital and privacy (Acquisti and Gross, 2006).

On a much smaller scale, the sawmill network (Michael, 1997) is comprised of nodes representing sawmill workers, and social interactions between them as edges. Analyzing the structure of this network demonstrated that the workers are divided by language, and thus helped identify the optimal way of disseminating information about strike negotiations that will reach all workers.

1.1.1 Clustering in biological networks

One area of research that has greatly benefited from advances in network analysis in the last years is biology. Cluster analysis in biological networks has received special attention. Indeed, the analysis of modules in biological networks, whether functional, protein interaction, or co-expression networks, has proven useful for gaining insights about the mechanism of the cell (Jeon et al., 2011). Another example is using modules to coarse-grain a network (such as a biochemical or metabolic one), allowing to simplify parameter estimation (Riel, 2006). Modules in dynamical networks can help decoding protein structure (Vijayabaskar and Vishveshwara, 2012).

Notably, we can study clusters in protein–protein interaction (PPI) networks. The nodes of PPI networks are proteins, and two proteins are connected by an edge if they interact. Thus, sets of interacting proteins come together to create *protein complexes* (Amoutzias and Peer, 2010; Price and L. Stevens, 1999). These complexes are the underlying element of many biological processes and together form molecular machinery that performs a vast array of biological functions. An example of a complex, Arp2/3, is shown in Figure 1.1. This complex is made of seven proteins, and is important in a variety of cell functions involving the actin cytoskeleton. To further the research into protein complexes, many computational methods have been developed to detect them automatically from PPIs. The underlying assumption of these methods is that protein complexes correspond to dense subgraphs of the network, as the proteins in a complex must interact with each other in order to perform the given function. It is natural, then, to formulate the problem of identifying protein complexes in PPI networks as a clustering problem, where the clusters correspond to the complexes. Further discussion of PPI networks and their analysis can be found in Chapter 3.

Another example of a biological network is a brain network, where nodes correspond to regions of the brain, and edges between them to similarity or correlation in measurements (Bassett et al., 2013; Bullmore and Sporns, 2012; Lynall et al., 2010; Meunier, Lambiotte, and Bullmore, 2010; A. A. Stevens et al., 2012). There are several standard toolboxes for constructing brain networks from imaging data and applying various network measures, for example the Brain Connectivity Toolbox (Rubinov and Sporns, 2010). The brain networks (sometimes: *connectomes*) of healthy subjects have been shown to exhibit a different *modular structure* than those of patients diagnosed with Alzheimer's disease (G. Chen et al., 2013), schizophrenia (Lynall et al., 2010), and epilepsy (Chavez et al., 2010). That is, clustering these networks (that share the same set of nodes) results in modules that are somehow different in structure: Having more edges between modules, smaller or larger average module size, or other such attributes, which are not easy to quantify.

Even within the same healthy individual, the modular structure changes when learning a new skill (Bassett et al., 2013), and such differences have even been linked to varying performance in short-term memory tests (A. A. Stevens

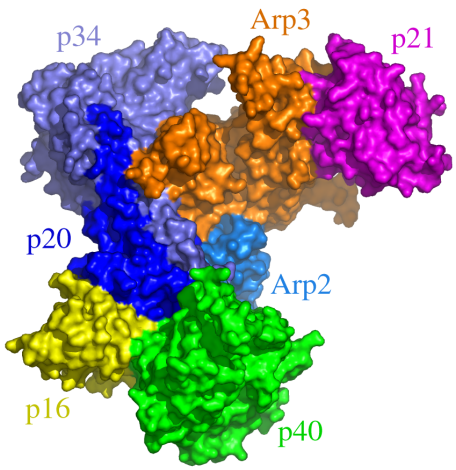


Figure 1.1: The atomic structure of the bovine Arp2/3 protein complex, PDB code 1K8K, surface representation (Splettstoesser, 2006). Rendered with PyMol

et al., 2012). We analyze and discuss a set of brain networks in Chapter 2.

The example of the PPI network demonstrates that studying the modular structure of a single network can allow us insights into how the represented system works. At the same time, brain networks demonstrate that comparing the modular structure of several networks can also be useful. In order to investigate this modular structure it is helpful to qualify and quantify the *modularity* of a network, to obtain a meaningful comparison. Thus it is valuable not only to identify clusters in the network, but also to explain the modular structure, a theme that will be explored in this thesis.

1.1.2 Types of clusters in real networks

Research into the analysis of modular networks has often been done under the assumption that the network must be fully partitionable, that is, every node should belong to at least (often, exactly) one module. However, we know that real networks are not necessarily fully partitionable. Revisiting the examples above, we can show that these networks contain nodes which are not naturally assigned to any module, whether because of the noise that is inherent in the measurements from which the network was derived, or simply because the underlying system is not completely modular.

For example, the Facebook network contains many fake and spammy profiles, up to 8.7%, as specified in the SEC (U.S. Securities and Exchange) filing[1]. These profiles exhibit different connectivity patterns, often serving as *hubs* that connect many existing groups (Fire, Katz, and Elovici, 2012). The nodes representing

[1] http://www.digitaltrends.com/mobile/8-7-percent-of-facebook-users-are-fake/

such profiles would not necessarily fit into modules in the same way as nodes representing real persons.

Figure 1.2 presents examples of two networks from unrelated fields: history and linguistics. The first network describes the membership in some organizations in the year 1772. The nodes represent important individuals from the period, who played some role in the American Revolution, and there is an edge between them if they are members in the same organization: The Loyal Nine, St. Andrews Lodge, and others. Simply drawing the network makes it clear that while there are dense clusters, the network also contains several nodes (individuals) that cannot be assigned to a single cluster, but rather bridge the groups. One such person, for example, is Paul Revere, who later played a major part in the revolution. A simple full partitioning of the network would not distinguish those possibly important individuals, but rather assign them to some cluster.

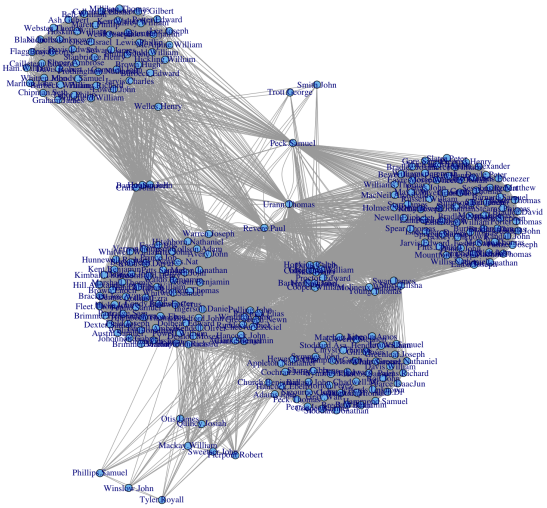
The second network displays the languages of Europe, connected according to the *linguistic distance* between them. Here the clusters are labeled by language category (Germanic, Baltic, etc.), but some clusters do no conform to our intuitive notion of density: The Greek cluster, for example, contains only one language (Greek). We can claim that it is not a cluster at all, but rather, as befitting its historical role, outside of the language clusters, affecting them all.

Although it can be interesting to discuss the modular structure of networks in various fields, this thesis focuses on the question of network modularity in biological networks. We begin with PPI networks. These networks contain both multiprotein complexes and "free" proteins, either because these proteins truly do not belong to any stable complex or because there is simply not enough data available about their interactions. Even for the well-studied model organism *Saccharomyces cerevisiae* (yeast), about 20% of proteins have no known, assigned function [2].

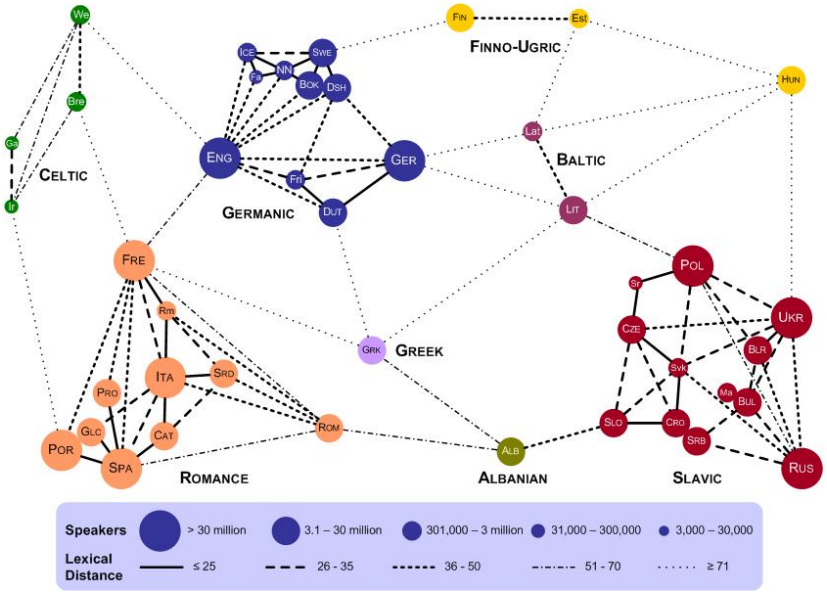
In general, nodes outside of modules can take on several roles. In protein interaction networks, they could, for example, be outliers, as in the case of the proteins whose function and interactions are not known. Alternatively, such nodes could be between modules, serving as attachment proteins between protein complexes (Gavin et al., 2006). They might also create entire structures that are transient and weakly connected. Studying the modular structure of the PPI network could potentially tell us something about the way the different PPI network parts, stable and transient, interact.

From the examples examined so far we see that it is useful to quantify the modular structure of a network. More specifically, we would like to answer the question: To what extent does a network contain modules? The answer can come in the summarized form of a single number, a *modularity score* for a network. With such a score we can better understand the organization of a particular network. We can additionally use it to compare between the modular content

[2] http://www.yeastgenome.org/



(a) Social Network of the American Revolution (1772). Source: http://kieranhealy.org/blog/archives/2013/06/09/using-metadata-to-find-paul-revere/



(b) Languages in Europe. Source: http://elms.wordpress.com/2008/03/04/lexical-distance-among-languages-of-europe/

Figure 1.2: Two networks from the fields of American History and linguistics, that naturally decompose into modules and nodes that bridge modules.

of different networks, or analyze the emergence of modularity in a network over time. Such an analysis can be useful, for example, for understanding brain networks.

Brain networks are indeed natural candidates for modularity analysis: As the modular structure of a network can differ between sick and healthy, a succinct summary of it in a single score could potentially be used as a biomarker (Chavez et al., 2010; G. Chen et al., 2013). A high modularity score can indicate the presence of modules, thus indicating that clustering this network is meaningful (G. Chen et al., 2013). For example, Chavez et al. (2010) have shown that when comparing patients with epilepsy to a control group, the control group exhibited sparser connectivity between modules than the sick. Similarly, Lynall et al. (2010) showed that networks measured from schizophrenia patients exhibited "reduced clustering".

The modular structure of the network in these cases was measured by the average clustering coefficient, or by Newman–Girvan modularity (Newman and Girvan, 2004). Both measures will be examined in Section 1.3. Notably, Newman–Girvan modularity and its variants have been used frequently to analyze brain networks, see e. g. Bassett et al. (2013), Meunier, Lambiotte, and Bullmore (2010), and A. A. Stevens et al. (2012). However, as we will soon see, this measure does not always capture our intuition of modules as dense, sparsely connected subgraphs.

We now explore the idea of assigning a network a modularity score which is high when many of its nodes belong to dense modules, and low if many nodes do not belong to dense modules at all. We will formulate this notion exactly in the following sections. We start with some basic definitions that will be used throughout the thesis.

1.2 Definitions

We will identify networks with undirected graphs and use the terms interchangeably. A network is then a graph $G = (V, E)$, where V is a finite set of *nodes* and E is the set of *edges*, which are unordered tuples of nodes, that is, $E \subseteq \{\{v_1, v_2\} \mid v_1, v_2 \in V\}$. Two nodes v_1, v_2 are *adjacent* if there is an edge $\{v_1, v_2\} \in E$. In this case we also say that v_1 and v_2 are *neighbors*, and $v_1 \in N(v_2), v_2 \in N(v_1)$. A node with no neighbors is called a *singleton*.

A graph can be *edge-weighted*, or simply *weighted*, in which case there is a function $w : E \to \mathbb{R}$ that assigns a weight to each edge, $w(\{v_1, v_2\})$ or simply $w(v_1, v_2)$. If v_1, v_2 are not adjacent, we set $w(v_1, v_2) = 0$. We can also define a weight on the nodes $w_v : V \to \mathbb{R}$ to obtain a *node-weighted* graph.

The *degree* of a node v is denoted $d(v)$, and is equal to the number of nodes in the graph that are adjacent to v. If the network is weighted, we can define the *strength* of a node as $s(v) = \sum_{u \in V} w(u, v)$.

The complete graph, or *clique* K_n for some n is the graph on n nodes that contains all possible $\binom{n}{2}$ edges. Every node in K_n has then degree $n-1$.

A node $v \in V$ is said to be *reachable* from a node $u \in V$ if there is a set of nodes $x_1, x_2, \ldots, x_l \in V$ such that $u = x_1, v = x_l$, and for every $1 \leqslant i < l$ there is an edge $\{x_i, x_{i+1}\} \in E$. In other words, there is a *path* between v and u. In undirected graphs, there is a path from u to v if and only if there is a path from v to u.

A *connected component* $H = (V_H, E_H)$ of G is a subgraph of G where there is a path between any two nodes $v_1, v_2 \in V_H$, but there is no path between any node in V_H and any node in $V \setminus V_H$.

The *induced subgraph* of G on a set of nodes $U \subseteq V$ is denoted $G[U]$. It is the subgraph of G containing the nodes in U and all edges between them.

1.2.1 Clusters, modules, partitions, and assignments

We are interested in associating network nodes into different sets usually called *modules*, *clusters*, or *communities*. Denote a set of disjoint subsets $V = V_1 \dot\cup V_2 \dot\cup \cdots \dot\cup V_k$ a *full partition* of V for some k.

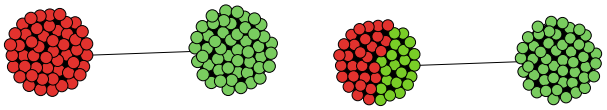
In an *overlapping partition* $V = \tilde{V}_1 \cup \tilde{V}_2 \cup \cdots \cup \tilde{V}_k$ we allow $\tilde{V}_i \cap \tilde{V}_j \neq \emptyset$. We will denote the node subsets in a partition *partitions* or *clusters*. Here we will assume that partitions are disjoint unless explicitly stated otherwise, and use the $\cup$ notation.

In contrast to a full partition, $\mathcal{M} = C_1 \cup \cdots \cup C_k \subseteq V$ are *modules* of G. The difference is that not every $v \in V$ must belong to a module. The set $\mathcal{M}$ is then the *modular region* of the network. We also define the *transition region* $\mathcal{T} = V \setminus \mathcal{M}$. The partition $V = \mathcal{M} \cup \mathcal{T}$ that associates each node either with the modular region or the transition region is called an *MT-partition*. When it is further known not only that a node is a part of the modular region, but also the module to which it belongs, we obtain an *assignment* $V = C_1 \cup C_2 \cup \cdots \cup \mathcal{T}$.

Thus, we differentiate between clusters and communities as general terms, and partitions and assignments in the full partition and non-full partition case, respectively. Further, a *fuzzy assignment* is an assignment where a node is not exclusively associated with a single module or the transition region. Instead, each node is associated with a vector whose entries correspond to the affiliation of the node to the various modules. An assignment can be obtained from a fuzzy assignment for example by associating each node with the module it has the highest affiliation to.

1.2.2 Networks as matrices

The adjacency matrix A of a network has $A_{ij} = 1$ (or simply $a(i,j) = 1$) if nodes i, j are adjacent, 0 otherwise. When we want to take into account edge weights $w : E \to \mathbb{R}$, we will use the weighted adjacency matrix W, where if there is an edge between nodes i, j then $W(i,j) = w(i,j)$, otherwise $W(i,j) = 0$. Since



(a) Assignment 1 (b) Assignment 2

Figure 1.3: Two assignments to the same network. The network is comprised of two cliques with 50 nodes each. The cliques are connected by a single edge. The nodes are colored green or red according to their assignment.

our networks are undirected, we have, for every $i, j \in V$, $A(i,j) = A(j,i)$ and $W(i,j) = W(j,i)$. That is, A and W are *symmetric*.

With these definitions, we can now discuss the literature regarding the evaluation of the modular structure of networks.

1.3 Known methods of evaluating modularity

In the following discussion, we distinguish between *global* and *local* methods of quantifying network modularity. Local methods evaluate modularity by scoring a particular assignment or clustering of the network nodes. That is, a score is computed according to how much the clusters of the given assignments are (i) dense, and (ii) sparsely connected to the rest of the network. It is then possible to compare assignments via these scores. For example, it is intuitively clear that the assignment that assigns the nodes in the two cliques to two different modules is better, or more modular, that one that splits a module over the two cliques (Figure 1.3).

Consequently, to compute a local modularity score of a network we adopt the score of its optimal, highest-scoring assignment. A natural way to do this is to first find this best assignment, and then compute its score. This approach has the added benefit of producing the assignment itself, which, as we saw, can be useful for further analysis of the network. Unfortunately, as exemplified below, computing this optimal assignment can be computationally prohibitive.

Using global measures, on the other hand, we compute a score for the network directly, without first computing an assignment. This is often done by utilizing well-known network measures. A popular example is the *average clustering coefficient*. The clustering coefficient of $v \in V$ quantifies v's participation in dense clusters. The idea is that nodes that take part in clusters have neighbors that are adjacent. Node v and any two neighbors u_1, u_2 that are adjacent induce a triangle in the graph. The node clustering coefficient is then the ratio of triangles

v participates in to the total number of possible triangles, that is:

$$\mathrm{ccoef}(v) = \frac{\sum_{i,j \in N(v)} a(i,j)}{|N(v)| \cdot (|N(v)| - 1)}. \tag{1.1}$$

Then the global modularity measure for a network is the average clustering coefficient across all its nodes. The average clustering coefficient (ACC) has been used to get a sense of the clustering of the graph in biological networks (Lynall et al., 2010; Ravasz et al., 2002). It has also been extended to weighted networks (Barrat et al., 2004; Saramäki et al., 2007). To understand what kind of networks have high or low ACC, recall that the score is based on counting triangles. That is, triangle-free graphs (trees) will always have ACC equal to 0. This is a desirable property, since we would like modules to be dense, and any connected subgraph of a tree will obviously be only minimally connected.

On the other side of the scale, any complete graph will have an ACC of 1, as all possible edges, and therefore triangles, exist. This is quite a disadvantage for our purposes, as we would like our high-scoring network to contain modules that can be easily broken off, while complete graphs (and complete bipartite graphs, among others) do not answer this requirement, despite having a perfect average clustering coefficient. Even outside these extreme cases, the ACC can be difficult to interpret. The network in Figure 1.4 is a random graph generated using an Erdős–Rényi (Erdős and Rényi, 1960) model with 100 nodes and probability $p = 0.5$ of any node to be adjacent to any other node. This graph, for which by construction we do not expect a modular structure, has average clustering coefficient 0.5. In contrast, the network on the right of Figure 1.4 contains two complete graphs of 140 nodes each connected by a tree with about 500 nodes. This network, which exhibits a clear modular structure with two dense, sparsely connected modules, also has an ACC of 0.5. Therefore, despite its desirable behavior on sparse networks, we cannot trust the average clustering coefficient as a modularity score.

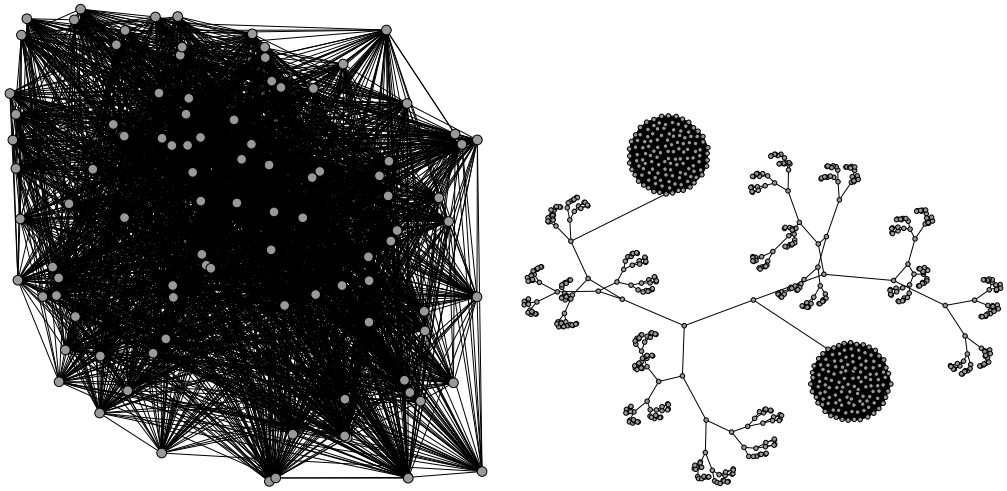
We will now study further common local, then global, methods of evaluating modularity, and discuss their advantages and drawbacks.

1.3.1 Local measures

We survey the well-known Newman–Girvan modularity (Newman and Girvan, 2004), considering its variants and extensions. We review the method's well-studied problems and discuss how they apply to our modularity definition. We then survey a similar approach called Surprise, and perform experiments to highlight its difficulties.

1.3.1.1 Newman–Girvan modularity

One of the most well-known local measures is the Newman–Girvan modularity score (Newman, 2006; Newman and Girvan, 2004). The Newman–Girvan modu-



(a) Random graph with ACC 0.5

(b) Two cliques connected by a tree with ACC 0.5

Figure 1.4: Two networks, one with a clear modular structure, one without, that have the same average clustering coefficient.

larity of a full partition into clusters is the fraction of the edges that fall within the given clusters, minus the expected such fraction if edges were distributed at random. The score lies in the range $[-\frac{1}{2}, 1]$, where a score of 1 denotes perfect modularity, and a positive score indicates that the number of edges within partitions is higher than expected by chance for a graph with the same degree sequence. For a full partition into two clusters, Newman–Girvan modularity Q can be expressed as

$$Q = \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{d(i)d(j)}{2m} \right) (s_i s_j + 1), \tag{1.2}$$

where A is the adjacency matrix, m is the number of edges, $s_l = 1$ if node l belongs to partition 1 and -1 if it belongs to partition 2. Then $\frac{d(i)d(j)}{2m}$ is the expected number of edges between i and j in a random graph with the same degree distribution as G.

While the score was originally constructed for a 2-partition, it can be generalized to a pre-determined number k of partitions in one of two ways. Newman (2006) proposes a hierarchical algorithm where at first the graph is split into two partitions, then each partition is recursively split into two partitions, still taking into account the edges between partitions in the higher level. A partition is split only if there is a way of splitting it that results in a higher modularity than when leaving it intact. Alternatively, Clauset, Newman, and Moore (2004) propose an

extension of the modularity formula (1.2) as

$$Q = \sum_{ij} \left(\frac{A_{ij}}{2m} - \frac{d(i)d(j)}{(2m)(2m)} \right) \delta(c_i, c_j), \tag{1.3}$$

where c_u is the cluster of node u, and $\delta(c_i, c_j)$ is 1 if nodes i and j belong to the same cluster, 0 otherwise. A more convenient form is

$$Q = \sum_{i=1}^{k} \left(\frac{e_i}{m} - \left(\frac{d_{e_i}}{2m} \right)^2 \right), \tag{1.4}$$

where e_i is the number of edges in cluster i, d_{e_i} is the total node degree in cluster i (this is $2e_i$ if the graph is unweighted), and there are k clusters. Thus, Q quantifies the difference between the connectivity within the observed clusters and the expected value of this connectivity for the null model, a random graph with the same degree sequence.

Extensions to this score for directed networks (Leicht and Newman, 2008) and allowing clusters to overlap (Nicosia et al., 2009) also exist, but to our knowledge there is no variant that specifically takes into account the existence of nodes not in modules. The score (1.2) is NP-hard to optimize (Brandes et al., 2008). Therefore, multiple heuristics have been proposed. Newman (2006) suggests a procedure similar to spectral clustering (Luxburg, 2007), where instead of computing the eigenvectors of the Laplacian matrix and clustering their entries, the eigenvectors of the modularity matrix $M_{ij} = A_{ij} - \frac{d_i d_j}{2m}$ are used, where A is the adjacency matrix. We will see in subsequent sections how the eigenvectors and eigenvalues of some special matrices can tell us about network modularity.

Other authors applied standard optimization techniques to the problem: simulated annealing (Danon et al., 2005; Guimerà and Amaral, 2005) or extreme optimization (Duch and Arenas, 2005). In parallel, some heuristics have been developed, most notably the Louvain method (Blondel et al., 2008), whose fast implementation is used in several software packages and applications, such as Gephi[3]. The heuristic is divided into two steps, where in the first step a full partition with maximal modularity is greedily obtained by merging nodes whose merge gives the highest gain in modularity. In the second step a network is constructed whose nodes are the partitions from the first step, and step 1 can be recursively applied to this network. These iterations continue until no further improvement in the modularity can be achieved. The fast running time in practice makes this algorithm applicable to large networks with millions of nodes (Blondel et al., 2008). Due to its popularity and speed, in subsequent sections, whenever computing Newman modularity we use an implementation of this heuristic.

Several authors have pointed out drawbacks of the Q-score. One such prominent issue is the existence of an implied *resolution limit*, where clusters having

[3] http://www.gephi.org/

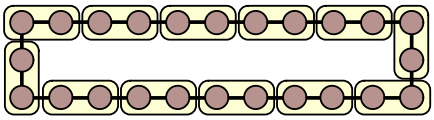


Figure 1.5: Clique ring (Good, Montjoye, and Clauset, 2010). Although the intuitive partition would be the partition into cliques, there is a number of cliques above which the optimal score is given to the partition that merges adjacent cliques.

size below a certain threshold are not identified (Fortunato and Barthélemy, 2007; Kumpula et al., 2007). The standard example is a graph comprising k cliques, each with c nodes. The cliques are connected to each other via a ring structure (Figure 1.5). Then, there is a value for k above which Q is higher for a partition that merges pairs of adjacent cliques (Good, Montjoye, and Clauset, 2010).

In this case, the null model expects the number of edges between clusters to decrease with k (in general, as the size of the network increases, the probability of any particular edge decreases). Therefore, the expected number of edges between two clusters in the null model can be < 1. In this case, even a single edge between two clusters as in the clique ring network implies a strong connection between them, and thus the solution with optimal modularity would merge them. Thus, Fortunato and Barthélemy (2007) show that in a network with m edges, clusters with fewer than $\sqrt{\frac{m}{2}}$ internal edges are not visible. In fact, this problem occurs not only for Newman–Girvan Modularity, but for all clustering algorithms where the investigated network is compared to some null network model (Potts-model-based community detection algorithms) (Kumpula et al., 2007).

Several modifications for Q were proposed to circumvent the resolution limit problem. For example, Arenas, Fernández, and Gómez (2008) add a *resolution parameter* γ that controls the importance of the null model term in the formula (1.3). By modifying the resolution parameter, it is possible to give preference to clusters of different sizes. However, since real-world networks can contain clusters of different sizes, this approach does not always scale. Adding self-loops to all nodes has also been shown to mitigate the resolution limit problem (Arenas, Fernández, and Gómez, 2008), however it cannot solve it in all cases.

While the resolution limit problem makes it difficult to identify what has been termed the most intuitive partition (Good, Montjoye, and Clauset, 2010), it has no immediate bearing on the score Q for the entire network, defined as the Q-value of the best partition. That is, we are interested in the highest score obtainable, regardless of partition, so we can compare the modular structure across networks. As Good, Montjoye, and Clauset (2010) point out, this is also not straightforward with Newman–Girvan modularity: Larger networks tend to

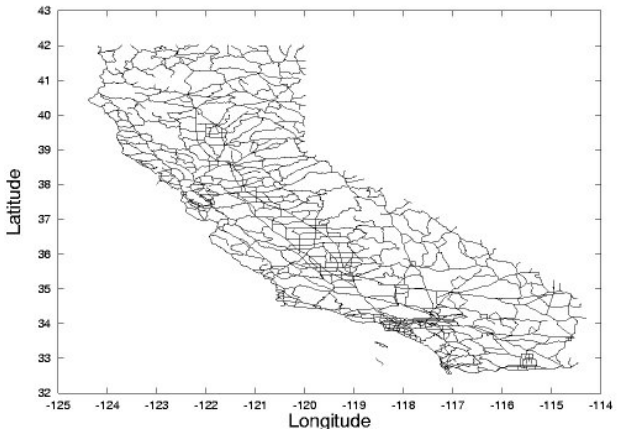


Figure 1.6: Road network of California. Source: http://www.cs.fsu.edu/~lifeifei. No apparent modular structure.

have a higher Q-score than smaller ones. This is again due to the null model: As the number of nodes increases, it is increasingly unlikely that any edges fall within a particular observed cluster, simply because of the huge number of possible connections. Therefore a large network ($\approx 10^6$ nodes in their example) is very different from a random graph with the same degree sequence, hence it will have high Q-score. This implies that it is not possible to simply compare the modularity of a large network with that of a small one, which makes this measure less useful.

In addition to the drawbacks described above, one other characteristic of the Newman score make it less suitable for our purpose: Ignoring the absolute density of modules. Specifically, networks that are tree and tree-like have a high Q-score (Bagrow, 2012). This means that sparse networks without any dense subgraphs have high modularity, despite containing no dense modules. As we would like our modules to be dense also in an absolute sense and not only relatively to the rest of the graph, the high scores obtained by tree graphs are a concern. To demonstrate this problem on real-world networks we used Gephi to compute the Q-score of several road networks, downloaded from http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm. The nodes of these networks are junctions, and edges are roads. They are naturally planar, and thus we would not expect to see dense modules (Figure 1.6). Indeed, these networks exhibit a very low average clustering coefficient (~ 0.01) and density (~ 0.0002). However, the average Newman–Girvan modularity is > 0.95. This property of the Q-score means that we cannot rely on it to quantify the existence of dense modules in networks.

1.3.1.2 Surprise

Around the time Newman and Girvan first proposed their modularity measure, Arnau, Mars, and Marín (2005) introduced a new measure of modularity for a full partition called *Surprise*. Recently (Aldecoa and Marín, 2011, 2013), some of the authors revisited Surprise and tested its performance on popular benchmark networks. The authors strongly claim that by maximizing Surprise one can determine the community structure of a network, much more accurately than with Newman–Girvan modularity (Aldecoa and Marín, 2013). The idea behind Surprise is also a comparison with a null graph model, but, in contrast to the Potts model variations (Kumpula et al., 2007), the comparison is based on probability: a hyper-geometric test for the number of edges between clusters. Given a full partition C with i_e edges within clusters and m edges in total, we first draw a graph uniformly at random from the set of all graphs on V having m edges. Then the Surprise of the partition is the probability that the random graph has at least i_e edges within the clusters defined in C. The lower this probability, the more surprising, and hence, better, the clustering. Fleck, Kappes, and Wagner (2014) describe it using an urn model: Given a partition with i_e edges within partitions, where i_p denotes the number node pairs that belong to the same cluster in C, our test is an urn with $\binom{n}{2} - i_p$ black balls and i_p white balls, and the Surprise S is the probability to draw at least i_e white balls when drawing m balls without replacement. The formula is then

$$S(C) = \sum_{i=i_e}^{m} \frac{\binom{i_p}{i}\binom{p-i_p}{m-i}}{\binom{p}{m}}, \tag{1.5}$$

where $p = \binom{n}{2}$. In later papers (Aldecoa and Marín, 2011, 2013), Surprise is defined as $S' = -\log(S)$, and thus ranges from 0 to infinity. A Surprise value of 0 is given to the partition into singletons and to the complete graph.

Maximizing Surprise is NP-complete on general graphs, but polynomial on trees (Fleck, Kappes, and Wagner, 2014). On general graphs no single algorithm has been shown to achieve consistently high results on accepted benchmark graphs. Surprise manages to bypass the resolution limit problem by taking into account the relative sizes of the partitions, optimizing over all partitions at once instead of summing over them as in Q.

Despite the success of Surprise in recovering the intuitive partition in the benchmark networks, it is difficult to test the claim that this is the definitive paradigm in community detection. This is because currently there are no good heuristics that consistently maximize Surprise, but rather only a "meta-heuristic" that selects the highest-scoring partition from those output by an ensemble of heuristics (Aldecoa and Marín, 2013).

Our preliminary experiments with the SurpriseMe software (Aldecoa and Marín, 2013) that implements this meta-heuristic raise some issues. Similarly to Q, it appears to suffer from the problem of larger and larger networks obtaining

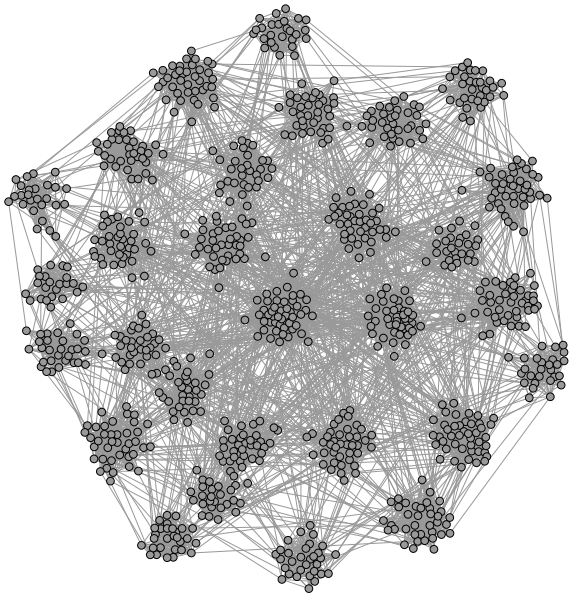


Figure 1.7: A modular network with 1000 nodes, from the SurpriseMe software package.

higher and higher Surprise values. This is most striking in trees. We generated balanced ternary trees with depth increasing from 2 to 8 (13 nodes to 9841, respectively). The maximum Surprise value obtained by SurpriseMe ranged from 2.3 for the small tree to 25103.6 for the large tree, increasing rapidly with tree depth. Similar results were obtained for random Erdős–Rényi (Erdős and Rényi, 1960) graphs.

For comparison, the intuitively highly modular network in Figure 1.7 obtains a maximum Surprise of 12812.1 with 1000 nodes. We see then that trees can obtain arbitrarily high Surprise values, despite containing no dense subgraphs. This example demonstrates that it is difficult to compare Surprise values across networks, as it is unclear whether a high Surprise is due to the number of nodes or to the modular structure of the network.

