Chapter 2
Community Detection

It is clear that communities are frequently present in networks, and often have a very natural interpretation. They allow researchers to understand better the network by reducing its complexity. Our goal here is to investigate how such communities might be uncovered. We will first briefly explain the most common method for detecting communities, known as "modularity" in this chapter. We will then derive modularity from a more general framework from which some other methods can also be derived. Some of these methods have some problems, and we will discuss and analyse them in some detail, and provide some solutions in Chap. 3. For example, it remains a challenge to see how "granular" partitions should be: is it better to partition the network in many smaller communities, or in a few large communities? We address this choosing of the correct resolution in Chap. 4. If negative weights are present in network, modularity (and some variants) do not work well, and we will analyse some possible alternatives in Chap. 5. Finally, we will discuss some applications of community detection in Chap. 6.

There are two good overviews of community detection methods and algorithms. One is provided by Fortunato [16] and another by Porter et al. [39]. For a good introduction in traditional graph theory one can refer to Diestel [12], while Newman [36] provides a "complex networks" perspective. A traditional introduction into social network analysis from a sociological perspective is provided by Wasserman and Faust [50].

2.1 Modularity

Although clustering and graph partitioning have already quite a long history, they are usually not applied to (social) networks. Sociologists have constructed methods known as block modelling [13, 50], which are closer to "role[1]" detection [42] than to

[1] A role describes nodes that have similar connections to other roles, something closely related to the concept of "regular equivalence" [42, 50].

V. Traag, *Algorithms and Dynamical Models for Communities and Reputation in Social Networks*, Springer Theses, DOI: 10.1007/978-3-319-06391-1_2,

community detection. Computer scientists have been interested in graph partitioning for quite some time as well [36]. But the detection of groups in social networks really started to take off with a seminal paper by Girvan and Newman [18] in 2002. Especially their follow-up paper [37] which introduced a measure known as modularity attracted an enormous interest by a large group of researchers.

Originally, they implemented an algorithm based on the removal of edges which are part of many shortest paths [18]. The idea was that links that fall between communities are part of many such paths, because there are only few links that connect vertices from one community to another. Removing them should then disconnect the network at some point, in which case the communities should become visible. However, it was not clear at which point to stop removing edges. In order to determine this point, they introduced modularity [37]. This function should give some idea about the quality of a certain partition, and hence a clue as to when the algorithm should stop removing edges.

The idea is that communities should have relatively many edges within communities, and only little in between. Let A be an adjacency matrix of some undirected graph, so that $A_{ij} = A_{ji} = 1$ if there is an edge (i, j) and zero otherwise. Let us assume we have some fixed partition, and denote by e_{cd} the number of edges between communities c and d, corresponding to a tabulation as follows

From community \ To community	1	2	$\cdots$	q	$\sum$
1	e_{11}	e_{12}	$\cdots$	e_{1q}	K_1
2	e_{21}	e_{22}	$\cdots$	e_{2q}	K_2
$\vdots$			$\ddots$		$\vdots$
q	e_{q1}	e_{q2}	$\cdots$	e_{qq}	K_q
$\sum$	K_1	K_2	$\cdots$	K_q	$2m$

(2.1)

Then $\sum_{cd} e_{cd} = 2m$ equals twice the number of edges, since we are dealing with an undirected graph, and we count each edge twice in this manner. We are interested in $\sum_c e_{cc}/2m$ the fraction of edges within communities. Looking at this quantity, one already gets an idea of how good the partition is. However, it should be compared to how many edges we would expect to fall between two communities. This is usually done by simply taking marginals—row/column totals—which are $K_c := \sum_d e_{cd} = \sum_d e_{dc}$, the total number of edges linked to community c, as indicated in Eq. 2.1. Of course then also $\sum_c K_c = \sum_{cd} e_{cd} = 2m$. We thus arrive at the expected number of edges of $K_c K_d$ between communities c and d, which proportional to $2m$ then becomes $K_c K_d/(2m)^2$. Since we are only interested in having as many links as possible within a community we arrive at the function

$$\mathcal{Q} = \sum_c \left[\frac{e_{cc}}{2m} - \left(\frac{K_c}{2m} \right)^2 \right]. \tag{2.2}$$

The derivation provided here is quick and dirty, and we will see how a more rigorous derivation will also lead to modularity in the next section.

This measure seemed to do what was intended. Indeed when there are relatively many edges within a community, this quantity is relatively high, and approaches 1 for the most modular network possible. If a partition of a network is no better than random then $\mathcal{Q} \approx 0$. It was thought (incorrectly) that values above about 0.30 would be a sign of modular structure [37].

Although their original algorithm worked reasonably well, it was quite slow, and quickly faster algorithms appeared [8, 14, 35]. But their measure of modularity turned out to be an interesting one. Instead of using it simply to measure how well the network was partitioned, people began to optimize the measure itself [14, 21, 38]. However, it has some deficits and problems, which we will discuss in the next chapter. But first we will derive this measure of modularity in a more general framework, and go over some of the other possible methods for community detection.

2.2 Canonical Community Detection

In this chapter we will derive modularity in a more general setting, starting from first principles, similar to Reichardt and Bornholdt [41]. As stated, this more general framework will be used throughout the thesis, and forms the backbone of our analysis. Although not all methods can be represented in this way, it is a reasonably general framework, and we therefore refer to it as the canonical community detection framework.

Let us first start with some basic notation. Let $G = (V, E)$ be an undirected graph with nodes $V = \{1, \ldots, n\}$ and $E = \{(i, j) : i, j \in V\}$ the undirected edges of the graph G. Furthermore, we denote by A the adjacency matrix of G, such that $A_{ij} = 1$ if there is an (i, j) link, and $A_{ij} = 0$ otherwise. For an undirected graph the adjacency matrix $A = A^\top$ is symmetric where $A^\top$ denotes the transpose (i.e. $A^\top_{ji} = A_{ij}$). In addition, each link might have an associated weight $w_{ij} \in \mathbb{R}$, which we assume to be positive for the moment (we will consider the possibility of negative weights explicitly in Chap. 5). It might sometimes be useful to have a weighted adjacency matrix where $A_{ij} = w_{ij}$ when there is an (i, j) link. If we use the weighted adjacency matrix, this will be stated explicitly. The unweighted case then also corresponds to a weight of $w_{ij} = 1$. We denote the partition by $\sigma_i \in \{1, \ldots, q\}$ where each σ_i indicates the community to which node i belongs, so σ is the membership vector. Alternatively, it is sometimes useful to denote communities as sets of nodes. We will use $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ to denote the set of community sets, such that each set $C_c = \{i \in V \mid \sigma_i = c\}$ contains the nodes which belong to community c. Any partition of the graph is assumed to be non-overlapping and complete. Stated differently, every node belongs to a single community, in other words, for any valid partition it holds that $\bigcup_{c=1}^{q} C_c = V$ (all nodes are in a community) and $C_c \cap C_d = \emptyset$ for $c \neq d$ (no node is in more than one community). The size of a community (the number of nodes in a community) will usually be denoted by $n_c = |C_c|$. When

referring to "the partition" this might be either to σ or to $\mathcal{C}$, and should be clear from context. We will mostly focus on undirected and unweighted graphs, but most of these quantities can be straightforwardly extended to directed and weighted graphs.

Although the overall objective—detect communities—might be clear, what exactly constitutes a community is not undisputed. For example, one can take into account the number of triangles within a community, the size of the largest clique, or k-connectedness, and so forth. For example, traditional clustering works with notions of distance $d(i, j)$ between node i and j [51]. We shall start from a first principle basis that is due to Reichardt and Bornholdt [41]. The basic idea is to only specify the general framework, which can be made more specific, for example by counting the number of triangles or common neighbours. A commonly accepted idea of a community is that it should be a relatively dense subgraph that is relatively well separated from the rest of the graph. This means there should be relatively:

1. many present links within communities;
2. few absent links within communities;
3. few present links between communities; and
4. many absent links between communities.

Taking these assumptions, we reward present links (a_{ij}) and punish absent links (b_{ij}) within communities, while we punish present links (c_{ij}) and reward absent links (d_{ij}) between communities. Summarizing, we have the following weights:

	$A_{ij} = 1$	$A_{ij} = 0$
$\delta(\sigma_i, \sigma_j) = 1$	a_{ij}	$-b_{ij}$
$\delta(\sigma_i, \sigma_j) = 0$	$-c_{ij}$	d_{ij}

where all weights $a_{ij}, b_{ij}, c_{ij}, d_{ij} \geq 0$ remain to be specified and δ is the Kronecker delta

$$\delta(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \tag{2.3}$$

so that $\delta(\sigma_i, \sigma_j) = 1$ if $\sigma_i = \sigma_j$ both i and j are in the same community, and 0 otherwise. We then denote by $\mathcal{H}$ the objective function

$$\begin{aligned} \mathcal{H}(\sigma) = -\sum_{ij} \Big[a_{ij} A_{ij} - b_{ij}(1 - A_{ij}) \Big] \delta(\sigma_i, \sigma_j) \\ + \Big[-c_{ij} A_{ij} + d_{ij}(1 - A_{ij}) \Big] (1 - \delta(\sigma_i, \sigma_j)). \end{aligned}$$

The minus sign is only a matter of convention, and in this case we would like to minimize this function. The optimization problem is then

$$\min_{\sigma} \mathcal{H}(\sigma), \tag{2.4a}$$

$$\text{s.t. } \sigma \in \{1, \ldots, q\}^n. \tag{2.4b}$$

We will refer to $\mathcal{H}(\sigma)$ as the cost of a partition σ, and so the optimal partition has minimal cost. Now if we suppose that links within communities should be equally rewarded/punished as links between communities, i.e. $a_{ij} = c_{ij}$ and $b_{ij} = d_{ij}$, we can simplify to

$$\mathcal{H}(\sigma) = -\sum_{ij} a_{ij} A_{ij} (2\delta(\sigma_i, \sigma_j) - 1) - b_{ij}(1 - A_{ij})(2\delta(\sigma_i, \sigma_j) - 1).$$

Since we are looking for the minimum of $\mathcal{H}(\sigma)$ we can remove factors that do not depend on σ, i.e. not depending on $\delta(\sigma_i, \sigma_j)$. Furthermore, any multiplication with a constant leaves the minimum unchanged. Using these observations, we can simplify to

$$\mathcal{H}(\sigma) = -\sum_{ij} (a_{ij} A_{ij} - b_{ij}(1 - A_{ij}))\delta(\sigma_i, \sigma_j). \tag{2.5}$$

This is the objective function we will analyse in this thesis, and forms the core of our enquiry. The weights a_{ij} and b_{ij} remain to be specified, but are assumed to be non-negative $a_{ij}, b_{ij} \geq 0$.

Irrespective of the specific weights chosen, any community should be connected. To show this, assume on the contrary there is a community C which is disconnected, so that for some partition $C = S \cup S'$, with $S \cap S' = \emptyset$, there are no edges from S to S'. In that case, if we split the community into S and S', we decrease the cost function assuming there is at least one $b_{ij} > 0$, so that it cannot be optimal.

