Scalable Algorithms for Community Detection in Very Large Graphs

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)
von der Fakultät für
Wirtschaftswissenschaften
am Karlsruher Institut für Technologie

genehmigte
Dissertation

von
Dipl.-Inform.Wirt Michael Ovelgönne

Tag der mündlichen Prüfung: 14. Juli 2011
Referent: Prof. Dr. Andreas Geyer-Schulz
Korreferent: Prof. Dr. Karl-Heinz Waldmann

Karlsruhe, 2011

Contents

List of Figures

List of Tables

List of Algorithms

1 Motivation

In various scientific disciplines data gets modeled as networks which describe the entities of some system and one or more relations between them. Many facts can naturally be described with this kind of model: Students and their friendships, countries and their trade relations, proteins and their interactions. Recently, huge sources of linked data originate from the so-called Web 2.0. The bulk of user-generated content on the World Wide Web (WWW) usually consists of text, picture or video contributions that are linked to their contributor and to others who e.g. bookmarked it. Articles on the online encyclopedia Wikipedia link to related articles, users of the social network site Facebook link e.g. to their friends, schools and favored movies, and via backlinks different blogs are interconnected. Another source of linked data is the semantic web where e.g. personal profile documents according to the Friend-of-a-Friend (FOAF) specification create a network of people, organizations and projects. To make use of the vast amount of linked data, algorithms are necessary that scale to the large size of these datasets. If for example social scientists want to analyze user behavior on social media sites they need network analysis tools that are capable of processing networks of at least hundreds of thousands of actors. Tools designed in the age of survey-originating data, where networks usually have at most a few hundred actors, are not suitable to analyze the much larger communities on the web.

In many settings we observe that the connections between entities form structures. Email networks will show dense groups of people which might be friends or colleagues and a Wikipedia cross reference network shows densely connected groups of articles that possibly belong to the same domain. Clustering networks means to identify these *natural groups* or *communities*. Cluster analysis gives insights into the structures of networks by identifying groups of closely related entities. Thus, cluster analysis reduces the complexity of a network by creating a smaller representation of a large graph that has the same structure of modules. Overall, clustering can be regarded as a general and widely-applicable analysis method. Because of its numerous fields of application, research on clustering methods is conducted in many fields of science, e.g. statistics, mathematics, oper-

ations research, physics, computational biology, sociology and computer science.

Many graph clustering algorithms require the knowledge of the number of clusters in advance. Other clustering methods generate a hierarchical structure of clusters and sub-clusters. Newman and Girvan [NG04] introduced the modularity function as a measure for the quality of a clustering. This function measures how different the cluster densities are from the densities of the same clusters when the edges of the graph are randomly rewired while maintaining the same degree distribution. Modularity clustering has the advantage that the measure can be used as an objective function that can be maximized. Modularity became popular very quickly as the measure seems to be easy to understand and its applicability for clustering can been analyzed independently from the analysis of algorithms that maximize it. Many researcher with experience in algorithm design could work on improved modularity maximization algorithms without the need to verify whether the results represent useful clusterings.

Maximizing the modularity of a graph clustering computationally is a challenging task. The complexity of finding a clustering with maximal modularity is in the class of NP-hard problems [BDG+08]. This makes it impossible to compute the optimal solution for larger networks in reasonable time. Even most of the so far proposed heuristics are not capable of processing very large datasets with millions of vertices and edges in reasonable time. This thesis will deal with the problem of developing a fast and scalable algorithm for finding a high modular clustering for very large graphs. Scalability will be considered in terms of time complexity as well as in terms of memory complexity. In this thesis clustering is primarily considered as a large combinatorial optimization problem. The applicability of specific clustering algorithms for solving scientific or industrial problems is only a minor aspect of this work.

The need to be able to analyze very large networks arises from the size of networks that have to be analyzed for current and upcoming services and systems. For example, the sizes of telephone call networks or friendship networks from social network site, whose cluster analysis is e.g. required for emergency assistance services as discussed in [GSOS10, OSGS10], will probably consist of at least tens of millions of subscribers as graph vertices and billions of calls indicating the edges. So, scalability is a major issue for algorithm design. Although a lot of algorithms have been presented in recent years, only a very few of them are capable of processing huge networks. An algorithm that is fast and memory efficient as well as capable of finding high quality clusters was missing. In this thesis such

algorithms will be presented. Furthermore, the results of the analysis of these algorithms and the underlying heuristic maximization strategies for the modularity measure are believed to be useful for the development of other optimization algorithms for combinatorial problems.

This thesis is organized as follows. In Chapter 2 a general introduction to graph clustering will be given. Then, the class of modularity-based graph clustering algorithms will be discussed in Chapter 3. The main contribution of this thesis, two innovative randomized greedy modularity clustering algorithms, will be introduced in Chapter 4. Finally, Chapter 5 provides a summary and an outlook.

Note on self-citations

Parts of the work described in this thesis have been previously published. The RG algorithm has been presented first at the 4th ACM/SIGKDD Workshop on Social Network Mining and Analysis [OGSS10]. The extension of RG, RG+, has been presented first at the 2010 IEEE Workshop on Optimization Based Methods for Emerging Data Mining Problems [OGS10]. Finally, the analysis of merger processes of agglomerative hierarchical modularity clustering algorithms has been presented at the 34th Annual Conference of the German Classification Society [OGS11]. Text, tables, figures and algorithms have been partly adopted without modifications and partly with modifications. Especially Chapter 4 on the algorithms RG and RG+ heavily draws from the previous publications.

2 Introduction to Graph Clustering

This chapter provides an overview of literature on data clustering in general and graph clustering in particular. The objective of this chapter is to introduce the preliminaries for the in-depth discussion of modularity maximization algorithms in Chapter 3.

Clustering will be introduced as a technique to group similar patterns (objects, data points). The result of a clustering technique will also be denoted as a clustering. A clustering where each pattern belongs to exactly one group will be denoted as a partition. As will be discussed later on, clustering is a generic technique that can be employed to solve various problems. Given that clustering techniques have now a history of over one hundred years (compare Section 2.1) and gained increasing popularity, it is hard to give an overview that truly respects the many different views various scientific fields developed. As the major contribution of this thesis is of algorithmic nature, the view on clustering may be biased from that point of view.