Similarly to the approach of Good, Montjoye, and Clauset (2010), we can test what happens when a new module of size $k > 1$ is added to an existing network with n nodes, increasing its size. Note that a partition with a high Surprise would have a large i_e and a small i_p (relative to p). In this case, i_p increases by k^2, but $p - i_p$ increases by kn. Therefore the ratio of white balls to black balls in the urn becomes smaller. In the case of a tree, it is easy to track i_e, as it receives exactly $k - 1$ new edges. In general, $i_e = o(n)$. Therefore on trees we see that i_e increases as the network size increases, while it becomes increasingly unlikely that a node pair drawn at random will belong to the same cluster, hence the increase in Surprise.

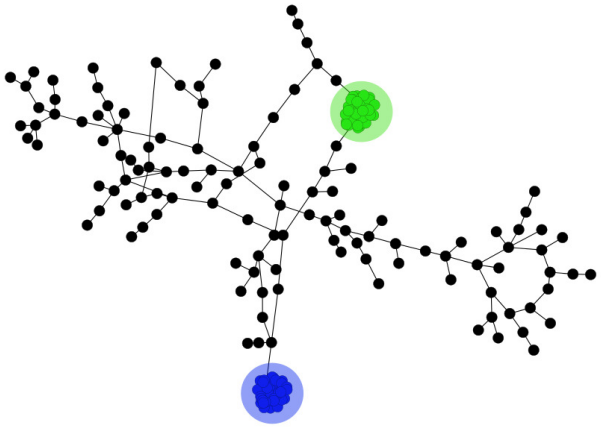


Figure 1.8: A modular network with two dense modules connected only via a sparse structure.

To conclude, Surprise is an interesting approach that sidesteps some of the pitfalls of Newman–Girvan modularity. However, much further research is needed to clarify whether this model can take the place of Newman–Girvan modularity for a full partition, and give consistently interpretable results.

Both the S and the Q scores described above are designed for a full partition. They are in that sense restrictive: only networks that can be neatly partitioned receive a high score. It is of course true that such a network would be modular; however, there are other classes of networks we would like to consider modular. Namely, we would like to think of networks that contain some very dense, well-separated subgraphs and sparse regions outside of them as modular, and test to what extent they contain modules. The network in Figure 1.8, which we'll denote the *adverse network* throughout this work, is an example of a network to which we would like to assign a high modularity score. Their S and Q scores do not reflect this: $Q = 0.59$, and $S = 1203$. The Newman–Girvan modularity points to the network being modular rather than randomly connected, but the score is not high. The Surprise value is difficult to interpret. We will return to this example in the next chapter, when we define modularity scores based on network dynamics.

1.3.2 Conductance: A graph-theoretic measure

The local scores presented above are all based on comparing a given full partition to some null model, with various trade-offs. In contrast, the graph theoretic measure of *Conductance* (Shi and Malik, 2000) is the basis for both a local score, since we can speak of the full partition with the highest Conductance, and a global score, since there is a property of a graph that quantifies the best such partition. We first state the definition of conductance for a node subset, then

extend the definition to a graph, following the introduction of Oveis Gharan and Trevisan (2014). The *Conductance* of a subset of nodes M in G is the ratio of edges going outside M (that is, having one endpoint in M and one in $\bar{M} := V \setminus M$) to the total number of edges adjacent to the nodes in M:

$$\Phi_G(M) = \frac{w(M, \bar{M})}{\nu ol(M)}, \tag{1.6}$$

where $\nu ol(M)$ is the sum of weights of edges in M (simply the number of edges if G is unweighted), and $w(M, \bar{M})$ is the weight of the *cut* between M and the rest of the graph, that is, the total weight of edges with one endpoint in M and one endpoint outside of M. The conductance of a graph is then the minimal conductance of any of its small subsets:

$$\Phi(G) = \min_{M : \nu ol(M) \leqslant \nu ol(V)/2} \Phi(M) \tag{1.7}$$

Intuitively, conductance of a graph is a measure of how "pull-apartable" it is. If it is easy to pull away a subset of nodes from the network, in the sense that it is loosely connected to the rest of the network, then this subset can be viewed as a module. We can then view Conductance as a local measure, where a natural approach for constructing a k-clustering of the network is to find k subsets of small conductance (Lee, Oveis Gharan, and Trevisan, 2012; Shi and Malik, 2000). Note that these k subsets are not necessarily a full partition! This can be expressed as:

$$\rho(k) = \min_{\text{disjoint } A_1, \ldots, A_k} \max_{1 \leqslant i \leqslant k} \Phi_G(A_i). \tag{1.8}$$

This is the maximum conductance of any k disjoint subsets of G.

While graph conductance captures the loose connectivity of a module to the rest of the graph, it gives no guarantees that the modules are dense. In order to fulfill this requirement there is another demand on the subgraphs induced by the subsets: That they, by themselves, have a high conductance (such graphs are called good *expanders* (Hoory, Linial, and Wigderson, 2006)). This way, we obtain clusters that are themselves not easy to pull apart, and are intuitively single units. For a subset M, define $\Phi(G[A_i])$ as the conductance of the graph induced by M in G. With this addition one can define a (Φ_{in}, Φ_{out}) clustering as k disjoint subsets $A_1, \ldots, A_k$ such that for all $1 \leqslant i \leqslant k$ (Oveis Gharan and Trevisan, 2014):

$$\Phi(G[A_i]) \geqslant \Phi_{in} \text{ and } \Phi_G(A_i) \leqslant \Phi_{out}. \tag{1.9}$$

A clustering is good if $\Phi(G[A_i])$ is large and $\Phi_G(A_i)$ is small.

It is interesting to ask whether there is an algorithmic way to find such a clustering, as opposed to exploring the entire solution space. Here, however, we are more concerned with determining whether a good clustering exists. Fortunately, there is a well-known connection between conductance and the

eigenvalues of the normalized Laplacian of G. Let A be the adjacency matrix of G with n nodes, and D the matrix with $D_{ii} = d_i$ and 0 everywhere else. Define the $n \times n$ normalized Laplacian $\mathfrak{L} := I - D^{-1/2}AD^{-1/2}$, where I is the identity matrix of order n. Let $v_1, \dots v_l$, respectively $\lambda_1, \dots \lambda_l$ be the eigenvectors and eigenvalues of $\mathfrak{L}$ so that $\mathfrak{L}v_i = \lambda_i v_i$. Then Cheeger's inequality states that

$$\frac{\lambda_2}{2} \leqslant \Phi(G) \leqslant \sqrt{2\lambda_2}. \tag{1.10}$$

That is, $\Phi(G)$ is bound from above and below by expressions involving the second eigenvalue of $\mathfrak{L}$. The graph G is then nearly disconnected (in the sense that one can pull apart a module) if λ_2 is close to 0. If $\lambda_2 = 0$ then the graph is disconnected. We will return to this property in the next chapter. More information about the Laplacian can be found, for example, in the Tutorial on Spectral Clustering (Luxburg, 2007).

Lee, Oveis Gharan, and Trevisan (2012) similarly bound $\rho(k)$ using the k-th eigenvalue:

$$\frac{\lambda_k}{2} \leqslant \rho(k) \leqslant O(k^2)\sqrt{\lambda_k}. \tag{1.11}$$

The authors extend this even further, connecting the eigenvalues of L with the existence of a good (Φ_{in}, Φ_{out}) clustering as follows: For any $k \geqslant 2$ if $\lambda_k > 0$, then for some $1 \leqslant l \leqslant k-1$ there is a l-partitioning of V into sets $P_1, \dots, P_l$ that is a $(\Omega(\rho(k)/k^2), O(l\rho(l))) = (\Omega(\lambda_k/k^2), O(l^3)\sqrt{\lambda_l})$ clustering. This is an l-partitioning of V such that the $\Phi(G[A_i])$ of each set A_i is significantly larger than its $\Phi_G(A_i)$. Therefore a large enough gap between λ_l and λ_k in the spectrum of $\mathfrak{L}$ implies a clustering that has dense clusters that are loosely connected to one another. Oveis Gharan and Trevisan (2014) additionally propose an algorithm running in polynomial time to find a partition that has slightly worse guarantees on (Φ_{in}, Φ_{out}). However, this algorithm needs to be seeded with modules having certain (Φ_{in}, Φ_{out}) values, and are then extended to full partitions. Since we are interested exactly in those subsets that are expected as input, this algorithm is not suitable for our purposes.

We therefore see that a gap in the spectrum of some matrix representing the network implies the existence of "good" modules. However, is the converse true? Does every network that has a good clustering also have a gap in the spectrum to match? In fact, it is not known that every good clustering is reflected in a sizeable spectral gap. When we next analyze network modularity through the lens of metastability, we will encounter other matrices connected with the network and make use of their spectrum, also revisiting the spectral gap as a score.

Chapter 2

Connecting network modularity and metastability

We introduce a new dynamics-based approach to network modularity. We start by providing background on time-discrete and time-continuous random walks, and equate network modules with metastable sets of a Markov process. We review important concepts such as committor functions and the eigenvector projection error. We then introduce our Markov State Model (MSM)-based score, termed I_m. In the subsequent sections we provide a formal proof that I_m matches our intuition on a specific class of networks, and support this with numerical experiments. We next compare the outcome of I_m with that of previously introduced modularity scores on networks where the optimal assignment is known by construction, and test I_m on some real world networks.

2.1 Random walks on networks

Our approach is based on the perspective of a *random walk* on the network. The idea is that of a random walker who traverses the network from node to node via the network edges, choosing at each time step a random neighbor of its current node to move to. This random walker would tend to stay longer in dense, highly connected regions of the network, moving between such regions rarely. That is, it would spend more time within modules. This is the behavior that many random-walk based algorithms, such as the popular Markov Clustering (MCL (Dongen, 2000)), try to capture. The following definitions and background can be found for example in the book by Bremaud (1999).

Define the (perhaps weighted) time-discrete random walk on $G(V, E)$. When in some node x, the random walker will jump to some neighbor node y with transition probability

$$p(x, y) = \frac{w(x, y)}{s(x)}, \tag{2.1}$$

where $s(x)$ is the weighted degree (strength) of x. We can view the sequence of nodes visited by the random walker as a *Markov chain* $\{X_n\}$, so that

$$\mathbb{P}\left[X_{n+1} = x \mid X_n = x_n, \ldots, X_1 = x_1\right] = \mathbb{P}\left[X_{n+1} = x \mid X_n = x_n\right]. \tag{2.2}$$

We will concentrate on time-homogeneous Markov chains, that is, $p(x, y)$ does not depend on the time step n, thus the chain can be defined by the transition function

$$P(x, y) = \mathbb{P}\left[X_{n+1} = y \mid X_n = x\right], \tag{2.3}$$

and the transition matrix P with entries $p(x, y)$. The probability to be in state y after k steps when starting in some x is

$$\mathbb{P}\left[X_{k+1} = y \mid X_1 = x\right] = P^k(x, y), \tag{2.4}$$

which are exactly the entries of the matrix P^k.

Let $\pi(x)$ be the initial distribution, $\pi(x) = \mathbb{P}[X_1 = x]$. Then the probability to be in node y after k steps when starting distributed according to π is

$$P[X_k = y] = \sum_{x \in V} \mathbb{P}[X_{k-1} = x]\mathbb{P}[X_k = y \mid X_{k-1} = x] = \sum_{x \in V} \pi(x)P^k(x, y). \tag{2.5}$$

It will be important to define the *invariant measure* (here also: *stationary distribution*) μ of a Markov chain with transition matrix P. This is a non-negative vector such that $\mu P = \mu$, thus it is a left eigenvector of P. Additionally we require $\sum_x \mu(x) = 1$. If we define as before $p(x, y) = w(x, y)/s(x)$ then:

$$\mu(x) = \frac{s(x)}{\sum_{y \in V} s(y)}. \tag{2.6}$$

From the fundamental theorem of Markov chains for the existence of a unique stationary distribution (Beichelt and Fatti, 2001), we have that regardless of the initial distribution, if the network is undirected, connected, and non-bipartite, after enough steps it will end up in state y with probability $\mu(y)$. Therefore we can assume that the Markov chain is initially distributed with the stationary distribution μ, that is, $\pi = \mu$.

For such networks, the corresponding Markov chain and its transition matrix $P_{n \times n}$ have some nice properties. Unlike the Laplacian matrix from Chapter 1, P is not symmetric. However, since the adjacency matrix A is symmetric (as the network is undirected), the random walk is *reversible* in time. Importantly, this means that *detailed balance* is satisfied:

$$\mu(x)p(x, y) = \mu(y)p(y, x), \forall x, y \in V \tag{2.7}$$

Then P is like a symmetric matrix. We can use μ to define the *weighted scalar product*:

$$\langle u, v \rangle_\mu = \sum_{x \in V} u(x)v(x)\mu(x), \tag{2.8}$$

Now, $\langle u, Pv\rangle_\mu = \langle Pu, v\rangle_\mu$. The Perron–Frobenius theorem (Meyer, 2000) states that:

1. P has a unique eigenvalue $\lambda_1 = 1$, with the corresponding right eigenvector $P\vec{1} = \vec{1}$, with $\vec{1} = (1, \ldots, 1)$ and the positive left eigenvector $\mu P = \mu$.

2. All other eigenvalues of P are also real and strictly smaller (in modulus) than λ_1: $1 = |\lambda_1| > |\lambda_2| \geqslant |\lambda_3| \geqslant \cdots \geqslant |\lambda_n|$.

This theorem forms the basis for our discussions of the spectral gap and the perturbation analysis in Section 2.3.

2.1.1 Spectral clustering and the spectral gap

We have already seen that existing scores fail to capture our intuition with regards to the modules being simultaneously dense and loosely connected to the rest of the graph. We have also seen that the spectrum of some matrix encoding the network can give us an indication of its modular content. Indeed, the eigenvalues and eigenvectors of graph matrices have long been used in connection with network partitioning.

Luxburg (2007) reviews a class of clustering algorithms called *spectral clustering*. These algorithms use the eigenvalues and eigenvectors of the Laplacian matrix $\hat{\mathfrak{L}} = D - A$, with adjacency matrix A and diagonal matrix D with the node degrees on the diagonal, and variants like the normalized Laplacian matrix $\mathfrak{L} = I - D^{-1/2}AD^{-1/2}$ introduced when discussing Conductance in Section 1.3.2.

The eigenvectors are used to identify clusters: when the goal is a k-partition, compute the first k eigenvectors $v_1, \ldots, v_k$ of $\mathfrak{L}$, and place their columns in a matrix Q. The rows of Q are then viewed as points in $\mathbb{R}^k$, and can be clustered using a standard clustering algorithm such as k-means. Node i corresponds to row i of Q, and we assign it to the same partition as point Q_i. This algorithm is based on the fact that the eigenvectors are a relaxation of the indicator vectors that optimizes

$$\text{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{w(A_i, \bar{A}_i)}{|A_i|} \tag{2.9}$$

Deuflhard et al. (2000) make similar use of the eigenvectors of the transition matrix $P = D^{-1}A$. They start by looking at the case where where each cluster is a connected component, and prove that rows of the eigenvector matrix Q corresponding to nodes in the same connected component have the same *sign structure*: $\text{sgn}(Q_{ik}) = \text{sgn}(Q_{jk})$ for all k and nodes i, j in the same connected component. Then using perturbation theory they prove that by adding edges between the modules to make the graph connected, as long as not too many edges are added, the sign structure of the rows of Q is also perturbed only a little. Thus the rows of Q can be used for clustering.

While the eigenvectors can be used for clustering, the eigenvalues can be used to determine the number of clusters, and thus indicate the existence of a modular structure: If there is a gap in the spectrum after k eigenvalues, it indicates the existence of a k-partition (Deuflhard et al., 2000). This can also be explained in terms of perturbation: In the uncoupled case, where the graph is disconnected into k connected components, it is well-known that P has k eigenvalues equal to 1, and these are the largest eigenvalues. When edges are added between components to connect the graph, the largest eigenvalue of P will still be 1, but the next $k-1$ eigenvalues will be close to 1, or "above the gap".

While a gap in the spectrum indicates the existence of modules, the converse is not always true. The by now familiar adverse network in Figure 1.8 contains two very dense clusters. We would expect the spectrum to reflect this by exhibiting a gap after the first two eigenvalues. However, there is no discernible gap. This is due to the fact that other substructures of the graph also have similar properties to the dense modules. Specifically, they also exhibit *metastability*, that is, they are *metastable sets* of the random walk process.

Next we will discuss metastability and its application to analyzing the modular structure of the network.

2.1.2 Modules and metastability

We take a closer look at modules in a given network $G(V, E)$. Assume we have k modules, defined as disjoint subsets $C_i \subseteq V$, $i = 1, \dots, k$. The union of all modules form the set $\mathcal{M} = \bigcup_i C_i$. We are not interested in a full partition, but rather in an assignment, containing both modules and non-modules. Thus we assume that $\mathcal{M}$ does not contain all nodes of the network, meaning, there is the non-empty set $\mathcal{T} = V \setminus \mathcal{M}$, which we call the *transition region*. Further define the set $\mathcal{M}_i = \mathcal{M} \setminus C_i$, the union of all modules except C_i. We would like to say that the modules are *metastable sets* of the random walk, that is, on the one hand they are attractive for the random walk while, on the other hand, they are well-separated in the sense that transitions between them are rare. More formally, the random walk has metastable sets, $C_i \subseteq V$, $i = 1, \dots, k$, if it exhibits a specific relation between two timescales: the typical *return time* R the random walk needs to enter one of the C_i, if started outside of any module, is small compared to its typical *waiting time* W between transitions from one of the C_i to another one.

In order to quantify these timescales we first denote by $\tau(A)$ the expected time the process needs to enter a set A. Then $\mathbb{E}_y(\tau(\mathcal{M}))$ denotes the expected entry time of the process into an arbitrary one of the C_i, if started in some node y in the transition region. Likewise, $\mathbb{E}_i(\tau(\mathcal{M}_i))$ denotes the expected entry time into one of the C_j with $j \neq i$, if it is started distributed according to the invariant measure μ (2.6) from C_i. The random walk is metastable with regards to the sets

C_i, $i = 1, \ldots, k$, if

$$R := \max_{y \notin \mathcal{M}} \mathbb{E}_y(\tau(\mathcal{M})) \ll \min_{i=1,\ldots,k} \mathbb{E}_i(\tau(\mathcal{M}_i)) =: W, \tag{2.10}$$

or, equivalently, if the relation of return and waiting time is small, $R/W \ll 1$. Note that there may be many different collections of disjoint sets C_i such that R/W is small. We could then view R/W as a score for an assignment that, unlike the scores reviewed in the previous chapter, does not assume that the network completely decomposes into modules.

It is then tempting to use R/W as a local score for the network, our first metastability-based score: The R/W score of a given network is exactly the R/W score of the best assignment. Unfortunately, we cannot simply define the optimal modules via the property of minimizing R/W since every full partition, that is, every choice with $\mathcal{T} = \emptyset$, leads to $R = 0$. Similarly, it is not straightforward to compare assignments with a different number of modules. For example, consider an assignment a_1 with 3 modules $\mathcal{M}^1 = C_1 \cup C_2 \cup C_3$. Compare it to assignment a_2 with 2 modules: $\mathcal{M}^2 = \gamma_1 \cup \gamma_2$, where $\gamma_1 = C_1$ and $\gamma_2 = C_2 \cup C_3$. The value of R is the same for both assignments, since the partition into T and $\mathcal{M}$ is the same. However, the expected waiting time W between transitions in a_2 is higher than in a_1, since any transitions between the original C_2, C_3 now count within γ_2. Such an assignment with fewer modules would always be as least as good as one with more modules. To summarize, while we can still use the R/W score to compare assignments, at least those with the same number of modules, we need to look elsewhere for a more widely-applicable, metastability-based, network modularity score.

We stay, therefore, with the idea of metastability. We have said that dense modules are metastable sets of the random walk process. Are there other regions (subgraphs) in the network that are metastable sets? Recall the definition of the expected waiting time W: this it the typical time it takes the random walker to transition to a module when starting in a different module. Putting aside the time spent in the transition region, the waiting time depends on the amount of time the random walker spends inside the module it starts from. If the module is large and contains many edges, relative to outgoing edges, then at any time step a random walk would, with high probability, jump to another neighbor within the module, rather than outside. Thus the waiting time is high. Now, consider a different type of subgraph: a long loop, or a linear chain, as exemplified in Figure 1.8. A random walker starting somewhere within such a structure, if large enough, would also stay in it for long timescales, going back and forth and being able to exit the structure only at the endpoints. We must conclude, then, that every method for approximating modularity that relies on metastability will suffer from effects of long chains and loops as metastable sets. Can we do anything to combat this? Sarich, Djurdjevac, et al. (2013) explore this issue by using a *time-continuous random walk*, which we will describe next.

2.1.3 The time-continuous random walk

We restrict ourselves now to unweighted networks, though the results below can be generalized to weighted networks as well. Now introduce a family of time-continuous random walks, that is, time-continuous processes on the finite state space of nodes V (also: Markov jump processes). Such processes are different from time-discrete processes by the addition of an exponentially-distributed *waiting time* before each jump from node to node. Thus, instead of jumping between nodes at discrete time steps $n = 1, 2, 3, \ldots$, jumps can occur at any time t after the waiting time specific to the node has passed. The process is further characterized by the jump probabilities between the nodes: After waiting the random walker chooses a neighbor to jump to based on the given probability distributions. The transition rules of the processes that interest us can be summarized as the following family of *rate matrices*

$$L(x,y) = \begin{cases} -\frac{1}{d(x)^p}, & x = y \\ \frac{k(x,y)}{k(x)d(x)^p}, & x \neq y, (x,y) \in E \\ 0, & \text{else,} \end{cases} \tag{2.11}$$

where $p \geqslant 1$, $k(x,y)$ are some non-negative weights of our choice such that $k(x,y) = 0$ if $(x,y) \notin E$ and $k(x,y) = k(y,x)$, and $k(x) = \sum_y k(x,y)$. The waiting times are encoded on the matrix diagonal of a rate matrix as follows: If being in node x, the expected waiting time until the next jump away from x is inversely proportional to $|L(x,x)|$, and $L(x,y)/|L(x,x)|$ is the probability that this jump leads to y. Therefore, in our particular family of rate matrices, the expected waiting time in a node is proportional to its degree $d(x)$, that is, the higher the degree, the higher the waiting time. Intuitively we can say that the more neighbors a node has, the longer, on average, it takes the random walker to "decide" where to go next. If we further set

$$k(x,y) = A(x,y) \cdot (1 + \langle a_x, a_y \rangle), \tag{2.12}$$

where a_z is the zth row of the adjacency matrix and $\langle \cdot, \cdot \rangle$ is the usual Euclidean scalar product, we can start to see the advantage of this approach over the discrete random walk: Nodes with high degree and high clustering coefficient become very attractive for the random walker. Now consider the behavior of such a random walker on the problematic chains and loops: the degree of the nodes in these regions is low, and their clustering coefficient is also low. Therefore, the random walker would spend little time in those regions, thus making them not metastable.

A Markov jump process with a rate matrix (*generator*) of this form has the unique invariant measure

$$\mu(x) = \frac{1}{Z} d(x)^p k(x) \tag{2.13}$$

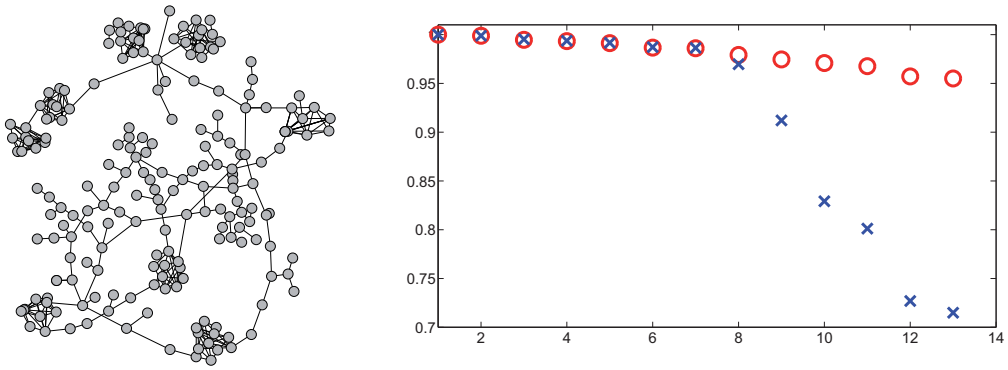


Figure 2.1: Example network and first 13 eigenvalues λ of standard random walk (red circles) and time-continuous random walk (blue crosses). Figure reproduced from Sarich, Djurdjevac, et al. (2013)

with the normalization constant Z so that the entries of μ sum up to 1. This process is always reversible, so it holds that $\mu(x)L(x,y) = \mu(y)L(y,x)$. From the rate matrix (2.11) we can also directly compute the transition matrices of the random walk by $P_t = \exp(tL)$. Its entries $P(t,x,y)$ denote the transition probability from node x to node y in time t.

We have then two interesting matrices: L and P_t, that characterize a random walk that favors dense regions. It is only natural to then check whether their spectrum reflects the modularity of a network without being distracted by sparse metastable structures.

Since our random walk is reversible, all eigenvalues Λ of the rate matrix L and λ of the associate transition matrix P_t are real with $\Lambda_0 = 0$, respectively $\lambda_0 = 1$ being the largest ones, and $\lambda = \exp(t\Lambda)$ in general. It is well-known that there is a direct relation between the existence of metastable sets and the largest eigenvalues of the transition matrix P_t of the random walk (Djurdjevac, Sarich, and Schütte, 2012; Huisinga and Schmidt, 2006; Sarich and Schütte, 2011; Schütte and Huisinga, 2003).

For example, in the case of just two metastable modules, there is exactly one other eigenvalue $\lambda_1 = \exp(t\Lambda_1)$ of P_t close to $\lambda_0 = 1$ such that $\mathbb{E}_1(\tau(\mathcal{M}_1))$ is approximately given by $1/|\Lambda_1|$, while all other eigenvalues of P_t are significantly further away from 0. Whenever there is a gap after the leading m eigenvalues $\lambda_0, \ldots, \lambda_{m-1}$ close to 1 we will find m metastable sets, and the longest relaxation timescales of the random walk, $t_i = 1/|\Lambda_i|$, are encoded by the leading eigenvalues $\Lambda_0, \ldots, \Lambda_{m-1}$.

Figure 2.1 depicts a network along with the spectrum of the standard and the time-continuous random walk.

While the spectrum of the standard transition matrix P does not offer a

clear gap to give an idea about the number of modules being present, the time-continuous random walk shows a small gap after 7 and a strong gap after 8 eigenvalues. This exemplifies how the continuous random walk leads to a better coherence of connectivity, the idea of modules, and metastability. We will test the performance of this score in Section 2.4.

Sarich, Djurdjevac, et al. (2013) go on to introduce a method of finding a good clustering by identifying modules as optimal metastable sets. We will elaborate on this method in the next chapter, when we discuss approaches for module finding in networks that are not fully partitionable. We will continue laying the groundwork for the metastability-based score. To do so we will need to define a few other concepts. A complete description can be found in several publications (Djurdjevac, 2006; Djurdjevac, Bruckner, et al., 2011; Djurdjevac, Sarich, and Schütte, 2012; Metzner, Schütte, and Vanden-Eijnden, 2009).

2.1.4 Fuzzy decomposition and committor functions

Since we are interested in networks that do not decompose completely into clusters, we consider a *fuzzy* decomposition of the network. Where in the full partition case we have a function that assigns each node to exactly one module, we now allow nodes to belong to any number of modules. Thus, for every node x we specify its affiliation $f_i(x) \in [0,1]$ to module i such that $\sum_i f_i(x) = 1$. Nodes that we can safely assign to a single module j will have $f_j = 1$, while nodes that can be viewed as shared between several modules would have approximately equal affiliation to all of them. On the other hand, nodes that can be said to belong to no module would have very low affiliation to any module. See Figure 2.2 for an example. The set of vectors associating each node to each module with some probability is a *fuzzy assignment*, as we defined in Section 1.2.

Given an assignment with modules C_i, we can use random walks to define this affiliation and obtain a fuzzy assignment. We start the random walk in node x and see which module it will enter next. Then, we set the affiliation $f_i(x)$ to be the probability that the next module to be entered is C_i. Such affiliation functions are called *committor functions*, or simply: *committors*. The committors can be computed very efficiently by solving sparse, symmetric and positive-definite linear systems. For sets $C_1, \dots, C_n$ Metzner, Schütte, and Vanden-Eijnden (2009) show that the committor f_i solves the linear system

$$\begin{aligned} (Lf_i)(x) &= 0 \quad \forall x \in \mathcal{T} \\ f_i(x) &= 1 \quad \forall x \in C_i \\ f_i(x) &= 0 \quad \forall x \in C_j, j \neq i. \end{aligned} \tag{2.14}$$

We will meet this formulation again in Section 2.3 when we approximate the committors for a specific class of networks.

This fuzzy decomposition can also be interpreted in the sense of a coarse graining of our random walk by Markov State Modeling (MSM) (Deuflhard

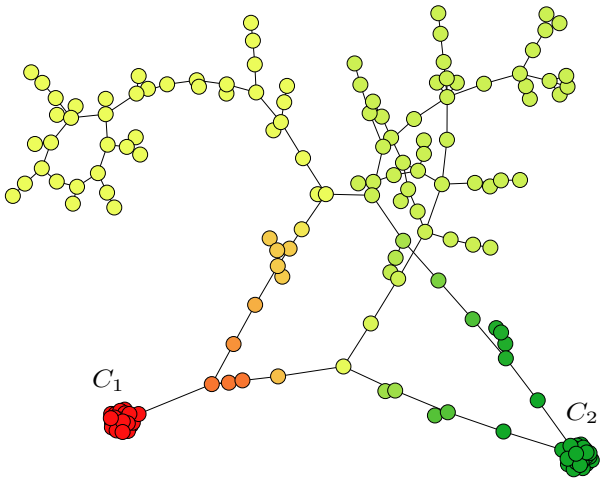


Figure 2.2: Affiliation of the nodes to the module C_1. Red color indicates values close to 1, green close to 0. The nodes that already belong to C_1 have the highest affiliation to to it, the nodes in a different module C_2 are not affiliated with C_1 at all, and the transition region nodes have a high or low affiliation, depending on their position in the network. Figure reproduced from Djurdjevac (2006)

et al., 2000; Djurdjevac, Bruckner, et al., 2011; Djurdjevac, Sarich, and Schütte, 2010, 2012; Schütte, Noé, et al., 2011). Thus we obtain a low dimensional approximation of the random walk process: instead of jumping between states (nodes), the random walker jumps between metastable sets (modules). In the full partition case, where every node belongs to exactly one module, this low dimensional space is the space spanned by step functions that are constant on the modules. In the fuzzy case, the space is spanned by the committor functions. In this case, we find a coarse grained random walk that jumps between the modules but now takes the dynamics on the transition region $\mathcal{T}$ into account, according to the committors (Sarich and Schütte, 2011). More formally, we find sets $C_1, \ldots, C_m$ such that the longest *relaxation timescales* of our random walk, encoded by m dominant eigenvalues $\lambda_0, \ldots, \lambda_{m-1}$ of the original random walk P, are close to the timescales of the coarse grained random walk, encoded by the eigenvalues $\hat{\lambda}_0, \ldots, \hat{\lambda}_{m-1}$ of the coarse-grained process denoted $\hat{P}$. We conclude, then, that to find a good fuzzy assignment we can find modules as metastable sets, and then determine the fuzzy decomposition of the transition region via the committors.

Djurdjevac, Sarich, and Schütte (2010, 2012) show that

$$\max_{i=0,\ldots,m-1} |\lambda_i - \hat{\lambda}_i| \leqslant \lambda_1 (m-1) \max_{i=0,\ldots,m-1} \delta_i^2, \tag{2.15}$$

Let Q be the orthogonal projection into the committor space. Then

$$\delta_i^2 = \|u_i - Qu_i\|_\mu^2 \tag{2.16}$$

is the associated projection error, where u_i is the normalized eigenvector of the random walk P with respect to λ_i. Previously, Djurdjevac, Sarich, and Schütte (2012) demonstrated that it is possible to get a good low-dimensional approximation (and thus a good fuzzy assignment) when the dominant eigenvalues of P are well approximated by the eigenvalues of the projected matrix $\hat{P}$. Now we can further quantify this by requiring that the projection error of the corresponding eigenvectors (2.16) is small enough. If there is a fuzzy assignment that induces a coarse graining that is a good approximation of the original dynamics, then we define the network as having good modularity. In the next section, we will use this projection error to define the modularity score.

2.2 A score based on Markov State Models

We can now finally introduce a modularity score that quantifies the modular content of a network via the best possible approximation of the dominant eigenvalues of the corresponding transition matrix P by a Markov State Model (MSM). Let u be some eigenvector of P, Q the projection matrix into the subspace spanned by the committors for some choice of modules and transition region, and μ the invariant measure. Then we know that the best approximation is achieved when the projection error $\|(\mathrm{Id} - Q)u\|_\mu^2$ is small for some appropriate choice of committors. That is, when the difference between the eigenvector and its projection into the committor space is small, the error is small.

We then define $\kappa = \|Qu\|_\mu^2$. As $1 - \kappa = \|(\mathrm{Id} - Q)u\|_\mu^2$ we get that a large κ implies a small projection error (2.16). For eigenvalues $1 = \lambda_1, \lambda_2, \ldots, \lambda_n$ and their corresponding eigenvectors $\vec{1} = u_1, u_2, \ldots, u_n$, we define $\kappa_2 = \|Qu_2\|_\mu^2$, and the score:

$$\boxed{I_m(P, q) = \frac{1}{2}(1 + \lambda_2\kappa_2)} \tag{2.17}$$

for the network specified by P and a fuzzy assignment specified by committors q, where λ_2 is the second eigenvalue of P.

A possible intuition is that we multiply κ_2 by λ_2 since a modular network (at least one whose transition matrix can be viewed as a perturbation of a block matrix) will have a second eigenvalue close to 1.

One noticeable aspect of I_m is that it is a local score: the score depends on the choice of modules and transition region. The score I_m of a network with the associated transition matrix P is then the maximum score obtainable for any set of committors. Therefore, if there is a choice of modules and transition region such that κ_2 is high, the score will be high.

Note that in this way we are not dependent at all on any sort of gap in the spectrum of P, which is unreliable, but rather on the existence of good committors.

We could ask, when composing the score, whether there can be an advantage to utilizing additional eigenvectors, corresponding to other leading eigenvalues. We can expect these additional leading eigenvalues in cases where there are additional modules. Consider the case of an optimal assignment (in the sense that it maximizes the score) and appropriate committors into k modules and a transition region. Define

$$\tilde{I}_m = \frac{1}{k}(1 + \lambda_2\kappa_2 + \lambda_3\kappa_3 + \cdots + \lambda_k\kappa_k). \tag{2.18}$$

Here $\kappa_i = \|Qu_i\|_\mu$. Additionally, u_i, λ_i are the ith eigenvector and eigenvalue of P_t. Again Q is the projection into the space of committors, where this time the committors are to k modules instead of two. For $k = 2$ we have $I_m = \tilde{I}_m$.