Different choices for the weights a_{ij} and b_{ij} lead to different methods for community detection. For example, we could imagine taking into account the number of common neighbours between i and j for absent links, so that $b_{ij} = |N(i) \cap N(j)|$, or the number of independent paths between i and j, similar to the original algorithm of Girvan and Newman [18]. Numerous choices could be made, and we will review some of the possibilities (for an overview, refer to Table 2.1).

2.2.1 Reichardt and Bornholdt

One choice consists of comparing the original network to a randomized network, a random null model, as considered by Reichardt and Bornholdt [41]. Let us assume the probability for a link is p_{ij}, which we will specify later. The weight of a missing link is $b_{ij} = \gamma_{\text{RB}} p_{ij}$, while the weight of a present link is $a_{ij} = w_{ij} - b_{ij}$, where w_{ij} is the weight of the (i, j) link, or $w_{ij} = 1$ if the graph is not weighed and γ_{RB} a parameter used to weigh the importance of the randomized network. Summarizing, the weights are

$$a_{ij} = w_{ij} - \gamma_{\text{RB}} p_{ij}, \tag{2.6a}$$
$$b_{ij} = \gamma_{\text{RB}} p_{ij}. \tag{2.6b}$$

In other words, whenever a link has more weight than expected in the randomized network, we reward that link if it is within a community. Including a missing link in a community would be punished slightly if the expected weight of a link is low. Working out this choice leads to

$$\begin{aligned}\mathcal{H}_{\text{RB}} &= -\sum_{ij} \left[(w_{ij} - \gamma_{\text{RB}} p_{ij}) A_{ij} - \gamma_{\text{RB}} p_{ij} (1 - A_{ij}) \right] \delta(\sigma_i, \sigma_j) \\ &= -\sum_{ij} \left[w_{ij} A_{ij} - \gamma_{\text{RB}} p_{ij} \right] \delta(\sigma_i, \sigma_j) \end{aligned} \tag{2.7}$$

In the following we will assume that the graph is unweighted and that $w_{ij} = 1$. We can rewrite Eq. (2.7) slightly to gain some additional insight. We gather the terms per community, and arrive at

$$\begin{aligned}\mathcal{H}_{rb} &= -\sum_{ij} (A_{ij} - \gamma_{\text{RB}} p_{ij}) \delta(\sigma_i, \sigma_j) \\ &= -\sum_{c} \sum_{ij} (A_{ij} - \gamma_{\text{RB}} p_{ij}) \delta(\sigma_i, c) \delta(\sigma_j, c).\end{aligned}$$

So if we write

$$e_c = \sum_{ij} A_{ij} \delta(\sigma_i, c) \delta(\sigma_j, c)$$

for the number[2] of edges in community c and

$$\langle e_c \rangle_{p_{ij}} = \sum_{ij} p_{ij} \delta(\sigma_i, c) \delta(\sigma_j, c)$$

for the expected number of edges in community c, we can rewrite Eq. (2.7) as

$$\mathcal{H}_{rb} = -\sum_{c} \left[e_c - \gamma_{\text{RB}} \langle e_c \rangle_{p_{ij}} \right].$$

In general, the average of some quantity will usually be denoted by $\langle \cdot \rangle$. In other words, this objective function considers the difference between the actual number of edges within a community and the expected number of edges within a community given a random null model. Hence, there are two ways for improving this function: by having more edges within a community, or by having less expected edges within

[2] Technically twice the number of edges in community c for undirected graphs.

a community. The expected edges weigh more heavily with higher γ_{RB}, so that it effectively constrains the community sizes. But we will get back to this later on.

Various random null models can be chosen to specify p_{ij}. One possibility is to take a simple Erdös-Renyí (ER) graph [5] where each link[3] appears with the same probability $p = m/n^2$, where $m = |E|$ the number of edges and n the number of nodes. We then set

$$p_{ij} = p = \frac{m}{n^2}.$$

The expected number of edges within a community is then simply

$$\langle e_c \rangle_p = p n_c^2$$

where n_c is the number of nodes of community c. In this case the density within a community is expected to be about the same as the density of the graph in general. The objective function as a sum over communities then simplifies to

$$\mathcal{H}_{\text{RB}} = \sum_c \left[e_c - \gamma_{\text{RB}} p n_c^2 \right].$$

However, an ER graph is not realistic in the sense that the degree $k_i = \sum_j A_{ij}$ of a node deviates from what is empirically expected. An ER graph has a Poissonian degree distribution so that

$$\Pr(k) = \frac{\langle k \rangle^k e^{-\langle k \rangle}}{k!},$$

while in reality the degree distribution is highly skewed and heavy tailed, and follows more a power law [36]

$$\Pr(k) \sim k^{-\tau}.$$

So, another common null model is the configuration model, which takes into account the degree. A simple way to construct a randomized network with the same degrees is to cut all links in half, so that each link has k_i stubs (one half of a link), and to connect all the stubs randomly. We then arrive at the expected number of links between i and j of

$$p_{ij} = \frac{k_i k_j}{2m}. \tag{2.8}$$

The derivation of the quantity is as follows. We have k_i ways to choose a stub from node i, since it has k_i stubs to connect. Similarly, we have k_j ways for choosing to connect to node j. Finally, we choose from $2m$ stubs (twice for each link). The expected number of links within a community is then

[3] We here include the possibility of self-loops.

$$\langle e_c \rangle_{\text{conf}} = \frac{K_c^2}{2m}, \tag{2.9}$$

where $K_c := \sum_i k_i \delta(\sigma_i, c)$ is the sum of the degrees of the nodes in community c. If the total degree is relatively high, we expect more edges to fall within the community. Notice that this no longer corresponds to the density of a community. The objective function becomes

$$\mathcal{H}_{\text{RB}} = \sum_c \left[e_c - \gamma_{\text{RB}} \frac{K_c^2}{2m} \right]. \tag{2.10}$$

The classical modularity can then be derived by taking $\gamma_{\text{RB}} = 1$, using the configuration model, and normalize by $\frac{1}{2m}$ and inverse the sign to arrive at

$$\mathcal{Q} = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(\sigma_i, \sigma_j). \tag{2.11}$$

or as a sum over communities, which is sometimes easier to use,

$$\mathcal{Q} = \sum_c \left[\frac{e_c}{2m} - \left(\frac{K_c}{2m} \right)^2 \right], \tag{2.12}$$

and we retrieve the definition provided in Eq. (2.2).

2.2.2 *Arenas, Fernández and Gómez*

A particular problem of modularity (and the RB model in general) is the so-called resolution limit, which we will analyse more in-depth later on (see Chap. 3). The basic problem in the resolution limit is that communities are merged together while they actually shouldn't. This problem can be addressed to a certain extent by the resolution parameter γ_{RB} in the RB model, but other solutions have been proposed. One noteworthy solution by Arenas et al. [2] (AFG) consists of adding self-loops to nodes so as to prevent these nodes from being merged. In other words, they use almost the same weights as RB, but then adapted for the added self-loops of strength γ_{AFG}. This idea translates into the weights

$$a_{ij} = w_{ij} - b_{ij}, \tag{2.13a}$$

$$b_{ij} = p_{ij}(\gamma_{\text{AFG}}) - \gamma_{\text{AFG}} \delta_{ij}, \tag{2.13b}$$

where $\delta_{ij} = \delta(i, j) = 1$ if $i = j$ and zero otherwise. The authors use the classical configuration model for the null-model, and use

$$p_{ij}(\gamma_{\text{AFG}}) = \frac{(k_i + \gamma_{\text{AFG}})(k_i + \gamma_{\text{AFG}})}{2m + n\gamma_{\text{AFG}}}. \tag{2.14}$$

Their model then becomes (up to multiplicative scaling)

$$\mathcal{H}_{\text{AFG}}(\sigma) = -\sum_{ij} \left(A_{ij}w_{ij} + \gamma_{\text{AFG}}\delta_{ij} - p_{ij}(\gamma_{\text{AFG}})\right)\delta(\sigma_i, \sigma_j) \tag{2.15}$$

which is simply Eq. (2.7) with self-loops added. The benefit of this method is that it leaves unchanged properties that depend on the eigenvectors or on the difference of the eigenvalues. In order to see that, observe that we could also have transformed the original matrix A to $A' = A + \gamma_{\text{AFG}} I_n$ where I_n is the $n \times n$ identity matrix, i.e. $I_n = \text{diag}(1, \ldots, 1)$. Now suppose that λ is an eigenvalue and v the corresponding eigenvector of A (i.e. $Av = \lambda v$), then also $A'v = Av + \gamma_{\text{AFG}} I_n v = (\lambda + \gamma_{\text{AFG}})v$ so that v is an eigenvector of A' and $\lambda + \gamma_{\text{AFG}}$ an eigenvalue of A'. Although the same idea could be investigated using the ER null model this has not been considered. Notice that the AFG model is indeed different from the RB null-model and that the two are only equal for $\gamma_{\text{AFG}} = 0$ and $\gamma_{\text{RB}} = 1$ in general.

2.2.3 Ronhovde and Nussinov

Ronhovde and Nussinov [43] (RN) do not include any null model, in order to avoid issues with the resolution limit, and in general set

$$a_{ij} = w_{ij}, \tag{2.16a}$$
$$b_{ij} = \gamma_{\text{RN}}, \tag{2.16b}$$

(although for specific networks, such as with negative weights, they allow some minor changes). Working this out we obtain

$$\mathcal{H}_{\text{RN}}(\sigma) = -\sum_{ij} (A_{ij}(w_{ij} + \gamma_{\text{RN}}) - \gamma_{\text{RN}})\delta(\sigma_i, \sigma_j). \tag{2.17}$$

Notice that for unweighted graphs (i.e. $w_{ij} = 1$) up to rescaling this is equal to

$$\mathcal{H}_{\text{RN}}(\sigma) = -\sum_{ij} \left(A_{ij} - \frac{\gamma_{\text{RN}}}{1 + \gamma_{\text{RN}}}\right)\delta(\sigma_i, \sigma_j). \tag{2.18}$$

If we compare this to the RB model with an ER null model, the RN model is equal to the RB model if

$$\gamma_{\text{RN}} = \frac{1 - \gamma_{\text{RB}} p}{\gamma_{\text{RB}}}.$$

For weighted graphs, the models are not necessarily the same however.

2.2.4 Constant Potts Model

A formulation that also has no null model, similar to Ronhovde and Nussinov [43], but which resembles more closely the RB model is provided by

$$a_{ij} = w_{ij} - b_{ij}, \tag{2.19a}$$
$$b_{ij} = \gamma_{\text{CPM}}, \tag{2.19b}$$

which results in

$$\mathcal{H}_{\text{CPM}} = -\sum_{ij}(A_{ij}w_{ij} - \gamma_{\text{CPM}})\delta(\sigma_i, \sigma_j). \tag{2.20}$$

We call this the Constant Potts Model because it only compares the network to a constant parameter γ_{CPM} [49].

As can be expected, this model is rather similar to the RN model and the RB model. The RB and RN model are equivalent if $\gamma_{\text{CPM}} = \gamma_{\text{RB}}\, p$ and the ER null model is used. The RN model is only equal to the CPM model for unweighted graphs, in which case we have $\gamma_{\text{CPM}} = \frac{\gamma_{\text{RN}}}{1+\gamma_{\text{RN}}}$.