The classic clustering techniques like linkage-based algorithms [Sne57, SS63, MS57] and k-means [Mac67] have been developed to group data points in an Euclidean space. In the Euclidean space, each data point is given by its coordinates and the distance measure is defined between the points in space. In a graph, each data point (vertex) is characterized just by the links to its neighbors, the coordinates in space are not given.

Graph clustering is getting increasing attention. The availability of very large sets of network data fostered the research on clustering techniques and their application to different types of problems. The identification of 'natural groups' helps to understand the structure of connections between the entities of a network. In addition, services like recommendation systems can be built on the basis of this technology.

Work in the field of graph clustering is conducted and discussed in several scientific communities: the *complex networks* community (see e.g. *Proceedings of the National Academy of Sciences of the United States of America*, *Physical Review E* by American Physical Society, *New Journal of*

Physics by Institute of Physics and Deutsche Physikalische Gesellschaft), the *data mining* community [RZ07] (see e.g. *ACM SIGKDD Conference on Knowledge Discovery and Data Mining* and *IEEE International Conference on Data Mining* series) and the *social networks* community (see e.g. *Social Networks* by Springer). The origin of the quality measure modularity and most approaches for this measure's optimization come from complex networks community. This community mostly refers to clustering as community detection.

As the application of graph clustering is not limited to the mentioned communities, the research on this topic is even more wide-spread. In bioinformatics, cluster analysis is used for classification of gene expression networks [XOX02] and protein interaction networks [BH03]. In logistics, for example methods to solve the *uncapacitated facility location problem* are based on graph clustering [CS03]. Overviews of applications of clustering techniques provide for example Schaeffer [Sch07] and Xu and Wunsch [XW05].

2.1 Clustering

A good generic definition of clustering (also referred to as cluster analysis) is provided by Jain et al. [JMF99, p. 1]: "Clustering is the unsupervised classification of patterns (observations, data items or feature vectors) into groups (clusters)". Independently of what the patterns actually are, those patterns that are similar with regards to some measure, should be classified into a common group and those patterns that are not similar should be classified into different groups.

Although some authors use a different denotation and speak e.g. of 'supervised clustering', in this thesis clustering is regarded, especially in distinction to supervised classification approaches, as a method that searches for groups of similar patterns without additional information like already partly grouped patterns.

Clustering data with help of computers started with Sneath's [Sne57] seminal work in 1957. However, first numerical classification methods have been described by Czekanowski [Cze09] in 1909 and by Kulczynski [Kul27] in 1927 (see [LL95]). A complete historical overview of (graph) clustering is out of the scope of this thesis. A good summary of the rich and diverse history of classification is provided by [JMF99]. Porter et al. [POM09] provide a short historical overview on community identification in social networks, the term used in the social sciences for clustering graph data,

from a perspective of that field.

The definition of clustering from above leads to the first problems. First, clustering requires an explicit or implicit similarity measure, second, a rule is required to decide if the similarity is strong enough to classify two patterns into the same group. While in some settings the (desired) number of clusters is known, in other settings the number of clusters has to be determined along with the assignment of patterns into groups.

2.1.1 Clustering with Distance Measures

Classic data clustering works with patterns that can be represented as points in an Euclidean space [JMF99]. The i-th pattern is represented by a d-dimensional feature vector $x_i = (x_{i,1}, \cdots, x_{i,d})$. The distance $d(x_i, x_j)$ between the patterns x_i and x_j is used for the determination of clusters. Problems of this kind of clustering are e.g. how to represent a pattern in the Euclidean space and which measure to use to determine the distances. An example for a clustering in a two-dimensional space is shown in Figure 2.1. A small example shows the problem of determining distances. Let us assume each data point represents a product, property 1 is the product price in Euro, property 2 is the number of sold units and the distance measure is the Euclidean distance. What would happen when the price is not given in Euro but in British Pound or U.S. Dollar? As only one axis would be scaled a non-linear transformation of the distances would occur. That could e.g. lead to a situation where points that previously would have been assigned to the same cluster are now in different clusters.

This example is a warning of oversimplified treatment of cluster analysis and its results. But assumed that the defined distances and the scaling of the dimensions of the feature space provide a good measure for the (dis-) similarity of patterns, clustering algorithms can find useful groups. As examples for clustering algorithms, the two well-known representatives k-means and DBSCAN will be introduced in the following.

One of the most popular clustering techniques is k-means [Mac67]. For a given constant k, k-means tries to identify k clusters in such a way that the sum of the squared distances of all data points to their respective cluster centroid is minimal. Here, the cluster centroid is defined as the arithmetic mean of each data point of the cluster for all dimensions. Let $X = \{x_1, ..., x_n\}$ be a set of data points, the k-means problem is to find a clustering $\{C_1, ..., C_k\}$, $C_t \subset X$, $\cup_t C_t = X$ so that the sum of squared distances distances between the data points x_i and the cluster centroid c_t is minimal. The function that has to be minimized is

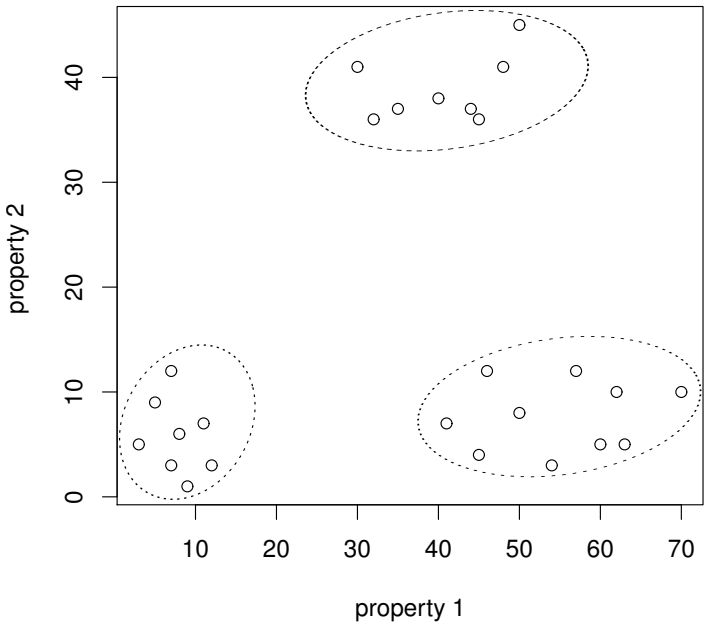

Figure 2.1: Example of a clustering of points in a two-dimensional space. The dashed ellipses indicate clusters.