There are some advantages to $\tilde{I}_m$ over I_m in some scenarios. It takes into account all the pertinent information about the modules, as encoded in the leading eigenvalues. This can be helpful in case the modular structure is complicated, for example if there are multiple modules with different densities. However, computing $\tilde{I}_m$ requires computing more eigenvalues and eigenvectors. It is also not easy, in this case, to compare scores of networks with a different number of modules in the optimal assignment. We can construct simple network classes where networks with fewer modules will have a higher score then similar networks with more modules. Therefore, we investigate the I_m score, and leave the analysis of $\tilde{I}_m$ to future work.

Having defined the score, the remainder of this section is dedicated to some examples of the performance of this score on special networks.

When computing I_m we must start from some assignment into modules and transition region. Without the best assignment, we cannot compute I_m optimally. If we are given an assignment obtained in some way, perhaps via some clustering algorithm, we can compute some $\kappa_{\text{practical}} < \kappa_2$ for this assignment, and obtain a lower bound for the score. We can of course always compute λ_2 exactly. In Chapter 3 we explore some methods for identifying good metastable modules and a transition region. In the meantime, for the following small examples, we can exhaustively enumerate assignments into 2 modules and a transition region, compute the score for each, and use the κ_2 of the best assignment.

Example 1: The complete graph. A clique K_n is the densest graph possible on n nodes, and thus we view it as a single module: It cannot be naturally partitioned into smaller modules. For example, it has the highest possible conductance. A clique is not modular, and therefore its modularity score should be low. To demonstrate, we tested all possible homeomorphic partitions of K_{20} into 2 modules and a transition region (Figure 2.3), and then computed I_m based

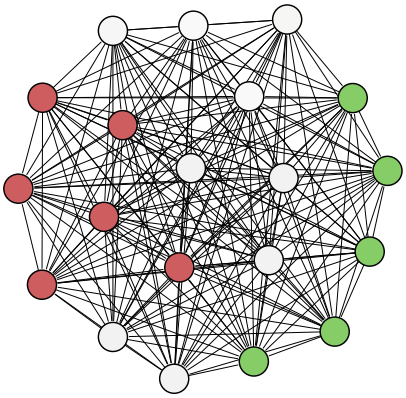
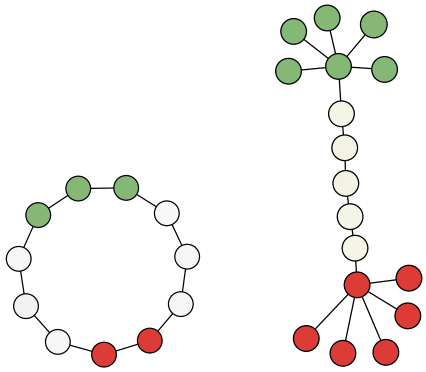


Figure 2.3: The complete graph K_{20} with one of the fuzzy assignments into two modules (green and red) and a transition region (white).

on P_t with $t = 100$ with the generator (2.11). We can derive this directly from the formula for I_m (2.17): The second eigenvalue of P_t is 0 or very close to it.

Example 2: Sparse graphs. In contrast to Newman modularity, which, as we have seen previously, assigns a high score to sparse networks, we would like sparse graphs such as trees to have a low modularity score. We can begin with two simple examples: a cycle and a path with stars (Figure 2.4). The advantage of the simple structure is that there is a limited number of possibilities for partitions, and hence we can feasibly try all possible ones. The graph C_{11} is the cycle on eleven nodes. As in the clique case, we test all its partitions (up to isomorphism) into 2 modules and a transition region. We use P_t for $t = 1, 100$, and 1000. We obtain $I_m = 0.5$ or close to 0.5 (minimum) in almost cases, as desired. Here we see the influence of the choice of t on the score: Computing the second eigenvalue we find that for $t = 1$, that is, for a very short time, $\lambda_2 \approx 0.8$. For the higher values of $t = 100$ or 1000, however, $\lambda_2 = 0$ and the score is accordingly minimal. To make the path case more interesting to test, we analyze the structure of a slightly more complicated graph: a path whose every end node is also the center of a star. For the numerical experiments, we generated a path of length 7, where each end node is adjacent to 5 additional degree-1 nodes. Here we notice a difference in the score when using different generators: When using the generator (2.11), the score is minimal. When taking $k(x, y) = 1$ for all x, y, that is, basing the dynamics on degree only and not on the clustering coefficient, we get a higher score of ≈ 0.7 for the assignment that associated the stars with modules and the path between them to the transition region.

Example 3: The adverse network. We return once again to the network that we tested when demonstrating the drawbacks of other scores (Section 1.3.1.2). This



(a) Cycle. (b) Path with stars.

Figure 2.4: Some sparse graphs that obtain intuitively correct I_m scores.

network contains two dense modules, connected by a sparse structure consisting of multiple long paths and cycles (Figure 1.8). We previously saw that the gap of the Laplacian does not indicate the modular structure of this network, and that Newman modularity is only ≈ 0.6. Computing the score for the planted assignment with the two intuitive modules gives $I_m = 0.996$, a high score. Thus I_m successfully captures the modularity of this difficult network.

2.3 Proof of performance on special networks

We would like to prove that I_m matches our intuition. That is, networks with intuitively high modular content obtain a high score, while those with low modular content have only a low score. We will show this for a specific class of networks.

The proof below is based on perturbation analysis. We prove that for a class of stochastic P constructed in a certain way as perturbations of some block matrix, the score is high when the perturbations are small. The basic idea is not new: Perturbation analysis has been used to motivate spectral clustering (Luxburg, 2007). Importantly, we can build on the ideas of Deuflhard et al. (2000). They follow the work of Stewart (1983) on general primitive stochastic matrices (corresponding to connected, non-bipartite networks), but extend it to reversible matrices. Luxburg (2007) summarizes the approach as follows: Perturbation theory studies how the spectrum of a matrix A changes if we add a small perturbation in the form of some matrix H with a small norm, that is, we compare the eigenvalues and eigenvectors of A and $\tilde{A} = A + H$. The comparison can be defined exactly by computing the distance (measured in *canonical angles*) between the subspaces spanned by the corresponding sets of eigenvectors from A and $\tilde{A}$. For symmetric matrices, this distance is bounded by a constant that depends on

the 2-norm of the perturbation H, and the "goodness" of the spectral gap: the more well-separated the first k eigenvalues are from the rest of the spectrum, the better the bound, and the closer the eigenvalues and eigenvectors of A and $\tilde{A}$.

The problem of fully partitioning the network can then be posed in perturbation theory terms: the matrix A corresponds to the uncoupled Markov chain (Deuflhard et al., 2000), or a network where each connected component is a module. This is an ideal modular network that would have an ideal modularity score. Then the perturbation H adds edges between the modules, making the resulting network represented by $\tilde{A}$ connected. The smaller the perturbation, the closer the network is to the ideal modular network, and the higher the modularity score. The matrix A in the traditional spectral clustering approaches is the Laplacian, which is symmetric and therefore amenable to the analysis above. Deuflhard et al. (2000) work with the transition matrix P, which is not symmetric, but reversible. They show that in this case the eigenvalues and eigenvectors of $P = D + E$, where D is this time block diagonal and $\|E\|$ is small, are close enough to the eigenvectors and eigenvalues of D so they can be used for clustering.

We are of course not interested in the full partition, but in a fuzzy assignment. Given $P_\varepsilon = P_0 + \varepsilon L$, where P_0 is a block matrix and εL is the perturbation matrix (formal definition below), we first prove that all components of the score: u_2, λ_2 of P_ε, the projection Q, and the committors q_i, are perturbations of the corresponding entities of P_0. In particular, this means that the eigenvectors are slight perturbations of step functions.

Why is this what we want? Assume we have some P_0, comprised of two modules and a transition region. We now look at two networks having different perturbations εL. Let the second network be more "problematic", hence less modular, than the first network: Suppose the modules are less dense due to the perturbation lowering the average node degree inside modules. This translates into a higher value of ε for the perturbation in the second network, and if we formulate the perturbation such that $\|L\|$ is constant for some norm (and we show that such a formulation is possible!) that means that the lower modularity implies larger perturbation.

Next we need to be convinced that a larger perturbation implies a lower score. By proving that the score components are perturbations of the step function, we get exactly that: The larger the perturbation, the further the components are from step functions. Specifically the committors would be far from step functions, and $\|Qu_2\|$ will be lower. When we have no perturbation at all, that is, $P_\varepsilon = P_0$, the score is optimal: In this case $\lambda_2 = 1$, $\kappa_2 = 1$, as the committors are also constant on the block functions, and $\|Qu_2\|_\mu^2 = \|u_2\|_\mu^2 = 1$, as needed.

2.3.1 Definitions

Let $P_\varepsilon = P_0 + \varepsilon L \in \mathbb{R}^{n\times n}$ such that P_ε is stochastic and reversible for all ε,L is some matrix with $\|L\| = 1$ (we will use the Frobenius norm), and P_0 is a block matrix with m blocks, having the following form:

$$P_0 = \begin{pmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_m \end{pmatrix}. \tag{2.19}$$

We additionally require that L is a rate matrix: Let $e = (1\dots1)^T \in \mathbb{R}^n$. Then $Le = 0$, where

$$L = \begin{pmatrix} L_{1,1} & L_{1,2} & \cdots & L_{1,m} \\ L_{2,1} & L_{2,2} & \cdots & L_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ L_{m,1} & L_{m,2} & \cdots & L_{m,m} \end{pmatrix} \tag{2.20}$$

Then $L_{i,j}$ refers to block i,j of L.

Define the block vectors

$$\chi_l(i) = \begin{cases} 0 & \text{if } i \notin \text{ Block } l \\ 1 & \text{if } i \in \text{ Block } l, \end{cases} \tag{2.21}$$

and define $\hat{\mu_l}$ as the invariant measure of D_l: $\hat{\mu}_l^T D_l = \hat{\mu}_l^T$. Now $\hat{\mu}_l^T \chi_l = 1$.

Consider the right and left eigenvectors of P_0 corresponding to eigenvalue $\lambda = 1$. Since P_0 is a block matrix and each block is stochastic and primitive (this translates to a network which is connected), the right eigenvectors can be written as a linear combination of the block vectors:

$$u_0^i = \sum_j \alpha_j^i \chi_j, \text{ with } u_0^1 \equiv \vec{1} \Rightarrow \forall j : \alpha_j^1 = 1. \tag{2.22}$$

The right eigenvectors of P_0 are then constant on the blocks. In particular, $P_0\chi_k = \chi_k$.

The left eigenvectors can be written as a linear combination of the invariant measures on the blocks:

$$v_0^i = \sum_j \beta_j^i \hat{\mu}_j, \text{ with} \tag{2.23}$$

$$v_0^1 = \mu_0 = \sum_j \beta_j^1 \hat{\mu}_j \text{ where } \forall j : \beta_j^1 > 0. \tag{2.24}$$

Define also μ_ε, the invariant measure of P_ε: $\mu_\varepsilon^T P_\varepsilon = \mu_\varepsilon^T$.

We now look at the asymptotic expansions of the first m eigenvalues and eigenvectors. These are simply perturbations of the corresponding eigenvalues and eigenvectors of P_0.

$$\lambda_\varepsilon^i = 1 + \varepsilon\lambda_1^i + O(\varepsilon^2) \tag{2.25}$$
$$u_\varepsilon^i = u_0^i + \varepsilon u_1^i + O(\varepsilon^2) \tag{2.26}$$
$$\mu_\varepsilon = \mu_0 + \varepsilon\mu_1 + O(\varepsilon^2). \tag{2.27}$$

We can compute explicitly what those zero and first-order terms look like. We begin with the eigenvalue asymptotics up to the first order. The vector u_ε is an eigenvector, therefore $P_\varepsilon u_\varepsilon^i = \lambda_\varepsilon^i u_\varepsilon^i$. Replace each expression with its asymptotic expansion (2.25) to obtain:

$$(P_0 + \varepsilon L)(u_0^i + \varepsilon u_1^i + O(\varepsilon^2)) = (1 + \varepsilon\lambda_1^i + O(\varepsilon^2))(u_0^i + \varepsilon u_1^i + O(\varepsilon^2)). \tag{2.28}$$

From this we get

$$P_0 u_0^i + \varepsilon(Lu_0^i + P_0 u_1^i) + O(\varepsilon^2) = u_0^i + \varepsilon(\lambda_1^i u_0^i + u_1^i) + O(\varepsilon^2). \tag{2.29}$$

Since u_0^i is an eigenvector of P_0 with $\lambda = 1$, the term of leading order disappears. Discarding the second order terms, we obtain

$$Lu_0^i + P_0 u_1^i = \lambda_1^i u_0^i + u_1^i. \tag{2.30}$$

We can use this to get an explicit expression for u_1^i given u_0^i, λ_1^i with

$$(P_0 - Id)u_1^i = (\lambda_1^i - L)u_0^i. \tag{2.31}$$

Then

$$u_1^i = (P_0 - Id)^{-1}(\lambda_1^i - L)u_0^i. \tag{2.32}$$

We would like to find a basis of eigenvectors u_0^i of P_0 that is orthonormal with regards to μ_0:

$$\langle u_0^i, u_0^j\rangle_{\mu_0} = \delta_{ij} = \sum_{k,l} \alpha_k^i \alpha_l^j \langle \chi_k, \chi_l\rangle_{\mu_0}. \tag{2.33}$$

Since $\langle \chi_k, \chi_l\rangle_{\mu_0} = \delta_{kl}\beta_k^1 \hat{\mu_k} = \delta_{kl}\beta_k^1$ (as the sum of entries of $\hat{\mu_k} = 1$), we get

$$\delta_{ij} = \sum_k \alpha_k^i \alpha_k^j \beta_k^1. \tag{2.34}$$

From this we obtain:

$$1 = \sum_k (\alpha_k^i)^2 \beta_k^1 \tag{2.35}$$
$$0 = \sum_k \alpha_k^i \alpha_k^j \beta_k^1 \quad \text{if } i \neq j. \tag{2.36}$$

From this, since we defined $\alpha_k^1 = 1$, we get for $\forall j \neq 1$ that

$$0 = \sum_k \alpha_k^j \beta_k^1. \tag{2.37}$$

Multiplying (2.30) from the left by a u_0^j that fulfills the orthonormality conditions above gives

$$\langle u_0^j, L u_0^i \rangle_{\mu_0} + \langle u_0^j, P_0 u_1^i \rangle_{\mu_0} = \lambda_1^i \langle u_0^j, u_0^i \rangle_{\mu_0} + \langle u_0^j, u_1^i \rangle_{\mu_0}. \tag{2.38}$$

Since P_0 is reversible, we have, for the second expression:

$$\langle u_0^j, P_0 u_1^i \rangle_{\mu_0} = \langle P_0 u_0^j, u_1^i \rangle_{\mu_0} = \langle u_0^j, u_1^i \rangle_{\mu_0}, \tag{2.39}$$

with the last equality again due to the fact that u_0^j is an eigenvector of P_0. We have then:

$$\langle u_0^j, L u_0^i \rangle_{\mu_0} = \lambda_1^i \langle u_0^j, u_0^i \rangle_{\mu_0} = \lambda_1^i \delta_{ij} \tag{2.40}$$

$$\langle u_0^i, L u_0^i \rangle_{\mu_0} = \lambda_1^i. \tag{2.41}$$

We obtained an expression for λ_1^i. Substituting it in asymptotic expansion (2.25), for the case $i = j$ we have

$$\lambda_\varepsilon^i = 1 + \varepsilon \langle u_0^i, L u_0^i \rangle_{\mu_0} + O(\varepsilon^2). \tag{2.42}$$

In the case of $i = j = 1$ with $u_0^1 = e$ (that is, $L u_0^i = 0$), we further get $\lambda_\varepsilon^1 = 1$. Still from (2.30), for the case $i \neq j$, we get:

$$\langle u_0^j, L u_0^i \rangle_{\mu_0} = 0. \tag{2.43}$$

Taking the definition of u_0^i from (2.22) we have

$$\sum_{kl} \alpha_k^j \alpha_l^i \langle \chi_k, L \chi_l \rangle_{\mu_0} = 0. \tag{2.44}$$

$\langle \chi_k, L \chi_l \rangle_{\mu_0}$ is just the μ_0-weighted sum of the rows of the block $L_{k,l}$ of L. The vector μ_0 restricted on this block has entries $\beta_k^1 \hat{\mu_k}$. Therefore, we can write the product as

$$\langle \chi_k, L \chi_l \rangle_{\mu_0} = \hat{\mu}_k^T L_{k,l} \chi_l \cdot \beta_k^1, \tag{2.45}$$

We now define the matrix $\mathcal{L} \in \mathbb{R}^{m \times m}$, with the entries $\mathcal{L}_{k,l} = \hat{\mu_k}^T L_{k,l} \chi_l$. Then we can rewrite (2.45) as

$$\langle \chi_k, L \chi_l \rangle_{\mu_0} = \beta_k^1 \mathcal{L}_{k,l}. \tag{2.46}$$

We show that the eigenvalues and eigenvectors of $\mathcal{L}$ can give us the zero and first-order terms in (2.25). First we obtain β^1, the coefficients of μ_0. The vector β^1 is just the invariant measure of $\mathcal{L}$. To see this, we turn again to the asymptotic

expansions of the eigenvalues and eigenvectors of P_ε from (2.25) and look at the invariant measure asymptotics up to the first order. Comparing the invariant measure for $P_0 + \varepsilon L$ to that of P_ε, we have:

$$\mu_0^T P_0 + \varepsilon \mu_0^T L + \varepsilon \mu_1^T P_0 = \mu_0^T + \varepsilon \mu_1^T. \tag{2.47}$$

Since $\mu_0^T P_0 = \mu_0^T$, we have

$$\mu_0^T L + \mu_1^T P_0 = \mu_1^T. \tag{2.48}$$

Multiplying from the right by χ_l we get

$$\mu_0^T L \chi_l + \mu_1^T P_0 \chi_l = \mu_1^T \chi_l. \tag{2.49}$$

Since $P_0 \chi_l = \chi_l$, the terms containing μ_1^T cancel out to give $\mu_0^T L \chi_l = 0$. Substituting the definition of μ_0 (2.24) we get

$$\sum_k \beta_k^1 \hat{\mu}_k^T L \chi_l = 0. \tag{2.50}$$

Since $\hat{\mu_j}$ is nonzero only on the block j, ${\hat{\mu_j}}^T L \chi_l$ is $\mathcal{L}_{j,l}$. Therefore, we obtain:

$$\sum_k \beta_k^1 \mathcal{L}_{k,l} = 0 \quad \forall l \quad \Rightarrow {\beta^1}^T \mathcal{L} = \vec{0}. \tag{2.51}$$

And β^1 is the invariant measure of $\mathcal{L}$ as required. Then

$$\mu_\varepsilon = \sum_j \beta_j \hat{\mu}_j + O(\varepsilon). \tag{2.52}$$

We continue analyzing $\mathcal{L}$. Substituting (2.45) into (2.44) we further obtain, for $i \neq j$:

$$\sum_{kl} \alpha_k^j \alpha_l^i \beta_k^1 \mathcal{L}_{k,l} = 0. \tag{2.53}$$

If additionally $j = 1, j \neq i$, since $\alpha_k^1 = 1$:

$$\sum_{kl} \beta_k^1 \mathcal{L}_{k,l} \alpha_l^i = 0 \Rightarrow {\beta^1}^T \mathcal{L} \alpha^i = 0. \tag{2.54}$$

Similarly, for $i = 1, j \neq i$ we have $\sum_{kl} \alpha_k^j \beta_k^1 \mathcal{L}_{k,l} = 0$.

We can rewrite (2.53) using the scalar product on $\mathbb{R}^n$

$$\langle u, v \rangle_\beta = \sum_k \beta_k^1 u_k v_k \tag{2.55}$$

as

$$\langle \alpha^j, \mathcal{L} \alpha^i \rangle_\beta = 0 \qquad \forall i \neq j. \tag{2.56}$$

Therefore the α^is are orthogonal with respect to $\langle \cdot, \cdot \rangle_\beta$.

1. The eigenvectors of $\mathcal{L}$ diagonalize $\mathcal{L}$ with respect to $\langle\cdot,\cdot\rangle_\beta$.
2. These eigenvectors are exactly those α_i whose entries we can use as coefficients for u_0^i, which we can use to approximate u_ε.
3. If $\hat{\lambda}_1 \dots \hat{\lambda}_m$ are the eigenvalues of $\mathcal{L}$, they can be sorted as $0 = \hat{\lambda}_1 > \hat{\lambda}_2 \geqslant \dots$, and we can use them to approximate λ_ε.

The last statement is derived as follows: From (2.41) we get that

$$\lambda_1^i = \langle u_0^i, L u_0^i \rangle_{\mu_0}. \tag{2.57}$$

Substituting (2.22) we get:

$$\lambda_1^i = \sum_{jk} \langle \alpha_j^i \chi_j, L \alpha_k^i \chi_k \rangle_{\mu_0} = \sum_{jk} \sum_l \beta_l^1 \hat{\mu}_l \alpha_j^i \chi_j^T L \chi_k \alpha_k^i = \sum_{lk} \beta_l^1 \alpha_l^i \alpha_k^i \hat{\mu}_l^T L \chi_k. \tag{2.58}$$

Now, since $\hat{\mu}_l^T L \chi_k = \mathcal{L}_{l,k}$, the last expression is equal to $\langle \alpha^i, \mathcal{L}\alpha^j \rangle_{\beta^1}$. We know that the α^is are eigenvectors of $\mathcal{L}$, and they are orthonormal with respect to β^1. Therefore,

$$\langle \alpha^i, \mathcal{L}\alpha^j \rangle_{\beta^1} = \langle \alpha^i, \hat{\lambda}_i \alpha^j \rangle_{\beta^1} = \hat{\lambda}_i, \tag{2.59}$$

and we get $\lambda_1^i = \hat{\lambda}_i$. Therefore, we can write:

$$u_\varepsilon = \sum_j \alpha_j^i \chi_j + \varepsilon u_1 + O(\varepsilon) \tag{2.60}$$

and

$$\lambda_\varepsilon = \lambda_0^i + \varepsilon \hat{\lambda}_i + O(\varepsilon^2) \tag{2.61}$$

$$\lambda_\varepsilon = 1 + \varepsilon \hat{\lambda}_i + O(\varepsilon^2). \tag{2.62}$$

2.3.2 Computing the perturbed committor functions

The standard committor functions between two blocks of P_ε are defined in the same way as (2.14).

For arbitrary blocks 1 and m we want q_ε so that:

$$(\mathrm{Id} - P_\varepsilon) q_\varepsilon = 0 \qquad \text{on blocks } 2 \dots m-1 \equiv C, \tag{2.63}$$

$$q_\varepsilon = 0 \qquad \text{on block } 1, \tag{2.64}$$

$$q_\varepsilon = 1 \qquad \text{on block } m. \tag{2.65}$$

The asymptotic expansion of q_ε is

$$q_\varepsilon = q_0 + \varepsilon q_1 + O(\varepsilon^2). \tag{2.66}$$

We want to find q_0. To do this, we notice that on C:

$$(\mathrm{Id} - P_\varepsilon)q_\varepsilon = 0 \tag{2.67}$$
$$(\mathrm{Id} - P_\varepsilon)(q_0 + \varepsilon q_1 + O(\varepsilon^2)) = 0 \tag{2.68}$$
$$(\mathrm{Id} - P_0 - \varepsilon L)(q_0 + \varepsilon q_1 + O(\varepsilon^2)) = 0 \tag{2.69}$$
$$q_0 - P_0 q_0 + \varepsilon(q_1 - P_0 q_1 - L q_0) + O(\varepsilon^2) = 0. \tag{2.70}$$

For this to be true, we then need the following two conditions, on C:

$$q_0 - P_0 q_0 = 0 \tag{2.71}$$
$$q_1 - P_0 q_1 = L q_0. \tag{2.72}$$

From the first condition, we obtain that q_0 restricted on C is additionally an eigenvector of P_0, and is therefore a linear combination of the block functions. It has the solution

$$q_0 = \sum_j \gamma_j \chi_j \text{ with } \gamma_1 = 0, \gamma_m = 1, \tag{2.73}$$

where the remaining $\gamma_2, \ldots, \gamma_m$ are free.

From the second condition we get, by multiplying from the left with χ_k:

$$\langle \chi_k, q_1 \rangle_{\mu_0} - \langle \chi_k, P_0 q_1 \rangle_{\mu_0} = \langle \chi_k, L q_0 \rangle_{\mu_0} \quad \forall k = 2, \ldots, m-1. \tag{2.74}$$

Since P_0 is reversible and χ_k is an eigenvector of P_0 we get

$$\langle \chi_k, P_0 q_1 \rangle_{\mu_0} = \langle P_0 \chi_k, q_1 \rangle_{\mu_0} = \langle \chi_k, q_1 \rangle_{\mu_0}. \tag{2.75}$$

The left side of (2.74) vanishes, and we have

$$\langle \chi_k, L q_0 \rangle_{\mu_0} = 0 \quad \forall k = 2 \ldots m-1. \tag{2.76}$$

Using $q_0 = \sum_j \gamma_j \chi_j$ this yields

$$\sum_{j=2}^{m-1} \gamma_j \langle \chi_k, L\chi_j \rangle_{\mu_0} = 0 \quad \forall k = 2 \ldots m-1. \tag{2.77}$$

From this, in combination with the boundary conditions, we get:

$$\mathcal{L}\gamma = 0 \tag{2.78}$$
$$\gamma_1 = 0 \tag{2.79}$$
$$\gamma_m = 1 \tag{2.80}$$

This is a new committor equation whose solution are the coefficients γ, thus determining q_0 as wanted. We can use q_0 to compute q_1 via the second condition (2.72) as

$$q_1 = (\mathrm{Id} - P_0)^{-1} L q_0. \tag{2.81}$$

From this we get, finally, using (2.66):

$$q_\varepsilon = \sum_j \gamma_j \chi_j + \varepsilon q_1 + O(\varepsilon^2). \tag{2.82}$$

2.3.3 Computing the score

From the different components (2.61), (2.60), (2.52), and (2.82), we can compose the score. We defined the score as $\frac{1}{2}(1+\lambda_2\kappa_2)$. From (2.61), we get

$$\lambda_2 = 1 + \varepsilon\hat{\lambda}_2 + O(\varepsilon^2). \tag{2.83}$$

We now need to compute κ_2, according to

$$\kappa_2 = \|Q_\varepsilon u_\varepsilon\|_\mu^2 \tag{2.84}$$

Then we write

$$\kappa_2 = \|Q_\varepsilon u_0\|_{\mu_0}^2. \tag{2.85}$$

The asymptotic expansion is

$$\begin{aligned}\lambda_2\kappa_2 = \|Q_0 u_0\|_{\mu_0}^2 + \varepsilon\hat{\lambda}^2\|Q_0 u_0\|_{\mu_0}^2 \\ + \varepsilon\|Q_1 u_0\|_{\mu_0}^2 + \varepsilon\|Q_0 u_1\|_{\mu_0}^2 + O(\varepsilon^2).\end{aligned} \tag{2.86}$$

The matrix Q_ε is just the projection into the space spanned by the committors q_ε.

We have shown that all components of the I_m score are only slight perturbations, depending on ε, of the eigenvectors of P_0, which are combinations of the block functions. Since P_0 achieves the optimal score, the smaller the perturbation, the higher the score of the network represented by P_ε.

2.3.4 Constructing the perturbed network classes

We describe two classes of networks whose transition matrices can be written as $P_\varepsilon = P_0 + L$, with P_0 the block matrix and $L = \varepsilon\hat{L}$ a a rate matrix representing the perturbation. We would like these networks to be constructed in such a way that they have both modules and a transition region, and the differences between the densities of the modules and the transition region are encoded in L. This way, decreasing ε (while the norm of L remains constant) would result in increasing I_m. Therefore, we write an explicit formula for L given P_0, as $L = \varepsilon\hat{L}$ with a small ε. Our networks will have 3 blocks: $\mathcal{M} = M_1 \cup M_2$ are modules, each of size n_m, and $\mathcal{T}$ is the transition region of size n_t.

We now consider two variants:

1. Within each module, every node is adjacent to at least N other nodes. In the transition region, each node is adjacent to $c_1 N$ nodes for some $c_1 < 1$. This way the modules are more densely connected than the transition region.

2. Within each module, every node has *strength* (weighted degree) at least N. Thus, the networks in this case are weighted, and we would intuitively like to have modules that contain many strong nodes. Accordingly, in the transition region, each node has strength $c_1 N$ nodes for some $c_1 < 1$. This way the modules are stronger than the transition region.

To connect the modules with the transition region, we add new edges between them such that every node is now adjacent to c_2 new neighbors for some small constant c_2. To complete the construction of L, we need to select an appropriate ε. In case (1) we take $\varepsilon = 1/N$, and in case (2) we can then choose $\varepsilon = 1/\sqrt{N}$.

2.3.4.1 Case 1: Unweighted networks

We set $\varepsilon = \frac{1}{N}$. We need to show that the norm of $\hat{L}$ is constant when constructing the unweighted networks. The Frobenius norm of $\hat{L}$ is computed as follows: Let $\hat{L} = N \cdot L$. For every node in M_1, M_2, each row of L has at least N entries of the form $\frac{1}{N+c_2} - \frac{1}{N}$. There are $2n_m$ such rows.

Taking the square of the corresponding entries of $\hat{L}$ and summing all $2n_m \cdot N$ entries, we get

$$2n_m N \cdot \frac{c_2}{(N + c_2)^2}. \tag{2.87}$$

This expression goes to 0 as N increases. Additionally, to account for the new edges, those rows of L corresponding to module nodes have c_2 additional entries of the form $\frac{c_2}{N+c_2}$. Then the sum of these squared entries in $\hat{L}$ is

$$2n_m \cdot c_2 \frac{N^2}{(N + c_2)^2}, \tag{2.88}$$

which goes to $2n_m \cdot c_2$ as N increases.

For every node in the transition region, each row of L has at least $c_1 N$ entries of the form $\frac{1}{c_1 N+c_2} - \frac{1}{c_1 N}$. There are n_t such rows. Taking the square of the corresponding entries of $\hat{L}$ and summing all $n_t c_1 N$ entries, we get

$$n_t \cdot N \cdot \frac{c_2^2}{(c_1 N + c_2)^2} \to 0. \tag{2.89}$$

To account for the new edges, rows of the transition region have c_2 entries with $\frac{1}{c_1 N+c_2}$. Then the sum of these squared entries in $\hat{L}$ is

$$n_t \cdot c_2 \frac{N}{(c_1 N + c_2)^2} \to n_t \cdot c_2. \tag{2.90}$$

The norm is constant as required.

Example. We construct actual networks having the structure mentioned above, and compute the perturbed (Section 2.3.3) and actual ((2.17)) I_m score for them. We use the following parameters: the network has a total of 400 nodes. Of these, 200 are in the transition region, and we also have two modules with 100 nodes each. Initially, each node in the modules has some N neighbors inside the module, where N ranges from 4 to 50. This implies that the density of the

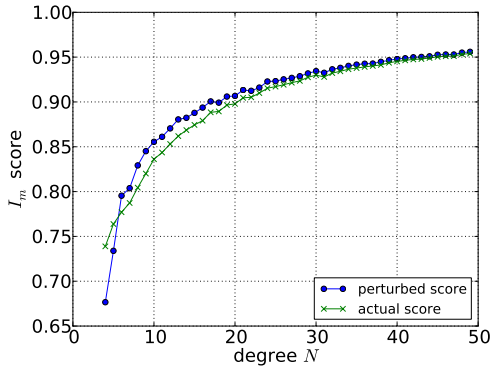


Figure 2.5: I_m for the unweighted network class: 400 nodes, 200 in transition region. Every node in the modules has initially N neighbors, while in the transition region every node has $\frac{1}{2}$N neighbors.

modules ranges from 0.04 to 0.5. We additionally set the number of neighbors each node in the transition region has initially to $\frac{1}{2}$N, that is, we set $c_1 = \frac{1}{2}$. Then the density inside the transition region ranges between 0.01 and 0.12, lower than that of the modules. Now we connect the modules and transition region such that each node has an additional $c_2 = 5$ neighbors.

We can compute the I_m for these networks in two ways: (1) exactly, and (2) using the components from the perturbation analysis. The perturbed and exact score are very similar, as demonstrated in Figure 2.5.

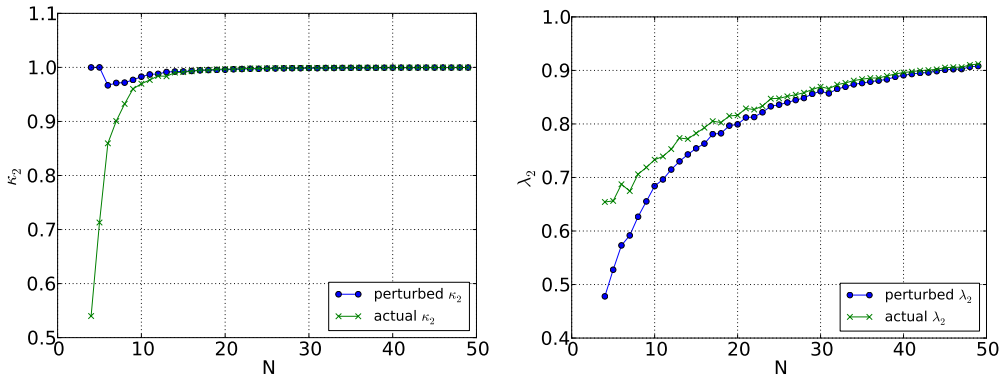
Figure 2.6 shows the score components. The perturbed κ_2 is still very close to 1 even when N is small, as opposed to the actual κ_2. However, the plots for λ_2 and its perturbed version match more closely, with the values of the perturbed λ_2 lower than those of the actual λ_2, thus compensating for the high perturbed κ_2 in the case of small N.

2.3.4.2 Case 2: Weighted networks

This network class is constructed along the same lines as the previous class, this time with variable node strength instead of number of neighbors. We then take $\varepsilon = \frac{1}{\sqrt{N}}$, where N is the strength of the nodes in the modules.

We show again that the norm of $\hat{L}$ is constant. The Frobenius norm of $\hat{L}$ is computed as follows: Let $\hat{L} = N \cdot L$. Let a^i_j be the weight of edge (i, j), and k_i the strength of node i. Then $\sum_j^{k_i} a^i_j = N$. For every node in M_1, M_2, row i of L has k_i entries of the form $\sqrt{N}(\frac{a^i_j}{N+c_2} - \frac{a^i_j}{N})$. There are $2n_m$ such rows.