2.2.5 Label Propagation

Finally, the label propagation (LP) method [40] can be shown to be equivalent to the Potts model [48]

$$a_{ij} = w_{ij}, \tag{2.21a}$$
$$b_{ij} = 0. \tag{2.21b}$$

which results in the trivially optimized

$$\mathcal{H}_{LP} = -\sum_{ij} A_{ij}w_{ij}\delta(\sigma_i, \sigma_j) \tag{2.22}$$

This model is equivalent to the RB model, the RN model and CPM as long as $\gamma_{\text{RB}} = \gamma_{\text{RN}} = \gamma_{\text{CPM}} = 0$. This is the least interesting formulation, since there is only one global optimum, namely all nodes belong to a single community, which is trivial. However, the local minima could be of some interest. Furthermore, these local minima can be relatively quickly found, rendering the complexity of the associated algorithm essentially linear [40].

Table 2.1 Overview of different methods

Method	a_{ij}	b_{ij}	Objective function
Modularity (p. 11)	$w_{ij} - \frac{k_i k_j}{2m}$	$\frac{k_i k_j}{2m}$	$\frac{1}{2m}\sum_{ij}\left(A_{ij}w_{ij} - \frac{k_i k_j}{2m}\right)$
RB (p. 15)	$w_{ij} - \gamma_{\text{RB}} p_{ij}$	$\gamma_{\text{RB}} p_{ij}$	$-\sum_{ij}(A_{ij}w_{ij} - \gamma_{\text{RB}} p_{ij})\delta(\sigma_i, \sigma_j)$
AFG (p. 18)	$w_{ij} - b_{ij}$	$p_{ij}(\gamma_{\text{AFG}}) - \gamma_{\text{AFG}}\delta_{ij}$	$-\sum_{ij}(A_{ij}w_{ij} + \gamma_{\text{AFG}}\delta_{ij} - p_{ij}(\gamma_{\text{AFG}}))\delta(\sigma_i, \sigma_j)$
RN (p. 19)	w_{ij}	γ_{RN}	$-\sum_{ij}(A_{ij}(w_{ij} + \gamma_{\text{RN}}) - \gamma_{\text{RN}})\delta(\sigma_i, \sigma_j)$
CPM (p. 20)	$w_{ij} - \gamma_{\text{CPM}}$	γ_{CPM}	$-\sum_{ij}(A_{ij}w_{ij} - \gamma_{\text{CPM}})\delta(\sigma_i, \sigma_j)$
LP (p. 20)	w_{ij}	0	$-\sum_{ij} A_{ij}w_{ij}\delta(\sigma_i, \sigma_j)$

2.2.6 Random Walker

There are also some other derivations of modularity (and some of the others models) in terms of a random walk on a graph, by Delvenne et al. [11]. They focus on the time it takes for a random walker to escape from a community. Since a random walker should be trapped within a community for a considerable time, if we try to maximize how long the walker will remain in the same community, we should find communities.

Let us take a look at how we can represent such a random walk on a graph. Suppose we start our walk with a certain probability $\pi(0)$ in some node, so that $\pi_i(0)$ gives the probability we start in node i. The random walker simply follows each link with uniform probability. So, from a node i, it follows the link (i, j) with probability $1/k_i$. If we define $M = (D^{-1}A)^\top$ where $D = \text{diag}(k_1, k_2, \ldots, k_n)$ has the degrees on the diagonal, then M_{ij} gives the transition probabilities for moving from node i to j. The probability we are in a certain node after a single step is then $\pi(t+1) = M\pi(t)$, and so $\pi(t) = M^t\pi(0)$. If we assume the network to be (strongly) connected and aperiodic, this matrix is primitive, and according to the Perron-Frobenius theorem, in the limit

$$\lim_t \pi(t) = \pi = M\pi \tag{2.23}$$

this probability becomes stationary, and π is the dominant eigenvector of M. So, after a sufficient long time, each node will be visited with probability π_i.

Now let us give each node some label σ_i. Suppose the random walker records the labels σ_i of nodes visited in a random variable X_t, so that if the random walker was in node i after t steps, then $X_t = \sigma_i$. As stated, we would like to know whether the random walker remains in the same community for a long time. Suppose that the label σ_i of a node indicates the community. If a random walker stays within the same

community, the random variable X_t is likely to be the same. This can be measured through the autocovariance between X_τ and $X_{\tau+t}$ with $t > 0$, which is defined as

$$\text{Cov}(X_\tau, X_{\tau+t}) = \mathbb{E}(X_\tau X_{\tau+t}) + \mathbb{E}(X_t)^2.$$

The expected value of X_t can be easily calculated, if we assume the random walk to become stationary. In that case, $\mathbb{E}(X_t) = \sum_i \sigma_i \pi_i = \sigma^\top \pi = \pi^\top \sigma$, and so

$$\mathbb{E}(X_t)^2 = (\sigma^\top \pi)(\pi^\top \sigma) = \sigma^\top \pi \pi^\top \sigma.$$

To calculate $\mathbb{E}(X_\tau X_{\tau+t})$ at stationarity we obtain that

$$\mathbb{E}(X_t X_{t+\tau}) = \sum_i \sigma_i \pi_i (M^t \sigma)_i = \sigma^\top \Pi M^t \sigma,$$

where $\Pi = \text{diag}(\pi)$. We encode $\sigma = S\alpha$ where $\alpha = (1, \ldots, q)$ and S is the $n \times q$ community matrix, such that $S_{ic} = 1$ if $\sigma_i = c$ node i is in community c and 0 otherwise (see also Sect. 2.3.4). We can then write the covariance as

$$\text{Cov}(X_\tau, X_{\tau+t}) = \alpha^\top R(t) \alpha$$

where

$$R(S, t) = S^\top (\Pi M^t - \pi^\top \pi) S$$

is the so called stability matrix. Each element $R(S, t)_{cd}$ denotes the probability to start in community c and go to community d after t steps minus the probability two random walkers are in c and d. Since we are interested in maximizing the time spent inside a community, we would like to maximize $R(S, t)_{cc}$. In other words, we would like to find $\max_S \text{Tr} R(S, t)$ where $\text{Tr} X = \sum_i X_{ii}$ is the trace of some matrix X. However, we should remain within the community for all time up to t. So we define the stability of a partition S at time t as

$$r(S, t) = \min_{\tau < t} \text{Tr} R(S, \tau).$$

and we would like to maximize this $r(S, t)$ for some t. In general, we can write

$$\text{Tr} R(S, t) = \sum_c S_c^\top (\Pi M^t - \pi^t \pi) S_c = \sum_{ij} (\Pi (M^t)_{ij} - \pi_i \pi_j) \delta(\sigma_i, \sigma_j).$$

If the random walk is undirected, we have that $\pi_i = \frac{k_i}{2m}$. Now suppose we look at only a single step, or $t = 1$, so that we obtain that

$$\mathrm{Tr}\, R(S,1) = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(\sigma_i,\sigma_j).$$

Hence, we recover exactly modularity for time $t = 1$ on undirected networks. For directed networks this quantity differs from the null model originally proposed for directed networks [32]. Approximating this equation around $t = 1$, a different interpretation of the resolution parameter for the RB model is obtained, namely that $\gamma_{\mathrm{RB}} \approx 1/t$. However, this only holds approximately. Furthermore, some related type of (continuous time) random walk gives an alternative derivation for the RB model with an ER null model [28].

2.2.7 Infomap

A quite successful method that unfortunately doesn't fit within this framework is Infomap [44, 45]. We include a brief description of this method since it is one of the best performing methods outside of this framework, although certainly not the only one (see [1, 31]). It is based on ideas of information theory, which we will briefly explain. Information theory concerns itself with the representation of information, and naturally involves also the compression of information. For example, if we have a very long piece of text which reiterates "Help! Help! Help! Help! Help!", it would be more efficient to simply write "Help! (5×)". In a similar fashion, one can imagine being able to compress other information, which these days is often used when creating `.zip` files, but also in videos (`.mp4`), images (`.jpg`) or music (`.mp3`).

Infomap focuses on trying to compress the list of nodes visited by a random walker on a graph. We record all the nodes a walker has visited, for example "1, 5, 3, 2", meaning that the walker first visited node 1 then 5, then 3 and finally 2, similar to the random variable X_t in the previous section. If we continue this walk for a very long time, we expect him to spend a reasonable amount of time in the same community. We may use this to represent the list of all nodes the walker has visited in a more efficient way. Hence, the idea of a random walker is similar to the previous section, although the objective is different: previously the focus was on staying in the same community as long as possible, while here the focus is on having a description of the random walk which is as short as possible.

Let us first briefly review the basics of information theory.

Information Theory

Information theory mostly deals with how information can be represented and quantified [9, 33]. The information value of a certain event is logarithmically inverse to the probability of it occurring. In other words, suppose that X is a random variable and that $\Pr(X = x) = p(x)$, then the information associated with event x is

$$I(x) = \log\frac{1}{p(x)} = -\log p(x). \tag{2.24}$$

This has two nice properties: (1) the information associated with two independent identically distributed events x is then $2I(x)$ so contains twice the information; and (2) if x is sure to happen, so when $p(x) = 1$, it contains no information and $I(x) = 0$. The maximum information about a certain event is then when $p(x) \to 0$, which makes sense. After all, if x happens almost never, it provides much information when it actually does happen.

Given a certain distribution $p(x)$ we can also ask what is the expected information associated with the random variable X. This measure is also known as the entropy, and can be written as

$$H(X) = \mathbb{E}(I(X)) = -\sum_x p(x) \log p(x). \tag{2.25}$$

If we look at the probability of X given Y, or $\Pr(X = x \mid Y = y) = p(x \mid y)$, the information content associated to x given y is then $I(x \mid y) = -\log p(x \mid y)$. The entropy of $H(X \mid Y = y)$ is then

$$H(X \mid Y = y) = -\sum_x p(x \mid y) \log p(x \mid y),$$

hence the conditional entropy is

$$\begin{aligned} H(X \mid Y) &= \mathbb{E}(H(X \mid Y = y)) = -\sum_y p(y) \sum_x p(x \mid y) \log p(x \mid y) \\ &= -\sum_{xy} p(x, y) \log \frac{p(x, y)}{p(y)}. \end{aligned} \tag{2.26}$$

Notice that if Y and X are independent random variables, then $H(X \mid Y) = H(X)$, and otherwise $H(X \mid Y) \leq H(X)$. In other words, conditioning always decreases the entropy. Furthermore, if X is completely determined by Y then $H(X \mid Y) = 0$, which makes sense since knowing Y we also know X. Similarly, the joint entropy can be defined as

$$H(X, Y) = -\sum_{xy} p(x, y) \log p(x, y), \tag{2.27}$$

and hence

$$\begin{aligned} H(X, Y) = H(Y, X) &= H(Y \mid X) + H(X), \\ &= H(X \mid Y) + H(Y). \end{aligned}$$

If X and Y are independent random variables then $H(X \mid Y) = H(X)$, and so $H(X, Y) = H(X) + H(Y)$. Since $H(X \mid Y) \geq 0$, we have $H(X, Y) \geq H(X)$ and $H(X, Y) \geq H(Y)$, and so the joint entropy is always larger than the entropy of a single random variable.