$$d = \sum_{t} \sum_{x_i \in C_t} d(x_i, c_t)^2. \tag{2.1}$$

Finding the partition with the minimal value of d is NP-hard [ADHP09]. The most widely used heuristic for this problem is the one developed by Llyod [Llo82], which in turn has been optimized in several ways (e.g. [KMN+02]). K-means partitions the set of data points into non-overlapping clusters. The algorithm belongs to the class of distance-based algorithms as the distance is the only factor taken into account.

Another well-known clustering algorithm is DBSCAN (Density-Based Spatial Clustering of Applications with Noise) by Ester et al. [EKSX96]. As the algorithm's name says this algorithm searches for dense regions of the data space and so belongs to the class of density-based algorithms. Unlike k-means DBSCAN does not assign every data point to a cluster but marks those points which are not in a dense region as noise. Another difference is that the number of clusters does not need to be known in advance.

DBSCAN clusters are defined as follows. Let D be a set of data points and $d(p, q)$ be the distance between the points $p, q \in D$. The

ϵ-neighborhood $N_\epsilon(p) = \{q \in D | d(p,q) \leq \epsilon\}$ of a point p are all other points in the maximal distance of ϵ. A point p is *directly density-reachable* from a point q ($ddr_q(p)$) if p is in q's ϵ-neighborhood and the size of q's ϵ-neighborhood is at least $MinPts$, where $MinPts$ is a parameter:

$$ddr_q(p) \Leftrightarrow p \in N_\epsilon(q) \wedge |N_\epsilon(q)| > MinPts \tag{2.2}$$

If p is not directly density-reachable from q but there is a chain of other directly density-reachable points connecting them, then p and q are *density-reachable* (denoted as $dr_q(p)$). Furthermore, two points p and q are *density-connected* (denoted as $dc(p,q)$) if there is a point o so that both are density-reachable from o. A cluster is then defined as a non-empty subset C of D which follows two conditions:

1. Maximality:

$$\forall p, q \in D : p \in C \wedge dr_p(q) \Rightarrow q \in C \tag{2.3}$$

2. Connectivity:

$$\forall p, q \in C : dc(p,q) \tag{2.4}$$

The procedure of DBSCAN is quite simple. The algorithm visits every point $p \in D$. If p has not been visited before, it either marks p as noise (if $|N_\epsilon(p)| < MinPts$) or creates a new cluster around p and adds all points in p's neighborhood to the new cluster (if they are not already part of another cluster). With each visited point p' in p's neighborhood, the neighborhood of p gets expanded by the neighborhood of p' when p' is directly density-reachable from p.

K-means and DBSCAN are only two examples of algorithms for clustering with a distance measure. A complete overview of cluster algorithms for the Euclidean space and related aspects of cluster analysis is out of the scope of this thesis. Several good introductory works provide this overview - see for example [JMF99].

2.2 Graph Clustering

The second large sub-domain of clustering is the clustering of graphs. Here, clustering means the identification of dense subgraphs that are only

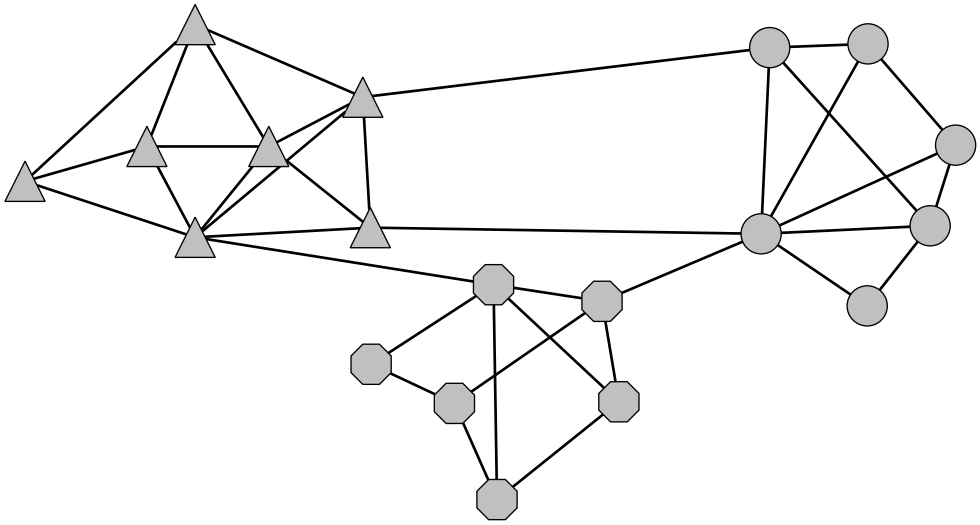

Figure 2.2: Example of a graph clustering. Vertex shapes represent clusters.

sparsely interconnected with each other. Graph clustering differs from distance-based clustering as the distance is not known for all pairs of patterns. Instead, for graph clustering the proximity is given only for some pairs of patterns. An example of a graph clustering is given in Figure 2.2.

Usually, graph clustering means the clustering of vertices but lately it has been argued that for some types of graphs the edges build natural groups rather than the vertices [ABL10]. But there is a duality between vertices and edges, too. Graphs can be transformed into dual graphs by creating a vertex for every edge from the dual graph and creating links between all new vertices when two edges in the dual graph are connected to the same vertex. In this thesis, all considerations are restricted to the clustering of vertices. Next, the graph and graph clustering terminology for the rest of this thesis will be introduced. Afterwards, an overview of algorithms for graph clustering will be given.