Taking the square of the corresponding entries of $\hat{L}$ and summing all $2n_m \cdot N$



(a) κ_2 (b) λ_2

Figure 2.6: Comparing the components of the score for the class of unweighted networks with $\varepsilon = \frac{1}{N}$.

entries we get:

$$\sum_{i=1}^{2n_m} \sum_{j=1}^{k_i} \sqrt{N} \left(\frac{a_j^i}{N + c_2} - \frac{a_j^i}{N} \right). \tag{2.91}$$

The inner sum is equal to

$$\frac{c_2^2}{\sqrt{N}(N + c_2)^2} \sum_{j=1}^{k_i} (a_i^j)^2. \tag{2.92}$$

Now, since $\sum_{j=1}^{k_i} (a_i^j) = N$, we get $\sum_{j=1}^{k_i} (a_i^j)^2 \leqslant N^2$, so that (2.91) goes to 0.

Additionally, to account for the new edges, those rows of L corresponding to module nodes have c_2 additional entries of the form $\frac{\sqrt{N}}{N+c_2}$. Then the sum of these squared entries in $\hat{L}$ is

$$2n_m \cdot c_2 \frac{N}{(N + c_2)^2}, \tag{2.93}$$

which goes to 0.

A similar computation for the nodes in the transition region block gives similar results.

Example. We construct weighted networks with the following parameters: As before, the network has a total of 400 nodes, 200 are in the transition region, and there are two modules with 100 nodes each. Both the modules and the transition region are generated using an Erdős–Rényi model to have density 0.1. Initially, each node in the modules has strength N, distributed over its neighbors inside the module, where N ranges from 1 to 400. The strength of the transition

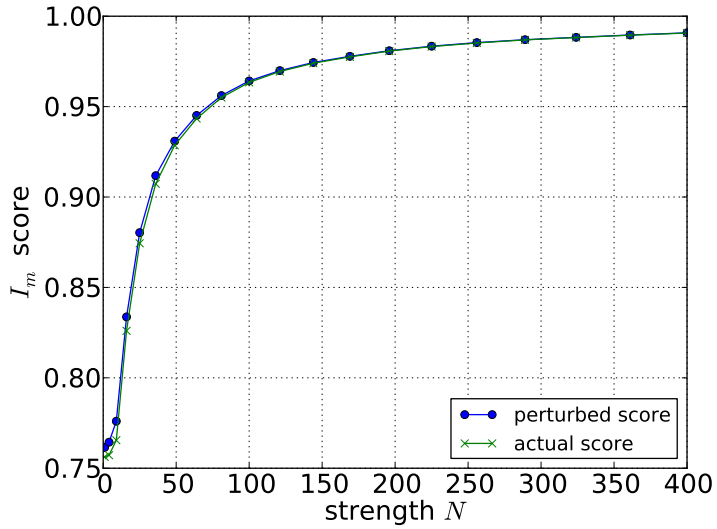


Figure 2.7: I_m for the weighted network class: 400 nodes, 200 in the transition region. Every node in the modules has initially strength N, while in the transition region every node has strength $\frac{1}{2}$N.

region nodes is initially $\frac{1}{2}$N, that is, we set $c_1 = \frac{1}{2}$. When connecting the modules and transition region each node acquires an additional $c_2 = 10$ neighbors. To distribute the strength N among the edges we iterate over the nodes, and if there is a node v where $s(v) <$ N is we distribute the difference N $-$ s uniformly to its adjacent edges.

We compute again I_m for these networks in two ways: (1) exactly, and (2) using the components from the perturbation analysis. See Figure 2.7 for the curves.

2.4 Experiments on simple network classes

After proving analytically that I_m behaves according to our intuition on a small class of networks, we now demonstrate experimentally that this is the case for more general classes of networks. We are chiefly interested in showing the following:

1. When the density of modules increases, so does the score.

2. When the proportion of modules to transition region in the network increases, so does the score.

3. The score should not depend on the number of modules in the network.

We will generate sets of benchmark networks for each case above, and test the performance of the metastability-based scores introduced in the previous sections: the R/*W*-score and I_m.

2.4.1 Experimental setting

To perform the experiments outlined above, we need to determine two aspects: First, the method of generating benchmark networks and the networks themselves must be described. Second, when designing the modularity scores, we have some freedom in choosing parameters, and those might alter the results. These parameters need to be explored.

The network construction is straightforward: We begin by generating an Erdős–Rényi (Erdős and Rényi, 1960) random graph, where each pair of nodes is connected with probability $p_t = 0.05$. This is our transition region. Then choose at random two nodes with the same degree, and remove them. In their place we insert two modules, as described next. In the first set of experiments these modules have increasing density p_m, and in the second set the modules increase in size. The modules are then inserted and connected to the rest of the network as follows: We add an edge between every node in the modules to every node in the transition region with probability p_t. If the resulting network is not connected, we simply restart the process.

We recap the modularity scores to be tested:

- The R/*W*-score is the ratio between the expected time to exit the transition region and to enter a different module than the one started in (Section 2.1.2).
- The I_m-score is the computed as $\frac{1}{2}(1 + \kappa_2\lambda_2)$, where κ_2 is μ-weighted norm of Qu_2, and u_2, λ_2 are the second eigenvector and eigenvalue of the matrix $P = \exp(Lt)$ for one of the generators specified next. The matrix Q is the projection matrix into the space spanned by the committors representing the fuzzy assignment.

When designing the score, recall that the metastability scores all rely on a choice of generator L. As we saw in Section 2.1.3, the choice of generator for the continuous random walk can have a substantial effect on the substructures considered metastable, and we can therefore expect it to influence the score. We therefore tested three generators, and compare them below.

1. L_a is the generator described by the formula (2.11). The waiting time $h(i)$ in each node is proportional to its degree, and the probability of jumping to a neighbor depends on the number of shared neighbors.
2. L_b is the generator where the probability $k(x, y)$ of jumping to a neighbor y from a node x is like in L_a, but the waiting time is now proportional to the clustering coefficient (equation Equation (1.1)) of the node rather than

its degree. Define $h(i) = 0.1 + c(i)$, where $c(i)$ is the clustering coefficient of node i, and the addition of 0.1 is done to ensure that $h(i) > 0$.

3. L_e is the *discrete generator* $P - Id$. In this case, when computing I_m, we take $P_t = P$.

Next, recall that two of the scores, R/W and I_m, require a fuzzy assignment as input. Since we construct our networks as combinations of modules and a transition region, we possess a *planted assignment*, which is intuitively very close to optimal. We will use this assignment as input. Lastly, I_m uses the eigenvectors and values of the matrix P_t. While the embedded Markov chain P depends only on the topology of the network, we still have to choose an appropriate time lag t. We do this by using the heuristic proposed by Sarich, Djurdjevac, et al. (2013) as follows: Let Γ_i be eigenvalues of the generator L. Suppose we can identify the spectral gap after k eigenvalues. Then a choice of t such that $\frac{1}{\Gamma_k} \leqslant t \leqslant \frac{1}{\Gamma_{k+1}}$ is appropriate.

2.4.2 Varying module density

In this experiment we vary the module density p_m from 1 (a clique) down to 0.03 (less than the average density of the network). The size of the transition region in the first stage is is 1000, each module contains 100 nodes. Then after replacing two nodes with modules, the final network contains 1198 nodes.

Figure 2.8 presents the I_m score for this set of networks in increasing module density, for the three generators. In all three cases I_m increases with the density of the modules, but it increases most gradually for L_a. In the case of the discrete generator L_e the score retains the minimal value 0.5 until the module density is more than 0.6, and only then increases.

It is also interesting to follow the different parts of the score as they increase with the module density (Figure 2.9). We see that κ_2 approaches the maximal value 1 very quickly, already with module density 0.4, while λ_2 rises more slowly. The eigenvalue is then more correlated with the density.

Figure 2.10 presents the R/W score for this set of networks in increasing module density, for the three generators. All three generators behave similarly when $p_m < 0.2$. At this point the score using the discrete generator increases slowly, while the scores based on the other two generators increase similarly, approaching the optimal score as the module density approaches 1.

2.4.3 Varying module size

This experiment is similar to the previous one, except that now the density of the modules remains constant (it is 1, they are cliques) but their size is varied from 10 to 400. The total number of nodes in the network remains constant. Thus, at a small size the modules comprise only a very small part of the network, and as

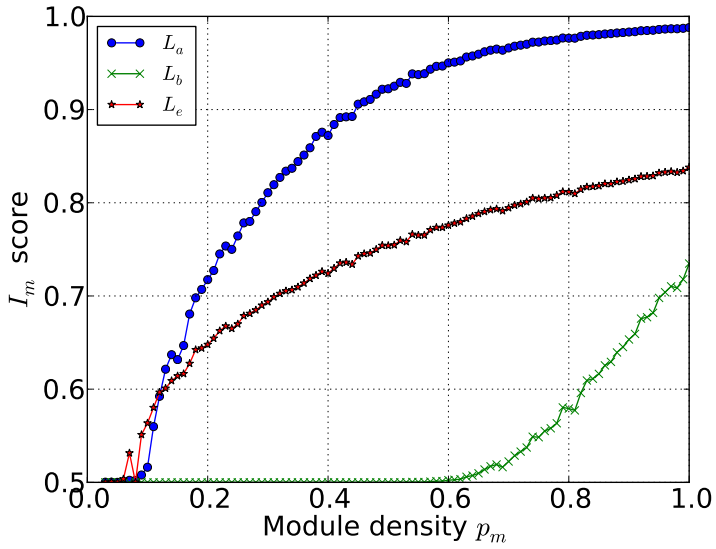


Figure 2.8: Comparing I_m for the three generators L_a, L_b, L_e on networks where the modules increase in density.

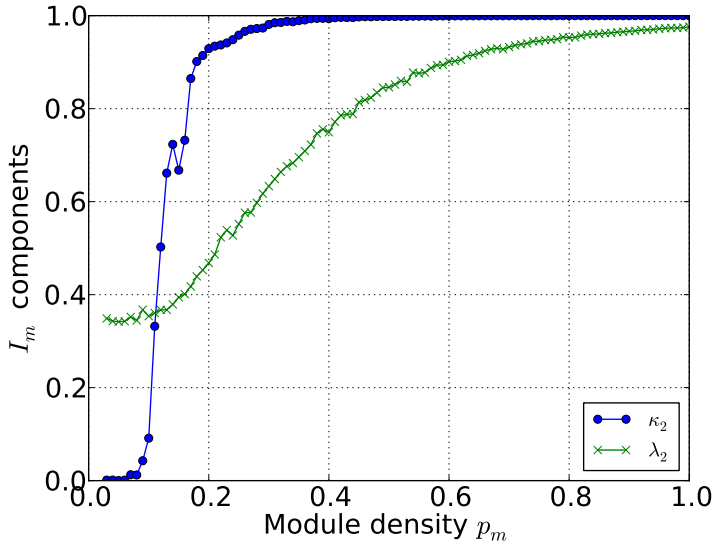


Figure 2.9: Comparing the components of I_m with generator L_a. Both κ_2 and λ_2 increase with density.

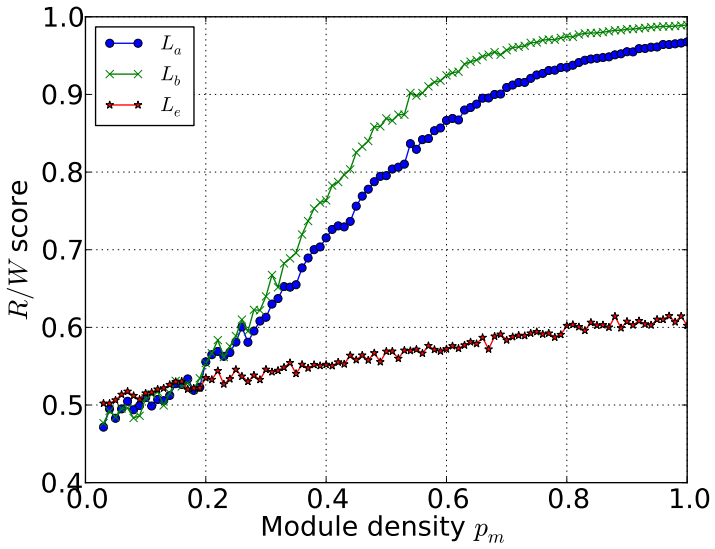


Figure 2.10: Comparing R/W for the three generators L_a, L_b, L_e on networks where the modules increase in density.

they approach 400 the total number of nodes in modules approaches 800, thus the transition region is relatively small at about 200 nodes. We would like our score to have the property that it increases as the modular region increases.

Figure 2.11 presents the I_m score for this set of networks in increasing module size, for the three generators. As the modules themselves are optimally dense (as cliques), I_m is high already at small sizes. From the three generators, only the score based on L_a discerns between networks with small modules and networks with larger modules, increasing with the module size. This is a clear advantage of this generator.

The results above become clear when breaking I_m based on L_a into its components (Figure 2.12). While $\kappa_2 = 1$ almost for every module size, λ_2 increases slowly, providing the variance in the final I_m score.

Figure 2.13 presents the R/W score for this set of networks in increasing module size, for the three generators. Here we see the same trend as in the density experiment, as all three scores are similar for very small modules, then L_a, L_b increase similarly while the score based on the discrete generator increases more slowly.

2.4.4 Varying number of modules

The modularity of a network should not depend on the number of modules, but rather on the relative size and densities of the transition region and modular region. We can test this by returning to the module density experiments, this time varying the number of modules from two to three. All the networks we test in

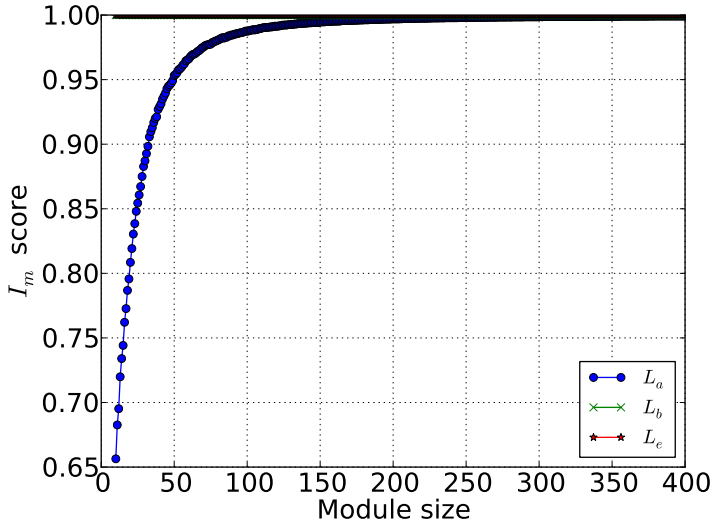


Figure 2.11: Comparing I_m for the three generators L_a, L_b, L_e on networks where the modules increase in size. While the scores based on L_b, L_e are constantly 1, the score based on L_a increases with the module size.

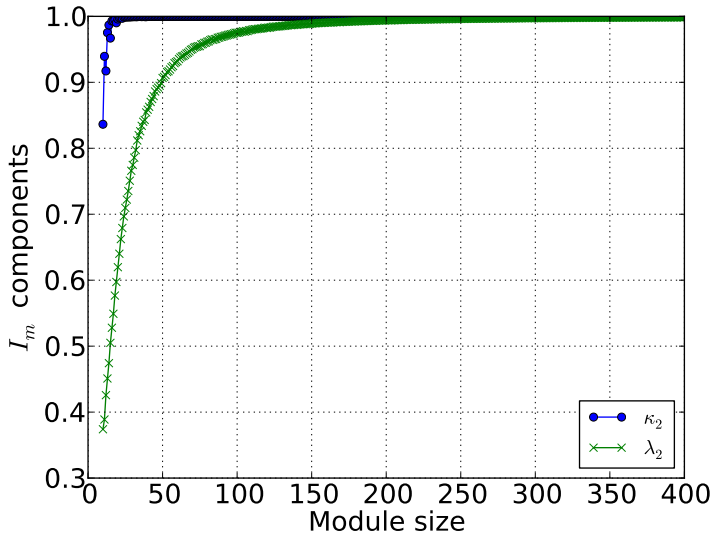


Figure 2.12: Comparing the components of I_m with generator L_a. Both κ_2 and λ_2 increase with size

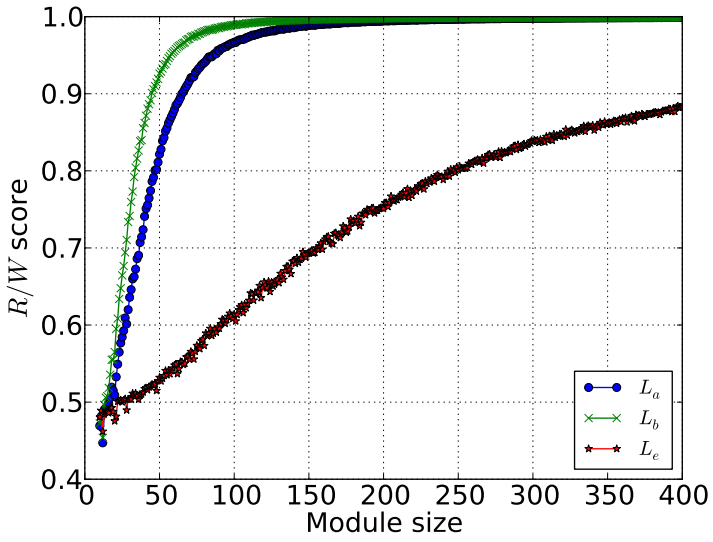


Figure 2.13: Comparing R/W for the three generators L_a, L_b, L_e on networks where the modules increase in size.

this section have an identical transition region, 1000 nodes with density $p_t = 0.05$. The modular region contains 300 nodes. These nodes are then partitioned into two or three modules, and we vary the number of edges within the modules to increase the module density until it reaches the required threshold. The modules then have different sizes (100 or 150), but the size of the modular region does not change.

We present in Figure 2.14 the results of the I_m score and the R/W score on each pair of networks, using the generator (2.11) (results for other generators are similar). The I_m score is the same for each pair of networks with the same density, thus fulfilling our requirement. The R/W score, on the other hand, is higher for two modules than for three, consistent with our observation in Section 2.1.3. These findings imply that by restricting the score to use only the second eigenvalue and eigenvector, we can compare networks with simple structures and not be influenced by the number of modules.

Having demonstrated the good properties of the I_m score on benchmark networks, where the correct assignment is known in advance, in the next section we show an example of analyzing real-world networks with this approach.

2.5 Application: Modularity of brain networks

We describe an example for a possible application of our modularity score: Determining the modularity of brain networks (connectomes), with the larger goal of using the modularity to help differentiate between typical and atypical

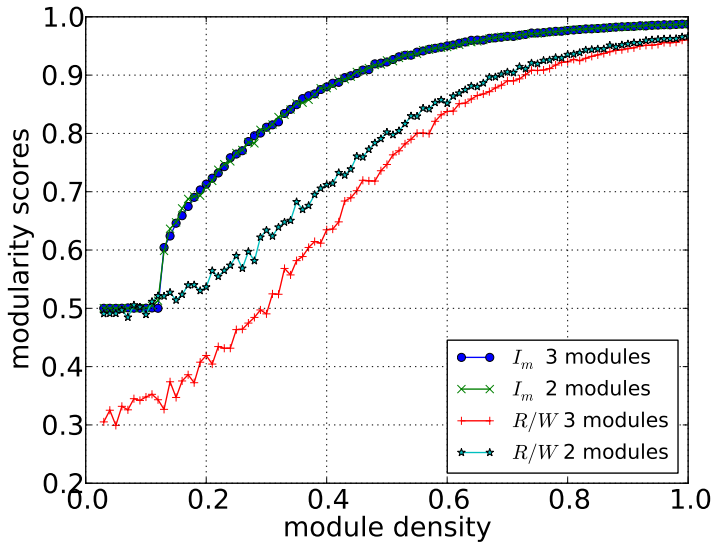


Figure 2.14: Comparing scores between networks with two or three modules. The transition region is identical for all networks, and for each density there is a corresponding pair of networks, one with two modules and the other with three. The $\mathrm{I_m}$ scores are identical, while the R/W scores differ.

brains according to some criteria. As surveyed in Section 1.1, such an approach is well-motivated: in recent years, the modular structure of brain networks has often been investigated in the context of disorders such as epilepsy (Chavez et al., 2010), Alzheimer's diseases (G. Chen et al., 2013), and autism (Rudie et al., 2012). When comparing groups of brain networks under different conditions (sick vs. healthy, before vs. after learning a task), the groups often exhibit different modularity. This is usually described qualitatively, by noting cluster shapes, or quantitatively: the networks in each group have a different number of clusters, larger or smaller clusters, etc. Such measurements can then be used to better understand the mechanism of the brain, or even as features in classifiers that can label an unknown network (Gamboa et al., 2013).

Due to the increased availability of brain network data for download and research, we are able to test the performance of our $\mathrm{I_m}$ score on real connectomes. We downloaded the UCLA autism dataset[1], first collected and analyzed by Rudie et al. (2012). This dataset contains contains 175 networks from children and teenagers, two from each of 51 children diagnosed with Autism Spectrum Disorder (ASD), and 43 typically-developed (TD) children.

The dataset contains two types of connectomes: *functional* and *structural*. The functional networks are derived from resting state functional magnetic resonance imaging (fMRI) experiments. All networks share the same node set, where in this

[1] http://umcd.humanconnectomeproject.org/umcd/default/index

case the authors chose a whole-brain parcellation scheme (Power et al., 2011) with 264 functional brain regions, resulting in 264 nodes. The fMRI measurements of every subject, after processing, lead to a time series associated with each node, and edge weights are then transformed z-scores of the correlation coefficient between the time series. Unfortunately, this implies that these networks contain negative edge weights, which makes our I_m score (along with other scores surveyed here) not applicable.

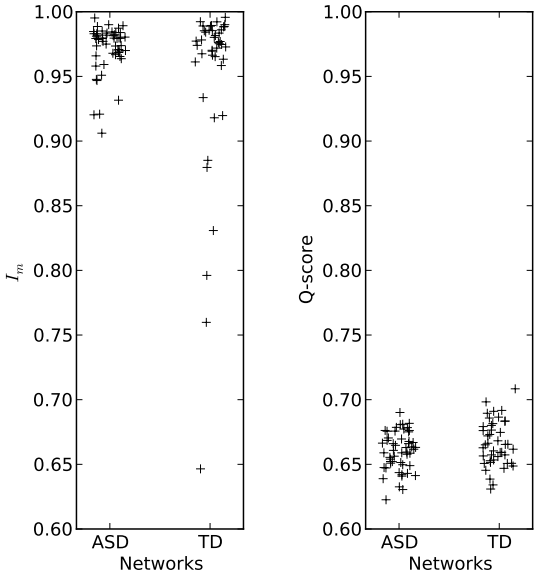
We are left, then, with the structural networks: Nodes correspond to the same 264 functional regions, and edges correspond to the number of fibers connecting the regions, measured using an MRI method called Diffusion MRI (dMRI). This implies integer weights on the edges, within the range $(1, 500)$. We focus, therefore, on 43 TD and 51 ASD structural networks. The question we would like to answer is, do ASD and TD networks have different I_m scores? If so, we can use this fact to gain both insight about the brain structure in autism and a feature for classification.

The networks in the experiments described below display the following behavior: All ASD networks obtain a high I_m score, while the distribution of I_m for the TD networks is more varied, with some networks obtaining low scores. Thus, given a structural network we have otherwise no information about, if it has a low I_m score we can assume that it comes from a typically-developed individual.

We first outline the results of the analysis done by Rudie et al. (2012). We follow this with the results of the Newman–Girvan Q-score (see Section 1.3.1.1) on this set of networks, and then our own I_m score. To fully describe I_m, we need to specify how we obtain an assignment to use in computing κ_2. We can then break down the score to show which components influence the variance in the TD network scores.

Rudie et al. (2012) analyzed both the functional and structural networks. Their analysis consisted of several parts: They fully partition the network and count the edges within and between clusters. Next they calculate several global measures on the average TD and ASD networks when thresholding the edge weights across several thresholds. These measures include the average clustering coefficient, characteristic path length, and four others (Rubinov and Sporns, 2010). While for the functional network some of these measures, such as the clustering coefficient, were clearly different between the TD and ASD group, this was not the case for the structural networks. For these networks no single measure was significantly different for both groups. Finally, the authors discuss the connection between the results on the functional and structural networks, along with the connection between some of the network measures and characteristics such as age of the child and severity of autism indicators. The overall results agree with previous work, which links the modularity of functional networks with neurological characteristics.

Rudie et al. (2012) interpreted the networks using several well-established



(a) I_m (b) Q

Figure 2.15: Modularity scores I_m and Q for ASD versus TD networks. The I_m score of the ASD networks is always high, while the TD is more spread out, with some low-scoring networks.

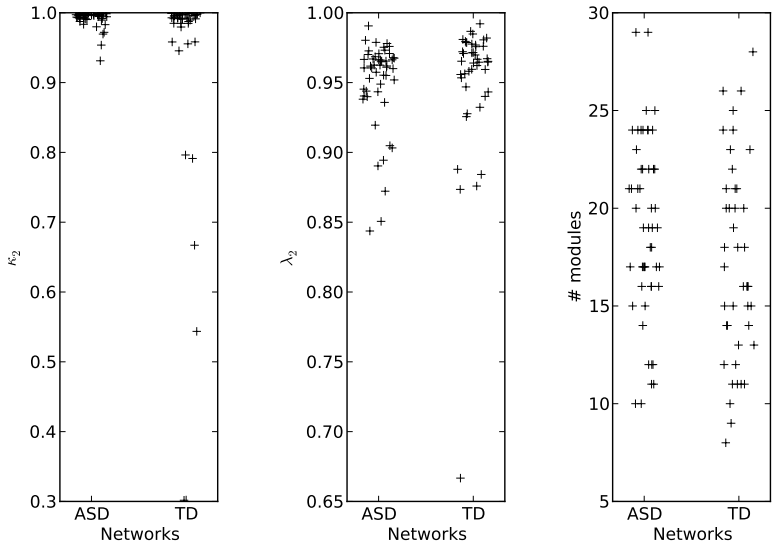
measures, but their interpretation of the network modularity is restricted: They first find a full partition with a high Q-score using the Louvain heuristic (Blondel et al., 2008), then count the edges inside and outside the clusters of this partition to conclude that the TD networks contain more dense, sparsely-connected clusters, that is, TD networks are more modular. With our extended notion of modularity, allowing for the possibility of assignments rather than full partitions, we can be more flexible. Another advantage of our approach is that it takes into account all edges of the network, including weights. In contrast, Rudie et al. (2012) create unweighted networks by discarding some percent of edges in each network having lowest weights and assigning weight 1 to the remaining edges, thus perhaps losing information. The Q-score, on the other hand, is designed to take edge weights into account, and summarize the modularity of a network in a single number, as we aspire to do. Unfortunately, the Q-score of the TD and ASD networks does not appear to differ (Figure 2.15). The scores vary between 0.6 and 0.7 in both cases, suggesting that the networks are indeed modular in some sense, but not differentiating between the two network types.

Can we do differently with I_m? In order to test this, we need to compute the components of the score (2.17). We begin with computing a fuzzy assignment, represented by a set of committors (2.14). For this we utilize the MSM clustering algorithm (Sarich, Djurdjevac, et al., 2013), fully described in Chapter 3. Briefly,

the algorithm consists of three steps: Separating the transition region from the modular region, fully-partitioning the modular region, and finally, assigning some transition region nodes to the modules via thresholding. We adopt only the first two steps, thus dropping the need to find an optimal thresholding parameter. Thus, the entire process is parameter-free:

1. Compute the generator L similarly to (2.11), replacing the degree $d(x)$ of every node with its strength $s(x)$ to account for the edge weights.
2. Determine an appropriate lag-time α from the spectral gap, as described by Sarich, Djurdjevac, et al. (2013).
3. Identify the transition region and compute the transition matrix P_α of the modular region.
4. Estimate the number of modules via the largest gap of $\hat{P_\alpha}$ and fully partition the modular region to obtain modules.
5. Compute the committor functions for these modules.
6. Compute from L the matrix P_t for the time lag α.
7. Calculate the score components: u_2, μ, Q, and then κ_2.
8. Compute the score as $\frac{1}{2}(1 + \lambda_2\kappa_2)$.

Running this algorithm on the TD and ASD networks, we obtain the results depicted in Figure 2.15. While all ASD scores are high (above 0.9), TD scores can also be lower, with 6 out of the 42 under 0.9 and of those, 3 less than 0.8. What is the source of these differences? Breaking the score down into its components (Figure 2.16), we find that, in contrast to the experiments on benchmark networks (Section 3.2.3), λ_2 is consistently high, while the fluctuations in κ_2 cause the fluctuations in the final score. Thus, the eigenvector error representing the difference between the committors of the assignment resulting from the clustering and the eigenvectors of P_t is responsible for the lower scores: If we work under the (not necessarily true) assumption that the clustering algorithm finds the optimal assignment and thus the κ_2 is optimal, then we must conclude that these TD networks with low scores simply do not have a good assignment into dense modules and a transition region. By plotting the number of modules found for every network, we further confirm, in agreement with our previous experiments on benchmark networks, that the fluctuations in the score are not related to the number of modules. No correlation was found also when comparing the size of modular and transition region: The modular transition region in the low-scoring network is not smaller than the average transition region in the remaining network. Thus, the transition region size does not explain the lower scores, either.



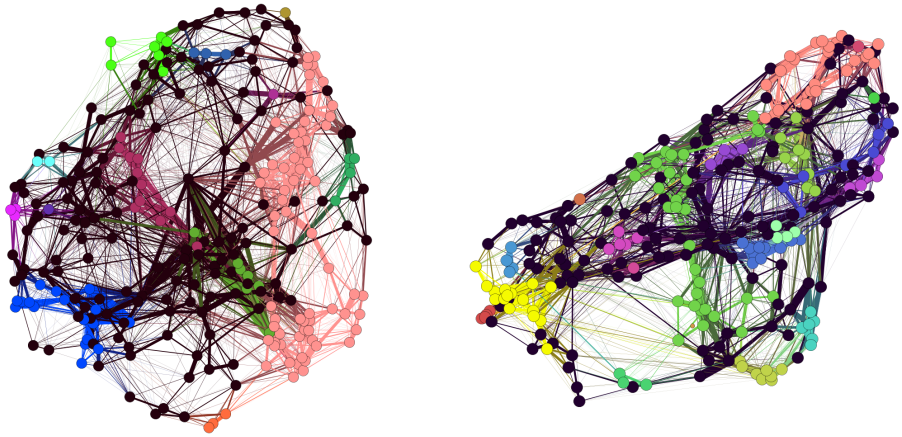
(a) κ_2 (b) λ_2 (c) number of modules

Figure 2.16: Analyzing the different factors contributing to the I_m score: The score components κ_2 and λ_2, and the number of modules determined by MSM clustering. The variance of I_m for these types of networks appears to stem from the variance in κ_2.

The difference between the modules detected in the high-scoring ASD networks and low-scoring TD networks is not easy to visualize. We plotted a pair of such networks, along with the modules of the corresponding assignments found by the MSM. clustering (Figure 2.17). No clear difference is discernible. The modules we find correspond to adjacent regions of the brain, but that is to be expected from the construction of the network. Further study is required to derive meaning from the particular modules we found.

The last test consists of computing the correlation between I_m and the age of the participants in the study. One major finding of Rudie et al. (2012) is correlation between Q and participants age: In the TD group, modularity sharply decreased with age, while in the ASD group it decreased more slowly. We noticed no such trend when comparing participant age and I_m score, aside from noticing a negative Pearson correlation between age and I_m in the TD group (Pearson correlation -0.3) and no correlation in the ASD group.

The major finding of our experiments is, therefore, that the I_m score for ASD networks is always high, and for TD it can also be low. What can we conclude from this? Mainly, how can we reconcile this with previous results, which point to reduced modularity in ASD brains? The first explanation is a technical one: As demonstrated above, previous attempts at quantifying modularity of brain networks must disregard some information (edge weights, in this case). The



(a) Network from TD brain, $I_m = 0.7961$, 11 modules

(b) Network from ASD brain, $I_m = 0.9952$, 24 modules

Figure 2.17: The assignment found using MSM clustering for one TD and one ASD brain networks. Thickness of edges is proportional to weight.

particular preprocessing chosen can influence results. For example, when we tried thresholding and binarizing the networks, the I_m scores for the TD and ASD groups were not differently distributed. Therefore, it can be valuable to use all available information.

The second explanation goes to the heart of the model: full partition versus assignment. In the full partition case, the partitions have been shown to correspond to functional or structural brain regions. Then for ASD brains, the partitions have fewer edges within them and more edges between them than the TD brains. Thus, the modularity of the TD brains is naturally higher. However, it is possible that a full partition is not the only useful way to model the brain. Rudie et al. (2012) suggest, for example, when discussing the functional networks, that the decreased modularity of the ASD brains is partially caused by some brain regions (default and sensorimotor systems) containing nodes that are connected to many other nodes in different clusters. A model that assigns such nodes to the transition region rather than to a module might better represent the underlying mechanism, and the modularity score should reflect this. Overall, it would be interesting to further study the non-full clustering of the brain, and the insights it could provide.

Chapter 3

Module and transition region identification

Previous chapters introduced some modularity scores and demonstrated their behavior and usefulness. We focused on the metastability-based I_m score, which assumes as input an optimal fuzzy assignment, defined via the committors. We will now discuss methods of finding such a fuzzy assignment. While finding the optimal assignment is difficult (Djurdjevac, Bruckner, et al., 2011; Sarich, Djurdjevac, et al., 2013), we can use heuristics to identify an assignment with good properties.

We then start by briefly describing the MSM clustering algorithm (Sarich, Djurdjevac, et al., 2013), designed specifically for fast identification of modules and a transition region. Since the problem of computing an assignment is interesting by itself (in the same way that finding a good full partition of a network is interesting), we additionally propose several algorithms that compute assignments that are good according to various criteria. We begin by describing simple modifications to several well-known full partition algorithms, so that they would output assignments. We summarize this section with our experiments on benchmark networks (Bruckner, Kayser, and Conrad, 2013).

The significant theoretical contribution of this chapter is in Section 3.3, where we develop a combinatorial formulation of an assignment, based on graph-theoretical concepts. Since we show that this formulation is NP-hard to optimize, we propose several heuristics and an integer linear program that optimally solves the problem. In the last section, we present experiments on protein–protein interaction networks, where we identify and analyze modules using the algorithms above.

3.1 Markov State Model-based clustering

The MSM clustering algorithm (Sarich, Djurdjevac, et al., 2013) is designed specifically to find modules and the transition region in modular networks.