Now suppose we wish to represent a series of random variables, which are independently identically distributed (iid) with distribution $p(x)$. In this context it is common to talk about symbols and a code to represent that symbol. For example, suppose that our distribution gives the symbol a with probability p_a and b with probability p_b, and likewise p_c and p_d. We will usually represent codes of symbols in binary code, and so we can represent the symbols by using the following code.

Symbol	Code
a	00
b	01
c	10
d	11

Here the code length $b_i = 2$ for all codes i. So, the code for the sequence "$adba$" is then "00110100". However, if we know that some symbols occur more often then others, we might want to assign shorter codes to symbols that are more often used. For example if the symbols occur with probabilities $p_a = 0.6$, $p_b = 0.2$ and $p_c = p_d = 0.1$, we could use the following codes.

Symbol	Code
a	0
b	10
c	110
d	111

Notice that the code for a is shorter $b_a = 1$, but the codes for c and d are longer, $b_c = b_d = 3$. The code for the same sequence as before is now "0111100", which has a total length of 7 bits, while the original code used 8 bits. Notice that we can identify the codes unambiguously, because no code appears in the beginning of another code, a property known as prefix-free. In general, if we look at the expected code length per symbol, this is

$$\sum_i p_i b_i = 0.6 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 = 1.6$$

using the adapted code, while for the original codes this was $\sum_i p_i b_i = 2$. So, we improved the representation of this sequence by changing the codes. The idea is now that the number of possibilities for a codeword of a length b_i should be inversely proportional to its probability, so that $2^{b_i} = 1/p_i$, or the number of bits[4] $b_i = -\log p_i$. Rare symbols then get long codes, and often occurring symbols shorter codes. The expected code length per symbol is then

[4] This could be expressed in a different base as well. Since the base only changes the properties up to a multiplicative constant, we ignore this and simply take the natural logarithm.

$$\sum_i p_i b_i = -\sum_i p_i \log p_i = H(X).$$

The amazing thing is that this is also the optimal code length per symbol. In other words, we cannot represent the information in a shorter code per symbol than the entropy. This is known as the famous Shannon source-coding theorem [9]. The actual codes attaining this bound are known as Huffman codes. For our purposes here, we do not need this machinery, and we will not discuss it further.

Compressing Random Walks

How can we use compression to find communities? As stated, we expected a random walker to remain in the same community for a substantial amount of time. The ingenious idea is then that as long as we remain in the same community we can use shorter codes for nodes in the same community. That is, we can use the same code for two different nodes in two different communities. Compare it to calling somebody on a land line. If you need to call someone within the same village (or even organisation) you usually only need a few numbers. For example, you dial your best friend with the phone number "1105". Now if you want to call somebody in another village (with number "38"), you will first have to dial out (using the code "0"), then dial the access code and then the phone number again. For example, your other friend lives in another town and you dial "0-38-1105". Notice that the actual phone number can be the same for both friends: "1105". This is the same idea for the random walker: nodes in different communities can reuse the same code.

If we do not consider any partition, by Shannon's source coding theorem, we can represent the list of nodes visited with $H(X) = -\sum_i \pi_i \log \pi_i$ bits per step, where π_i are the stationary probabilities of the random walker as derived in Eq. (2.23). If we do consider a partition σ, we can reuse the same codes for nodes in different communities, which should shorten the average code length for that community.

The probability that a random walker stays within a community is then

$$\rho_c = \frac{e_c}{K_c}$$

where $e_c = \sum_{ij} A_{ij}\delta(\sigma_i, \sigma_j)$ the total number of edges as before, and $K_c = \sum_i k_i \delta(\sigma_i, c)$ the total degree. The probability to leave a community is then of course $1 - \rho_c$. The probability a certain community is visited is then

$$q_c = \sum_i \pi_i \delta(\sigma_i, c).$$

We should also define a code for moving outside a community to another community, similar to dialling a "0" for dialling out. We include this code for exiting from community c in the entropy, in order to take it into account. The entropy for moving within a community c (or exiting) is then

$$H_c = -\sum_i \frac{\pi_i}{q_c + (1-\rho_c)} \log \frac{\pi_i}{q_c + (1-\rho_c)} - \frac{1-\rho_c}{q_c + (1-\rho_c)} \log \frac{1-\rho_c}{q_c + (1-\rho_c)},$$

so that we can choose optimal codes of average code length H_c for that community. In addition, if the random walker exits from a community, the average code length for indicating to which community the random walker goes is then

$$H_q = -\sum_c q_c \log q_c$$

With probability q_c we then incur the average code length of H_c while with probability $(1-\rho) := \sum_c (1-\rho_c)$ we incur the cost of switching communities. So, the total expected code length is then

$$\mathrm{L}(\sigma) = (1-\rho) H_q + \sum_c q_c H_c. \tag{2.28}$$

This is known as the map equation, and we try to minimize this expected code length. The derivation here is slightly different from the original [44], but is similar in spirit. Unlike the other models, we will not analyse this model in great detail, but it is included for the sake of completeness.

2.2.8 Alternative Clustering Methods

As stated earlier, the approach of community detection is somewhat recent, and different approaches have been used before. There exists a multitude of general clustering techniques, such as hierarchical clustering or k-means clustering, which are usually applied to datasets in some Euclidean space [15, 23, 27, 51]. By using some graph similarity (or distance) type of measure, it is possible to apply these existing techniques on graphs [46]. Hierarchical clustering for example merges two groups depending on the similarity of the two groups (taking a greedy outlook), thus resulting in a dendrogram of merges. The k-means method tries to iteratively minimize the average within cluster distances by minimizing the distance to some cluster average.

Similarities between nodes can be derived in many different ways. One such similarity measure can for example be derived by considering the expected commuting time to go from node i to node j in a random walk on a graph [52]. This can be based on the graph Laplacian, which is defined as

$$\mathcal{L} = D - A, \tag{2.29}$$

where A is the adjacency matrix and $D = \text{diag}(k_1, \ldots, k_n)$ is the diagonal degree matrix. Notice that

$$\begin{aligned} u^\top \mathcal{L} u &= \sum_{ij} u_i L_{ij} u_j \\ &= \sum_{ij} \left[u_i \delta_{ij} k_i u_j \right] - \sum_{ij} \left[u_i A_{ij} u_j \right] \\ &= \sum_{ij} A_{ij} (u_i - u_j)^2 \end{aligned}$$

so that $\mathcal{L}$ is positive-semidefinite and has only non-negative eigenvalues. We won't go into the details, but the expected commuting time C_{ij} to go from node i to node j can be expressed as [17]

$$C_{ij} = 2m(e_i - e_j)\mathcal{L}^+(e_i - e_j) \tag{2.30}$$

where e_i is the ith basis vector and L^+ is the pseudo inverse of the Laplacian

$$\mathcal{L}^+ = \left(\mathcal{L} - \frac{1}{n} \right)^{-1} + \frac{1}{n}. \tag{2.31}$$

It can be proven that C_{ij} is a proper distance metric, which can then be used in other clustering techniques for further processing.

Another approach also based on the Laplacian is that of spectral graph partitioning (for details, see [3, 36]). This idea is based on trying to minimize the cut-size. Assume we have some vector $s \in \{-1, 1\}^n$, where $s_i = -1$ indicates node i is in group 1 and if $s_i = 1$ it is in group 2. Then the total number of edges running between the two groups can be written as

$$\sum_{ij} A_{ij} \frac{1}{2}(1 - s_i s_j) = \frac{1}{2} s^\top \mathcal{L} s. \tag{2.32}$$

Realising that $\frac{1}{2}(1 - s_i s_j) = 1 - \delta(\sigma_i, \sigma_j)$, we then recognize the trivially optimized label propagation method (LP) from Eq. 2.22. The trivial solution is simply $s = (1, \ldots, 1)$ in which case $s^\top \mathcal{L} s = 0$. That is why often in this context an additional constraint is imposed, namely that the two groups should be of roughly equal size. Solving this leads to the eigenvector u_2 corresponding to the second-smallest eigenvalue λ_2 of the Laplacian $\mathcal{L}$, and setting $s_i = \text{sgn}(u_{2i})$. This eigenvector is also known as the Fiedler vector. The first eigenvalue $\lambda_1 = 0$, and the second eigenvalue λ_2 is only zero if the graph is disconnected. For this reason, it is also known as the algebraic connectivity. There are also other variants of spectral graph partitioning, for example based on the normalized Laplacian $D^{-1}\mathcal{L}$, but we won't treat them here.

2.3 Algorithms

In this section we will review some of the more common algorithms for optimizing modularity (and some of its alternatives). The problem of community detection is NP-hard in general [6], so that there is no (known[5]) efficient (polynomial time) algorithm for optimizing the objective function. The algorithms presented will thus be heuristics, and usually involve some stochasticity. This implies that it will not necessarily always find exactly the same partition. In fact, modularity often seems to have many near optimal partitions, making it difficult to obtain the global optimum, and the other methods are expected to show a similar degeneracy [19].

In order to test whether an algorithm is working correctly, and performs well, it is useful to construct test networks. These test networks—also known as benchmark networks—are constructed such that the community partition is known beforehand. Comparing the known partition to the partition detected by the algorithm provides evidence of how well the algorithm is performing. We will test some of the methods, and present their results. In spite of the NP-hardness of the problem, and that the algorithms are only heuristic, we will see they work reasonably well.

2.3.1 Simulated Annealing

Simulated Annealing (SA) is a general optimization technique [26]. The idea is that the search is allowed to explore a large part of the landscape at the beginning, but as the algorithm progresses, follows more and more the steepest descent trajectory (greedily) towards a (local) minimum. The basic idea is to analyse the difference in the objective function $\Delta\mathcal{H} = \mathcal{H}_{\text{after}} - \mathcal{H}_{\text{before}}$ when making a certain change to the partition. We will use $\Delta\mathcal{H} = \mathcal{H}_{\text{after}} - \mathcal{H}_{\text{before}}$ throughout this thesis, so that $\Delta\mathcal{H} < 0$ will always mean there is an improvement after some change, while $\Delta\mathcal{H} > 0$ indicates the prior situation was better (remember we are minimizing $\mathcal{H}$). Such a change can take many forms, but the changes usually considered are: moving a single node from one community to another; merging two communities; or splitting a community.