For further reading two recent and very comprehensive review articles on graph clustering published by Schaeffer [Sch07] and Fortunato [For10] are highly recommended. Both give an overview on graph clustering techniques as well as on related topics like evaluating and benchmarking clustering methods and applications of graph clustering. Another good introductory work has been written by Gaertler [Gae05].

2.2.1 Terminology Graphs

In the following paragraphs the most important graph terminology for this thesis will be introduced.

Graph A graph in terms of graph theory is a tuple $G = (V, E)$ with $V = \{v_1, ..., v_n\}$ the set of vertices and $E = \{e_1, ..., e_m\}$ the set of edges. Accordingly, $n = |V|$ is the number of vertices of G and $m = |E|$ the number of edges.

Undirected and directed edges A directed graph is a graph where the edges have directions, i.e. traversing an edge is only possible in one direction. In contrast the undirected edges of undirected graphs can be traversed in both directions.

An undirected edge is denoted by $\{v_x, v_y\}$ and a directed edge (arc) by (v_x, v_y) with $v_x, v_y \in V$.

Vertex degree For undirected graphs, the number of edges a vertex is connected to is called the vertex degree: $d(v_x) = |\{\{v_x, v_y\} \in E, v_y \in V\}|$.

In directed graphs, each vertex has an indegree $d^{in}_{v_x} = |\{(v_y, v_x) \in E, v_y \in V\}|$ and an outdegree $d^{out}_{v_x} = |\{(v_x, v_y) \in E, v_y \in V\}|$ which describe the number of edges that end, respectively, start at vertex v_x.

Neighbor The neighbors of a vertex v_x are all other vertices that are connected to v_x by an edge. That means, for undirected graphs the neighbors of vertex v_x are $N(v_x) = \{v_y \in V | \{v_x, v_y\} \in E\}$ and for directed graphs the neighbors are $N(v_x) = \{v_y \in V | (v_x, v_y) \in E \vee (v_y, v_x) \in E\}$.

Successor and predecessor In directed graphs the set of neighbors can be divided into two subsets: the successors and the predecessors. The successors of v_x, $Succ(v_x) = \{v_y \in V | (v_x, v_y) \in E\}$, are all vertices where an edge starting in v_x ends. Correspondingly, the predecessors of v_x, $Pred(v_x) = \{v_y \in V | (v_y, v_x) \in E\}$, are all vertices where an edge ending in v_x starts.

Weighted graphs A graph is called weighted when there are weights assigned to the edges. That means, the graph is a triple $G = (V, E, \omega)$ with the weight function $\omega : E \to X$ and X is an arbitrary number system,

e.g. real values $\mathbb{R}$. If no weights are attached to the edges a graph is called unweighted.

Loop An edge who's endpoints are both attached to the same vertex is called a loop: $\{v_x, v_x\}$.

Simple graph A graph is called simple when it is undirected, loop-free and there is at most one edge between any distinct pair of edges.

Path A path is a series of vertices which are connected by edges. Two vertices $v_1, v_t \in V$ are connected by a path when there is a series of vertices $v_1, v_2, \ldots, v_{t-1}, v_t$ with $\forall i \in 1, \ldots, t-1 : \{v_x, v_{x+1}\} \in E$. The number of vertices of a path is called the length of the path.

In the case of directed graphs, two vertices $v_1, v_t \in V$ are connected by a path when there is a series of vertices $v_1, v_2, \ldots, v_{t-1}, v_t$ with $\forall i \in 1 \ldots t-1 : (v_x, v_{x+1}) \in E$.

Shortest path The shortest path between a pair of vertices is that path between them with the minimum number of vertices in between. Of course, there can be several shortest paths between two vertices. In the following the length of the shortest path between two vertices $v_1, v_2 \in V$ is denoted by $sp(v_1, v_2)$.

Connected graph An undirected graph is called connected when there is a path between any pair of vertices. A directed graph is called weakly connected when there is a path between any pair of vertices when the edge directions are disregarded. A directed graph is called strongly connected when there is a path between any pair of vertices while taking the edge directions into account.

Diameter The diameter of a graph is the maximal length of any shortest path between two vertices of the network: $diam(G) = \max_{v_x, v_y \in V} sp(v_x, v_y)$.

Clique A clique is a complete subgraph, i.e. a set of vertices with an edge between every pair of vertices. A maximal clique is a clique that is not part of a larger clique.

Density The term density refers to the ratio of edges of a graph to the maximal number of possible edges. The number of edges of a dense graph is near to the maximum number of edges while the number of edges in a sparse graph is far below this number. The maximum number of edges in a simple graph with n vertices and m edges is $n(n-1)/2$ and therefore the density D^{ud} is

$$D^{ud} = \frac{2m}{n(n-1)}. \tag{2.5}$$

In a directed graph without loops or multiple edges, the maximum number of edges is twice the number of the undirected variant as edges can go in both directions. Accordingly, the density D^d of directed graphs is

$$D^d = \frac{m}{n(n-1)}. \tag{2.6}$$

2.2.2 Terminology Graph Clustering

The following terms related to graph clustering will be used in this thesis:

Non-overlapping clustering A clustering of a graph is the assignment of vertices to groups of vertices called clusters. A non-overlapping clustering $C = \{C_1, ..., C_k\}$ is a partition of the vertices of a graph $G = (V, E)$ into groups $C_i \subset V$ so that $\forall i, j, i \neq j : C_i \cap C_j = \emptyset$ and $\cup_i C_i = V$.

Overlapping clustering An overlapping clustering $C^o = \{C_1^o, ..., C_k^o\}$ is a set of clusters $C_i^o \subset V$ with $\cup_i C_i^o = V$ and $\exists i, j : C_i^o \cap C_j^o \neq \emptyset$.

Cluster size The cluster size denotes the number of vertices a cluster consists of: size of cluster $C_i := |C_i|$.