3.1.1 Identification of modules

When constructing the I_m modularity score in Section 2.2, we scored a network according to the optimal fuzzy assignment. A good assignment is defined as one where the eigenvector projection error of the coarse-grained random walk is small. More precisely, we can suppose that we do not have a full partition but just the modules C_i, $i = 1, \dots, k$, with the remaining nodes belonging to the transition region $\mathcal{T}$, and k committor functions f_i with $\sum_i f_i(x) = 1$ for all nodes x. We find a coarse-grained random walk that jumps between the modules and takes the dynamics on the transition region into account (Sarich and Schütte, 2011). The process has an $k \times k$ projected transition matrix $\hat{P} = QPQ$, where Q is again the orthogonal projection onto the space spanned by the committor functions, as in the I_m score we introduced.

Sarich and Schütte (2011) show that for any eigenvalue λ_i of P_t and the corresponding normalized eigenvector u_i it holds that

$$\delta_i \leqslant p(u_i) + 2\mu(\mathcal{T}) p_{\max}(u_i) + r(\mathcal{T})(1-\lambda_i) \left(\sum_{x \in \mathcal{T}} u_i(x)^2 \mu(x) \right)^{1/2} \tag{3.1}$$

with

$$\begin{aligned} r(\mathcal{T}) &= \sup_{\substack{\|v\|=1 \\ v=0 \text{ on } \mathcal{M}}} \left(\frac{1}{\sum_{x \in T} (v(x) - (Pv)(x))^2 \mu(x)} \right)^{1/2} \\ p(u_i) &= \|e_i\|, \quad p_{\max}(u_i) = \|e_i\|_\infty \\ e_i(x) &= \begin{cases} 0 & \text{if } x \in T, \\ \frac{1}{\mu(C_j)} \sum_{y \in C_j} u_i(x) - u_i(y)\mu(y) & \text{if } x \in C_j. \end{cases} \end{aligned} \tag{3.2}$$

From this, we conclude that modules should satisfy two things in order to ensure the small projection errors $\|Q^\perp u_i\| = 1 - \|Q u_i\|$ for the dominant eigenvectors, hence providing a good approximation of the largest eigenvalues. Recall that these eigenvalues correspond to metastability of the random walk, similarly to the motivation behind I_m. First, from the transition region the random walker should always enter some module quickly enough such that $r(\mathcal{T})(1-\lambda_i)$ is small enough. This corresponds to a fast return time R (Section 2.1.2). The more eigenvalues of P we want to approximate, the faster $\mathcal{T}$ has to be left. Second, the dominant eigenvectors should be close to step functions, that is, almost constant on the modules to guarantee small values of $p(u_i)$ and $p_{\max}(u_i)$. Fortunately, the

error bound decomposes into these two parts. The factor $r(\mathcal{T})(1-\lambda_i)$ takes only the transition region into account, and the errors $p(u_i)$ and $p_{max}(u_i)$ depend only on the partitioning of $\mathcal{M} = V \setminus \mathcal{T}$ into the modules.

We can exploit this when constructing methods for finding fuzzy assignments. Strictly from an algorithmic perspective, the task divides into two parts: (1) identifying the transition region, and (2) clustering the remaining nodes into modules. Viewed this way, we can in principle solve each of these subtasks independently.

3.1.2 Algorithm

Sarich, Djurdjevac, et al. (2013) compute the modules and the fuzzy decomposition in three separate steps:

1. Identify the transition region $\mathcal{T}$.
2. Cluster the remaining nodes of the network into modules $C_1, \ldots, C_k$.
3. Compute the fuzzy decomposition as the committors with respect to the modules.

Identifying the transition region first reduces the problem to that of finding a full partition. If the transition region is large, the number of nodes in modules is comparably small, and the running time of the full-partition clustering step would be reduced. That is, we would have a full partition with respect to the nodes belonging to $\mathcal{M}$ only, which would be simpler since nodes that can be affiliated with several modules, and therefore pose difficulties for the algorithm, are removed.

Step 1. We want to choose a transition region such that the random walker leaves it quickly. If we also want to consider modules that are less metastable, we will have to approximate eigenvalues which are less close to 1: the factor $(1-\lambda_i)$ in (3.1) is larger, $r(\mathcal{T})$ must be smaller, and therefore, the region T has to be left faster. If, on the other hand, we are only interested in the very metastable modules, only the eigenvalues close to 1 need to be approximated. This can be controlled by a choice of the lag time parameter α. The higher we choose the value for α, the more metastable the modules we find. Suppose we can place random walkers distributed according to some stationary distribution μ^* (uniformly or according to some other scheme (Sarich, Djurdjevac, et al., 2013)) on the nodes of the network. Now we propagate the random walk with the generator (2.11) and compare the distribution of walkers on the nodes to μ^*:

$$\mathcal{M}^\alpha = \{x \in V \mid (P_\alpha^T \mu^*)(x) > \mu^*(x)\} \tag{3.3}$$

We choose nodes $x \in V$ for the modular region for which there are, informally, more random walkers after time $\alpha > 0$ than there were according to the original

distribution. Thus, this set (3.3) is exactly the region that rather attracted random walkers in the ensemble than let them leave within the time α.

Step 2. Having identified the set $\mathcal{M}^\alpha$, we have effectively reduced the problem to that of finding a full partition for those nodes. There are two possible strategies: First, we can remove the transition region entirely, and cluster the remaining nodes without taking into account the dynamics on the transition region. We can use any full-partition algorithm for this, such as MCL (Dongen, 2000). Alternatively, we can view the time lag α as a *resolution parameter*, allowing us to define a metastable hierarchy of modules: On the lowest level of the hierarchy, we have the most metastable modules, those corresponding to the longest lag times. The higher levels of the hierarchy contain modules that are less and less metastable. Viewed this way, a particular choice of $\mathcal{M}^\alpha$ gives a single level of the hierarchy, but by including information about the transition region dynamics in the input provided to the full-partition clusterer we can influence the full partition. For example, a set of nodes $U \subseteq V$ can be split by the clusterer into two separate modules U_1, U_2, or be merged into a single module if there are multiple transition region nodes that have neighbors in both U_1 and U_2. Which strategy to choose depends on the application. We will see examples for the effects of both choices in Section 3.1.3.

For the second strategy, Sarich, Djurdjevac, et al. (2013) consider the random walk only on the nodes belonging to $\mathcal{M}^\alpha$ with transition matrix

$$\hat{P}_\alpha(x, y) = \sum_{z \in V} P(x, z) q_y(z), \quad x, y \in \mathcal{M}^\alpha, \tag{3.4}$$

where $q_y(z)$ is the probability that y will be the next node from $\mathcal{M}^\alpha$ that is hit by the random walk starting in z. This can be computed via the committors (2.14) by treating each node as a module and computing the probability of hitting this module. That is, $\hat{P}_\alpha(x, y)$ describes the transition probabilities between the nodes of $\mathcal{M}^\alpha$. From there it is possible to use a full-partition algorithm to split $\mathcal{M}$ into modules $C_1, \ldots, C_m$. Since $\hat{P}_\alpha$ describes the dynamics only within the modular region, the absence of nodes with affiliation to several modules makes the spectrum of $\hat{P}_\alpha$ usually very amenable for interpretation, as exemplified when we compute the eigenvalues of $\hat{P}_\alpha$ on the adverse network (Figure 1.8). The result is shown in Figure 3.1, where the spectrum of $\hat{P}_\alpha$ (red crosses) is compared with that of the discrete random walk P. There is a clear gap after two eigenvalues, corresponding to the two modules we want to identify, but only after removing the transition region.

Step 3. We can of course stop here: We have modules and the transition region, that is, we have an assignment, and for some applications this would be enough. However, for other purposes, in particular for computing the I_m score, we need to compute the affiliation of all nodes to the modules: a fuzzy assignment. The

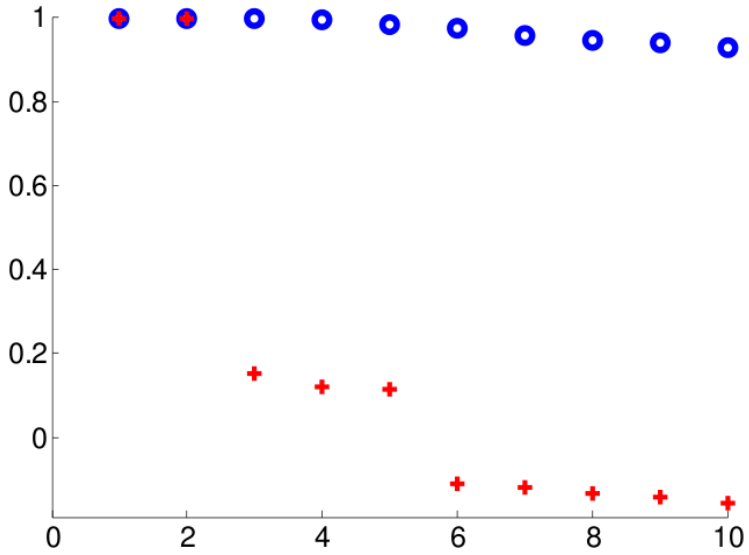


Figure 3.1: The eigenvalues of $\hat{P}_\alpha$ (red crosses) versus those of the standard transition matrix P (blue circles) (Sarich, Djurdjevac, et al., 2013) for the adverse network in Figure 1.8.

affiliation of the nodes in $\mathcal{M}^\alpha$ is clear, but we still need to compute the affiliation of the transition region nodes to the modules.

Thus, this step consists of computing the committors with respect to the modules $C_1, \ldots, C_k$ to get a fuzzy clustering of the remaining nodes. As mentioned above, committors can be computed by solving positive definite, symmetric linear systems which will be as sparse as the adjacency matrix. Such computations can be performed efficiently, even for large systems.

Performing this step has the additional advantage of allowing extra flexibility in determining which nodes belong to the transition region and which belong to the modules: After obtaining the affiliation of each transition node to each module, we can choose to assign transition region nodes to modules to which they are strongly affiliated. This can be done, for example, with thresholding: every node that has an affiliation higher than θ for some chosen parameter θ to some module C_j can be viewed as belonging to C_j. Sarich, Djurdjevac, et al. (2013) summarize the algorithm as follows:

Algorithm summary.
1. Input: $\alpha > 0$, matrix A
Compute L according to (2.11), with e. g. $p = 1$.
Solve

$$\frac{d}{dt}\nu_t = L^T\nu_t, \quad \nu_0 = \mu^* \tag{3.5}$$

until $t = \alpha$, so $\nu_\alpha = e^{L^T\alpha}\mu^*$. Set

$$\mathcal{M}^\alpha = \{x \in V \mid \nu_\alpha(x) > \mu^*(x)\}. \tag{3.6}$$

2. Compute committors $q_y(z), y \in \mathcal{M}^\alpha, z \in T$ with respect to the single nodes of $\mathcal{M}^\alpha$ and for $x, y \in \mathcal{M}^\alpha$

$$\hat{P}_\alpha(x, y) = \sum_{z \in V} P(x, z) q_y(z). \tag{3.7}$$

Choose number of modules n according to the spectrum of $\hat{P}_\alpha$ and use hard clustering method, e. g. (Deuflhard et al., 2000).

3. Compute committors $f_i(x)$ for every $x \in T$ with respect to the modules $C_1, \ldots, C_k$.

Running time analysis. Let the number of nodes in the network be n, and the number of nodes in the modular region $\mathcal{M}^\alpha$ be denoted by N_m. The running time of the algorithm described above can be decomposed as follows (Sarich, Djurdjevac, et al., 2013):

Step 1: This requires solving a system of linear differential equations. Al-Mohy and Higham (2011) show that the computational effort is dominated by matrix multiplications, dominated by $O(n)$ for large, sparse matrices.

Step 2: This has two parts: computing $\hat{P_\alpha}$ and performing full-partition clustering. Thus first we have to solve a symmetric, positive definite linear system for N_m right-hand sides. Since the matrix L is large and sparse, conjugate gradient methods allow to compute the solution in $O(kn)$ point operations. Then, we have to compute a full partition with respect to the coarse grained random walk with $N_m \times N_m$ transition matrix $\hat{P}_\alpha$. Many algorithms exist for this, for example Deuflhard et al. (2000) and E, T. Li, and Vanden-Eijnden (2008), performing in running time $O(N_m^2 \log N_m)$.

Step 3: Again, we have to solve linear systems in order to compute committors as in step 2 with the same matrix L.

Thus the overall running time is dominated by Step 2, where we have to fully partition the N_m nodes in the modular region. If $N_m \ll n$, that is, if the number of nodes in modules is much smaller than the number of nodes not assigned to modules, then the running time scales linearly with the total number of nodes. To summarize, the different stages of the algorithm mostly require solving linear systems of equations, a task whose running time depends on the method used.

The running time then depends on the size of the network and the number of nodes in the modular region.

3.1.3 Drawbacks

The algorithm described above performs well on some benchmark networks, as we will demonstrate in Section 3.2.3. However, several issues must be noted when applying the algorithm to general networks, as exemplified in the network described below.

We construct a network with 2 modules and a transition region. Each module is a random graph on 50 nodes, with edge density 0.6. Hence, the average degree inside a module is $0.6 \cdot 49 \approx 30$. The transition region is a random graph on 1000 nodes, with a much lower density of 0.05, and thus an average degree of about 50. We then connect every node in a module with every node in the transition region with probability 0.05. Thus the average degree of a module node in the connected network is 80, and the average degree of a transition region node is about 52. This structure is depicted in Figure 3.2. This network is still modular, and the degree of module nodes is higher than that of transition region nodes on average, but the degree inside the modules is lower than the degree in the transition region. With this network we can demonstrate two issues with the MSM clustering algorithm:

1. A high degree is the best indicator of module membership, as modeled by the waiting time of the continuous Markov process being proportional to the degree.

2. In Step 2 of the algorithm, the modular region is fully partitioned but the jump probabilities between the nodes also take the transition region into account.

We first run the MSM clustering algorithm exactly as described in the previous section. From the spectrum of the generator, with a clear gap after the first three eigenvalues, we determine a lag time of $\alpha = 138$ and separate the modular and transition region. The modular region (Figure 3.2, colored) is clearly misidentified, the nodes corresponding to some high-degree nodes in the network. We can attribute this to the generator (2.11), which rewards high-degree nodes with a relatively high waiting time. This is demonstrated by running the algorithm with a different generator L_b, first defined in Section 2.4. Recall that the generator is defined through the transition probabilities $k(x, y)$ and the waiting time, denoted $h(i)$. Keeping $k(x, y)$ the same, we replaced $h(i) = d(i)$ in the degree-based generator by $h(i) = 0.1 + c(i)$, where $c(i)$ is the clustering coefficient of node i. The spectrum of L_b suggests a small time lag of $\alpha = 3$. When separating the transition region and modular region, we find that the MT-partition is much closer to the original network construction, with only a few exceptions (Figure 3.2). This points to a possible problem with the MSM

clustering algorithm: when the structure of the network is complicated, with nodes of various degrees inside and outside of modules, the simple reliance on degree can lead to incorrect results. This could be a problem when analyzing real-world networks.

Clustering this network also emphasizes a second issue: We see that even when the modular region is almost correctly identified, as when we use the clustering-coefficient-based generator L_b, and even as the correct number of cores (two) can be determined from the spectrum of $\hat{P_\alpha}$, the clustering heuristic fails to partition the modular region correctly. In this case, one module is comprised of a single node, and the remaining nodes are clustered together. This can either be attributed to the deterministic full-partition algorithm itself, or to its input: The $\hat{P_\alpha}$ matrix, giving the probability of transitioning between each pair of nodes in the modular region, is computed via the committors. Thus a pair of nodes that is not directly connected but has many short paths between them through the transition region would have a high probability in $\hat{P_\alpha}$, and thus could be clustered together. When we instead take the induced graph on the modular region nodes and run a standard full-partitioning algorithm like Newman–Girvan (Section 1.3.1.1), the partition into two clusters matches the planted partition. Whether we want to maintain the transition region information or not will depend on the application.

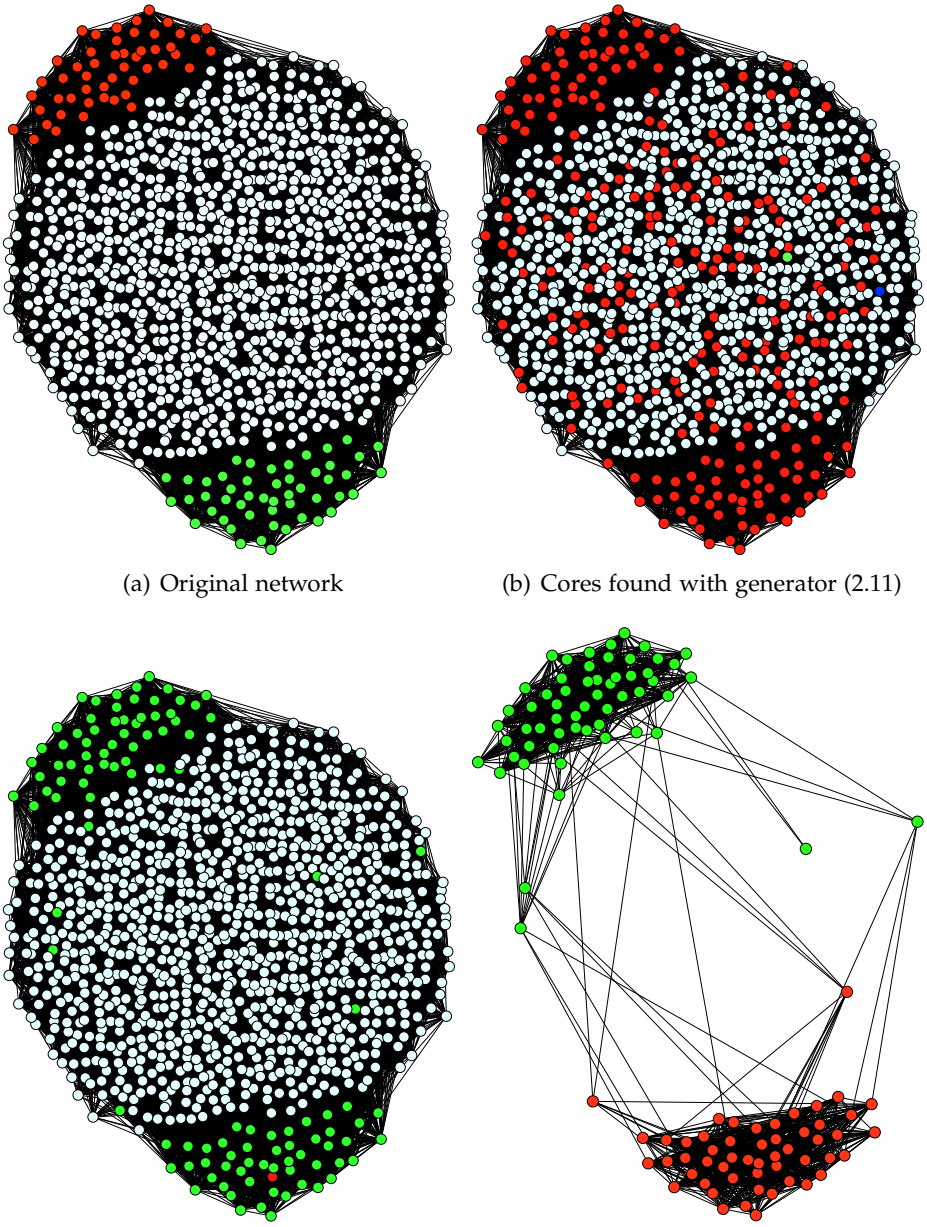
We conclude that while the MSM clustering algorithm can resolve networks with a straightforward structure with modules of similar density that is different to that of the transition region, further adjustments must be made to handle real-world networks. A different generator and a different approach to the full-clustering step could help in resolving such networks.

3.2 Modifications to prominent full-partition algorithms

In the previous section, we dealt directly with the question of how to identify the best fuzzy assignment in the sense of metastability. However, the question of finding a good assignment has a wider scope. As detailed in the first chapter, clustering algorithms are usually synonymous with full-partition algorithms. It is useful then to consider algorithms for identifying assignments, since there is no clear state-of-the-art algorithm for this task. A natural place to start is taking a closer look at some of the more prominent clustering algorithms and testing whether they can be extended to identify fuzzy assignments.

3.2.1 Simple adjustments to MCL and SCAN

There is a rich offering of clustering algorithms for different classes of networks (many surveys are available, for example Nascimento and Carvalho (2011), Porter, Onnela, and Mucha (2009), and Santo (2010)). To our knowledge, the majority of algorithms exist for the task of fully partitioning a network: Every node must



(a) Original network

(b) Cores found with generator (2.11)

(c) Cores found with L_b

(d) Clusters found by optimizing Q

Figure 3.2: The original network has 1000 nodes in the transition region (white), 50 in each module (red and green). The modular region identified by MSM clustering in Step 1 contains many transition region nodes, and the subsequent full partition is also unintuitive. The modular region identified when using L_b is much closer to the planted modular region, but the full partition into two cores is again non-intuitive. When fully partitioning the modular region using Newman–Girvan modularity, the resulting partitions match the original network.

belong to exactly one module. Another, smaller, category of algorithms that has received attention recently contains algorithms that allow nodes to belong to more than one module, that is, allowing overlaps (what we termed *overlapping partitions*). In this category we also find a subset of algorithms that permits nodes to be labeled as *outliers*, nodes that belong to no module.

We wish to test how we can modify algorithms from these categories to find assignments. Therefore, we now select representatives from the literature for each category, and explain how we interpret their output in the sense of our model.

3.2.1.1 SCAN

The SCAN algorithm (Xu et al., 2007) is, to our knowledge, one of the first algorithms to handle networks that are not fully partitionable, using the notion of *hubs*: nodes that bridge many clusters and therefore can be affiliated with more than one cluster, and *outliers*: nodes that cannot be said to belong to any cluster. Under this model, SCAN identifies clusters, hubs, and outliers in large networks, where the network is comprised mostly of modules, while hubs and outliers are more rare.

The algorithm defines and utilizes *structural similarity*: Two nodes are assigned to a cluster according to how they share neighbors. If their neighborhoods are structurally similar, they will be in the same cluster. If they are not similar enough to any other node they will be labeled as outliers, or hubs if they are structurally similar to nodes that end up in separate clusters. The algorithm then employs *structural reachability*: A node is structurally-reachable from another if there is a chain of nodes between them such that each node is structurally-reachable from the previous node. During the run of the algorithm, clusters are expanded from single-node seeds to contain nodes that are structurally reachable from one another. Nodes that do not fit in any cluster are then classified into outliers or hubs according to the number of clusters they are adjacent to. SCAN requires two user-defined parameters, μ and ε, where μ is the minimum size of a module, and ε controls the structural reachability. SCAN has been successfully applied to very large graphs with millions of nodes. The source code is available by request from the authors.

The two special classes of nodes defined by SCAN, *hubs* and *outliers*, partly correspond to our notion of transition region nodes: Outliers are not a part of any single module. Hubs bridge between different modules and therefore do not strictly belong to any. Therefore we count outliers and hubs found by SCAN as transition nodes, and assign them to the transition region.

3.2.1.2 Markov clustering

The MCL (Markov Clustering) algorithm (Enright, Dongen, and Ouzounis, 2002) is a popular algorithm for fully partitioning networks that is based on simulation

of random walks. Its central idea is that a random walk on a network would remain "stuck" in dense regions for a long time, and will therefore will go infrequently from one natural cluster to another.

MCL simulates random walks on networks by alternating two operators on the transition matrix: *expansion* and *inflation*. In the Expansion step the algorithm computes the power of the matrix using the normal matrix product (matrix squaring). The Inflation step corresponds with taking the Hadamard power of a matrix (taking powers entrywise), followed by a scaling step, such that the resulting matrix is stochastic again, and the matrix elements (on each column) correspond to probability values. As the process converges the matrix exhibits a structure close to a block structure, with high probabilities between nodes belonging to the same clusters and close to zero everywhere else. The original code by the authors, written in C++, is available from micans.org/mcl. MCL is designed to return a full partition of the network, and therefore must be adjusted to be able to take outliers into account. It has been demonstrated (for example by Satuluri, Parthasarathy, and Ucar (2010)) that MCL tends to produce imbalanced clusterings, consisting of a few large clusters and many small clusters of size two or three and singletons. Usually viewed as a shortcoming of the algorithm, we now interpret this tendency to our advantage: We introduce a parameter μ similar to SCAN to set a minimum size for a module. All modules with less than μ nodes are assigned to the transition region.

Using these modifications to SCAN and MCL we essentially formulated two algorithms for finding assignments. We are then equipped with three algorithms whose performance we can test and compare: modified SCAN, modified MCL, and the MSM clustering algorithm. We can test their performance on benchmark networks: We construct a simple class of networks with a planted assignment that we will assume is an optimal assignment. We can then compare this optimal assignment with the assignment determined by the three algorithms. We next describe how to compare two assignments to evaluate algorithm performance, and then outline our class of benchmark networks.

3.2.2 Evaluating the similarity of two assignments

We propose methods to score the performance of the clustering algorithms described above on our benchmark networks. Continuing along the line we began when describing the MSM clustering algorithm, we evaluate the algorithms on the two parts of the assignment task: (1) Correctly assigning nodes to the modular region or to the transition region (MT-partition), and (2) Correctly assigning the modular region nodes to the different modules (full partition). Then we combine the two scores to obtain a single evaluation.

At the basis of our scoring scheme is a well-known evaluation measure for clustering methods. The adjusted Rand index (ARI (Rand, 1971)) is a measure of agreement between two clustering partitions that can be used as a metric for

evaluating clustering methods (Santos and Embrechts, 2009). Compared to the original Rand index, the adjusted version corrects for just-by-chance bias.

ARI is based on a matrix N, where the rows correspond to one clustering A and the columns to the other clustering B. The number of nodes in cluster i from partition A that belong to cluster j of partition B is $N(i,j) = n_{ij}$. The number of clusters of some partition P is c_P. The sum over row i (that is, the number of nodes in cluster i from partition A) is $n_{i.}$, sum over column j (number of nodes in cluster j from partition B) is $n_{.j}$. The ARI is then computed as:

$$\rho = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}/\binom{n}{2}}{\frac{1}{2}(\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}) - \sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}/\binom{n}{2}}. \tag{3.8}$$

This formula can only be applied to compare full partitions of a node set, without taking the transition region into account. To compare two fuzzy assignments on the tasks outlined above, we construct the appropriate full partitions.

For testing how well the algorithm classifies the nodes into the modular and transition region we apply the ARI formula to an appropriate MT-partition. That is, we score the similarity between the modular regions and the transition regions in both assignments. This is the ρ^{MT} score.

For testing how well the algorithm splits the modular region into modules we apply the ARI formula to the assignments of a smaller subset of the nodes: Only those that are assigned to the modular region in *both* the ground-truth (planted) assignment and the one output by the algorithm. This is the ρ^{M} score.

The combined ARI score. In order to get a single cohesive score that combines the performance of the algorithm on the two tasks defined above we introduce the ρ^c score. The main challenge in designing ρ^c is to find a way to account for the transition region. Two extreme approaches are possible: First, we can merge all transition region nodes into a single new module. Alternatively, we could assign each transition node to a single new module. Both approaches have the same drawback: for networks with a large ground-truth transition region, any algorithm that assigns almost all or all of the nodes to the transition region will attain a high score. This is because a a large proportion of the assignments in the calculated clustering and the ground-truth agree. To circumvent this we modify the assignments as follows: Denote the two assignments A and B. Construct A' and B' as follows: Every node assigned to the transition region in assignment A and to a module in assignment B becomes a singleton module in A' and vice versa for B'. Every node assigned to the transition region in both assignments is removed in both. Finally, every node assigned to a module in both clusterings retains its module number.

Define the following subsets of nodes:

- $\mathcal{M}_A$: Modular region of the planted assignment A.

- $\mathcal{T}_A$: Transition region of the planted assignment A.
- $\mathcal{M}_B$: Modular region of the algorithm result B.
- $\mathcal{T}_B$: Transition region of the algorithm result B.

For each node v:

1. $v \in \mathcal{M}_B \cup \mathcal{T}_A$. Then node v belongs to a module in B but to the transition region in A. Assign v to a new singleton module in A.
2. $v \in \mathcal{T}_B \cup \mathcal{M}_A$. This is the symmetric situation. Node v in the transition region of A but assigned to a module in B Assign v to a new singleton module in B.
3. $v \in \mathcal{M}_B \cup \mathcal{M}_A$. Here v belongs to a module in both assignments. Do the same in A', B'.
4. $v \in \mathcal{T}_B \cup \mathcal{T}_A$. In this case, v is assigned to the transition region in both assignments. Node v is then not a part of A' or B' and thus not considered in the evaluation.

This leads to two "full partition" assignments of a subset of the nodes which can be compared using the standard ARI formula. Since the modified assignments contain only nodes that are assigned to one module in at least one of the assignments, the ARI is not biased by the size of the transition region of the original network. We can thus evaluate the similarity of two assignments, both in the modular and the transition region, using this ρ_c score.

3.2.3 Experiments on benchmark networks

Before presenting results of the modified algorithms, we first describe how we construct a simple class of benchmark networks.

The networks are constructed as follows: each of the n nodes is first assigned to either exactly one of the modules, or to the transition region. This is the planted partition, the "ground truth". To create the networks, we generate the transition region and modules separately as random Erdős–Rényi (Erdős and Rényi, 1960) graphs. To ensure the connectivity of each ER graph we add a random spanning tree between its nodes. The next step is to connect the parts into a network. We select a node at random from the transition region and from each module and connect these nodes via a random spanning tree. Finally, each possible edge between a module node and a transition region node is added with some small probability.

More formally, this class has the following parameters: *Network size* (total number of nodes) N, *number of modules* M, *total number of nodes in modules* N_M, *module density* p_m, *transition region density* p_t and *inter-connection density* p_i. The

inter-connection density is an indicator for the number of edges between modules and the transition region. We also define the number of nodes in the transition region $N_T := N - N_M$ and the number of nodes per module $N_m := N_M/M$. The module size is then determined by the other parameters.

For the experiments we describe next we use the following set of parameters: $N = 1000$ nodes, $M = 5$ modules, $p_m = 0.6$ module density, $p_t = 0.01$ transition region density, and $p_i = 0.01$ interconnection density. The minimal module size under these constraints is then 10 nodes.

3.2.3.1 Algorithm parameters

SCAN, MCL, and MSM all require specifying values for input parameters. When choosing the parameters, we attempted to stay as close as possible to the default values stated in the documentation of the algorithms. In some cases, when a range of values was possible, we ran the experiments with different values and selected the value for which ρ^c was highest.

In SCAN the parameters are ε and μ. We take $\mu = 10$ as discussed above. To choose ε, a parameter controlling the distance between nodes in a module, we followed the guidelines set in the SCAN documentation and tried $\varepsilon = 0.5 \ldots 0.8$. We then set $\varepsilon = 0.5$, as for this value we obtained the best results for all experiments.

MCL has many tunable parameters. We opt for the default values as described by Enright, Dongen, and Ouzounis (2002). We experimentally determined that altering those parameters does not improve the results. We add the post-processing parameter $\mu = 10$ as a minimum module size as described in Section 3.2.1.2. The members of any module with less than ten nodes are then assigned to the transition region.

In MSM the parameters are α, which determines how metastable the detected modules must be, and the threshold θ for adding transition region nodes to existing modules according to their committor values. Sarich, Djurdjevac, et al. (2013) set the parameters of MSM to $\theta = 0.9$ in all experiments, and we duplicate this here. The case of α is more complicated, since it is treated as a resolution parameter (Section 3.1.2): higher values of α result in only the most metastable modules detected, while lower values allow for less metastable structures to also be detected as modules. Our experiments showed that the exact value of α does not make a difference to the results on our benchmark set, probably due to the simple nature of those networks: all the modules have the same size and density. We therefore set $\alpha = 1000$, as in the examples of Sarich, Djurdjevac, et al. (2013). The generator we choose is the degree-based one (2.11) as used by Sarich, Djurdjevac, et al. (2013), which prefers assigning high-degree nodes into modules.

3.2.3.2 Experiment results

Our first set of experiments concerns varying the proportions between the sizes of the different parts of the network: The relative size of the transition region, and the size and number of the modules in the modular region. While the particular structure of the modular region appears not to affect the score of the clustering algorithms' output, the relative sizes of the modular and transition region influence the scores dramatically: The larger the transition region, the worse the performance of the algorithms.

Experiment 1: Varying transition region size. In this experiment, our goal is to evaluate the behavior of the different algorithms on networks where the transition region comprises between 0% and 90% of the network. In the case of 0% transition region, the modular region occupies the entire network, and the problem will again be that of full partitioning. We hypothesize that the algorithms should perform better on networks with a small transition region, as they are closer to the full partition case: SCAN looks for hubs and outliers but those are usually single nodes, not entire regions; MCL was originally designed for full partitions. Since MSM does not make assumptions about the size of the transition region, it is possible that this algorithm performs the same on the networks regardless of the transition region size.

Indeed, our experiments show that for a transition region covering 80% or less of the network, all algorithms perform optimally ($\rho^c = 1$ for MSM) or close to optimally ($\rho^c > 0.92$). For larger transition regions, all algorithms perform progressively worse.

Figure 3.3 shows the performance of the algorithms, giving the ρ^c score averaged over 5 networks for each transition region size. SCAN and MCL both identify only two or three modules, assigning the rest to the transition region. SCAN additionally identifies no hubs or outliers, thus the transition region is a result of small clusters, just as in the case of MCL. MSM separates the modular and transition region well ($\rho^{MT} > 0.95$), identifies five modules in the modular region, but partitions it less than optimally (average $\rho^M = 0.77$). MSM begins to deteriorate a little later than the others, at 89%, but the score decreases fast, with a score of 0 (all nodes are identified as transition region nodes) from 92%. Therefore, MSM is clearly the choice in case the transition region is large, but not too large.