There are several choices available for accepting such a change. The idea is to also accept changes that worsen the partition (i.e. when $\Delta\mathcal{H} > 0$) with some probability that decreases as the algorithm progresses. The implementation from Reichardt and Bornholdt [41] works as follows. Consider moving node i from community c to d, and let the new communities be c' and d'. In terms of the community set we thus have that $C'_c = C_c \setminus i$ and $C'_d = C_d \cup i$. The change in the objective function is then

$$\Delta\mathcal{H}(\sigma_i = c \mapsto d) = (e_{id'} - \gamma_{\text{RB}}\langle e_{id'}\rangle) - (e_{ic'} - \gamma_{\text{RB}}\langle e_{ic'}\rangle) \tag{2.33}$$

[5] It is unlikely that any efficient algorithm will ever be found, part of the famous P = NP problem.

where $e_{ic'} = e_c - e_{c'} = \sum_j A_{ij}\delta(\sigma_i, c')$ is the number of edges from node i to community c' and $\langle e_{ic'}\rangle = \langle e_c\rangle - \langle e_{c'}\rangle = \sum_j p_{ij}\delta(\sigma_i, c')$ the expected number of edges from i to community c', and similarly so for d'. We consider all communities to which node i is connected, and the associated change in the objective function of $\Delta\mathcal{H}(\sigma_i = c \mapsto d)$. We then choose the new community with probability

$$\Pr(\sigma_i = d) = \frac{1}{Z}\exp\bigl[-\beta\Delta\mathcal{H}(\sigma_i = c \mapsto d)\bigr], \tag{2.34}$$

where $Z = \sum_d \exp\beta\Delta\mathcal{H}(\sigma_i = c \mapsto d)$ is the normalization factor. This is known as the Boltzmann probability distribution [24]. The parameter $\beta = 1/T$ is known as the inverse temperature. A high temperature (low β) gives nearly uniform probabilities, so that every change is chosen with almost equal probabilities. As the algorithm progresses, the temperature is lowered, for example after n changes, usually via $T' = \alpha T$ where $0 < \alpha < 1$ is some decay factor. Lower temperature leads to more narrow choices, and in the limit of $T \to 0$ only the moves with the maximum improvement of the objective function are chosen.

An alternative scheme was proposed by Guimerà et al. [20, 22]. Instead of considering all possible changes, we simply choose a random new community for a node. Similarly, a change can consist of merging two communities. Finally, a change can consist of splitting a community in two. All changes have a certain associated change in the objective function of $\Delta\mathcal{H}$ and the change is accepted with probability

$$\Pr(\text{accept change}) = \begin{cases} 1 & \text{if } \Delta\mathcal{H} < 0, \\ \exp(-\beta\Delta\mathcal{H}) & \text{if } \Delta\mathcal{H} \geq 0. \end{cases} \tag{2.35}$$

The change for moving a node i from community c to community d is already provided in Eq. (2.33). The change when merging two communities c and d into one new community c' is then

$$\Delta\mathcal{H}(\{c, d\} \mapsto c') = -e_{cd} + \gamma_{\text{RB}}\langle e_{cd}\rangle_{p_{ij}} \tag{2.36}$$

while the splitting of community c' into c and d is just the opposite

$$\Delta\mathcal{H}(c' \mapsto \{c, d\}) = -\Delta\mathcal{H}(\{c, d\} \mapsto c'), \tag{2.37}$$

with $e_{cd} = \sum_{ij} A_{ij}\delta(\sigma_i, c)\delta(\sigma_j, d)$ the number of edges between c and d and $\langle e_{cd}\rangle$ the expected number of such edges. A random split is unlikely to improve the partition, so some additional effort should be made to find a reasonably good candidate split, for example by using the eigenvector split (see Sect. 2.3.4), but we will not consider that here.

For both implementations the general idea remains the same. We consider a number of changes, which are accepted with a certain probability. After a certain number of changes, we lower the temperature, and repeat the procedure. When the objective

function is no longer improved, the procedure terminates. The method moving only nodes is provided in Algorithm 1.

The exact calculations depend on the null model used. For the configuration null model, we have that $\langle e_c \rangle = K_c^2/2m$, and if we work out we obtain

$$\Delta\mathcal{H}(\sigma_i = c \mapsto d) = e_{id'} - e_{ic'} - \gamma_{\text{RB}} \frac{k_i}{m}(K_d - K_c + k_i) \tag{2.38a}$$

$$\Delta\mathcal{H}(\{c, d\} \mapsto c') = \gamma_{\text{RB}} \frac{K_c K_d}{2m} - e_{cd} \tag{2.38b}$$

$$\Delta\mathcal{H}(c' \mapsto \{c, d\}) = e_{cd} - \gamma_{\text{RB}} \frac{K_c K_d}{2m}, \tag{2.38c}$$

for respectively joining nodes, merging communities and splitting communities. For the ER null model, with $\langle e_c \rangle = pn_c^2$ where n_c is the size of community c, we obtain

$$\Delta\mathcal{H}(\sigma_i = c \mapsto d) = e_{id} - e_{ic} - \gamma_{\text{RB}} p((n_d + 1) - (n_c - 1)) \tag{2.39a}$$

$$\Delta\mathcal{H}(\{c, d\} \mapsto c') = \gamma_{\text{RB}} p n_c n_d - e_{cd} \tag{2.39b}$$

$$\Delta\mathcal{H}(c' \mapsto \{c, d\}) = e_{cd} - \gamma_{\text{RB}} p n_c n_d. \tag{2.39c}$$

Similar calculations can be derived for the other models.

Algorithm 1 Simulated Annealing (SA) method

```
function SA(Graph G)
    initialize σ_i ← i for all nodes i
    T ← some high number, β ← 1/T
    while improvement do
        for all nodes i do
            C_neigh ← {σ_j | (i, j) ∈ E} ∪ σ_i          ▷ Communities of neighbours
            for all communities d ∈ C_neigh do
                P_d ← exp(β ΔH(σ_i = c ↦ d))
            end for
            σ_i ← RANDSAMPLE(P)                        ▷ Draw random community
        end for
        T ← α * T, β ← 1/T                             ▷ Lower temperature
    end while
    return σ
end function
```

2.3.2 Greedy Improvement

Graph partitioning itself is not new, and one heuristic method that has long been used, and which resembles the steps from Simulated Annealing (SA), is Kernighan-Lin (KL) improvement [25]. Although in the original formulation two nodes are swapped from their communities in order to keep the community sizes the same, this is not necessary for modularity optimization. So, the greedy improvement we consider here simply amounts to moving nodes from one community to another.[6] The difference with SA is that we choose greedily the best new community. In other words, the method loops (randomly) over all nodes, and determines for each node the community with the largest $\Delta\mathcal{H}$. It repeats these steps as long as there remain improvements.

More specifically, when considering node i we greedily check the increase in the objective function $\Delta\mathcal{H}(\sigma_i = c \mapsto d)$ if the node was moved from community c to d, as was already calculated in Eq. (2.33). Now instead of choosing the new community with a certain probability as defined in Eq. (2.34), we simply choose the community

$$s^* = \arg\max_s \Delta\mathcal{H}(\sigma_i = r \mapsto s) \tag{2.40}$$

which maximizes the change. This can be seen as the limit of the simulated annealing process for which $T \to 0$ (or $\beta \to \infty$). We consider all nodes (perhaps in random order), and repeat until no further improvement can be made.

Algorithm 2 Greedy method

```
function GREEDY(Graph G)
    initialize σ_i ← i for all nodes i
    while improvement do
        for all nodes i do
            C ← {σ_j | (i, j) ∈ E} ∪ σ_i          ▷ Communities of neighbours
            for all communities d ∈ C do
                Δ_d ← ΔH(σ_i = c ↦ d)
            end for
            σ_i ← arg max_d Δ_d                    ▷ Greedily, maximum choice
        end for
    end while
    return σ
end function
```

[6] There are some other greedy algorithms as well, for example [7, 8].

2.3.3 Louvain Method

The Louvain method for optimizing modularity [4] is one of the fastest and best algorithms available for optimizing modularity [29]. It makes changes to the partition similar to the greedy improvement, i.e. it always makes the optimal change at that moment. The trick that makes it so fast and yet work well, is that whenever no more changes can be made by moving nodes, we aggregate the graph, and rerun the same algorithm on the aggregated graph. This is then repeated until modularity can be no further increased.

Algorithm 3 Louvain method

```
function LOUVAIN(Graph G)
    σ ← GREEDY(G)                          ▷ Initial Greedy
    Σ ← σ                                  ▷ Use Σ for aggregate
    while improvement do
        G ← AGGREGATE(G, Σ)
        Σ ← GREEDY(G)                      ▷ Greedy on aggregate graph
        σ_i ← Σ_{σ_i} for all i            ▷ Correct σ according to Σ
    end while
    return σ
end function
```

The important detail is then of course that moving nodes in the aggregated graph should be equivalent to merging communities in the original graph. Hence, the aggregate method depends on the exact cost function used. Using the configuration null-model allows for a particularly straightforward aggregation. In that case, the new aggregated weighted adjacency matrix A' is constructed as follows

$$A'_{cd} = \sum_{ij} A_{ij}\delta(\sigma_i, c)\delta(\sigma_j, d) = e_{cd}$$

which simply creates a new node c for each community, and an edge to another new node community d has as weight the total number of edges between community c and d. The essential thing is now that joining two nodes in this graph A' should be equivalent to merging two communities in A. The benefit for joining nodes c and d in A' is

$$\Delta\mathcal{H} = -\left(A'_{cd} - \gamma_{\text{RB}}\frac{k'_c k'_d}{m}\right)$$

which is equivalent to joining communities c and d in A since $A'_{cd} = e_{cd} = \sum_{ij} A_{ij}\delta(\sigma_i, c)\delta(\sigma_j, d)$ the number of edges between communities c and d and $k'_c = \sum_d A'_{cd} = \sum_{ij} k_i\delta(\sigma_i, c)$ is the total degree in community c. Hence, joining two nodes is indeed equivalent to merging two communities as specified in Eq. (2.38b). This special feature of the configuration model (and modularity) allows this formulation to exploit this.

Algorithm 4 Aggregation for configuration null-model

function AGGREGATE(Graph G, Community σ)
  $A \leftarrow$ ADJACENCY(G)
  $A'_{cd} \leftarrow \sum_{ij} A_{ij}\delta(\sigma_i, c)\delta(\sigma_j, d)$
  return A'
end function

When using the ER null model this way of aggregating does not work correctly. Let us assume for an instance that we aggregated a graph according to this method. The benefit of merging node c and d in this aggregate graph, according to the ER null model is then

$$\Delta\mathcal{H} = A_{cc} + A_{dd} - A'_{cd} - \gamma_{\text{RB}} p$$

while this should actually be

$$\Delta\mathcal{H} = A_{cc} + A_{dd} - A'_{cd} - \gamma_{\text{RB}} p n_c n_d$$

where n_c and n_d are the number of nodes in community c and d. Using this method of aggregating then clearly does not work.

In order to make this step of aggregating the graph work for the ER null-model we need to introduce the node size. In the aggregate graph, the node size will then represent the number of nodes in the community (i.e. the community size). So, for the initial graph we set the node size to $n_i = 1$ for all nodes, and upon aggregating we will set the node size $n_c = \sum_i n_i \delta(\sigma_i, c)$ of community c, i.e. the new node in the aggregated graph, equal to the sum of the node sizes within the community.

Notice that we can use the same type of aggregation for CPM (and by extension RN). Since we can also apply the greedy algorithm to CPM, the Louvain method is easily applied to CPM as well.