Singleton A cluster of size 1, i.e. a cluster that consists of exactly one vertex, is called singleton or singleton cluster.

Intra-cluster edge An edge $\{v_x, v_y\}$ is called intra-cluster edge, if it connects vertices within the same cluster: $v_x \in C_i, v_y \in C_j \Rightarrow i = j$. The number of intra-cluster edges of a vertex $v_x \in C_i$ is denoted as $d_{v_x}^{in}(C_i)$.

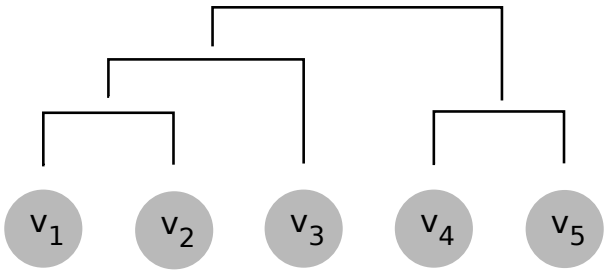

Figure 2.3: Example of a dendrogram

Inter-cluster edge An edge $\{v_x, v_y\}$ is called inter-cluster edge, if it connects vertices from different clusters $v_x \in C_i, v_y \in C_j \Rightarrow i \neq j$. The number of inter-cluster edges of a vertex $v_x \in C_i$ is denoted as $d_{v_x}^{out}(C_i)$.

Dendrogram A dendrogram is a figure showing the decomposition of a hierarchical cluster structure with clusters and subclusters. An example of a dendrogram is shown in Figure 2.3.

2.2.3 Graph Properties

Graphs as the representations of networks can be described by various structural properties. The two basic properties diameter and density have already been introduced in Section 2.2.1. In this section, the properties degree distribution, average path length and transitivity will be discussed. Natural, self-organized networks have characteristic values of these properties. As we will discuss later on in Chapter 4, the structural properties of graphs are important for the design of graph clustering algorithms.

The entities and links of a network can basically arise from two different sources. First, a network can result from a central planning process. For example, a country's railroad system is (ideally) the result of a route optimization process. An employee of the railroad company takes factors like travel demand and geography into account when deciding which cities should be connected. In contrast to this central planning process, the friendship network of a social network site is created decentralized and self-organized. The decision about the creation of a link is made by the involved persons individually.

The properties of natural networks have been the subject of intensive research. Albert and Barabási [AB02], Newman [New03] as well as Leskovec et al. [LLDM08] published comprehensive reviews on the structure of nat-

ural networks. Natural networks usually evolve over time and the static networks that get analyzed are snapshots of the entities and their connections at some point in time. Some properties of natural networks are a result of an evolutionary generation process. For example, when two persons A and B are friends, and B is also a friend of C, it is more likely that A and C become friends in the future than in a situation where they do not have a common friend B.

In the following, some important properties of natural networks will be discussed.

2.2.3.1 Degree Distribution

The degree distribution in simple undirected networks is the distribution of the number of neighbors of a vertex. In directed graphs, each vertex has an indegree and an outdegree. Accordingly, directed graphs have an indegree distribution and an outdegree distribution.

The following description of the degree distribution is for undirected graphs but applies analogous to indegree and outdegree distributions of directed graphs. Let p_k be the fraction of vertices with a degree k, i.e. p_k is the probability that a randomly picked vertex has degree k. The analysis of many natural networks revealed that they usually have a power-law degree distribution [AB02]. That means, that there is a α so that

$$p_k \sim k^{-\alpha}. \tag{2.7}$$

Newman [New03] reports α values between 2 and 3 for various types of networks. But not all networks have a power-law degree distribution. In each network category (social, information, technological and biological), Newman found some networks that have a power-law degree distribution and some that do not have a power-law degree distribution. Among the analyzed social networks are networks like a film actors network (links between actors that have appeared in the same movie), which has a power-law degree distribution and networks like co-authorship networks (links between researchers that have co-authored a paper), which do not have this kind of degree distribution. But even those natural networks without a power-law distribution have still a highly skewed degree distribution with predominantly low-degree vertices and few high-degree vertices.

2.2.3.2 Average Path Length

While the diameter is the maximal length of the shortest path between any pair of vertices the average path length

$$apl(G) = \frac{1}{n(n-1)/2} \sum_{v_x, v_y \in V, i>j} sp(v_x, v_y). \tag{2.8}$$

is the average length of the shortest paths connecting any two vertices in V.

The low average path length in real-world social networks is called the small-world phenomenon. This term goes back to the famous experiment of Milgram [Mil67] in the 1960s. Milgram asked participants of his experiment to forward letters to a designated target person by sending the letter to somebody they know and who is likely to be "closer" to the target person. The result of the experiment was that the letter reached the target in only about six steps.

This property of a low average path length despite of a huge number of vertices can be found in many natural graphs. And as every vertex is kind of nearby from any position in the graph, cluster analysis can not easily separate different regions. Leskovec et al. [LLDM08] report for various social networks average path lengths between about 4 and 8.

The small-world phenomenon is popularly known by two "numbers": the Erdős number and the Bacon number. The Erdős number gives the co-authoring distance to the mathematician Paul Erdős and the Bacon number gives the co-appearing distance to the movie actor Kevin Bacon.

2.2.3.3 Transitivity

The transitivity or clustering effect is a property of many natural graphs that is most comprehensibly described by the example of friendship: When Alice is friend with Bob and Charly, there is a high probability that Bob and Charly are friends as well.

The clustering coefficient measures how densely connected the neighborhoods of the graph's vertices are. Different formulations for the clustering coefficient CC of a graph $G = (V, E)$ have been proposed [New03]. The global clustering coefficient compares the total number of triangles in the graph to the total number of connected triples of vertices:

$$CC_g(G) = \frac{3 \cdot \text{number of triangles in G}}{\text{number of connected triples in G}} \tag{2.9}$$

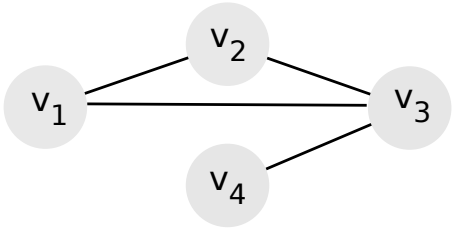

Figure 2.4: The vertex triples $\{v_1, v_2, v_3\}$, $\{v_1, v_3, v_4\}$ and $\{v_2, v_3, v_4\}$ are connected triples, but only the first triple is also a triangle.