Experiment 2: Varying module size. As we increase the size of the transition region in Experiment 1, the size of a module decreases automatically, since fewer nodes are now divided into a constant number of 5 modules. Specifically, for a transition region which covers 80% of the network, the corresponding module size is 40, and for 90% it is already 20. To test whether the difference in scores in Experiment 1 is a result of varying the transition region size or of varying the

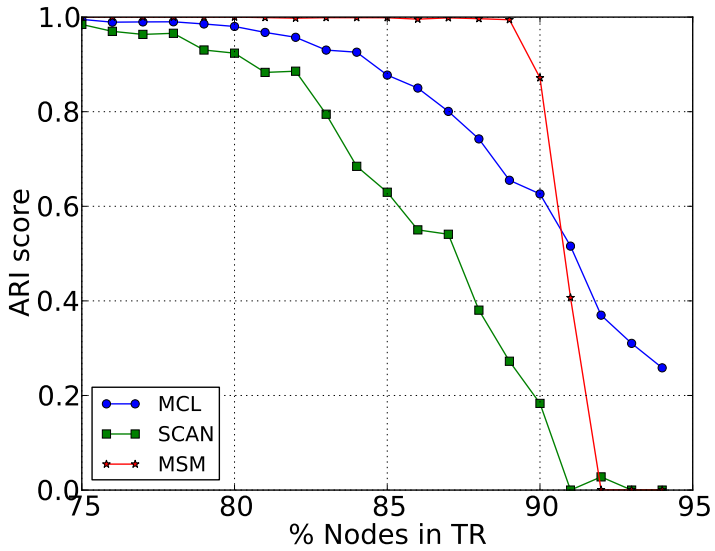


Figure 3.3: Comparing the ρ^c score for SCAN, MCL, and MSM on networks with varying transition region size. All networks were generated with 1000 nodes and 5 modules, having the default densities.

module size, we run a set of experiment where we directly vary the module size. The module size is between 20 and 200, there are 5 modules as before, and the transition region comprises 50% of the network, a value for which all algorithms in Experiment 1 performed perfectly ($\rho^c = 1$). Naturally, to preserve the same proportion of transition region to modular region while varying the total size of the modular region, the overall network size has to change as well, varying from 200 to 2000, respectively. All 3 algorithms performed perfectly for all module sizes: All three ρ scores were 1 or > 0.99. We additionally tested module size < 20, but the results were unstable due to the small difference between the true and the minimal module size: in some cases the modules were detected correctly, in others, only 9 nodes from a 10-node module were detected, and were assigned to the transition region, causing low scores. Therefore, while a very small module size can negatively influence the algorithm, this effect disappears for slightly larger module sizes, and the low scores of Experiment 1 cannot be fully attributed to the module size, but must be due to the proportion of transition region nodes.

In the next set of experiments we keep the proportions between the network components constant but vary their densities.

Experiment 3: Varying module and transition region densities. In this experiment, we test combinations of the module density p_m and the transition region density p_t. We take as before networks with 1000 nodes and 5 modules, with the

transition region comprising 50% of the network. We additionally set $p_i = 0.01$. For these parameters and the default densities $p_t = 0.01, p_m = 0.6$ all three algorithms performed optimally in the previous experiments.

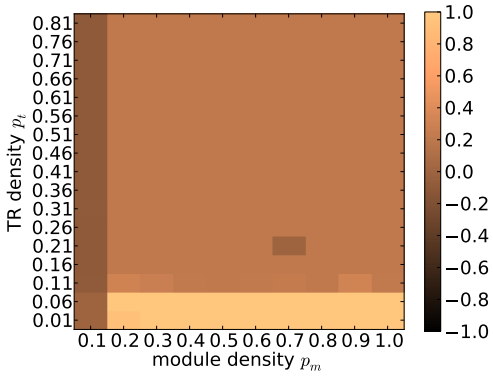
We set $p_m = 0.1, 0.2, \ldots, 0.9, 1$, and $p_t = 0.01, 0.06, 0.11, \ldots, 0.81$. Figure 3.4 shows a heatmap for each of the algorithms, giving the ρ^c score for each combination of transition region density and module density. Intuitively, we expect the algorithms to do well when the module density is high and the transition density is low. Indeed, we see that this is the case for all algorithms. SCAN performs the best, erring only when $p_t > 0.45$. The other two algorithms perform optimally when $p_t < 0.06$ and $p_m = 0.8$, and performance quickly deteriorates. Looking more closely at the ρ^{MT} and ρ^M scores, we see that the ρ^M score is perfect while ρ^{MT} is low: the entire transition region is detected as a single module in all these cases.

The poor performance of MSM could perhaps be attributed to the fact that the algorithm tends to reward (with a high waiting time) those nodes that have a relatively high degree (see Section 3.1.3). Those nodes end up being assigned to modules more often. As the density of the transition region increases, so does the average degree. Since we have fixed p_i at 0.01, and as the modules are smaller than the transition region (each module has size 100, compared to 500 for the transition region), the average degree of nodes in the module is also bounded, and for some values of p_m and p_t, the degrees are about the same, and thus MSM cannot tell them apart as well.

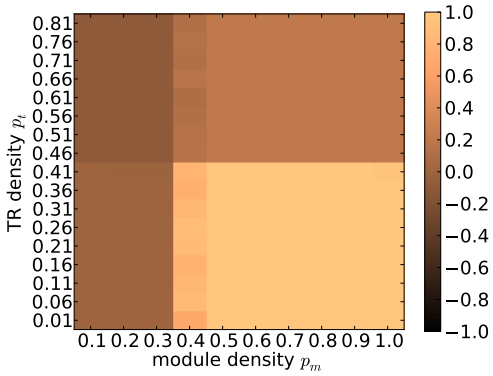
Discussion. We must conclude that no algorithm comes out the clear leader in every case. MSM identifies modules even when the transition region is large, but does not perform so well when the average degree in the transition region is high. While SCAN performs better than the other algorithms whenever the densities of the transition region and modules are close, in many cases it too identifies the transition region as a module.

With regards to the different steps of module identification, we first note that MSM performs best the task of guessing the correct number of modules. SCAN and MCL both under-estimate the module number, identifying modules that are too small and are therefore assigned to the transition region. No algorithm over-estimated the number of modules throughout our experiments. On the task of separating the transition region and the modular region (assessed with the ρ^{MT} measure), the three algorithms had successes and shortcomings: In Experiments 1 and 2 the errors were a result of nodes from the modular region being assigned to the transition region. In Experiment 3, the error resulted from the transition region being identified as a single module.

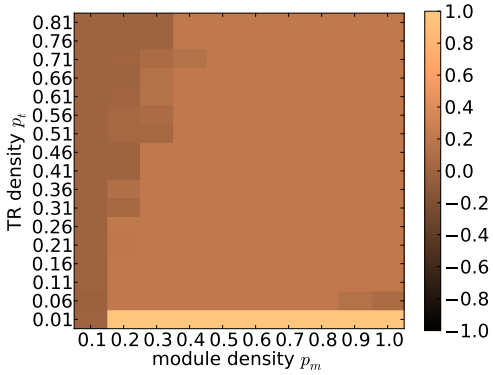
We conclude that even on very simple networks, where all modules have the same size and density, the algorithms we introduced do not perform perfectly. One major disadvantage of these approaches is their heuristic nature: They do not promise to find the optimal assignment, only a good such assignment.



(a) Value of ρ^c for MCL.



(b) Value of ρ^c for SCAN.



(c) Value of ρ^c for MSM.

Figure 3.4: Plotting the ρ^c score for the three algorithms for different combinations of p_m and p_t. All networks have 1000 nodes and 5 modules with 100 nodes each.

In the next section we introduce a new approach for the assignment problem with a goal function that is more amenable to exact optimization. For this, we use a combinatorial, graph-theory based approach.

3.3 A combinatorial approach to finding assignments

The approach described below is grounded on the foundation of the core–periphery structure of networks. Networks having this structure are characterized by a dense *core*, which is a single subset of the nodes that is highly connected, and a *periphery*, the remaining network nodes that are sparsely connected amongst themselves, but can be well-connected to the core. That is, it can be said that the periphery nodes are connected to one another mainly via the core. Then the general idea is to view a network as a union of multiple core–periphery structures of various sizes. The analogy to an assignment is then clear: the cores constitute the modules, while the peripheries constitute the transition region. We begin by reviewing the literature on core–periphery.

Core–periphery model. A formal definition of this model for networks was first given by Borgatti and Everett (1999) in the context of social networks. The authors discuss several intuitive notions of core and periphery as they appeared in literature: The network contains a *core*, a connected region which is very dense, and a *periphery*, which is sparsely connected. The connectivity between the core and periphery region can vary. The authors focus on the variant of a fully connected core, and a periphery that is fully connected to the core, but is in itself completely disconnected. The methods of Borgatti and Everett (1999) and in subsequent papers (e. g. (Brusco and Steinley, 2009; Muñiz and Carvajal, 2006)) to identify the best partition into core and periphery are based on comparison to an ideal matrix block structure. Further approaches are covered in a recent review from Csermely et al. (2013).

Only a few methods attempt to identify multiple cores and peripheries, and use them to analyze real data. Puck Rombach et al. (2014) again try to optimize the model where full connectivity between the core and periphery region is required, but this time for multiple cores. They do this by optimizing a continuous variant of the objective function of Borgatti and Everett (1999) to obtain an assignment into core and periphery. These authors apply their algorithms to discover communities in citation networks.

In graph theory, a core–periphery network is known by another name: a *split graph* is a union of a clique, defined previously (Section 1.2), and an *independent set*:

Definition 3.1. *A graph* G *is an* independent set *if no node in* G *is adjacent to any other node in* G.

Then we define a split graph:

Definition 3.2. *A graph* $G = (V, E)$ *is a* split graph *if* V *can be partitioned into* V_1 *and* V_2 *such that* $G[V_1]$ *is an independent set and* $G[V_2]$ *is a clique.*

Note that the partition for a split graph is not always unique. Another useful concept from graph theory is that of a *cluster graph*:

Definition 3.3. *A graph* G *is a* cluster graph *if each connected component of* G *is a clique.*

The cluster graph corresponds to a notion of a "perfect" full partition: Every connected component is a module. Then the problem of finding a full partition can be framed as the problem of transforming the original graph into a cluster graph. This is the optimization problem known as Cluster Editing (Shamir, Sharan, and Tsur, 2004), also known as Correlation Clustering (Bansal, Blum, and Chawla, 2004). We look specifically at transformations through *edge modifications*: adding or removing edges from the graph. From the perspective of an application, this is an error-correction problem: The graph is the result of some measurements, perhaps the edges denote similarity between the objects represented by the nodes. Now, because of measurement noise some similarities are measured as non-similarities and vice versa. Consequently, the input graph is not a cluster graph as expected. The task is to recover the underlying cluster graph from the input graph. We assume parsimony, that is, under the assumption that the errors are independent, the most likely cluster graph is one that disagrees with the input graph on a minimum number of edges. Such a graph can be found by a minimum number of edge modifications.

We now describe two core–periphery-based models in detail. The first one is based on the split graph described above:

Definition 3.4. *A* G *graph is a* split cluster graph *if every connected component of* G *is a split graph.*

Our first fitting model is thus described by the following optimization problem.

> Split Cluster Editing (SCE)
> Input: An undirected graph $G = (V, E)$.
> Task: Transform G into a split cluster graph by applying a minimum number of edge modifications.

In a variation of the model, we want to allow the vertices in the periphery to not only be attached to one core, but to an arbitrary number, thereby connecting the cores. In this model, we thus assume that the cores are disjoint cliques and the vertices of the periphery are an independent set. Such graphs are called monopolar (Z. A. Chernyak and A. A. Chernyak, 1986) and are natural generalizations of bipartite and split graphs.

Definition 3.5. *A graph is* monopolar *if its vertex set can be two-partitioned into* V_1 *and* V_2 *such that* $G[V_1]$ *is an independent set and* $G[V_2]$ *is a cluster graph. The partition* (V_1, V_2) *is called* monopolar partition.

> Monopolar Editing
> Input: An undirected graph $G = (V, E)$.
> Task: Transform G into a monopolar graph by applying a minimum number of edge modifications and output a monopolar partition.

The connection to modular networks. The split cluster graph resulting from SCE and the monopolar graph resulting from *monopolar editing* can both be viewed as assignments. These assignments are not fuzzy, since we do not have the affiliation function of the nodes in the periphery to the modules in the Monopolar case. In the SCE case we can interpret the independent set of each split graph as affiliated with the clique of that split graph with probability 1. This formulation naturally lends itself to a new modularity score for a network. The number k of edge modifications represents the minimal distance between the input graph and the split cluster graph. We can design a simple global score based on k:

$$I_k = \frac{k}{\binom{n}{2}}, \tag{3.9}$$

where $\binom{n}{2}$ thus represents all possible edge insertions and deletions. This score has a major drawback: We need to know k. As we show in the next sections, this is not easy to do. We first survey the literature on similar problems, then give some hardness proofs for the problems we defined above.

3.3.1 Related work

The previously examined optimization problem most closely related to our approach is Split Editing, which asks to transform a graph into a split graph by at most k edge modifications. Split Editing is, somewhat surprisingly, solvable in polynomial time (Hammer and Simeone, 1981); in fact, the number of required modifications depends only on the degree sequence. Thus, in particular, split graphs are recognizable by their degree sequence. This algorithm was rediscovered by Lip (2011) in the context of core–periphery models.

Cluster graph modification problems. Cluster Editing can be seen as one instance of a popular way of rigorously defining graph clustering problems: to ask for the minimum number of graph modifications that yield a graph in which every connected component (*cluster*) is dense in some sense. Many criteria on whether a graph on n vertices is considered dense have been considered: it is a clique (Bansal, Blum, and Chawla, 2004; Shamir, Sharan, and Tsur, 2004), each vertex has degree at least $n - s$ for some constant s (Guo, Komusiewicz, et al.,

2010), each vertex has degree more than $n/2$ (Hüffner et al., 2013), the cluster has diameter at most two (Liu, Zhang, and Zhu, 2012), and others (e. g. (Guo, Kanj, et al., 2011)). Allowed modifications are typically *deletion* (deleting edges), *insertion* (inserting edges), *editing* (deletion and insertion of edges), and *vertex deletion*.

Cluster Editing is NP-hard, even on graphs with maximum degree four (Komusiewicz and Uhlmann, 2012) and APX-hard (Charikar, Guruswami, and Wirth, 2005). The fastest parameterized algorithm, after a long series of improvements, runs in $O(1.619^k + n + m)$ time (Böcker, 2012). Under some complexity-theoretic assumptions, Cluster Editing cannot be solved within a running time that is subexponential in k (Fomin et al., 2013; Komusiewicz and Uhlmann, 2012). There are several variations of this model, for instance, one may assume that the clusters do not have to be perfectly dense, but each vertex has a limited number of missing edges (Guo, Komusiewicz, et al., 2010). Cluster Editing is also known as Transitivity Editing and has, for example, been applied to cluster protein families by sequence similarity (Wittkop et al., 2007). Several experimental studies have examined exact approaches for solving Cluster Editing on real-world instances (Böcker, Briesemeister, and Klau, 2011; Dehne et al., 2006).

3.3.2 Combinatorial properties and complexity

Preliminaries. As before, we consider undirected simple graphs $G = (V, E)$ where $n := |V|$ denotes the number of nodes (also: vertices) and $m := |E|$ denotes the number of edges. We denote the *neighborhood of a set* U by $N(U) := \bigcup_{u \in U} N(u) \setminus U$. For two disjoint vertex sets U and W, we use $E(U, W) := \{\{u, w\} \in E \mid u \in U \wedge w \in W\}$ to denote the edges between U and W.

Before presenting concrete algorithmic approaches for split cluster editing and monopolar editing, we show some properties of the types of graphs that we want to obtain which will be useful for the various algorithms. Furthermore, we present computational complexity results for the problems which will justify the use of integer linear programming (ILP) and heuristic approaches.

3.3.2.1 Split cluster graphs

For Split Cluster Editing, each connected component of the solution has to be a split graph. These graphs can be characterized by *forbidden induced subgraphs*. A graph class $\mathbb{F}$ has a forbidden subgraph characterization if a graph G belongs to $\mathbb{F}$ if and only if it does not contain any of the forbidden induced subgraphs (see for example Diestel (2005)). It is well-known that the graph classes that can be thus characterized are exactly those graph classes that are *hereditary*: A graph class $\mathbb{F}$ is called hereditary if for any $G \in \mathbb{F}$ also all induced subgraphs of G are in $\mathbb{F}$. Many interesting graph classes are characterized by forbidden subgraphs. Some of these classes have an infinite set of minimal forbidden subgraphs: bipartite graphs, which contain no odd-length cycles, are an immediate example. Others

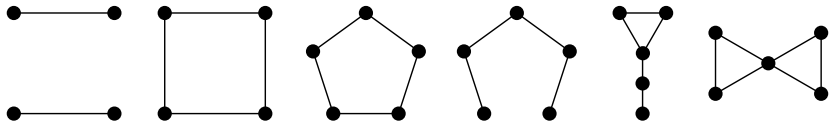


Figure 3.5: Forbidden subgraphs.

have a finite set of minimal forbidden subgraphs. Examples include the class of cluster graphs defined above, and cographs, which are P_4-free, denoting that they cannot contain a path of length 4 as an induced subgraph. Having a finite set of forbidden subgraphs gives an easy way of recognizing the graph class in polynomial time. We first quote a result for split graphs:

Theorem 3.1 (Foldes and Hammer (1977))**.** *A graph* G *is a split graph if and only if* G *does not contain an induced subgraph that is a cycle of four or five edges or a pair of disjoint edges (that is,* G *is* $(C_4, C_5, 2K_2)$*-free (see Figure 3.5)).*

Using this known characterization for split graphs, we obtain one for split cluster graphs. For this, the following lemma is helpful.

Lemma 3.1. *If a connected graph contains a* $2K_2$ *as induced subgraph, then it contains a* $2K_2 = (V', E')$ *such that there is a vertex* $v \notin V'$ *that is adjacent to at least one vertex of each* K_2 *of* (V', E')*.*

Proof. Let G contain the $2K_2$ $\{x_1, x_2\}, \{y_1, y_2\}$. Without loss of generality, let the shortest path between any x_i, y_j be $P = \{x_1 = p_1, p_2, \ldots, p_k = y_1\}$. Clearly, $k > 2$. If $k = 3$, then x_1 and y_1 are both adjacent to p_2. Otherwise, if $k = 4$, then $\{x_2, x_1 = p_1\}, \{p_3, p_4 = y_1\}$ is a $2K_2$ and x_1 and p_3 are both adjacent to p_2. Finally, if $k > 4$, then P contains a P_5. The four outer vertices of this P_5 induce a $2K_2$ whose K_2's each contain a neighbor of the middle vertex. □

Theorem 3.2. *A graph* G *is a split cluster graph* $\Leftrightarrow$ G *is* $(C_4, C_5, P_5,$ *necktie*, *bowtie*$)$*-free (see Figure 3.5).*

Proof. Let G be a split cluster graph, that is, every connected component of G is a split graph. Clearly, G does not contain a C_4 or C_5. If a connected component of G contains a P_5, then omitting the middle vertex of the P_5 yields a $2K_2$, which contradicts that the connected component is a split graph. The same argument shows that the graph cannot contain a necktie or bowtie.

Conversely, let G be $(C_4, C_5, P_5, \text{necktie}, \text{bowtie})$-free. Clearly, no connected component contains a C_4 or C_5. Assume for a contradiction that a connected component contains a $2K_2$ consisting of the K_2's $\{a, b\}$ and $\{c, d\}$. Then according to Lemma 3.1, there is a vertex v that is without loss of generality adjacent to a and c. If no other edges between the $2K_2$ and v exist, then $\{a, b, v, c, d\}$ is a P_5. Adding exactly one of $\{b, v\}$ and $\{d, v\}$ creates a necktie, and adding both

edges results in a bowtie. No other edges are possible, since there are no edges between $\{a, b\}$ and $\{c, d\}$. □

Theorem 3.3. *A forbidden subgraph for a split cluster graph can be found in* $O(n + m)$ *time.*

Proof. For each connected component, we run the algorithm by Heggernes and Kratsch (2007) that in linear time checks whether a graph is a split graph, and if not, produces a $2K_2$, C_4, or C_5. If the forbidden subgraph is a C_4 or C_5, we are done. If it is a $2K_2$, we can find in linear time a P_5, necktie, or bowtie, using the method described in the proof of Lemma 3.1. □

Hence, split cluster graphs can be recognized in linear time. In contrast, Split Cluster Editing is NP-hard even in restricted cases.

Theorem 3.4. Split Cluster Editing *is NP-hard even on graphs with maximum degree 11. Further, it is APX-hard and cannot be solved in* $2^{o(k)} \cdot n^{O(1)}$ *or* $2^{o(n)} \cdot n^{O(1)}$ *time if the exponential-time hypothesis (ETH) (Impagliazzo, Paturi, and Zane, 2001) is true.*

Proof. We reduce from Cluster Editing:

> Input: An undirected graph $G = (V, E)$ and an integer k.
> Question: Can G be transformed into a cluster graph, that is, a union of vertex-disjoint cliques, by applying at most k edge modifications?

Cluster Editing is NP-hard (Křivánek and Morávek, 1986) even if the maximum degree of the input graph is five (Fomin et al., 2013) and it cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ time assuming ETH (Fomin et al., 2013; Komusiewicz and Uhlmann, 2012).

The reduction works as follows. Given an instance (G, k) of Cluster Editing, build a graph $G' = (V', E')$ that has the same vertices and edges as G and an additional $\deg_G(v) + 1$ new degree-one vertices attached to each $v \in V$.

We show that G can be transformed by at most k edge modifications into a cluster graph if and only if G' has a split cluster editing set of size at most k. First, if a set S of at most k edge modifications transforms G into a cluster graph $\tilde{G}$, then performing the same modifications on G' converts G' into a split cluster graph $\tilde{G}'$: Each connected component of $\tilde{G}'$ contains a clique K of $\tilde{G}$ plus $\deg_G(v) + 1$ degree-one vertices adjacent to each $v \in K$. The set of these degree-one vertices is an independent set.

For the other direction, we show that if a set S' of $\leqslant k$ edge modifications transforms G' into a split cluster graph, then performing the same edits on G transforms it into a cluster graph. First we show that the new edges to the degree-one vertices are never deleted in an optimal solution. Clearly, it is never beneficial to delete just some of the edges: If v is to be assigned to a clique, then the new vertices are a part of the clique's independent set, and no edges have to

be deleted. If, otherwise, v is assigned to the independent set, then assume x of them remain adjacent to v. This causes least $\deg_G(v) + 1 - x$ edge deletions plus $x \cdot (x-1)/2$ edge insertions. This is always at least $\deg_G(v)$. In this case, however, a solution that is at least as cheap is to disconnect v from all other $u \in V$ and create a new connected component that is a split graph with v as the clique and its $\deg_G(v) + 1$ degree-one neighbors as the independent set. The cost here is only $\deg_G(v)$. Hence, we can assume that S' does not delete edges incident with the degree-one vertices in $V' \setminus V$.

Similarly, it can be shown that in an optimal solution, no edge can be inserted with one or both endpoints being a degree-1 vertex.

Thus, the new degree-one vertices in G' are part of the independent sets of $\tilde{G}'$. Consequently their neighbors, the vertices in V, belong to some clique. Furthermore, these cliques are vertex disjoint since $\tilde{G}'$ is a split cluster graph. Hence, S' transforms G into a cluster graph.

Hence, the reduction is correct. The hardness results follow from the previous hardness results and the fact that the solution size remains the same and that the maximum degree of the constructed graph G' is exactly twice the maximum degree of G plus one. □

This hardness result motivates the study of the *parameterized complexity* of Split Cluster Editing for the parameter number of edge modifications k. Parameterized complexity is a recent way of dealing with NP-hard problems. The NP-hardness of a problem implies that there are some worst-case instances where there is (we assume) no algorithm that can solve the problem in polynomial time. In practice, however, there could be many interesting instances (or classes of instances) that have a more restricted structure. On such instances it is possible that we can find an algorithm with an acceptable running time. The idea is to accept an exponential running time but confine the exponential part to a parameter which is assumed to be small in the application. This then makes it possible to solve NP-hard problems on real-world instances. See for example the textbooks by Niedermeier (2006) or Downey and Fellows (2013). A problem is called *fixed-parameter tractable* with respect to some parameter k if there is an algorithm that decides it in time $\mathcal{O}(f(k)n^c)$, where f is some function (usually exponential) and c is a constant.

In this case, the fixed-parameter tractability of Split Cluster Editing follows from a search tree algorithm that checks whether the graph contains a forbidden subgraphs, and, if this is the case, recursively branches into the possibilities to destroy this subgraph. In each recursive branch, the number of allowed edge deletions decreases by one. Furthermore, since the largest forbidden subgraph has five vertices, at most 10 possibilities for edge insertions or deletions have to be considered to destroy a forbidden subgraph. By Theorem 3.3, forbidden subgraphs can be found in $O(n+m)$ time. Altogether, this implies the following.

Theorem 3.5. Split Cluster Editing *can be solved in* $O(10^k \cdot (n+m))$ *time.*

3.3.2.2 Monopolar graphs

In contrast to the recognition of split cluster graphs, which is possible in linear time by Theorem 3.3, deciding whether a graph is monopolar is NP-hard (Farrugia, 2004). The class is hereditary, and so is characterized by forbidden induced subgraphs, but the set of forbidden induced subgraphs is infinite. Recent research is focused on the recognition problem for special graph classes. A fairly general such approach uses a 2-SAT formulation (Churchley and Huang, 2014; Nevries and Le, 2011). Since Monopolar Editing (that is, the problem of adding and deleting up to k edges to transform a graph into a monopolar graph) is NP-hard already for $k = 0$, it cannot be fixed-parameter tractable with respect to k unless $P = NP$.

3.3.3 Algorithms for Split Cluster and Monopolar Editing

From the forbidden subgraph characterization we already get algorithms with $10^k n^{O(1)}$ for Split Cluster Editing, $6^k n^{O(1)}$ for Split Cluster Deletion and Split Cluster Insertion, and $5^k n^{O(1)}$ for Split Cluster Vertex Deletion. These algorithms are based on a search tree approach as described above.

Continuing our previous thread of dividing the task of finding an assignment into two parts, we make the following observation: If we correctly guess the partition into clique and independent set vertices, we can get a simpler characterization of split cluster graphs by forbidden subgraphs.

Lemma 3.2. *Let $G = (V, E)$ be a graph and $C \cup I = V$ a partition of the vertices. Then G is a split cluster graph with clique vertices C and independent set vertices I iff it does not contain an edge with both endpoints in I, nor an induced P_3 with both endpoints in C.*

Proof. "⇒": Let $(u, v) \in E$. Then u and v belong to the same connected component $H \subseteq G$. Since G is a split cluster graph, H is a split graph, so edges are possible only if $u \in C$ or $v \in C$, therefore it is not possible to have $u, v \in I$. Let the induced graph on some u–v–w be a P_3. Again u, v, w belong to the same connected component H. Let $u, w \in C$. Then they must belong to the clique on vertices $C_H \subseteq C$, but there is no edge between u, w, a contradiction.

"⇐": We prove that if G is not a SCG with clique vertices C and independent sets I, it must contain an edge with both endpoints in I or a P_3 with endpoints in C. In this case there is a connected component H such that C_H is not a clique or I_H is not an independent set. If I_H is not an independent set, then there are $u, v \in I_H \subseteq I$ that are connected by an edge. If C_H is not a clique then there are $u, v \in C_H$ such that $(u, v) \notin E$. If there is a vertex $w \in H$ such that $(u, w), (w, v) \in E$, then the induced graph on u, v, w is a P_3 with both endpoints in C. Otherwise, since u and v belong to the same connected component H, they are connected by a path of vertices from H. Let $P = u = p_1, \ldots, p_k = v$ for some $k > 3$ be the shortest path between u, v. Assume w.l.o.g. that we picked u, v with

the minimum path length k. Then $p_i \in I$ for all $i \neq 1, k$, otherwise there is a subpath of P with endpoints in C with length $< k$, contrary to the minimality of k. Since $k < 3, p_i \in I$, P must contain 2 adjacent vertices $w_1, w_2 \in I$, and thus there is an edge $(w_1, w_2) \in E$ with both endpoints in I. □

With a very similar proof, we can get a simpler set of forbidden subgraphs for annotated monopolar graphs.

Lemma 3.3. *Let $G = (V, E)$ be a graph and $C \cup I = V$ a partition of the vertices. Then G is a monopolar graph with clique vertices C and independent set vertices I iff it does not contain an edge with both endpoints in I, nor an induced P_3 whose vertices are contained in C.*

Proof. "⇒": Let $(u, v) \in E$. Then u, v belong to the same connected component $H \subseteq G$. Since G is monopolar, H is a clique, so edges are possible only if $u \in C$ or $v \in C$, therefore it is not possible to have $u, v \in I$. Let the induced graph on some u–v–w be a P_3. Then u, v, w belong to the same connected component H. If additionally $u, v, w \in C$, then the P_3 must belong to the cluster graph $C_H \subseteq C$. But P_3s are forbidden subgraphs of cluster graphs (Shamir, Sharan, and Tsur, 2004), a contradiction.

"⇐": We prove that if G is not monopolar with clique vertices C and independent sets I, it must contain and edge with both endpoints in I or a P_3 with all vertices in C. In this case there is a connected component H such that the graph induced on C_H is not a cluster graph or the graph induced on I_H is not an independent set. If I_H is not an independent set, then there are $u, v \in I_H \subseteq I$ that are adjacent. If C_H is not a cluster graph then according to Shamir, Sharan, and Tsur (2004) it must contain a P_3. All vertices of this path are then contained in $C_H \subseteq C$. □

3.3.3.1 Integer Linear Programming

From Lemma 3.2, we can directly derive an integer linear programming formulation for Split Cluster Editing. We introduce binary variables e_{uv} indicating whether $\{u, v\}$ is an edge in the edited graph and binary variables c_u indicating whether a vertex u is part of the core. Defining $\bar{e}_{uv} := 1 - e_{uv}$ and $\bar{c}_u := 1 - c_u$, and fixing an arbitrary order on the vertices, we have

$$\text{minimize} \sum_{\{u,v\} \in E} \bar{e}_{uv} + \sum_{\{u,v\} \notin E} e_{uv} \text{ subject to} \tag{3.10}$$

$$c_u + c_v + \bar{e}_{uv} \geqslant 1 \; \forall u, v \tag{3.11}$$

$$\bar{e}_{uv} + \bar{e}_{vw} + e_{uw} + \bar{c}_u + \bar{c}_w \geqslant 1 \; \forall u \neq v, v \neq w > u. \tag{3.12}$$

For Monopolar Editing, we can replace constraint 3.12 by

$$\bar{e}_{uv} + \bar{e}_{vw} + e_{uw} + \bar{c}_u + \bar{c}_v + \bar{c}_w \geqslant 1 \; \forall u \neq v, v \neq w > u. \tag{3.13}$$

Since this formulation has $O(n^3)$ constraints, we use row generation (lazy constraints) and, in a solver callback, add only constraints that are violated by the incumbent solution.

3.3.3.2 Data reduction

A common strategy for designing fixed parameter algorithm is using *data reduction rules* (Downey and Fellows, 2013; Niedermeier, 2006). Those are efficient transformations that reduce the problem instance to a smaller instance that still allows to retrieve a solution to the original instance from the reduced instance. This is a powerful classic approach that can be combined with heuristics, approximation algorithms, and others, to solve instances more efficiently.

Rules for Split Cluster Editing. We define *setting an edge* e *permanent* as the following operation: First, add e if it is not already in the graph. Second, label e as permanent. In this way we indicate that e must be a part of the solution, the resulting split cluster graph. Similarly, *setting an edge* e *forbidden* is the following: First, delete e if it is already in the graph. Second, label the vertex pair e as forbidden. This indicates that e cannot be a part of the optimal solution.

Rule 3.1. *If there is a degree-one vertex v whose neighbor has degree larger than one, then label v as periphery.*

Proof of correctness. Assume v is a core vertex in some solution. Let u be the neighbor of v. If $\{u, v\}$ was deleted, we can just make v a periphery vertex. Otherwise, if u is a core vertex, we can also make v a periphery vertex. Finally, if u is a periphery vertex, then $\{u, v\}$ form a two-vertex connected component in the solution, and we can make u a core vertex and v a periphery vertex. □

With similar arguments, we can prove the correctness of the following two rules.

Rule 3.2. *Set an edge e between two vertices labeled as periphery to forbidden.*

Rule 3.3. *If there is a vertex v of degree at least two such that v has at least as many degree-one neighbors as other neighbors, then label v as core and label each edge between v and a degree-one neighbor as permanent.*

Monopolar Editing

Rule 3.4. *Remove all degree-one vertices.*

3.3.3.3 Heuristics

The integer linear programming approach, even with data reduction, is not able to solve the hardest of our instances. Thus, we need to revert to heuristic approaches.

We first tried heuristics based on the forbidden subgraph characterization: for instance, find the vertex pair contained in the most forbidden subgraphs, and edit it. Unfortunately, enumerating all forbidden subgraphs is prohibitively slow for the instances of interest.

We obtained better results with the well-known *Simulated Annealing* heuristic. This is a local search method, where we try a random modification of our current solution, and accept it if it improves the objective; but to escape local minima, we also accept it with a small probability if it makes the objective worse. More precisely, a change in the objective of Δ is accepted with probability $\exp(\Delta/T)$, where the factor T is reduced over the course of the algorithm down to zero, such that the algorithm initially explores a larger part of the search space, but eventually settles in a local minimum. We repeat the simulated annealing algorithm with a fixed number of steps until the user-defined time limit is exceeded.

For Split Cluster Editing, we start with a clustering where each vertex is a singleton. As the random modification, we move a vertex to a cluster that already contains one of its neighbors. Since this would allow the number of clusters to only shrink monotonically, we also allow moving a vertex into an empty cluster. For Monopolar Editing, we also allow moving a vertex into the independent set.

3.3.3.4 Summary.

After some experimentation, we settled on the following pipeline for finding a good assignment for a given network:

1. Apply data reduction rules to the network to obtain a simpler instance.
2. Use the simulated annealing heuristic to find a good solution.
3. Solve the ILP formulation with the IBM CPLEX 12.5 solver, using the solution from the heuristic as a starting point.

3.4 Application: Protein interaction networks

Protein–protein interaction networks (PPI networks or PPINs) are a snapshot of physical interactions between proteins. The nodes therefore represent proteins, and there is an edge between two proteins if they interact under some condition. This interaction can be either observed directly in a lab experiment, or implied as a result of a different experiment or a computational prediction. How are PPINs

created? There are different experimental methods (see Jaimovich (2010) and Rivas and Fontanillo (2010) for reviews). Some, like yeast two-hybrid (Ito et al., 2001) output binary interactions. That is, it is possible to test directly whether two proteins interact. Others, like co-immunoprecipitation (Fields and Song, 1989) output sets of proteins that are related. Then, a network can be constructed based exclusively on the experimental results, on the experimental results with added statistical testing and thresholding, or by integrating the results of several previous experiments. There also exist networks that integrate physical interactions with other types of interactions, for example STRING (Jensen et al., 2009) or ConsensusPathDB (Kamburov et al., 2011). Some state-of-the-art PPI networks for various species can be found for example in BioGrid (Chatr-aryamontri et al., 2013). The *Saccharomyces cerevisiae* (herein: yeast) network from BioGrid will be analyzed later in this chapter.