Algorithm 5 Aggregation for ER null-model & CPM

function AGGREGATE(Graph G, Community σ)
  $A \leftarrow$ ADJACENCY(G)
  $A'_{cd} \leftarrow \sum_{ij} A_{ij}\delta(\sigma_i, c)\delta(\sigma_j, d)$
  $n_c \leftarrow \sum_i n_i \delta(\sigma_i, c)$
  return A', n'
end function

2.3.4 Eigenvector

We can also take a matrix analysis perspective [38]. If we define the modularity matrix B with entries

$$B_{ij} = a_{ij}A_{ij} - b_{ij}(1 - A_{ij}) \tag{2.41}$$

and S the $n \times q$ community matrix, such that $S_{ic} = 1$ if node i is in community c and 0 otherwise, we can write our objective function as

$$\mathcal{H} = -\sum_{ij}\sum_{c} B_{ij}S_{ic}S_{jc} = -\mathrm{Tr}S^{\top}BS, \tag{2.42}$$

since $S_{ic}S_{jc} = 1$ if $\sigma_i = \sigma_j = c$ and 0 otherwise, so that $\sum_c S_{ic}S_{jc} = \delta(\sigma_i, \sigma_j)$, and $\sum_i S_{ic}S_{id} = 0$ for $c \neq d$. Here $S^{\top}$ denotes the transpose of S (i.e. $S^{\top}_{ij} = S_{ji}$). Since each node should be in exactly one community, we have the constraint that $S_{ic} \in \{0, 1\}$ and $\sum_c S_{ic} = 1$. From this it also follows that $\mathrm{Tr}S^{\top}S = n$ and that the columns of S are mutually orthogonal. For undirected graphs B is symmetric (i.e. $B = B^{\top}$), and we can decompose $B = U\Lambda U^{\top}$ where Λ is a diagonal vector containing the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ with U an orthogonal matrix (i.e. $UU^{\top} = I_n$ is the identity matrix) containing the associated eigenvectors. Plugging this in leads to

$$\begin{aligned}\mathcal{H} &= -\mathrm{Tr}S^{\top}U\Lambda U^{\top}S \\ &= -\mathrm{Tr}\Lambda U^{\top}SS^{\top}U,\end{aligned}$$

So, for all $\lambda_i > 0$ we should put as much weight as possible in $U_i^{\top}SS^{\top}U_i$. Without the constraint that $S_{ic} \in \{0, 1\}$ this would be simply optimized by taking the column S_i proportional to u_i for $\lambda_i > 0$, and the rest 0. Because of the constraints that $S_{ic} \in \{0, 1\}$ this is not straightforward, and usually only a partitioning in two groups is considered. This is known as (recursive) spectral bisectioning. The basic idea is to recursively split communities, until we can no longer divide the sub parts.

For spectral bisectioning, it is simpler to use a single vector s to indicate two groups as $s_i = -1$ if i is in group 1 and $s_i = 1$ if i is in group 2. Then $\frac{1}{2}(s_i s_j + 1) = \delta(\sigma_i, \sigma_j)$, and we can write

$$\mathcal{H} = -\sum_{ij} B_{ij}\frac{1}{2}(s_i s_j + 1)$$

which is up to a multiplicative and additive constant equivalent to

$$\mathcal{H} = -s^{\top}Bs, \tag{2.43}$$

with $s^{\top}s = n$. If we relax the problem by allowing s to take on real values, $s^{\top}Bs$ is similar to a Rayleigh quotient, for which it is well known that it is maximized by taking s proportional to u where u is the eigenvector associated to λ_1 the largest eigenvalue of B. Hence, if we take

$$s_i = \begin{cases} 1 & \text{if } u_i \geq 0, \\ -1 & \text{if } u_i < 0, \end{cases}$$

this is the vector s with $s_i \in \{-1, +1\}$ for which $\|s - u\|$ is minimal.

We can then recursively apply this method to a single community. Let B^c be the $n_c \times n_c$ submatrix of B corresponding to community c. The improvement of $\mathcal{H}$ by dividing community c in two, again denoted by the vector $s \in \{-1, +1\}^{n_c}$, can then be described by

$$\Delta\mathcal{H} = -\sum_{ij} B^c_{ij} \frac{1}{2}(s_i s_j + 1) - B^c_{ij}$$

which by removing parts that don't depend on the optimization reduces to

$$\Delta\mathcal{H} = -\sum_{ij} B^c_{ij} s_i s_j = -s^\top B^c s \tag{2.44}$$

similar as before. So, we follow the same procedure. However, we must ensure that the total contribution is positive still, so that $\Delta\mathcal{H}$ in Eq. (2.44) must obey

$$\Delta\mathcal{H} = -s^\top B^c s < -e^\top B^c e$$

with $e = (1, \ldots, 1)$ the vector of all ones. In other words, as long as subdividing puts more weight within the subdivided community as there is in total within the community, we should continue splitting. Notice that this is similar to the condition that $\Delta\mathcal{H}(c' \mapsto \{c, d\}) > 0$ for splitting community c' into community c and d in Eq. (2.37). Furthermore, notice that for the RB model with $\gamma_{\text{RB}} = 1$ we have that $e^\top B e = 0$ by definition of modularity, so that we can use the same condition.

Algorithm 6 Recursive eigenvector bisection

```
function EIGENVEC(Modularity matrix B)
    u ← largest eigenvector of B
    σ_i ← { −1  if u_i ≥ 0,
          {  1  if u_i < 0.
    if σ^⊤ B σ > e^⊤ B e then                          ▷ If improvement
        Σ_1 ← EIGENVEC(B(σ = −1, σ = −1))              ▷ Submatrix for σ_i = −1
        Σ_2 ← EIGENVEC(B(σ = 1, σ = 1))                ▷ Submatrix for σ_i = 1
        σ ← Combine Σ_1 and Σ_2
    else
        σ_i ← 1                                        ▷ Otherwise, don't split
    end if
    return σ
end function
```

2.4 Benchmarks

In order to know whether these algorithms and methods work effectively, we now turn to methods for testing them. This involves two parts. First we have to construct good test networks with some planted partition, so that we can check if some community detection method is able to uncover this planted partition. Secondly, we need some measure to compare the computed partition to the planted partition. Finally, we will provide some results comparing different methods.

2.4.1 Test Networks

One of the first problems in generating test networks is that there is no definitely agreed upon definition of a community. However, as stated earlier, there is some consensus on some common features: the communities should be relatively dense, and relatively well separated from the rest of the network. Although specific details might not be agreed upon exactly, this often is the foundation upon which test networks are constructed. Still, we should keep in mind that different definitions of communities or good partitions might yield a partition different from the planted partition. This does not necessarily imply the method does not work correctly, because the definition of community simply differs. Nonetheless, if some method is unable to detect correctly the planted partition whereas other methods do, it does indicate it might not be the appropriate method for these type of test networks.

The first to propose such test networks were [18], and remained the common benchmark for some time [10]. In general, test networks are constructed as follows. We wish to build a network of q communities of each n_c nodes with average degree $\langle k \rangle$. The total number of nodes is then $n = qn_c$ and the total number of edges $m = \langle k \rangle n/2$. Furthermore, we would like to control the difficulty of detecting communities. The denser communities are, and the better separated from the rest of the network, the easier it is to detect such communities. Hence, we will introduce a mixing parameter $0 \leq \mu \leq 1$ such that each node will have about $(1-\mu)\langle k \rangle$ edges within its community, and about $\mu \langle k \rangle$ edges outside its community. Such a network can be easily constructed as follows. We pick a random node i and with probability μ we will link to a node outside of its community, and with probability $1-\mu$ we link to a node within its community. We will add in total $\langle k \rangle n/2$ edges. Easily partitioned networks are constructed using a low μ and this gets progressively more difficult for higher μ. The common test setting introduced by Mark Newman used $q = 4$ communities of $n_c = 32$ nodes each, with μ varying from 0 to 1.

One question concerns until what point μ we expect communities to exist. A reasonable limit is that the average density within a community should be higher than the average density between communities. Beyond this threshold communities become very fuzzy (regardless of the definition) and are unlikely to be detected by any method.

Let us first calculate the inner density for a community of size n_c. Each of the n_c nodes has on average $(1-\mu)\langle k\rangle$ edges within its community, and the density is therefore

$$p_{\text{in}} = \frac{(1-\mu)\langle k\rangle}{n_c - 1}. \tag{2.45}$$

The rest of the $\mu\langle k\rangle$ edges per node will be distributed across the rest of the network. Since these edges get distributed over $n - n_c$ nodes, they will be more dispersed in general. The average density is then simply

$$p_{\text{out}} = \frac{\mu\langle k\rangle}{n - n_c}. \tag{2.46}$$

A community of n_c nodes in the test network is then well-defined as long as $p_{\text{in}} > p_{\text{out}}$, which yields

$$\mu < \frac{n - n_c}{n - 1} \approx \frac{q-1}{q}. \tag{2.47}$$

In other words, the probability for a link within a community μ should be smaller than the proportion of nodes outside the community. Notice this is independent of the total size of the network, the average degree, and the size of the communities, and depends only on the number of communities q (up to a correction term of $\frac{1}{n_c}$). For the regular test setting of $q = 4$ communities this yields $\mu < 0.75$, contrary to what was believed earlier that the communities would be defined up to $\mu = 0.5$.

In fact, such a test network most closely resembles a random network around $\mu \approx (q-1)/q$. For smaller μ the network exhibits a community structure. For higher μ however, the network still has a very particular structure. In that case, there are few links within communities, and many between communities. In other words, it starts to show a multi-partite structure.

Although such a test network is fine, it is far from realistic. Most networks show a skewed degree distribution with a fat-tail. They have many nodes with a low degree, and some nodes with an extremely high degree. The above test networks on the other hand have a Poissonian degree distribution, such that most of the nodes have about the same degree $k_i \approx \langle k\rangle$. Most empirical results of community detection suggests the community sizes are also highly skewed, while in these test networks each community is of exactly the same size. This could lead to a potential bias when benchmarking methods, since it only looks to whether a method can find communities in this particular test setting. In order to overcome these issues it was suggested to create test networks that have a power-law degree and community size distribution by Lancichinetti et al. [30], now commonly known as the LFR benchmark. Additionally, weights of links can be introduced, which realistically should also take a power-law distribution. These weights can again be distributed differently within and between communities.

Furthermore, many complex networks show some form of hierarchical structure [30]. In order to test for this, hierarchical test networks would be needed. So, instead

of only having a single partition in communities, each community at the lowest level is embedded in increasingly larger communities. Instead of specifying then a single μ for the probability of having links outside the community, we specify $\mu_1, \mu_2, \ldots, \mu_l$ for l different levels, with each level i being embedded in the $i-1$ level. Level 1 is then the coarsest, highest level, and l the lowest most refined level. Of course, these probabilities are limited to $\sum_l \mu_l < 1$.

The limits of the densities remain rather similar, but now depend on the level we are looking at. Let us take a look to a two level hierarchy. The corresponding densities then are

$$\begin{aligned} p_1^{in} &= (1-\mu_1)\langle k\rangle/(n_{c,1}-1) \\ p_1^{out} &= \mu_1\langle k\rangle/(n-n_{c,1}) \\ p_2^{in} &= (1-\mu_1-\mu_2)\langle k\rangle/(n_{c,2}-1) \\ p_2^{out} &= (\mu_1+\mu_2)\langle k\rangle/(n-n_{c,2}) \end{aligned}$$

where $n_{c,1}$ is the community size at level 1 and $n_{c,2}$ the community size at level 2. The second level then remains detectable until

$$\mu_1 + \mu_2 < \frac{n-n_{c,2}}{n-1}.$$

Similarly, the first level is well defined until

$$\mu_1 < \frac{n-n_{c,1}}{n-1}.$$

Both limits are similar to the original limit in Eq. (2.47) but, there is a trade-off between the fine (μ_2) and course level (μ_1). Whenever the coarse level is less well defined, the corresponding limit for the finer level becomes smaller.