In effect, the clustering effect measures the ratio of the number of triple where all three possible edges exist to the number of triples where at least two edges exist (compare Figure 2.4). Each triangle (v_x, v_y, v_k) with the edges $\{v_x, v_y\}$, $\{v_y, v_k\}$ and $\{v_k, v_x\}$ has to be counted three times because it belongs to three connected triples (v_x, v_y, v_k), one with the edges $\{v_x, v_y\}$ and $\{v_y, v_k\}$, one with the edges $\{v_y, v_k\}$ and $\{v_k, v_x\}$, and one with the edges $\{v_k, v_x\}$ and $\{v_x, v_y\}$.

In contrast to the global variant, the local variant of the clustering coefficient first normalizes the number of triangles connected to a vertex with the number of triples connected to the vertex and aggregates the results afterwards:

$$CC_l(G) = \frac{1}{n} \sum_{v_x \in V} \frac{\text{number of triangles connected to } v_x}{\text{number of triples connected to } v_x} \tag{2.10}$$

The transitivity (measured by the local clustering coefficient) is very high for collaboration networks (between about 0.6 and 0.8) and lower for contact-/friendship networks (between about 0.1 and 0.3) [LLDM08].

2.2.4 Graph Clusters as Cohesive Subgroups

At the beginning of Chapter 2, clustering has been defined as the classification of patterns into groups where similar patterns should be classified into the same group and non-similar patterns should be classified into different groups. For a graph clustering this generic definition can be specialized to: a cluster should be a subgroup of vertices that is densely connected inside and weakly connected to the rest of the graph.

Wasserman and Faust [WF94] list several notions of cohesive subgroups. The most tightly interconnected subgroup is a clique where each vertex in the subgroup is connected with all other vertices in the subgroup. A less

strict definition than the clique is a k-plex, where all vertices in a subgroup of size s have to be connected to at least $s - k$ other subgroup members. Further definitions are based on the nodal-degree, the reachability or the diameter.

But there are other definitions, too. Radicchi et al. [RCC+04] define a community in the *strong sense* as a cluster $C_i \subset V$ where every vertex in that cluster has more intra-cluster edges than inter-cluster edges:

$$d_{v_x}^{in}(C_i) > d_{v_x}^{out}(C_i), \forall v_x \in C_i \tag{2.11}$$

Furthermore, they define a community in the *weak sense* as a cluster where the total number of intra-cluster edges exceeds the total number of inter-cluster edges:

$$\sum_{v_x \in C_i} d_{v_x}^{in}(C_i) > \sum_{v_x \in C_i} d_{v_x}^{out}(C_i) \tag{2.12}$$

Zhang et al. [ZWW+09] added the definition of clusters in the *most weak sense* to this classification. A cluster in the most weak sense is a cluster where twice the number of intra-cluster edges is never less than the number of edges joining the cluster to any other cluster:

$$2 \cdot \sum_{v_x \in C_i} d_{v_x}^{in}(C_i) > \sum_{v_x \in C_i} d_{v_x}^{out}(C_i) \tag{2.13}$$

From the lack of a clear definition of what a cohesive subgroup or community is follows the lack of the generally accepted definition of the quality of a clustering (as will be discussed later in Section 2.2.7).

2.2.5 Cluster Hierarchy

The generic cluster definition demands intra-cluster density and inter-cluster sparsity. If a set of patterns has been clustered in such a way, one might experience that the patterns inside one group are similar in relation to their degree of dissimilarity to the other groups of patterns, but still not all patterns within one group are equally similar to each other. Then, groups of patterns can be divided into smaller subgroups, where the degree of similarity of the patterns of one subgroup is even higher than in the super-group.

That many natural networks consist of communities on different levels of granularity becomes obvious when thinking of the usual communication

structures most people will experience every day. For example, think of a communication network in an organization. The members of a team will probably interact very frequently. The members of different teams of one department will probably communicate less frequently. But the level of communication of members of different departments will be even lower.

An early discussion of the hierarchy of systems and subsytems is Simon's seminal work on complex systems from 1962 [Sim62]. Lately, this topic got increasing attention. Hierarchies of clusters have been identified for example in metabolic networks [RSM+02, SPGMA07], communication networks [SPGMA07] and networks of species [CMN08]. Likewise, the router-level topology of the Internet is hierarchical [LAWD04].

In contrast to these works, Ahn et al. [ABL10] see the hierarchical network structures as a hierarchy of groups and subgroups of edges and not of vertices. Regardless of whether vertices or edges actually form hierarchical structures, there is much evidence for natural hierarchies in the topology of networks.

The clustering hierarchy is in so far an interesting property of many natural networks as it challenges cluster analysis. Does it make sense to calculate a flat clustering for "naturally" hierarchical structures? If yes, how to determine clusters at the "correct" level of granularity? How to validate that all clusters are on the same level of granularity? And first of all, how to determine whether a network has a hierarchical cluster structure or not? Although these questions are very interesting, they are out of the scope of this thesis.

2.2.6 Graph Clustering Algorithms

An example for an early motivation for partitioning graphs is the problem that motivated the development of the well-known algorithm of Kernighan and Lin [KL70]. Their problem was to assign the components of electronic circuits to circuit boards so that the number of connections between the boards is minimal (minimum-cut problem). This problem is different from the problem discussed in the following as the number of clusters is predefined. In other related problems the maximal size of a cluster or restrictions on the size of the clusters (e.g. equal size) might be given. In the most generic case of clustering, no restrictions on the number or size of clusters is predefined and a clustering algorithms has to determine the number of clusters and their components in parallel.