Several proteins can come together to form *protein complexes*. These complexes are the underlying elements of many biological processes and together form the molecular machinery that performs a vast array of biological functions. Some examples are the proteasome for molecular degradation, itself comprising subcomplexes like the 20S core, the metabolon for oxidative energy generation, and parts of the large and complex ribosome for protein synthesis. Complexes can be permanent or transient, and a single protein can participate in different complexes having diverse functions under different conditions (Amoutzias and Peer, 2010; Price and L. Stevens, 1999). Along with experiments that aim at studying a single complex, large-scale experiments can be performed to determine multiple protein complexes simultaneously (Babu et al., 2012; Gavin et al., 2006). Computational methods can also be leveraged: The CYC2008 database of yeast complexes (Pu, Vlasblom, et al., 2007; Pu, Wong, et al., 2009) was constructed by running MCL clustering (see Section 3.2) on a yeast PPI network and biologically evaluating the resulting clusters as candidate protein complexes. Thus catalogs of protein complexes for different species can be produced, such as MIPS (Mewes et al., 2002) and the aforementioned CYC2008.

Computationally inferring complexes. Many computational methods have been developed to infer protein complexes automatically from PPINs. The underlying assumption of these methods is that protein complexes correspond to dense subgraphs of the PPIN, as the proteins in a complex must interact with each other in order to perform the given function. It is natural, then, to formulate the problem of identifying protein complexes in PPINs as a clustering problem, where the clusters correspond to the complexes. Srihari and Leong (2012) provide a recent review of such methods.

One way of classifying these methods is by the kind of clustering they apply: Does every protein belong to a complex or can there be proteins not assigned to a complex? In other words, does the method return a full partition of the network or an assignment? Additionally, we can ask whether a protein must to belong

to *exactly one* complex, or whether it be a part of multiple complexes. That is, are the clusters found by the algorithm allowed to overlap? As hinted in the construction of the CYC2008 database, MCL and its variants are popular tools for this task: MCL has been shown to give consistently good results in comparison to other full-clustering algorithms (Brohee and Helden, 2006). It has also been extended to allow overlaps (Babu et al., 2012; Enright, Dongen, and Ouzounis, 2002) and integrate functional information for even higher accuracy (Srihari, Ning, and Leong, 2010).

An important class of computational methods for identifying complexes in PPI networks is closer in spirit to the assignment approach: These are algorithms that do not fully partition the network, but search for dense subgraphs that are loosely connected to the rest of the network and are maximal in some sense. Proteins that do not belong to any dense subgraph are ignored. Possibly the first dense-cluster-based method to be applied to PPINs is MCODE (Bader and Hogue, 2003). It assigns weights to the nodes according to topological features, identifies seeds and controls their expansion via user-defined parameters. Another interesting example is MCL-CAw (Srihari, Ning, and Leong, 2010), an extension of MCL.

Many of these algorithms can be classified as *seed-growing algorithms*, where we start with a single protein or a small, connected set of proteins and extend it to include neighboring proteins under some constraints, see for example the work of B. Chen et al. (2013) and Georgii et al. (2009).

The idea of overlapping clusters in the scope of a full partition has been more widely adopted, and a multitude of methods seek to identify such clusters, whether under the assumption that the overlap between them is very high (clique propagation methods (Adamcsek et al., 2006)), or small relative to the size of the clusters (see, for example, Lei et al. (2013)). In the second case, proteins that can be associated with more than one complex are sometimes called "shared components" (Krause et al., 2004). Shared components are particularly interesting to study, for example in their role as *linker proteins*, facilitating the communication between different processes (Vaggi et al., 2012).

An advantage of the full-partition model for identifying complexes is that it can assign putative membership in a complex to proteins whose function is otherwise unknown. This is because every protein must be assigned to some module, and therefore to some complex. We can then in principle use this information to discover the function of these proteins, as it should be similar to that of the other proteins in the complex. However, even in well-studied species such as yeast, not all protein interactions are known. For example, Babu et al. (2012) published just recently about two thousand(!) new interactions between membrane proteins, which comprise around 40% of the genome in the case of human and yeast. These interactions had until now not been detected due to technical limitations, and are thus not a part of any of the gold standard networks used for analysis. Interactions between membrane proteins and their assembly

into complexes are not so easy to study, despite their importance and the large body of work dedicated to inferring them: there is still much to be discovered about complexes and proteins. The problematic nature of full-partition clustering of PPINs is most acute for less-studied proteins, for which fewer interactions with other proteins are known: Even if a protein is part of a complex, it might not have many edges to it in the gold standard PPIN. A full-partition algorithm might force such a protein into a complex to which it does not really belong.

Core–attachment model. An interesting subclass of algorithms to find complexes via identification of dense subgraph adopts the *core–attachment* model (Leung et al., 2009; Luo et al., 2009; Pang et al., 2008; Wu et al., 2009), where each complex is assumed to contain a core of proteins that belong to it alone and is quite dense, and an attachment with proteins that are more loosely connected to the core, and can also be shared between complexes. The core–attachment model was presented by Dezső, Oltvai, and Barabási (2003) in small-scale experiments in yeast, and demonstrated that core proteins are more co-expressed than the attachment proteins, concluding that attachment proteins are spurious and short-lived. Later, Gavin et al. (2006), who performed the first genome-wide screen for complexes in yeast, observed the same. COACH (Wu et al., 2009) is an example of an algorithm that detects protein complexes under this model, first looking for cores and then extending them. Yu, Gao, and K. Li (2010) presented the LDRW algorithm that first identifies cores with local density and then extends them using random walks with restarts. This model of course immediately brings to mind the core–periphery model discussed in Section 3.3. Indeed, we can now use the combinatorial algorithms we introduced to analyze PPI networks and identify putative protein complexes. For completeness, we also run the MSM clustering and SCAN algorithms on these networks and compare the results.

3.4.1 Finding assignments in protein interaction networks

In both the Split Cluster Editing and Monopolar Editing models described in Section 3.3.3, we assume that all proteins of the cores interact with each other, implying that the cores are cliques. We also assume that the proteins in the periphery interact only with the cores but not with each other. Hence, the peripheries are independent sets. This is in line with the aforementioned idea that complexes in a PPI network are strongly connected amongst themselves and weakly connected to other complexes (Spirin and Mirny, 2003), hence in our models the cores correspond to the functional units and the peripheries connect to the cores as needed, and can sometimes be shared between cores (Gavin et al., 2006).

In the SCE model, we further assume that ideally the protein interactions give rise to *vertex-disjoint* core–periphery structures, that is, there are no interactions between different cores and no interactions between cores and peripheries of

other cores. Then each connected component has at most one core which is a clique and at most one periphery which is an independent set. This is admittedly a rather generous assumption, but the relative simplicity of SCE over the more difficult-to-compute Monopolar Editing makes it still interesting to study. Furthermore, it is clear that both models are simplistic and cannot completely reflect biological reality. For example, subunits of protein complexes consisting of two proteins that first interact with each other and subsequently with the core of a protein complex are supported by neither of our models. Nevertheless, our models are less simplistic than pure clustering models that attempt to divide protein interaction networks into disjoint dense clusters. Furthermore, there is a clear trade-off between model complexity, algorithmic feasibility of models, and interpretability.

We test exact algorithms and heuristics for SCE and Monopolar Editing on several PPI networks, and perform a biological evaluation of the modules found. We compare the results for SCE and Monopolar Editing with MSM clustering, and the SCAN algorithm tested in Section 3.2.3. When running SCAN, the resulting periphery or transition region comprises all hubs and outliers. The algorithm is ran with several parameter combinations, obtaining different results. For consistency, we select the results where the clusters have the highest Q-score, as reported by the SCAN program itself. In practice, the parameters for these high modularity clusters turn out to be $\varepsilon \approx 0.5$ and minimal cluster size $\mu = 2$, and the modularity reported is low, ranging from 0.2 to 0.45.

3.4.2 Experimental setup

Data. We perform all our experiments on networks from BioGrid (Chatr-aryamontri et al., 2013). BioGrid networks contain both physical interactions, which serve as the edges in our networks, and genetic interactions, which we can use for evaluations. We focus here on subnetworks of the yeast PPI network. From the complete BioGrid yeast network with 6377 nodes and 81549 edges, we extract three subnetworks, corresponding to three essential processes: cell cycle, translation, and transcription. To determine the protein subsets corresponding to each process, we query BioMart (Kasprzyk, 2011) for all yeast genes annotated with the relevant GO terms: GO:0007049 (cell cycle), GO:0006412 (translation), and GO:0006351 (DNA-templated transcription). For each subset we extracted the physical interactions from BioGrid.

Biological Evaluation. We evaluate our results using several well-accepted measures. First, as is standard when inferring complexes, we perform GO term analysis (Ashburner et al., 2000). GO is an ontology that annotates each protein with terms describing its function, process, or location in the cell. We test the coherency of the GO terms in our modules: The cores should contain nodes that are annotated with similar GO terms, while the peripheries should contain nodes

annotated with different GO terms: The idea is that the cores are stable and the peripheries less so. We test only terms relating to process, not function, since proteins in the same complex share a process. The semantic similarity algorithm we apply is G-SESAME (Du et al., 2009). This test is applicable to the SCE results, as the Monopolar editing, MSM clustering and SCAN return multiple cores and a only single periphery or transition region. Therefore we cannot compare the coherence of the cores against the coherence of the peripheries.

Second, we compare our resulting complexes with existing protein complex repositories. We use the CYC2008 database (Pu, Wong, et al., 2009) of yeast complexes. As the networks we analyze are subnetworks of the larger yeast network, we create, for each network, a restricted list of complexes with only proteins contained in the networks. We first test the overlap between the algorithm results and these complexes. We treat these complexes as the "ground truth" or planted assignment. However, since the list of complexes is incomplete, we cannot compare the results and complexes using the Rand index variants developed in Section 3.2.2.

Finally, we analyze the genetic interactions between and within modules. In addition to the physical interactions we use to construct the networks, BioGrid contains information about genetic interactions. Some will be positive, some negative. A negative genetic interaction between two proteins implies a "synthetic lethal" or "synthetic sick" relationship between them: knocking out both genes kills or cripples the cell, while knocking out any one of them has no such effect. In contrast, positive interactions occur when knocking out both genes has a smaller negative effect on the cell than expected. Ideally, we will expect significantly more genetic interactions outside of cores than within them. This is supported by the *between pathways model* (Kelley and Ideker, 2005), which proposes that different complexes can back one another up, thus disabling one would not harm the cell, but disabling both complexes would reduce its fitness or kill it. Here, when counting genetic interactions, we are interested only in genetic interactions that occur between proteins that do not physically interact.

The selected yeast subnetworks. Some statistics for these networks are given in Table 3.1.

The *cell cycle*, or cell-division cycle, is the series of events that take place in a cell leading to its replication, producing, in the case of *Saccharomyces cerevisiae*, a daughter cell via budding (Spellman et al., 1998). This network contains several important complexes, such as the cyclin-CDK complexes that promote the expression of transcription factors or the anaphase promoting complex marking proteins for degradation. This network contains 215 proteins and 797 edges. Additionally, there are 1151 genetic interactions among the proteins.

Transcription is the first step of gene expression, in which a particular segment of DNA is copied into RNA by the enzyme RNA polymerase. This network contains, for example, the Mediator complex, a transcription coactivator in-

	n	m	n_{lcc}	m_{lcc}	C	$\lvert\mathcal{T}\rvert$	A_C	i_g
cell cycle	215	797	192	795	17	148	3.9	1151
transcription	215	786	198	776	9	54	22.8	1479
translation	236	2352	186	2351	30	88	5.1	174

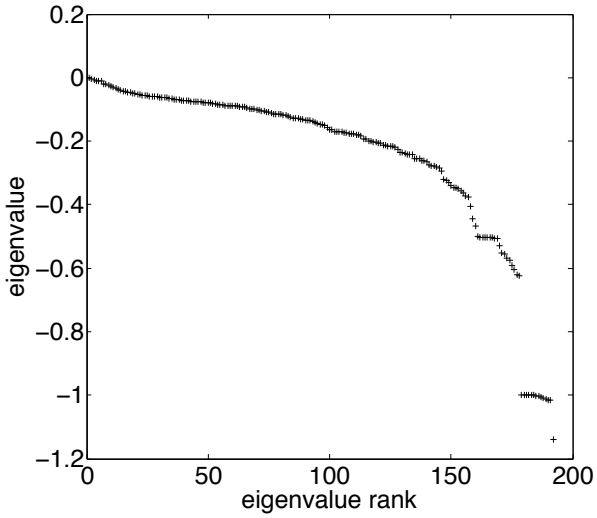
Table 3.1: Network statistics. Here, n is the number of proteins in the network, ignoring singletons, and m is the number of edges. The number of nodes in the largest connected component is n_{lcc}, and the number of edges is m_{lcc}. We provide some data on known complexes, where C is the number of complexes from the CYC2008 database with at least two proteins in the network, $|\mathcal{T}|$ is the number of proteins that do not belong to these complexes, and A_C is the average complex size. Finally, i_g is the number of genetic interactions observed between proteins that are not physically interacting.

creasing gene expression via transcription factor binding, along with several important polymerase-related complexes. This network contains 215 proteins and 786 physical interactions between them. There are additionally 1479 genetic interactions between the proteins.

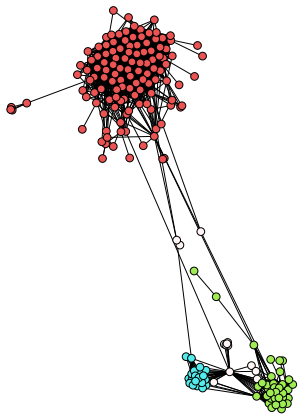
As stated in the official GO term definition, *translation* refers to "[t]he cellular metabolic process in which a protein is formed, using the sequence of a mature mRNA molecule to specify the sequence of amino acids in a polypeptide chain." A major component of this network is the ribosome, a large complex which synthesizes the translation process. The ribosome is comprised of two parts: the *small subunit* monitors the contact between tRNA anticodon and mRNA, while the *large subunit* catalyzes peptide bond formation. The network contains 236 proteins and 2352 physical interactions between them, along with 174 genetic interactions.

Implementation details. The data reduction (Section 3.3.3.2), Integer Linear Programming (Section 3.3.3.1), and simulated annealing heuristic (Section 3.3.3.3) were implemented in C++ and compiled with the GNU g++ 4.7.2 compiler. As ILP solver, we used CPLEX 12.5.1. For the ILP, we initially add all independent set constraints (3.11), and in a cutting plane callback, add the 1000 most violated constraints of type (3.12) or (3.13). In the simulated annealing heuristic, we use 10^6 steps and an initial $T_0 = 1$ for Split Cluster Editing, whereas for Monopolar Editing we use only $2 \cdot 10^4$ steps and $T_0 = 0.1$. That is, for Monopolar Editing we use more frequent restarts and a "more hill-climbing" strategy; we found this to give better results for our instances, probably because it is harder in Monopolar Editing to escape a region of local minima.

The test machine is a 4-core 3.6 GHz Intel Xeon E5-1620 (Sandy Bridge-E) with 10 MB L3 cache and 64 GB main memory, running under Debian GNU/Linux 7.0.



(a) Spectrum of L for the cell cycle network



(b) The ribosome network with the assignment found by MSM clustering.

Figure 3.6: MSM clustering results.

3.4.3 Results

Results of MSM clustering. Unfortunately, the MSM clustering algorithm does not perform well on our chosen biological networks. Although the process of removing the transition region can help in sharpening the eigenvalue gap, making the clustering process easier, when the structure of the network is complicated this is not enough. When there is no gap at all in the spectrum of the generator L (2.11), the choice of the appropriate lag-time α is arbitrary (Figure 3.6(a)). Removing the transition region results in many small connected components, but the gap of the $\hat{P_\alpha}$ matrix indicates the existence of of only a few modules. Altogether, the process results in few large clusters and several clusters containing only one or two proteins. We have some success applying MSM clustering to the network corresponding to the ribosome (GO:0005840). This network has a simpler structure, with three clear clusters. These clusters (see Figure 3.6(b)) correspond to the major components of the ribosome: The cytoplasmic ribosomal large subunit (red), the mitochondrial ribosomal small subunit (green), and the mitochondrial ribosomal large subunit (light blue).

Here we use the generator L_b introduced in Section 2.4, where the waiting time is proportional to the clustering coefficient. Thus the nodes between the two small clusters are correctly assigned to the transition region despite their high degree.

The results for the remaining algorithms are summarized in Table 3.2.

	cell-cycle				transcription				translation			
	K	$\lvert\mathcal{T}\rvert$	k	c	K	$\lvert\mathcal{T}\rvert$	k	c	K	$\lvert\mathcal{T}\rvert$	k	c
SCE	12	108	321	0.63	14	112	273	0.56	6	94	308	0.72
Monopolar Editing	21	75	126	0.57	27	78	106	0.60	11	129	240	0.56
SCAN	25	48	—	0.60	33	58	n/a	0.58	11	25	—	0.58

Table 3.2: Experimental results. Here, K is the number of clusters with at least two vertices, $\lvert\mathcal{T}\rvert$ is the size of the transition region, k is the number of edge edits, and c is the average coherency within the cores.

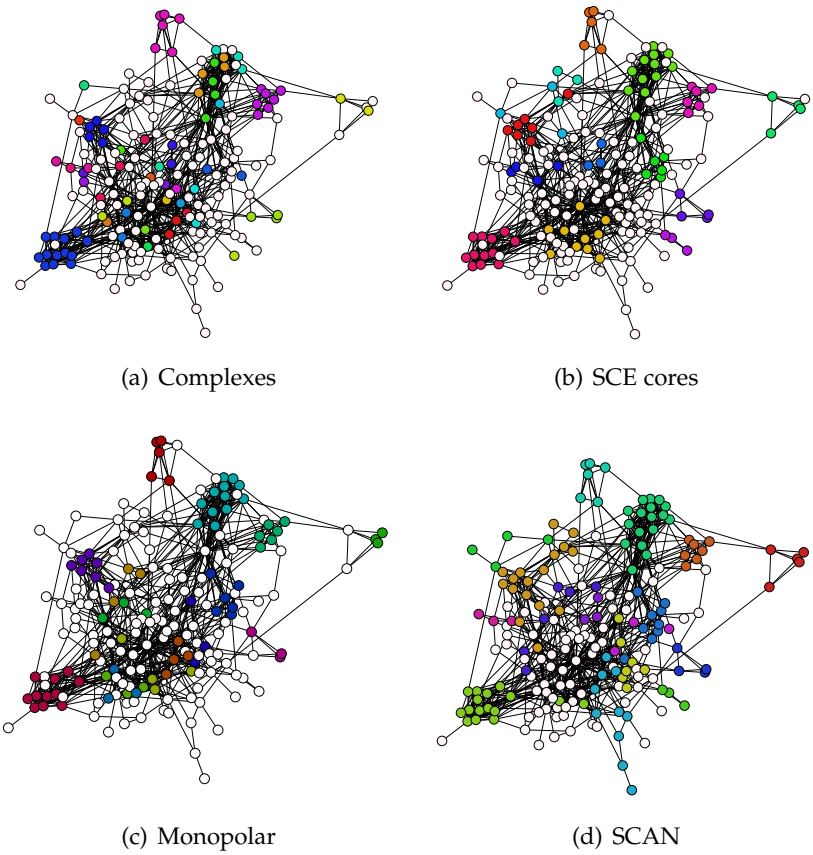
Results for SCE and Monopolar Editing. We apply the pipeline from Section 3.3.3.4. From the data reduction rules, unfortunately only Rules 3.1 and 3.2 were applicable; these rules allow determining that an edge between certain vertex pairs never needs to be inserted, which decreases the number of variables in the ILP.

Unfortunately, even after data reduction, the exact approach was not able to solve the SCE and Monopolar Editing instances, with CPLEX running out of memory. Only for SCE for the ribosome network, when given the solution from the simulated annealing heuristic as a starting point, CPLEX was able to prove its optimality after 80 minutes. Thus, we use the heuristic solution for all six clusterings. From experiments with other networks, we conjecture that the three SCE solutions are optimal; we are less sure about the Monopolar Editing.

The cell cycle network. We describe the results of Split Cluster Editing, Monopolar Editing, and SCAN on the cell cycle network (Figure 3.7). We begin with SCE. The algorithm identifies eight clusters that contain at least two nodes in the core and one in the periphery, along with four clusters containing only cores, and some singletons. When testing the similarity of GO terms within the cores versus the peripheries, we find that seven of the eight clusters have higher coherence of terms in the core than the periphery, as expected. The average coherence in the cores is 0.63, and the average coherence in peripheries is 0.39.

We compared the overlap of the known complexes with the set of cores and the set of cores and their associated peripheries. We find that the complexes correspond to the cores rather to the cores and peripheries. For example, we find a core corresponding to the anaphase promoting complex. All 12 proteins in the core are members of this complex of size 14. The periphery contains six proteins, only two of them a part of the complex. We find cores corresponding to the nuclear condensin complex, DASH complex, MIND complex, and the other complexes having at least two proteins in the cell cycle network.

The Monopolar Editing heuristic returned more clusters than SCE, 21 cores containing at least two proteins. The average coherence in these cores is 0.57,



(a) Complexes (b) SCE cores

(c) Monopolar (d) SCAN

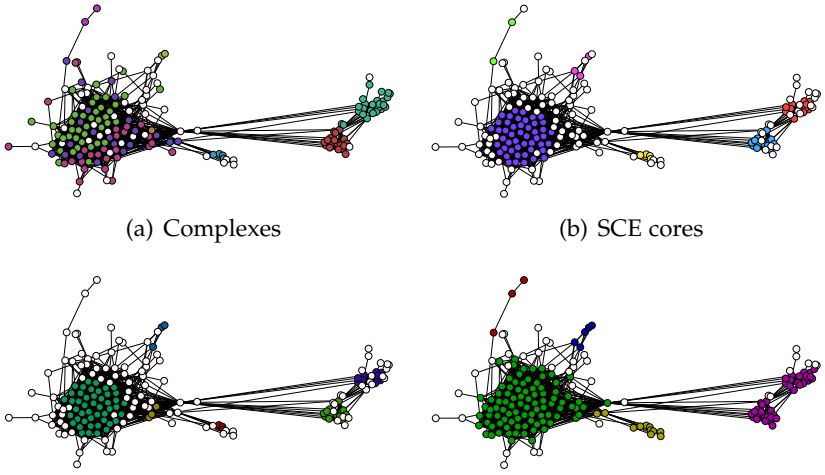
Figure 3.7: The results of the three algorithms on the cell-cycle network. The periphery is in white, remaining nodes are colored according to their clusters.

lower than for SCE.

SCAN identifies 7 hubs and 41 outliers, which then comprise the transition region. There are 25 clusters. These contain at least two proteins, as $\mu = 2$. The average coherence in these clusters is 0.6, lower than the coherence obtained by the cores found by SCE but higher than the monopolar results. The clusters found by SCAN correspond to the same complexes as those found by SCE. The SCAN clusters are slightly larger than the SCE cores, containing additional proteins that do not belong to the complex. This might account for their lower GO-term coherence.

The translation network. The results of the three algorithms on this network, along with the known complexes containing at least 3 proteins, can be seen in Figure 3.8.

When running SCE, we find five clusters containing at least two nodes in the core and one in the periphery, along with one cluster containing only cores,



(a) Complexes (b) SCE cores

(c) Monopolar (d) SCAN

Figure 3.8: The results of the three algorithms on the translation network. The periphery is in white, remaining nodes are colored according to their clusters.

and some singletons. As opposed to the cell cycle network, when testing the similarity of GO terms within the cores versus the peripheries we find that three of the clusters have more coherent cores while the other two have more coherent peripheries. The average coherence in the core is 0.72, while in the peripheries it is lower, 0.68.

We compared the overlap of the known complexes with the set of clusters. Most complexes correspond to detected clusters: the mitochondrial ribosomal small and large subunits (27 and 73 proteins in the network, respectively), and the cyoplasmic ribosomal large subunit. The cytoplasmic ribosomal small subunit, consisting of 42 proteins, is not detected.

The Monopolar Editing heuristic returned more clusters than SCE, 11 cores containing at least two proteins. The average coherence in these cores is 0.56, lower than the SCE.

SCAN identifies no hubs and 25 outliers, comprising the transition region. There are 11 clusters, of which two are large, containing 91 and 39 proteins. The average coherence in these clusters is 0.58, again lower than the coherence obtained by the cores found by SCE but slightly higher than the Monopolar Editing results. With respect to the overlap of clusters and complexes, the results are similar to those of the cell cycle network: The same complexes are detected by SCAN and SCE, with the exception of the mitochondrial ribosomal small subunit, consisting of 19 proteins in this network. This complex is covered by a 13-protein cluster in the SCE results, and not detected at all by SCAN. As in the cell cycle case, the clusters detected by SCAN contain additional proteins that are not a part of the complexes, which could explain the lower coherence.

The transcription network. The results of the three algorithms on this network, along with the known complexes containing at least 3 proteins, can be seen in Figure 3.9. When running SCE, we find twelve clusters containing at least two nodes in the core and one in the periphery, along with two clusters containing only cores, and some singletons. As opposed to the cell cycle network, when testing the similarity of GO terms within the cores versus the peripheries, we find that half of the clusters have more coherent cores, while the other half have more coherent peripheries. The average coherence in the cores and in the peripheries is the same, about 0.56.

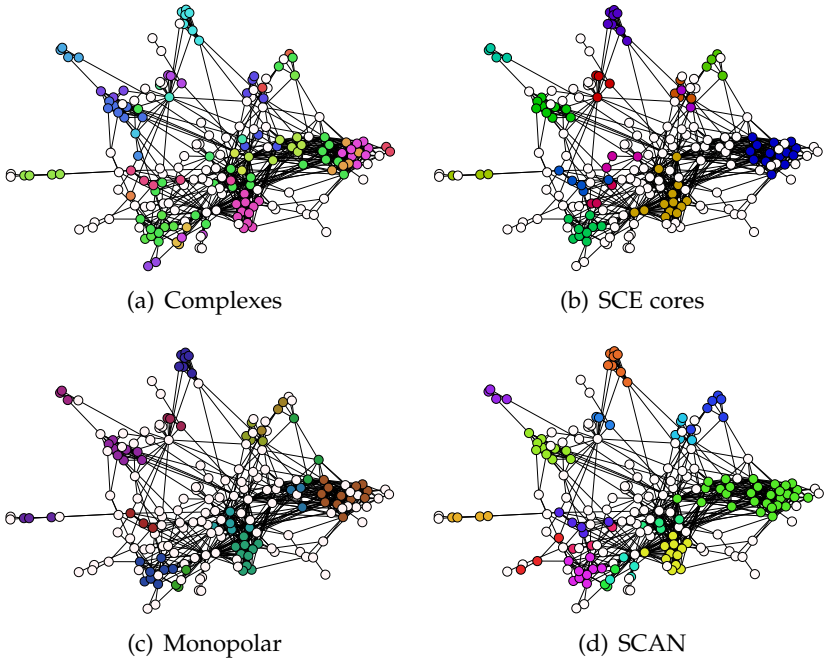
This network contains 29 complexes of at least two proteins. From these, our cores overlap with only eleven. We find, for example, the 5-protein Rpf3L complex (one of the cores matches it completely), the ARGR complex, and the UTP-A complex. We miss diverse complexes such as Swr1p, RSC, and the DNA-directed RNA polymerase II.

The Monopolar Editing heuristic returned more clusters than SCE, 27 cores containing at least two proteins, of which 15 contains at least three proteins. The average coherence in these cores is 0.6. This is higher than the coherence of the SCE cores, but is still a low score overall.

SCAN identifies 7 hubs and 51 outliers, comprising the transition region. There are 33 clusters, mostly of the minimal size 2. The average coherence in these clusters is 0.58, higher than in the clusters found by SCE but still lower than the Monopolar Editing clusters. The coverage of the complexes is better than in the SCE case, as the clusters cover 15 complexes. Those remaining clusters that do not correspond to any known complexes generally contain just two proteins.

Counting genetic interactions. In light of the fact that the clusters we identify largely correspond to known protein complexes, it is perhaps not surprising that we identify a higher than expected number of genetic interactions between these complexes. We applied the binomial test to check whether the frequency of genetic interactions in the periphery is significantly higher than the frequency in the entire network. We obtain p-values lower than $4 \cdot 10^{-7}$. In some cases, for example SCAN on the transcription network, the p-value is less than 10^{-41}. This is not the case when testing the peripheries of the clusters found by SCE: Recall that this algorithm returns set of cores and their associated peripheries. When testing whether the edges within these peripheries are similar to those within cores in their enrichment for genetic interactions, we see that they are more similar to the remaining periphery edges (which include edges between peripheries and between different cores). The frequency of genetic interactions within these peripheries is more than 100 times higher than in the cores.

Experiments conclusion. From these experiments, we can conclude that it useful to view our networks as unions of cores and peripheries (or a single periphery). Mainly, many clusters we identify correspond to known protein



(a) Complexes (b) SCE cores

(c) Monopolar (d) SCAN

Figure 3.9: The results of the three algorithms on the cell-cycle network. The periphery is in white, remaining nodes are colored according to their clusters.

complexes. The proteins in the cores are annotated with similar GO terms. The cores found by the Split Cluster Editing algorithms are more coherent than those found by the other two algorithms. This can be due to the fact that the underlying structure of the network is more similar to multiple cores and peripheries rather than to multiple cores and a single periphery. An alternative theory is that the Monopolar Editing heuristic does not return solutions that are close enough to optimal, and a better algorithm would return better results. Similarly, SCAN might return better results with a more careful consideration of parameters. It would be interesting to continue improving the algorithms to make it possible to find the model that best describes the network. Already our experiments on just three subnetworks hint that different biological processes might be organized differently: Our results for the cell cycle and translation networks are better than those for the transcription network, according to our biological evaluation. This could indicate that rather than trying to fit a single model of modularity for an entire PPI network, it could be useful to analyze its parts separately. It will be interesting to test this hypothesis by testing our algorithms on many subnetworks, thus making it possible to obtain better statistics and a deeper understanding of the modularity of protein interaction networks.

3.5 Outlook

In this thesis we explored the notion of networks that cannot be decomposed fully into modules, but do contain dense, sparsely connected subsets. We developed two main perspectives: a random-walk-based approach, and a combinatorial graph-theory one. For each approach we introduced a modularity score and an algorithm for finding good assignments. It is natural then to ask whether these two approaches can somehow be compared. While there is no reason to think the two should give identical results, it is interesting to consider the types of networks that are highly modular both in the random-walk sense and in the combinatorial sense, and, in contrast, those that are modular according to only one approach. We could, for example, try to obtain a formal description of a class of networks that has a high I_k (3.9) score. Then we could extend the proof (Section 2.3) and formally determine the behavior of the I_m score on these networks.

In general, it is interesting to consider the modularity of more complicated network classes. We focused in this thesis on networks having a single transition region with a fixed density, and modules having identical density and size. By varying any of these attributes we can construct networks that are more complex: networks containing modules with differing sizes and densities, a transition region that is itself comprised of several connected components, or more sophisticated models than the Erdős–Rényi for generating these network components. We can test the behavior of our modularity scores analytically, or, as demonstrated in Section 2.4, experimentally.

With respect to finding good assignments, it would be beneficial to have new and improved algorithms for this task. This is true both for the combinatorial Split Cluster Editing and Monopolar Editing approaches, and for the MSM clustering algorithm. We saw in Section 3.4.3 that the ILP cannot resolve the yeast subnetworks we were interested in, and we had to resort to heuristics. Improved exact algorithms and heuristics can help us analyze these and larger, more complex networks.

The MSM clustering algorithm has some drawbacks when analyzing such complex networks (Section 3.1.3). It would be interesting to modify the various parts of the algorithms as we outlined in that section, replacing the greedy deterministic clustering algorithm by a different one and using different generators for the time-continuous Markov process underlying the algorithm. It is also possible to consider other random-walk based algorithms that can return a fuzzy assignment. We can, for example, employ heuristics to search the solution space of potential assignments, looking for one that would maximize the I_m score. Such improvements to the algorithm can make them applicable to more complicated networks, and in turn, to real-world networks.

We demonstrated some possible real-world applications of the model: Scoring the modularity of brain networks, and identifying protein complexes in protein

interaction networks. It would be interesting both to better understand the results we have obtained, and see if they extend to similar networks. We have observed that the modularity scores of typically-developed brain networks are distributed differently than the modularity scores of autistic brains (Section 2.5). It is promising to consider what this implies about brain organization, especially since it has often been assumed that autistic brains are less modular (and thus less organized in some sense) than typical brains. The role of the transition region in this case is especially intriguing for further research, as we could try to identify those nodes that play a role in the communication between the modules. Then we could see if the differences in distributions persist in other brain network datasets, both of autism and other conditions. For the protein interaction networks, it would be useful to test improved algorithms on a wider range of networks, corresponding to different species, or, as we've done here, cellular processes (Section 3.4.2).

Summary

Networks are commonly used to model complex real-world systems such as the cell or the brain. An important notion in analyzing networks is modularity. Usually, a network is considered modular if all its nodes can be partitioned into dense, connected subsets that are sparsely connected to one another. In this thesis, we extend the definition of modularity to cover networks that do contain dense modules that are sparsely connected to the rest of the network, but some nodes can belong to the region outside of modules, the so-called transition region. Accordingly, in contrast to full partitions, this work deals considers *assignments*: The association of every node to a module in the modular region or to the transition region.

The first task to be tackled in this thesis is the construction of a formal definition of modularity. After surveying some of the prominent existing definitions in the literature, we found that they cannot be directly applied to networks that are not fully-partitionable. We then developed an approach based on time-continuous random walks for analyzing networks. Here we built on a rich literature on the topic of spectral clustering, equating network modules with metastable sets of a random walk. This allows us to define a network as modular if there is a choice of modules that produces a coarse-grained Markov process whose dynamics well-approximate those of the original random walk on the network. We first demonstrate that this score matches our intuition of modularity on synthetic networks, increasing with the density of the modules and in the case where the modular region is large. Perturbation analysis is then used to formally prove the correct behavior of the score on two specifically constructed classes of networks.

Motivated both by the need to provide a good assignment as input to the modularity score and by the established usefulness of clustering methods, the second task to be tackled is that of finding good assignments. Continuing with the approach of Markov models, we first describe an algorithm based on a continuous random walk on the network. Next, modifications to several leading full-partition algorithms are described so that they output modules and transition regions rather than full partitions. The performance of all algorithms is then tested on a class of benchmark networks. Finally, we developed a combinatorial model of networks that are not fully partitionable as a union of split graphs, and posed the problem of finding modules and transition regions as a graph editing problem, proving its hardness and providing exact algorithms and heuristics.