2.4.2 Comparing Partitions

Once a test network with a known partition is available, we need a measure for stating how well a certain method is able to recover this known partition. Various measures are suitable for this, but two of the most common ones are the normalized mutual information (NMI) and the variation of information (VI). The NMI measures how much information we have about one partition knowing the other. The VI is a true metric, and is closely related to the NMI. Benchmark results are usually provided in NMI, but VI seems somewhat more sensitive to small deviations.

Both measures have their origins in information theory, of which the basics have been provided in Sect. 2.2.7 (see pp. 23–26). The mutual information is defined as

$$I(X,Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X)$$
$$= H(X) + H(Y) - H(X,Y).$$

Hence, if X and Y are two independent variables, $H(X,Y) = H(X) + H(Y)$ and $I(X,Y) = 0$. On the other hand, if X is completely determined by Y then $H(X,Y) = H(X) = H(Y)$ and $I(X,Y) = I(X,X) = H(X)$. Hence, we can normalize $I(X,Y)$ by $H(X) + H(Y)$ and arrive at at the normalized mutual information

$$\text{NMI}(X,Y) = \frac{2I(X,Y)}{H(X) + H(Y)}, \tag{2.48}$$

which is always $0 \leq \text{NMI}(X,Y) \leq 1$. The Variation of Information (VI) can then be defined as

$$\text{VI}(X,Y) = H(X) + H(Y) - 2I(X,Y), \tag{2.49}$$
$$= 2H(X,Y) - H(X) - H(Y), \tag{2.50}$$

Since $I(X,Y) = H(X)$ if and only if X is completely determined by Y then $\text{VI}(X,X) = 0$. Otherwise, since $2I(X,Y) \leq H(X) + H(Y)$, we have that $\text{VI}(X,Y) \geq 0$. Furthermore, notice that $\text{VI}(X,Z) \leq \text{VI}(X,Y) + \text{VI}(Y,Z)$, since the inequality

$$2H(X,Z) - H(X) - H(Z) \leq 2H(X,Y) + 2H(Y,Z)$$
$$- H(X) - 2H(Y) - H(Z)$$

is equivalent to

$$H(X,Z) \leq H(X,Y) + H(Y,Z) - H(Y)$$
$$H(X \mid Z) \leq H(X \mid Y) + H(Y \mid Z).$$

The last inequality holds because

$$H(X \mid Y) + H(Y \mid Z) - H(X \mid Z)$$
$$\geq H(X \mid Y,Z) + H(Y \mid Z) - H(X \mid Z)$$
$$= H(X,Y \mid Z) - H(X \mid Z) \geq 0$$

In other words, the $\text{VI}(X,Y)$ is a true metric, and can be interpreted to provide a distance between the random variables X and Y. There are several ways to normalize this quantity, for example by dividing by $I(X,Y)$ or by $\max\{H(X), H(Y)\}$, but this is not often considered [29, 34].

When it comes to comparing partitions, these quantities are used as follows. Let C and D be two partitions, such that there are n_c nodes in community c in C, n_d nodes in community d in D and n_{cd} nodes that are in community c in C and in community

d in D. The probability a random node is in community c is then $p_c = n_c/n$, and likewise we can define the probability $p_{cd} = n_{cd}/n$. Working this out for mutual information, we thus arrive at

$$I(C, D) = -\sum_{cd} \frac{n_{cd}}{n} \log\left(n \frac{n_{cd}}{n_c n_d}\right)$$

and

$$H(C) = -\sum_{c} \frac{n_c}{n} \log \frac{n_c}{n}.$$

The other quantities follow readily. The baseline is that NMI $= 1$ (and so VI $= 0$) whenever $C = D$ the two partitions are equal. So when comparing a method to the known partition, if a method works well, NMI ~ 1, and VI ~ 0.

Other well known measures for comparing partitions are the (adjusted) rand index and Jaccard index [15, 47, 51]. This is based on checking how many pairs of nodes are clustered in the same manner. The number of pairs of nodes that are clustered in the same way in both partitions can be obtained as

$$a = \sum_{cd} n_{cd}, \tag{2.51}$$

where n_{cd} denotes the number of nodes that are in community c in partition C and in community d in partition D. The number of pairs of nodes that are clustered both in different communities—so the number of pairs of nodes i and j such that they are not in the same community in partition C and neither in partition D can be described by

$$b = \binom{n}{2} + \sum_{cd} n_{cd} - \sum_{c} n_c^C - \sum_{c} n_c^D \tag{2.52}$$

where n_c^C refers to the number of nodes in community c in partition C. Then the rand index is defined as

$$\mathrm{RI}(C, D) = \frac{a + b}{\binom{n}{2}}, \tag{2.53}$$

namely the fraction of pairs of nodes that are classified in the same manner (belonging both to the same community is both partitions are both to different communities in both partitions). This measure varies between 0 and 1 with 1 indicating two identical partitions C and D while 0 indicates two completely different partitions. There exists an adjusted version which takes into account the fact that the rand index for two random partition already attains some similarity. The Jaccard index is defined as

$$\mathrm{J}(C, D) = \frac{a}{\binom{n}{2} - b}. \tag{2.54}$$

Compared to the VI both measures have some drawbacks [34], although no measure is perfectly fit for all situations. For benchmarks in community detection however, the NMI has become the standard, although the rand index, Jaccard index and other variants are used in other domains.

2.4.3 Results

Not all models work equally well. We have tested extensively the RB model using the configuration null model and the ER null model, CPM and Infomap. For the RB model the "natural" parameter is $\gamma_{\text{RB}} = 1$, which then corresponds to modularity for the configuration model. For Infomap there is no parameter present, so there is little to choose there. For CPM there is no such "natural" parameter, and one would have to look which γ_{CPM} works best (we will touch upon this issue in Sect. 4.1). However, given that we know how we generate the benchmark networks, we can calculate the optimal parameter γ^*_{CPM} for uncovering the planted partition. Since the CPM model and the RB model are equal for the ER null model when using $\gamma_{\text{CPM}} = \gamma_{\text{RB}} p$, this also corresponds to the optimal parameter for the RB model with the ER null model. For the configuration null model we can choose a similar optimal parameter value, in order to detect the planted partition as well as possible.

Let us calculate this optimal parameter value. We denote by p_{in} the average density within a community, and by p_{out} the average density between a community and the rest of the network. For CPM to correctly detect these communities we should set $\gamma_{\text{CPM}} > p_{\text{in}}$ so that it doesn't split communities of that density, while $\gamma_{\text{CPM}} < p_{\text{out}}$ so that it doesn't merge communities either. We have already calculated these densities before in Eqs. (2.45) and (2.46), and we set

$$\gamma^*_{\text{CPM}} = \gamma^*_{\text{RB}} p = \frac{\langle p_{\text{in}} \rangle + \langle p_{\text{out}} \rangle}{2}$$

where $\langle p_{\text{in}} \rangle$ indicates we have taken the average p_{in} over all community sizes.

In order to calculate a similar optimal resolution parameter for the configuration model, notice that we should have that the inner "degree density" $\tilde{p}_{\text{in}} = \frac{e_c}{\langle e_c \rangle_{\text{conf}}}$ should be lower than γ_{RB}, while the outer "degree density" should be higher than γ_{RB}. The number of edges within a community is simply $e_c = n_c \langle k \rangle (1 - \mu)$, and the expected sum of degrees $K_c = n_c \langle k \rangle$. Furthermore, the total number of expected edges is $2m = n \langle k \rangle$, so that we obtain

$$\begin{aligned} \tilde{p}_{\text{in}} &= \frac{e_c}{\langle e_c \rangle_{\text{conf}}} \\ &= \frac{n_c \langle k \rangle (1 - \mu)}{\frac{(n_c \langle k \rangle)^2}{n \langle k \rangle}} \\ &= \frac{n(1 - \mu)}{n_c}. \end{aligned}$$

The outer "degree density" can be similarly calculated. The number of external edges remains $e_{c*} = n_c \mu \langle k \rangle$ as before (where the $*$ denote the rest of the network). The expected number of edges is $\langle e_{c*} \rangle = K_c K_* / 2m$, and so becomes $\langle e_{c*} \rangle = n_c (n - n_c) \langle k \rangle^2 / 2m$, so that the outer "degree density" is

$$\begin{aligned} \tilde{p}_{\text{out}} &= \frac{e_{c*}}{\langle e_{c*} \rangle_{\text{conf}}} \\ &= \frac{n_c \mu \langle k \rangle}{\frac{n_c (n - n_c) \langle k \rangle^2}{n \langle k \rangle}} \\ &= \frac{\mu n}{n - n_c}. \end{aligned}$$

Similar as before, we set the RB resolution parameter for the configuration model at

$$\gamma^*_{\text{RB}} = \frac{\langle \tilde{p}_{\text{in}} \rangle + \langle \tilde{p}_{\text{out}} \rangle}{2}$$

Notice that we can do a similar analysis as before, trying to calculate the point at which communities are no longer well defined, but use the "degree densities" to do so. Working out the inequality $\tilde{p}_{\text{in}} > \tilde{p}_{\text{out}}$ we obtain that up until

$$\mu < \frac{n - n_c}{n} \approx \frac{q - 1}{q}$$

the communities are well defined. Hence, this does not change anything in comparison to our earlier analysis in Eq. (2.47).

The results for the different methods are displayed in Fig. 2.1. On the y-axis it shows the NMI as defined earlier, while on the x-axis the mixing parameter μ is shown. For each value of the mixing parameter μ we generate 100 LFR benchmark networks. We have used the Louvain algorithm for all models, since earlier analysis showed the Louvain algorithm works at least as well as many other algorithms, but is much faster. For a more extensive comparison between different algorithms, refer to Lancichinetti and Fortunato [29].

It can be clearly seen that CPM performs well. The difference in performance of the CPM model in comparison to the RB model using the ER null model is especially striking. Obviously then, setting $\gamma_{\text{CPM}} = p$ is in general not a very good strategy, and for general networks one should carefully analyse at which resolution the network contains meaningful partitions, a topic we will review briefly in Sect. 4.1.