2.2.6.1 Linkage Algorithms

The agglomerative hierarchical linkage-algorithms are the oldest clustering algorithms. The first linkage algorithm has been proposed by Sneath [Sne57] in 1957 to cluster bacteria. Subsequently, further linkage algorithms like complete-linkage [SS63] and average-linkage [MS57] have been proposed.

Single linkage, complete linkage as well as average linkage are agglomerative hierarchical. The linkage algorithms build a dendrogram by merging step by step those clusters whose distance is minimal. Let $d(v_x, v_y)$ be the distance of the vertices v_x and v_y and $d(C_i, C_j)$ the distance of two clusters C_i and C_j. The three linkage algorithms differ from each other only in the definition of the cluster distance function.

Single linkage algorithms merge those two clusters C_i and C_j where the distance between any vertex from cluster C_i and any vertex from cluster C_j is minimal:

$$d(C_i, C_j) = \min_{v_x \in C_i, v_y \in C_j} d(v_x, v_y) \tag{2.14}$$

Complete linkage algorithms merge those two clusters C_i and C_j where the maximal distance between any vertex from cluster C_i and any vertex from cluster C_j is minimal:

$$d(C_i, C_j) = \max_{v_x \in C_i, v_y \in C_j} d(v_x, v_y) \tag{2.15}$$

Average linkage algorithms merge those two clusters C_i and C_j where the average distance between any vertex from cluster C_i and any vertex from cluster C_j is minimal:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{v_x \in C_i, v_y \in C_j} d(v_x, v_y) \tag{2.16}$$

All linkage algorithms create dendrograms, but provide no rule at which level to cut it to receive a flat clustering. If the number of "natural" or desired clusters is known it is easy to get a clustering, otherwise an additional rule to determine the cut level is necessary.

Linkage algorithms have been originally introduced for clustering data points in a vector space. Usually it is hard to define a distance measure for vertices of a graph and so the applicability of these methods for graph clustering is limited.

2.2.6.2 Splitting algorithms

While the linkage algorithms are agglomerative hierarchical and provide a rule which clusters have to be merged in every step, several splitting algorithms have been proposed that are divisive hierarchical and use different rules to determine the next split. The following summary of such algorithms follows the good discussion provided in [Gae05].

Minimum-Cut A minimum-cut clustering algorithm recursively bi-partitions a graph by splitting the vertex set V into two subsets so that the number of edges with one endpoint in each subset is minimal. That means, if $w(X, Y)$ denotes the number of edges with one edge endpoint in X and one edge endpoint in Y, then the cut $\emptyset \subset X \subset V$ is a minimum cut when for all $\emptyset \subset Y \subset V$ the following holds: $w(X, V \setminus X) \leq w(Y, V \setminus Y)$.

$$S_{min}(V) = \min_{\emptyset \neq X \subset V} w(X, V \setminus X) \tag{2.17}$$

The computation of the optimal cuts is usually NP-hard in the worst case and therefore heuristic approaches are required [Gae05]. These algorithms create dendrograms like the linkage algorithms discussed in Section 2.2.6.1 and they suffer from the same problems: If a flat clustering is desired, there is no rule how to extract it from the dendrogram.

Other common cut measures are:

Ratio cuts minimize the ratio of the number of edges between the two cuts to the product of the cut sizes:

$$S_{ratio}(V) = \min_{\emptyset \neq X \subset V} \frac{w(X, V \setminus X)}{|X|(|V \setminus X|)} \tag{2.18}$$

Balanced cuts minimize the ratio of the number of edges between the two cuts to the size of the smaller cut side:

$$S_{balanced}(V) = \min_{\emptyset \neq X \subset V} \frac{w(X, V \setminus X)}{\min(|X|, (|V \setminus X|))} \tag{2.19}$$

Conductance cuts Conductance Cuts are based on the conductance measure [Bol98] and they are similar to Balanced Cuts. Once again, the idea is to normalize the cut size in relation to the smaller cut side. While

the Balanced Cut normalizes with the number of vertices, the Conductance Cut normalizes with the total degree of the smaller cut side:

$$S_{conductance}(V) = \min_{\emptyset \neq X \subset V} \frac{w(X, V \setminus X)}{\min(d(X), d(V \setminus X))} \tag{2.20}$$

where $d(X)$ denotes the total degree of the vertices in X.

Bisectors are like Minimal Cuts but cut a graph in two equally sized subgraphs:

$$S_{bisector}(V) = \min_{X \subset V, \lfloor |V|/2 \rfloor \leq |X| \leq \lceil |V|/2 \rceil} w(X, V \setminus X) \tag{2.21}$$

2.2.6.3 Edge Betweenness Clustering

Newman and Girvan [NG04] proposed an algorithm that iteratively removes the edge with the highest betweenness centrality from a graph. The betweenness centrality of a vertex v is defined as the fraction of shortest paths between any two other vertices that pass v (Equation 2.22). If there are several shortest paths between a pair of vertices, the fraction of shortest paths that include a specific edge is counted.

$$C_B(v) = \sum_{u \neq v \neq w \in V} \frac{\sigma_{uw}(v)}{\sigma_{uw}} \tag{2.22}$$

where σ_{uw} is the number of shortest paths between the vertices u and w and $\sigma_{uw}(v)$ is the number of shortest paths between the vertices u and w that pass the vertex v.

The edge betweenness measure [Fre77] favors edges between communities over edges within communities. Obviously, the dense connections of vertices within a cluster make intra-cluster edges highly redundant for shortest paths. However, an inter-cluster edge will be part of many shortest paths because different clusters are only sparsely connected and shortest-paths have to pass these vertices when connecting vertices of different 'natural' clusters.

By iteratively removing the edges the once connected graph is decomposed into more and more unconnected subgraphs. That means, a hierarchy of clusters and subclusters is created and once again the same problem

of selecting the 'optimal' partition (as e.g. for the linkage algorithms - see 2.2.6.1) occurs.