The discussion of each of the two tasks is completed by an example of a biological application. Identifying good assignments gives us a way to detect protein complexes in protein interaction networks, a well-studied topic. The flexibility of assigning proteins to modules or to the transition region helps identify more complicated structures as complexes. The modularity score is used to analyze the brain networks of autistic and typically-developed children, where the distribution of modularity scores is different between the two types. In future work it will be interesting to further investigate the modularity of biological networks and draw conclusions about their organization.

Zusammenfassung

Netzwerke werden häufig benutzt, um komplexe reale Netzwerke wie die Zelle oder das Gehirn zu modellieren. Ein wichtiger Begriff zur Analyse von Netzwerken ist Modularität. Normalerweise wird ein Netzwerk als modular angesehen, wenn alle Knoten in dichte zusammenhängende Teilmengen partitioniert werden können, die untereinander nur schwach verbunden sind. In dieser Arbeit erweitern wir die Definition von Modularität, um auch Netzwerke zu erfassen, die dichte Module enthalten, die nur schwach mit dem Rest des Netzwerks verbunden sind, aber in denen einige Knoten zum Bereich außerhalb der Module gehören können, dem sogenannten Übergangsbereich. Dementsprechend betrachtet diese Arbeit im Gegensatz zu vollständigen Partitionen die sogenannten *Zuordnungen*: die Zuweisung von jedem Knoten zu einem Modul im Modulbereich oder zum Übergangsbereich.

Die erste Aufgabe, die in dieser Arbeit in Angriff genommen wird, ist die Konstruktion einer formalen Definition von Modularität. Nachdem wir eine Bestandsaufnahme einiger bekannten existierenden Definitionen in der Literatur erstellt haben, fanden wir heraus, dass sie nicht direkt für Netzwerke angewendet werden können, die nicht vollständig partitionierbar sind. Wir entwickelten dann einen Ansatz, der auf zeitkontinuierlichen Random Walks zur Analyse von Netzwerken basiert. Dabei haben wir auf einer reichhaltigen Literatur zum Thema spektrales Clustering aufgesetzt, bei der Netzwerkmodule mit metastabilen Mengen des Random Walks gleichgesetzt werden. Dies erlaubt uns, ein Netzwerk als modular zu definieren, wenn es eine Auswahl von Modulen gibt, die einen grobkörnigen Markow-Prozess erzeugt, dessen Dynamik die des ursprünglichen Random Walks auf dem Netzwerk gut approximiert. Wir zeigen zuerst, dass dieser Score auf künstlichen Netzwerken mit unserer Intuition übereinstimmt und mit der Dichte der Module und der Größe des Modulbereichs zunimmt. Perturbationsanalyse wird dann benutzt, um das korrekte Verhalten des Scores auf zwei speziell konstruierten Klassen von Netzwerken formal zu beweisen.

Motiviert sowohl durch die Notwendigkeit, eine gute Zuordnung als Eingabe für den Score zu finden, als auch durch die bekannte Nützlichkeit von Clustering-Methoden ist die zweite anzugehende Aufgabe die des Findens einer guten Zuordnung. Den Ansatz von Markow-Modellen fortsetzend, beschreiben wir einen Algorithmus, der auf einem zeitkontinuierlichen Random Walk auf dem Netzwerk beruht. Als nächstes werden Modifikationen für mehrere etablierte Algorithmen zum vollständigen Partitionieren beschrieben, die bewirken, dass sie Module und Übergangsbereich ausgeben anstelle von vollständigen Partitionen. Die Performance aller Algorithmen wird dann auf einer Klasse von Benchmark-Netzwerken getestet. Schließlich entwickeln wir ein kombinatorisches Modell von Netzwerken, die nicht vollständig partitionierbar sind, als Vereinigung von Split-Graphen, und formulieren das Problem, Module und den Übergangsbereich zu finden, als Grapheditierungsproblem; wir beweisen kombinatorische Schwierigkeit und zeigen exakte Algorithmen und Heuristiken.

Die Diskussion jeder der beiden Aufgaben wird vervollständigt durch ein Beispiel einer biologischen Anwendung. Die Identifikation guter Zuordnungen gibt uns eine Möglichkeit, Proteinkomplexe in Proteininteraktionsnetzwerken zu finden, ein gut untersuchtes Thema. Die Flexibilität, ein Protein einem Modul oder dem Übergangsbereich zuzuordnen, hilft dabei, kompliziertere Strukturen als Komplexe zu erkennen. Der Modularitäts-Score wird benutzt, um die Gehirnnetzwerke von autistischen und typisch entwickelten Kindern zu analysieren, wo die Verteilung von Modularitäts-Scores zwischen den beiden Gruppen unterschiedlich ist, was uns Vermutungen über den Grund für diesen Unterschied anstellen lässt. In weiteren Arbeiten wird es interessant sein, die Modularität biologischer Netzwerke weiter zu erforschen und Schlussfolgerungen über ihre Organisation zu ziehen.

Modularity in Biological Networks

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe.

Berlin, April 2014 Sharon Hüffner

Curriculum Vitae

For reasons of data protection, the curriculum vitae is not published in the online version.

Bibliography

Acquisti, Alessandro and Ralph Gross (2006). "Imagined communities: awareness, information sharing, and privacy on the Facebook". In: *Proceedings of the 6th International Workshop on Privacy Enhancing Technologies (PET '06)*. Volume 4258 in Lecture Notes in Computer Science, pages 36–58, Springer (cited on page 5).

Adamcsek, Balázs, Gergely Palla, Illés J. Farkas, Imre Derényi, and Tamás Vicsek (2006). "CFinder: locating cliques and overlapping modules in biological networks". *Bioinformatics* 22(8), pages 1021–1023 (cited on page 91).

Aldecoa, Rodrigo and Ignacio Marín (2011). "Deciphering network community structure by surprise". *PLOS ONE* 6(9), e24195 (cited on page 18).

— (2013). "Surprise maximization reveals the community structure of complex networks". *Scientific reports* 3, 1060 (cited on page 18).

Alon, Uri (2007). "Network motifs: theory and experimental approaches". *Nature Reviews Genetics* 8(6), pages 450–461 (cited on page 1).

Amoutzias, Grigoris and Yves van de Peer (2010). "Single-gene and whole-genome duplications and the evolution of protein–protein interaction networks". In: *Evolutionary Genomics and Systems Biology*, pages 413–429, Wiley (cited on pages 6, 90).

Arenas, Alex, Alberto Fernández, and Sergio Gómez (2008). "Analysis of the structure of complex networks at different resolution levels". *New Journal of Physics* 10(5), 053039 (cited on page 16).

Arnau, Vicente, Sergio Mars, and Ignacio Marín (2005). "Iterative cluster analysis of protein interaction data". *Bioinformatics* 21(3), pages 364–378 (cited on page 18).

Ashburner, Michael et al. (2000). "Gene Ontology: tool for the unification of biology". *Nature genetics* 25(1), pages 25–29 (cited on page 93).

Babu, Mohan et al. (2012). "Interaction landscape of membrane-protein complexes in Saccharomyces cerevisiae". *Nature* 489(7417), pages 585–589 (cited on pages 1, 90, 91).

Bader, Gary D. and Christopher W. V. Hogue (2003). "An automated method for finding molecular complexes in large protein interaction networks". *BMC Bioinformatics* 4(1), 2 (cited on page 91).

Bagrow, James P. (2012). "Communities and bottlenecks: trees and treelike networks have high modularity". *Physical Review E* 85(6), 066118 (cited on page 17).

Bansal, Nikhil, Avrim Blum, and Shuchi Chawla (2004). "Correlation clustering". *Machine Learning* 56(1–3), pages 89–113 (cited on pages 80, 81).

Barrat, Alain, Marc Barthélemy, Romualdo Pastor-Satorras, and Alessandro Vespignani (2004). "The architecture of complex weighted networks". *PNAS* 101(11), pages 3747–3752 (cited on page 13).

Bassett, Danielle S., Nicholas F. Wymbs, M. Puck Rombach, Mason A. Porter, Peter J. Mucha, and Scott T. Grafton (2013). "Task-based core-periphery organization of human brain dynamics". *PLOS Computational Biology* 9(9), e1003171 (cited on pages 6, 10).

Beichelt, Frank and L. Paul Fatti (2001). Stochastic processes and their applications. CRC Press (cited on page 24).

Blondel, Vincent D., Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre (2008). "Fast unfolding of communities in large networks". *Journal of Statistical Mechanics: Theory and Experiment*, P10008 (cited on pages 15, 56).

Böcker, Sebastian (2012). "A golden ratio parameterized algorithm for cluster editing". *Journal of Discrete Algorithms* 16, pages 79–89 (cited on page 82).

Böcker, Sebastian, Sebastian Briesemeister, and Gunnar W. Klau (2011). "Exact algorithms for cluster editing: evaluation and experiments". *Algorithmica* 60(2), pages 316–334 (cited on page 82).

Borgatti, Stephen P. and Martin G. Everett (1999). "Models of core/periphery structures". *Social Networks* 21(4), pages 375–395 (cited on page 79).

Borgatti, Stephen P., Ajay Mehra, Daniel J. Brass, and Giuseppe Labianca (2009). "Network analysis in the social sciences". *Science* 323(5916), pages 892–895 (cited on page 1).

Brandes, Ulrik, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner (2008). "On modularity clustering". *IEEE Transactions on Knowledge and Data Engineering* 20(2), pages 172–188 (cited on page 15).

Bremaud, Pierre (1999). Markov chains. Gibbs fields, Monte Carlo simulation, and queues. Volume 31 in Texts in Applied Mathematics, Springer (cited on pages 2, 23).

Brohee, Sylvain and Jacques van Helden (2006). "Evaluation of clustering algorithms for protein-protein interaction networks". *BMC Bioinformatics* 7(1), 488 (cited on page 91).

Bruckner, Sharon, Bastian Kayser, and Tim O. F. Conrad (2013). "Finding modules in networks with non-modular regions". In: *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA '13)*. Volume 7933 in Lecture Notes in Computer Science, pages 188–199, Springer (cited on page 61).

Brusco, Michael J. and Douglas Steinley (2009). "Integer programs for one- and two-mode blockmodeling based on prespecified image matrices for structural

and regular equivalence". *Journal of Mathematical Psychology* 53(6), pages 577–585 (cited on page 79).

Bullmore, Edward T. and Olaf Sporns (2012). "The economy of brain network organization". *Nature Reviews Neuroscience* 13(5), pages 336–349 (cited on page 6).

Charikar, Moses, Venkatesan Guruswami, and Anthony Wirth (2005). "Clustering with qualitative information". *Journal of Computer and System Science* 71(3), pages 360–383 (cited on page 82).

Chatr-aryamontri, Andrew et al. (2013). "The BioGRID interaction database: 2013 update". *Nucleic Acids Research* 41(D1), pages D816–D823 (cited on pages 90, 93).

Chavez, Mario, Miguel Valencia, Vincent Navarro, Vito Latora, and Jacques Martinerie (2010). "Functional modularity of background activities in normal and epileptic brain networks". *Physical Review Letters* 104(11), 118701 (cited on pages 6, 10, 54).

Chen, Bolin, Jinhong Shi, Shenggui Zhang, and Fang-Xiang Wu (2013). "Identifying protein complexes in protein–protein interaction networks by using clique seeds and graph entropy". *Proteomics* 13(2), pages 269–277 (cited on page 91).

Chen, Guangyu, Hong-Ying Zhang, Chunming Xie, Gang Chen, Zhi-Jun Zhang, Gao-Jun Teng, and Shi-Jiang Li (2013). "Modular reorganization of brain resting state networks and its independent validation in Alzheimer's disease patients". *Frontiers in Human Neuroscience* 7, 456 (cited on pages 6, 10, 54).

Chernyak, Zh. A. and A. A. Chernyak (1986). "About recognizing (α, β) classes of polar graphs". *Discrete Mathematics* 62(2), pages 133–138 (cited on page 80).

Churchley, Ross and Jing Huang (2014). "Solving partition problems with colour-bipartitions". *Graphs and Combinatorics* 30(2), pages 353–364 (cited on page 86).

Clauset, Aaron, Mark E. J. Newman, and Cristopher Moore (2004). "Finding community structure in very large networks". *Physical Review E* 70(6), 066111 (cited on page 14).

Csermely, Peter, András London, Ling-Yun Wu, and Brian Uzzi (2013). "Structure and dynamics of core/periphery networks". *Journal of Complex Networks* 1(2), pages 93–123 (cited on page 79).

Danon, Leon, Albert Díaz-Guilera, Jordi Duch, and Alex Arenas (2005). "Comparing community structure identification". *Journal of Statistical Mechanics: Theory and Experiment* 2005(09), P09008 (cited on page 15).

Dehne, Frank K. H. A., Michael A. Langston, Xuemei Luo, Sylvain Pitre, Peter Shaw, and Yun Zhang (2006). "The cluster editing problem: implementations and experiments". In: *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*. Volume 4169 in Lecture Notes in Computer Science, pages 13–24, Springer (cited on page 82).

Deuflhard, Peter, Wilhelm Huisinga, Alexander Fischer, and Christof Schütte (2000). "Identification of almost invariant aggregates in reversible nearly

uncoupled Markov chains". *Linear Algebra and its Applications* 315, pages 39–59 (cited on pages 25, 26, 30, 35, 36, 66).

Dezső, Zoltán, Zoltán N. Oltvai, and Albert-László Barabási (2003). "Bioinformatics analysis of experimentally determined protein complexes in the yeast Saccharomyces cerevisiae". *Genome Research* 13(11), pages 2450–2454 (cited on page 92).

Diestel, Reinhard (2005). Graph Theory. Volume 173 in Graduate Texts in Mathematics, Springer (cited on page 82).

Djurdjevac, Nataša (2006). "Methods for analyzing complex networks using random walker approaches". PhD thesis. Institut für Mathematik, Freie Universität Berlin (cited on pages 30, 31).

Djurdjevac, Nataša, Sharon Bruckner, Tim O. F. Conrad, and Christof Schütte (2011). "Random walks on complex modular networks". *Journal of Numerical Analysis, Industrial and Applied Mathematics* 6(1–2), pages 29–50 (cited on pages 2, 30, 31, 61).

Djurdjevac, Nataša, Marco Sarich, and Christof Schütte (2010). "On Markov state models for metastable processes". In: *Proceedings of the International Congress of Mathematicians (ICM 2010)*. Invited lecture. Chapter 188, pages 3105–3131, World Scientific (cited on page 31).

— (2012). "Estimating the eigenvalue error of Markov state models". *Multiscale Modeling & Simulation* 10(1), pages 61–81 (cited on pages 29–32).

Dongen, Stijn van (2000). *A cluster algorithm for graphs*. Technical report INS-R0010. National Research Institute for Mathematics and Computer Science in the Netherlands, pages 1–40 (cited on pages 23, 64).

Downey, Rodney G. and Michael R. Fellows (2013). Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer (cited on pages 85, 88).

Du, Zhidian, Lin Li, Chin-Fu Chen, Philip S. Yu, and James Z. Wang (2009). "G-SESAME: web tools for GO-term-based gene similarity analysis and knowledge discovery". *Nucleic Acids Research* 37(suppl. 2), W345–W349 (cited on page 94).

Duch, Jordi and Alex Arenas (2005). "Community detection in complex networks using extremal optimization". *Physical review E* 72(2), 027104 (cited on page 15).

E, Weinan, Tiejun Li, and Eric Vanden-Eijnden (2008). "Optimal partition and effective dynamics of complex networks". *PNAS* 105, pages 7907–7912 (cited on page 66).

Enright, Anton J., Stijn van Dongen, and Christos A. Ouzounis (2002). "An efficient algorithm for large-scale detection of protein families". *Nucleic Acids Research* 30(7), pages 1575–1584 (cited on pages 70, 74, 91).

Erdős, Paul and Alfréd Rényi (1960). "On the evolution of random graphs". *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* 5, pages 17–61 (cited on pages 13, 19, 48, 73).

Farrugia, Alastair (2004). "Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard". *The Electronic Journal of Combinatorics* 11(1), R46 (cited on page 86).

Fields, Stanley and Ok-kyu Song (1989). "A novel genetic system to detect protein–protein interactions". *Nature* 340, pages 245–246 (cited on page 90).

Fire, Michael, Gilad Katz, and Yuval Elovici (2012). "Strangers intrusion detection – detecting spammers and fake profiles in social networks based on topology anomalies". *Human Journal* 1(1), pages 26–39 (cited on page 7).

Fleck, Tobias, Andrea Kappes, and Dorothea Wagner (2014). "Graph clustering with surprise: complexity and exact solutions". In: *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '14)*. Volume 8327 in Lecture Notes in Computer Science, pages 223–234, Springer (cited on page 18).

Foldes, Stéphane and Peter L. Hammer (1977). "Split graphs". *Congressus Numerantium* 19, pages 311–315 (cited on page 83).

Fomin, Fedor V, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger (2013). "Tight bounds for parameterized complexity of cluster editing". In: *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS '13)*. Volume 20 in Leibniz International Proceedings in Informatics (LIPIcs), pages 32–43, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (cited on pages 82, 84).

Fortunato, Santo and Marc Barthélemy (2007). "Resolution limit in community detection". *PNAS* 104(1), pages 36–41 (cited on page 16).

Gamboa, Olga L., Enzo Tagliazucchi, Frederic von Wegner, Alina Jurcoane, Mathias Wahl, Helmut Laufs, and Ulf Ziemann (2013). "Working memory performance of early MS patients correlates inversely with modularity increases in resting state functional connectivity networks". *NeuroImage*. To appear (cited on page 54).

Gavin, Anne-Claude et al. (2006). "Proteome survey reveals modularity of the yeast cell machinery". *Nature* 440(7084), pages 631–636 (cited on pages 8, 90, 92).

Georgii, Elisabeth, Sabine Dietmann, Takeaki Uno, Philipp Pagel, and Koji Tsuda (2009). "Enumeration of condition-dependent dense modules in protein interaction networks". *Bioinformatics* 25(7), pages 933–940 (cited on page 91).

Good, Benjamin H., Yves-Alexandre de Montjoye, and Aaron Clauset (2010). "Performance of modularity maximization in practical contexts". *Physical Review E* 81(4), 046106 (cited on pages 16, 19).

Guimerà, Roger and Luís A. Nunes Amaral (2005). "Functional cartography of complex metabolic networks". *Nature* 433(7028), pages 895–900 (cited on page 15).

Guo, Jiong, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann (2011). "Editing graphs into disjoint unions of dense clusters". *Algorithmica* 61(4), pages 949–970 (cited on page 82).

Guo, Jiong, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann (2010). "A more relaxed model for graph-based data clustering: s-plex cluster editing". *SIAM Journal on Discrete Mathematics* 24(4), pages 1662–1683 (cited on pages 81, 82).

Hammer, Peter L. and Bruno Simeone (1981). "The splittance of a graph". *Combinatorica* 1(3), pages 275–284 (cited on page 81).

Heggernes, Pinar and Dieter Kratsch (2007). "Linear-time certifying recognition algorithms and forbidden induced subgraphs". *Nordic Journal of Computing* 14(1–2), pages 87–108 (cited on page 84).

Hoory, Shlomo, Nathan Linial, and Avi Wigderson (2006). "Expander graphs and their applications". *Bulletin of the American Mathematical Society* 43(4), pages 439–561 (cited on page 21).

Hüffner, Falk, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier (2013). "Partitioning biological networks into highly connected clusters with maximum edge coverage". *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. To appear. (cited on page 82).

Huisinga, Wilhelm and Bernd Schmidt (2006). "Metastability and dominant eigenvalues of transfer operators". In: *New Algorithms for Macromolecular Simulation*. Volume 49 in Lecture Notes in Computational Science and Engineering. Chapter 11, pages 167–182, Springer (cited on page 29).

Impagliazzo, Russell, Ramamohan Paturi, and Francis Zane (2001). "Which problems have strongly exponential complexity?" *Journal of Computer and System Sciences* 63(4), pages 512–530 (cited on page 84).

Ito, Takashi, Tomoko Chiba, Ritsuko Ozawa, Mikio Yoshida, Masahira Hattori, and Yoshiyuki Sakaki (2001). "A comprehensive two-hybrid analysis to explore the yeast protein interactome". *PNAS* 98(8), pages 4569–4574 (cited on page 90).

Jaimovich, Ariel (2010). "Understanding Protein-protein Interaction Networks". PhD thesis. Hebrew University, Israel (cited on page 90).

Jensen, Lars J. et al. (2009). "STRING 8—a global view on proteins and their functional interactions in 630 organisms". *Nucleic Acids Research* 37(suppl. 1), pages D412–D416 (cited on page 90).

Jeon, Jouhyun et al. (2011). "Network clustering revealed the systemic alterations of mitochondrial protein expression". *PLOS Computational Biology* 7(6), e1002093 (cited on page 6).

Kamburov, Atanas, Konstantin Pentchev, Hanna Galicka, Christoph K. Wierling, Hans Lehrach, and Ralf Herwig (2011). "ConsensusPathDB: toward a more complete picture of cell biology". *Nucleic Acids Research* 39(suppl. 1), pages D712–D717 (cited on page 90).

Kasprzyk, Arek (2011). "BioMart: driving a paradigm change in biological data management". *Database* 2011, bar049 (cited on page 93).

Kelley, Ryan and Trey Ideker (2005). "Systematic interpretation of genetic interactions using protein networks". *Nature Biotechnology* 23(5), pages 561–566 (cited on page 94).

Komusiewicz, Christian and Johannes Uhlmann (2012). "Cluster editing with locally bounded modifications". *Discrete Applied Mathematics* 160(15), pages 2259–2270 (cited on pages 82, 84).

Krause, Roland, Christian von Mering, Peer Bork, and Thomas Dandekar (2004). "Shared components of protein complexes—versatile building blocks or biochemical artefacts?" *Bioessays* 26(12), pages 1333–1343 (cited on page 91).

Křivánek, Mirko and Jaroslav Morávek (1986). "NP-hard problems in hierarchical-tree clustering". *Acta Informatica* 23(3), pages 311–323 (cited on page 84).

Kumpula, Jussi M., Jari Saramäki, Kimmo Kaski, and János Kertész (2007). "Limited resolution in complex network community detection with Potts model approach". *The European Physical Journal B* 56(1), pages 41–45 (cited on pages 16, 18).

Lee, James R., Shayan Oveis Gharan, and Luca Trevisan (2012). "Multi-way spectral partitioning and higher-order Cheeger inequalities". In: *Proceedings of the 44th Symposium on Theory of Computing (STOC '12)*, pages 1117–1130, ACM (cited on pages 21, 22).

Lei, Xiujuan, Shuang Wu, Liang Ge, and Aidong Zhang (2013). "Clustering and overlapping modules detection in PPI network based on IBFO". *Proteomics* 13(2), pages 278–290 (cited on page 91).

Leicht, Elizabeth A. and Mark E. J. Newman (2008). "Community structure in directed networks". *Physical Review Letters* 100(11), 118703 (cited on page 15).

Leung, Henry C. M., Qian Xiang, Siu-Ming Yiu, and Francis Y. L. Chin (2009). "Predicting protein complexes from PPI data: a core-attachment approach". *Journal of Computational Biology* 16(2), pages 133–144 (cited on page 92).

Lip, Sean Z. W. (2011). "A fast algorithm for the discrete core/periphery bipartitioning problem". arXiv:1102.5511 [physics.soc-ph] (cited on page 81).

Liu, Hong, Peng Zhang, and Daming Zhu (2012). "On editing graphs into 2-club clusters". In: *Proceedings of the Joint International Conference on Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM '12)*. Volume 7285 in Lecture Notes in Computer Science, pages 235–246, Springer (cited on page 82).

Luo, Feng, Bo Li, Xiu-Feng Wan, and Richard Scheuermann (2009). "Core and periphery structures in protein interaction networks". *BMC Bioinformatics* (suppl. 4), S8 (cited on page 92).

Luxburg, Ulrike von (2007). "A tutorial on spectral clustering". *Statistics and Computing* 17(4), pages 395–416 (cited on pages 15, 22, 25, 35).

Lynall, Mary-Ellen, Danielle S. Bassett, Robert Kerwin, Peter J. McKenna, Manfred Kitzbichler, Ulrich Muller, and Edward T. Bullmore (2010). "Functional connectivity and brain networks in schizophrenia". *The Journal of Neuroscience* 30(28), pages 9477–9487 (cited on pages 6, 10, 13).

Metzner, Philipp, Christof Schütte, and Eric Vanden-Eijnden (2009). "Transition path theory for Markov jump processes". *Multiscale Modeling & Simulation* 7(3), pages 1192–1219 (cited on pages 2, 30).

Meunier, David, Renaud Lambiotte, and Edward T. Bullmore (2010). "Modular and hierarchically modular organization of brain networks". *Frontiers in Neuroscience* 4, 200 (cited on pages 6, 10).

Mewes, Hans-Werner et al. (2002). "MIPS: a database for genomes and protein sequences". *Nucleic Acids Research* 30(1), pages 31–34 (cited on page 90).

Meyer, Carl D. (2000). Matrix Analysis and Applied Linear Algebra. SIAM (cited on page 25).

Michael, Judd H. (1997). "Labor dispute reconciliation in a forest products manufacturing facility". *Forest Products Journal* 47(11-12), pages 41–45 (cited on page 5).

Al-Mohy, Awad H. and Nicholas J. Higham (2011). "Computing the action of the matrix exponential, with an application to exponential integrators". *SIAM Journal on Scientific Computing* 33(2), pages 488–511 (cited on page 66).

Muñiz, Ana Salomé García and Carmen Ramos Carvajal (2006). "Core/periphery structure models: an alternative methodological proposal". *Social Networks* 28(4), pages 442–448 (cited on page 79).

Nascimento, Mariá C. V. and André C. P. L. F. de Carvalho (2011). "Spectral methods for graph clustering: a survey". *European Journal of Operational Research* 211(2), pages 221–231 (cited on page 68).

Nevries, Ragnar and VanBang Le (2011). "Recognizing polar planar graphs using new results for monopolarity". In: *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC '11)*. Volume 7074 in Lecture Notes in Computer Science, pages 120–129, Springer (cited on page 86).

Newman, Mark E. J. (2006). "Modularity and community structure in networks". *PNAS* 103(23), pages 8577–8582 (cited on pages 13–15).

Newman, Mark E. J. and Michelle Girvan (2004). "Finding and evaluating community structure in networks". *Physical Review E* 69(2), pages 413–421 (cited on pages 2, 10, 13).

Nicosia, Vincenzo, Giuseppe Mangioni, Vincenza Carchiolo, and Michele Malgeri (2009). "Extending the definition of modularity to directed graphs with overlapping communities". *Journal of Statistical Mechanics: Theory and Experiment* 2009(03), P03024 (cited on page 15).

Niedermeier, Rolf (2006). Invitation to Fixed-Parameter Algorithms. Volume 31 in Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press (cited on pages 85, 88).

Oveis Gharan, Shayan and Luca Trevisan (2014). "Partitioning into expanders". In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*, pages 1256–1266, SIAM (cited on pages 21, 22).

Pang, Chi Nam Ignatius, James Robert Krycer, Angela Lek, and Marc Ronald Wilkins (2008). "Are protein complexes made of cores, modules and attachments?" *Proteomics* 8(3), pages 425–434 (cited on pages 1, 92).

Porter, Mason A., Jukka-Pekka Onnela, and Peter J. Mucha (2009). "Communities in networks". *Notices of the American Mathematical Society* 56(9), pages 1082–1097 (cited on page 68).

Power, Jonathan D. et al. (2011). "Functional network organization of the human brain". *Neuron* 72(4), pages 665–678 (cited on page 55).

Price, Nicholas C. and Lewis Stevens (1999). Fundamentals of enzymology: The cell and molecular biology of catalytic proteins. Oxford University Press (cited on pages 6, 90).

Pu, Shuye, Jim Vlasblom, Andrew Emili, Jack Greenblatt, and Shoshana J. Wodak (2007). "Identifying functional modules in the physical interactome of Saccharomyces cerevisiae". *Proteomics* 7(6), pages 944–960 (cited on page 90).

Pu, Shuye, Jessica Wong, Brian Turner, Emerson Cho, and Shoshana J. Wodak (2009). "Up-to-date catalogues of yeast protein complexes". *Nucleic Acids Research* 37(3), pages 825–831 (cited on pages 90, 94).

Puck Rombach, Michaela, Mason A. Porter, James H. Fowler, and Peter J. Mucha (2014). "Core-periphery structure in networks". *SIAM Journal on Applied Mathematics* 74(1), pages 167–190 (cited on page 79).

Rand, William M. (1971). "Objective criteria for the evaluation of clustering methods". *Journal of the American Statistical Association* 66(336), pages 846–850 (cited on page 71).

Ravasz, Erzsébet, Anna Lisa Somera, Dale A. Mongru, Zoltán N. Oltvai, and Albert-László Barabási (2002). "Hierarchical organization of modularity in metabolic networks". *Science* 297(5586), pages 1551–1555 (cited on page 13).

Riel, Natal A. W. van (2006). "Dynamic modelling and analysis of biochemical networks: mechanism-based models and model-based experiments". *Briefings in Bioinformatics* 7(4), pages 364–374 (cited on page 6).

Rivas, Javier De Las and Celia Fontanillo (2010). "Protein–protein interactions essentials: key concepts to building and analyzing interactome networks". *PLOS Computational Biology* 6(6), e1000807 (cited on page 90).

Rubinov, Mikail and Olaf Sporns (2010). "Complex network measures of brain connectivity: uses and interpretations". *NeuroImage* 52(3), pages 1059–1069 (cited on pages 6, 55).

Rudie, Jeffrey D. et al. (2012). "Altered functional and structural brain network organization in autism". *NeuroImage: Clinical* 2, pages 79–94 (cited on pages 54–56, 58, 59).

Santo, Fortunato (2010). "Community detection in graphs". *Physics Reports* 486(3-5), pages 75–174 (cited on page 68).

Santos, Jorge M. and Mark Embrechts (2009). "On the use of the adjusted Rand index as a metric for evaluating supervised classification". In: *Proceedings of the 19th International Conference on Artificial Neural Networks (ICANN '09):*

Part II. Volume 5769 in Lecture Notes in Computer Science, pages 175–184, Springer (cited on page 72).

Saramäki, Jari, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertész (2007). "Generalizations of the clustering coefficient to weighted complex networks". *Physical Review E* 75(2), 027105 (cited on page 13).

Sarich, Marco, Nataša Djurdjevac, Sharon Bruckner, Tim O. F. Conrad, and Christof Schütte (2013). "Modularity revisited: a novel dynamics-based concept for decomposing complex networks". *Journal of Computational Dynamics*. To appear. (cited on pages 27, 29, 30, 49, 56, 57, 61–66, 74).

Sarich, Marco and Christof Schütte (2011). "Approximating selected non-dominant timescales by Markov state models". *Communications in Mathematical Sciences* 10(3), pages 1001–1013 (cited on pages 29, 31, 62).

Satuluri, Venu, Srinivasan Parthasarathy, and Duygu Ucar (2010). "Markov clustering of protein interaction networks with improved balance and scalability". In: *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology (BCB '10)*, pages 247–256, ACM (cited on page 71).

Schütte, Christof and Wilhelm Huisinga (2003). "Biomolecular conformations can be identified as metastable sets of molecular dynamics". In: *Handbook of Numerical Analysis*. Volume 10, pages 699–744, Elsevier (cited on page 29).

Schütte, Christof, Frank Noé, Jianfeng Lu, Marco Sarich, and Eric Vanden-Eijnden (2011). "Markov state models based on milestoning". *The Journal of Chemical Physics* 134, 204105 (cited on page 31).

Shamir, Ron, Roded Sharan, and Dekel Tsur (2004). "Cluster graph modification problems". *Discrete Applied Mathematics* 144(1–2), pages 173–182 (cited on pages 80, 81, 87).

Shi, Jianbo and Jitendra Malik (2000). "Normalized cuts and image segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8), pages 888–905 (cited on pages 20, 21).

Spellman, Paul T. et al. (1998). "Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization". *Molecular Biology of the Cell* 9(12), pages 3273–3297 (cited on page 94).

Spirin, Victor and Leonid A. Mirny (2003). "Protein complexes and functional modules in molecular networks". *PNAS* 100(21), pages 12123–12128 (cited on page 92).

Splettstoesser, Thomas (2006). *Atomic structure of bovine Arp2/3 complex*. URL: http://en.wikipedia.org/wiki/File:Arp2_3_complex.png (visited on 2014-03-24) (cited on page 7).

Srihari, Sriganesh and Hon Wai Leong (2012). "A survey of computational methods for protein complex prediction from protein interaction networks". *Journal of Computational Biology and Bioinformatics* 11(2), 1230002 (cited on pages 1, 90).

Srihari, Sriganesh, Kang Ning, and Hon Wai Leong (2010). "MCL-CAw: a refinement of MCL for detecting yeast complexes from weighted PPI networks by incorporating core-attachment structure". *BMC Bioinformatics* 11(1), 504 (cited on page 91).

Stevens, Alexander A., Sarah C. Tappon, Arun Garg, and Damien A. Fair (2012). "Functional brain network modularity captures inter-and intra-individual variation in working memory capacity". *PLOS ONE* 7(1), e30468 (cited on pages 6, 10).

Stewart, G. W. (1983). "On the structure of nearly uncoupled Markov chains". In: *Proceedings of the International Workshop on Computer Performance and Reliability*, pages 287–302, North-Holland (cited on page 35).

Vaggi, Federico et al. (2012). "Linkers of cell polarity and cell cycle regulation in the fission yeast protein interaction network". *PLOS Computational Biology* 8(10), e1002732 (cited on page 91).

Vijayabaskar, M. S. and Saraswathi Vishveshwara (2012). "Insights into the fold organization of TIM barrel from interaction energy based structure networks". *PLOS Computational Biology* 8(5), e1002505 (cited on page 6).

Wilson, Robert E., Samuel D. Gosling, and Lindsay T. Graham (2012). "A review of Facebook research in the social sciences". *Perspectives on Psychological Science* 7(3), pages 203–220 (cited on page 5).

Wittkop, Tobias, Jan Baumbach, Francisco P. Lobo, and Sven Rahmann (2007). "Large scale clustering of protein sequences with FORCE – a layout based heuristic for weighted cluster editing". *BMC Bioinformatics* 8, 396 (cited on page 82).

Wu, Min, Xiaoli Li, Chee-Keong Kwoh, and See-Kiong Ng (2009). "A core-attachment based method to detect protein complexes in PPI networks". *BMC Bioinformatics* 10(1), 169 (cited on page 92).

Xu, Xiaowei, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger (2007). "SCAN: a structural clustering algorithm for networks". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '07)*, pages 824–833, ACM (cited on page 70).

Yu, Liang, Lin Gao, and Kui Li (2010). "A method based on local density and random walks for complexes detection in protein interaction networks". *Journal of Bioinformatics and Computational Biology* 8(supp01), pages 47–62 (cited on page 92).