A similar effect also shows for modularity (or the RB model using the configuration model), such that when γ_{RB} is chosen appropriately (i.e. using $\gamma_{\text{RB}} = \gamma^*_{\text{RB}}$) the method will perform better than at the ordinary resolution $\gamma_{\text{RB}} = 1$. Indeed, the results of the CPM model and the RB model using the configuration null model using γ_{RB} are rather comparable, although the latter's performance drops less quickly, and then outperforms CPM. Interestingly, when we use the ordinary resolution $\gamma_{\text{RB}} = 1$, it becomes more difficult to detect communities in large networks using the

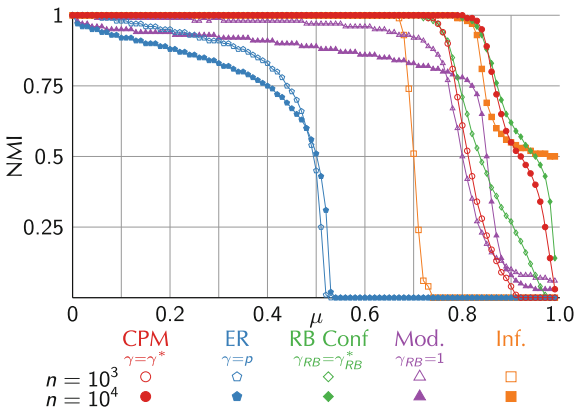

Fig. 2.1 Benchmark results

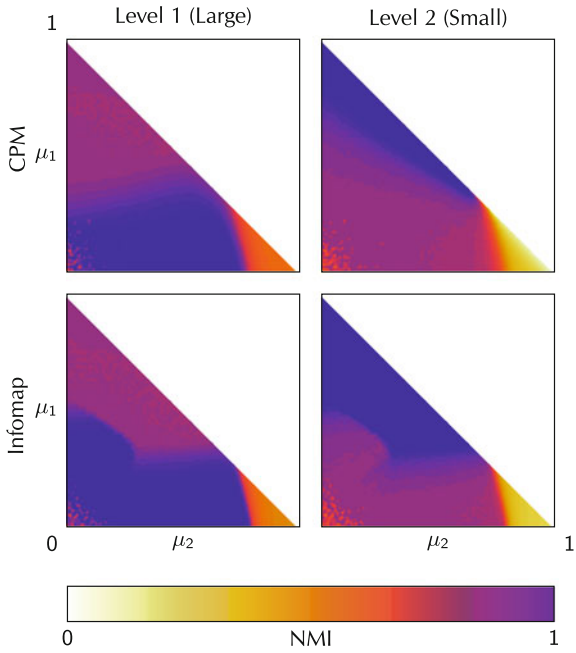

Fig. 2.2 Hierarchical benchmark results

configuration model. This contrasts with the results when we choose the appropriate resolution parameter γ^*_{CPM}, γ^*_{RB} and indeed also for the Infomap method. Indeed the communities should become more clearly discernible for larger networks when the community sizes remain similar. The limit of community detection as calculated earlier is about $\mu^* = \frac{q-1}{q} \approx 0.92$ for $n = 10^3$ and $\mu^* \approx 0.99$ for $n = 10^4$. The models with the tuned resolution parameters work quite well and approach this upper limit to some extent. Surprisingly, both methods outperform the Infomap method, which

performed superbly in previous tests [29], when the appropriate resolution parameter is chosen.

We have also performed extensive tests on hierarchical networks, where the method also performs well, and is able to extract the two different levels of communities effectively, as displayed in Fig. 2.2. For relatively low $\mu_2 \lessapprox 0.7$, the first (larger) level becomes more clear for low μ_1, while the second (smaller) level becomes more clear for larger μ_1. This is both the case for a recent hierarchical version of the Infomap method [45] and the CPM method. The Infomap method seems to be slightly better at detecting the planted communities, but the CPM method remains highly competitive. The possibility for having various scales of description of the network seems important, as many networks seem to have at least some hierarchical structure.

References

1. Aldecoa R, Marín I (2013) Surprise maximization reveals the community structure of complex networks. Sci Rep 3:1060. doi:10.1038/srep01060
2. Arenas A, Fernandez A, Gomez S (2007) Analysis of the structure of complex networks at different resolution levels. New J Phys 10(5):23. doi:10.1088/1367-2630/10/5/053039. arXiv:physics/0703218
3. Bichot CE, Siarry P (2011) Graph partitioning. Wiley, New York. ISBN 184821233X
4. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech Theory Exp 2008(10):P10008. doi:10.1088/1742-5468/2008/10/P10008
5. Bollobás B (2001) Random graphs, 2nd edn. Cambridge University Press, Cambrige
6. Brandes U, Delling D, Gaertler M, Goerke R, Hoefer M et al (2006) Maximizing modularity is hard. arXiv:physics 0608255. arXiv:physics/0608255
7. Brandes U, Delling D, Gaertler M, Gorke R, Hoefer M et al (2008) On modularity clustering. IEEE Trans Knowl Data Eng 20(2):172–188. doi:10.1109/TKDE.2007.190689
8. Clauset A, Newman M, Moore C (2004) Finding community structure in very large networks. Phys Rev E 70(6):1–6. doi:10.1103/PhysRevE.70.066111
9. Cover, TM and Thomas, JA (2012). Elements of Information Theory, vol 2012. Wiley, New York. ISBN 1118585771
10. Danon L, Díaz-Guilera A, Duch J, Arenas A (2005) Comparing community structure identification. J Stat Mech Theory Exp 2005:P09008. doi:10.1088/1742-5468/2005/09/P09008
11. Delvenne JC, Yaliraki SN, Barahona M (2010) Stability of graph communities across time scales. Proc Natl Acad Sci USA 107(29):12755–12760. doi:10.1073/pnas.0903215107
12. Diestel R (2010) Graph theory, 4h edn. Springer, Berlin. ISBN 978-3-642-14278-9
13. Doreian P, Batagelj V, Ferligoj A (2005) Generalized blockmodeling. Cambridge University Press, Cambridge. ISBN 9780521840859
14. Duch J, Arenas A (2005) Community detection in complex networks using extremal optimization. Phys Rev E 72(2):1–4. doi:10.1103/PhysRevE.72.027104
15. Everitt BS, Landau S, Leese M (2001) Cluster analysis. Wiley, New York. ISBN 9780340761199
16. Fortunato S (2010) Community detection in graphs. Phys Rep 486(3–5):75–174. doi:10.1016/j.physrep.2009.11.002
17. Fouss F, Pirotte A, Renders JM, Saerens M (2007) Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Trans Knowl Data Eng 19(3):355–369. doi:10.1109/TKDE.2007.46

18. Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci USA 99(12):7821–7826. doi:10.1073/pnas.122653799
19. Good BH, de Montjoye YA, Clauset A (2010) Performance of modularity maximization in practical contexts. Phys Rev E 81(4):046106. doi:10.1103/PhysRevE.81.046106
20. Guimerà R, Mossa S, Turtschi A, Amaral LAN (2005) The worldwide air transportation network: anomalous centrality, community structure, and cities' global roles. Proc Natl Acad Sci USA 102(22):7794–7799. doi:10.1073/pnas.0407994102
21. Guimerà R, Nunes Amaral LA (2005) Functional cartography of complex metabolic networks. Nature 433(7028):895–900. doi:10.1038/nature03288
22. Guimerà R, Sales-Pardo M, Amaral L (2004) Modularity from fluctuations in random graphs and complex networks. Phys Rev E 70(2):025101. doi:10.1103/PhysRevE.70.025101
23. Jain A, Dubes R (1988) Algorithms for clustering data. Prentice-Hall, Englewood Cliffs. ISBN 978-0130222787
24. Jaynes E (1957) Information theory and statistical mechanics. Phys Rev 106(4):620–630. doi:10.1103/PhysRev.106.620
25. Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J 49(1):291–307
26. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science (NY) 220(4598):671–680. doi:10.1126/science.220.4598.671
27. Kolaczyk ED (2009) Statistical analysis of network data: methods and models. Springer, Berlin. ISBN 9780387881461
28. Lambiotte R, Delvenne JC, Barahona M (2008) Laplacian dynamics and multiscale modular structure in, networks, pp 1–29. arXiv:0812.1770
29. Lancichinetti A, Fortunato S (2009) Community detection algorithms: a comparative analysis. Phys Rev E 80(5):056117. doi:10.1103/PhysRevE.80.056117
30. Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):46110. doi:10.1103/PhysRevE.78.046110
31. Lancichinetti A, Radicchi F, Ramasco JJ, Fortunato S (2011) Finding statistically significant communities in networks. PloS ONE 6(4):e18961. doi:10.1371/journal.pone.0018961
32. Leicht E, Newman M (2008) Community structure in directed networks. Phys Rev Lett 100(11):1–4. doi:10.1103/PhysRevLett.100.118703
33. MacKay D (2003) Information theory, inference and learning algorithms. Cambridge University Press, Cambridge. ISBN 9780521642989
34. Meilă M (2007) Comparing clusterings-an information based distance. J Multivar Anal 98(5):873–895. doi:10.1016/j.jmva.2006.11.013
35. Newman M (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6). doi:10.1103/PhysRevE.69.066133
36. Newman M (2010) Networks: an introduction. Oxford University Press, Oxford. ISBN 0199206651
37. Newman M, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113. doi:10.1103/PhysRevE.69.026113
38. Newman MEJ (2006) Finding community structure in networks using the eigenvectors of matrices. Phys Rev E 74(3):036104+. doi:10.1103/PhysRevE.74.036104
39. Porter MA, Onnela JP, Mucha PJ (2009) Communities in networks. Not AMS 56(9):1082–1097. arXiv:0902.3788
40. Raghavan U, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106. doi:10.1103/PhysRevE.76.036106
41. Reichardt J, Bornholdt S (2006) Statistical mechanics of community detection. Phys Rev E 74(1):016110+. doi:10.1103/PhysRevE.74.016110
42. Reichardt J, White DR (2007) Role models for complex networks. Eur Phys J B 60(2):217–224. doi:10.1140/epjb/e2007-00340-y
43. Ronhovde P, Nussinov Z (2010) Local resolution-limit-free Potts model for community detection. Phys Rev E 81(4):046114. doi:10.1103/PhysRevE.81.046114. arXiv:0803.2548v4

44. Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad SciUSA 105(4):1118–1123. doi:10.1073/pnas.0706851105
45. Rosvall M, Bergstrom CT (2011) Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. PloS ONE 6(4):e18209. doi:10.1371/journal.pone.0018209. arXiv:1010.0431
46. Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64. doi:10.1016/j.cosrev.2007.05.001
47. Theodoridis S, Koutroumbas K (2006) Pattern recognition. Academic Press, New York. ISBN 9780080513614
48. Tibély G, Kertész J (2008) On the equivalence of the label propagation method of community detection and a Potts model approach. Phys A Stat Mech Appl 387(19–20):4982–4984. doi:10.1016/j.physa.2008.04.024
49. Traag VA, Van Dooren P, Nesterov Y (2011) Narrow scope for resolution-limit-free community detection. Phys Rev E 84(1):016114. doi:10.1103/PhysRevE.84.016114. arXiv:1104.3083
50. Wasserman S, Faust K (1994) Social network analysis. Cambridge University Press, Cambridge
51. Xu R, Wunsch D (2008) Clustering. Wiley, New Jersey. ISBN 9780470382783
52. Yen L, Fouss F, Decaestecker C, Francq P, Saerens M (2009) Graph nodes clustering with the sigmoid commute-time kernel: a comparative study. Data Knowl Eng 68(3):338–361. doi:10.1016/j.datak.2008.10.006

http://www.springer.com/978-3-319-06390-4

Algorithms and Dynamical Models for Communities and Reputation in Social Networks

Traag, V.

2014, XIV, 229 p. 40 illus., 19 illus. in color., Hardcover
ISBN: 978-3-319-06390-4