Tests have shown that good clustering results can only be achieved when the betweenness centrality of all edges is recalculated after each edge removal. This is a major issue as this leads to the very high time complexity of the edge betweenness clustering algorithm of $O(mn^2)$. Therefore, the algorithm can only be used for small networks. It seems worthwhile to study the effect of approximating heuristics for betweenness centrality on the results of clustering approach.

2.2.6.4 Objective Functions

Cluster detection methods can be divided into two groups. There are techniques that optimize an explicit objective function. The objective function is a quality measure of a clustering. The algorithm how to find the clustering with the best objective function value is exchangeable. Then, there are other cluster methods without an explicit objective function that define clusterings operationally, i.e. where the solution is the result of a specific algorithm. The previously discussed linkage, splitting and edge betweenness algorithms are all operationally defined.

Let Ω be the set of all partitions of a graph $G = (V, E)$ and let $F : \Omega \to \mathbb{R}$ be an objective function where higher values of F indicate better partitions, then the graph clustering problem is to find a partition C^* with

$$F(C^*) = \max_{C \in \Omega} F(C). \tag{2.23}$$

Shi et al. [SCY+10] compiled a comprehensive list of objective functions for the detection of non-overlapping communities in networks. The principle of most of these objective functions is to normalize the number of inter-cluster edges in some way. As inter-cluster sparsity is one of the abstract ideas of clustering, the inter-cluster edges contribute to a violation this objective. A problem of most objective functions is that they are (contrary to their name) not actually suitable for finding a good partition through optimizing them. For the trivial solution where all vertices belong to the same cluster the number of inter-cluster edges is minimal. Of course, this result provides no information on the structure of the graph.

In detail, the explicit objective functions compiled by Shi et al. are given in the following. In this section, $d_{v_x}^{in}$ and $d_{v_x}^{out}$ denote as usual the indegree, respectively the outdegree of vertex v_x and $|V|$ and $|C_i|$ denote

the number of vertices in the graph $G = (V, E)$, respectively in the cluster C_i.

Conductance measures the ratio of inter-cluster edges to the total edge number:

$$P_1(C) = \sum_{C_i \in C} \frac{\sum_{v_x \in C_i} d_{v_x}^{out}(C_i)}{\sum_{v_x \in C_i} d_{v_x}} \tag{2.24}$$

Expansion measures the number of inter-cluster edges per vertex:

$$P_2(C) = \sum_{C_i \in C} \frac{\sum_{v_x \in C_i} d_{v_x}^{out}(C_i)}{|C_i|} \tag{2.25}$$

Cut ratio measures the fraction of all possible edges that are inter-cluster edges:

$$P_3(C) = \sum_{C_i \in C} \frac{\sum_{v_x \in C_i} d_{v_x}^{out}(C_i)}{|C_i|(|V| - |C_i|)} \tag{2.26}$$

Normalized cut

$$P_4(C) = \sum_{C_i \in C} \left(\frac{\sum_{v_x \in C_i} d_{v_x}^{out}(C_i)}{\sum_{v_x \in C_i} d_{v_x}} + \frac{\sum_{v_x \in C_i} d_{v_x}^{out}(C_i)}{\sum_{v_x \notin C_i} d_{v_x}} \right) \tag{2.27}$$

Maximum-ODF measures the maximum fraction of inter-cluster edges per vertex:

$$P_5(C) = \sum_{C_i \in C} max_{v_x \in C_i} \frac{d_{v_x}^{out}(C_i)}{d_{v_x}} \tag{2.28}$$

Average-ODF measure the average fraction of inter-cluster edges per vertex:

$$P_6(C) = \sum_{C_i \in C} \frac{1}{|C_i|} \sum_{v_x \in C_i} \frac{d_{v_x}^{out}(C_i)}{d_{v_x}} \tag{2.29}$$

Flake-ODF measures the fraction of vertices that have less intra-cluster edges than inter-cluster edges:

$$P_7(C) = \sum_{C_i \in C} = \frac{\left|\left\{v_x : v_x \in C_i, d_{v_x}^{out} < \frac{d(v_x)}{2}\right\}\right|}{|C_i|} \tag{2.30}$$

Description length measures how a clustering reduces the information needed to encode a graph's topology. A similar information-based measure is used for the Infomap algorithm discussed in Section 2.2.6.7.

$$P_8(C) = n \log l + \frac{1}{2} l(l+1) \log m + log \left(\prod_{i=1}^{l} \binom{n_i(n_i-1)/2}{m_i} \prod_{i>j} \binom{n_i n_j}{c_{ij}} \right) \tag{2.31}$$

n_i and m_i are the number of vertices, respectively, the number of edges in cluster i and c_{ij} is the number of edges connecting vertices in cluster i with vertices in cluster j.

Community score is the square of the average intra-cluster edge count per vertex:

$$P_9(C) = \sum_{C_i \in C} \left(\frac{\sum_{v_x \in C_i} d_{v_x}^{in}(C_i)}{|C_i|} \right)^2 \tag{2.32}$$

Internal density measures the density of the clusters:

$$P_{10}(C) = \sum_{C_i \in C} \left(1 - \frac{\sum_{v_x \in C_i} d_{v_x}^{in}(C_i)}{|C_i|(|C_i|-1)/2} \right) \tag{2.33}$$

Modularity measures the difference between the fraction of intra-cluster edges and the expected value of the fraction of intra-cluster edges:

$$P_{11}(C) = \sum_{C_i \in C} \left(\frac{d^{in}(C_i)}{m} - E\left[\frac{d^{in}(C_i)}{m}\right] \right) \tag{2.34}$$

where $d^{in}(C_i)$ is the number of intra-cluster edges of cluster C_i and $E[\cdot]$ the expected value.

The comparison to the expected value makes modularity a suitable measure for finding good partitions by maximization. While other objective functions have problems with the trivial solution (one cluster contains all vertices), modularity does not suffer from this problem. This property made modularity very popular and is the reason why modularity is the topic of this thesis. The objective function will be discussed in detail in Chapter 3. Several further objective functions derived or inspired by modularity will be discussed in Chapter 3, as well.

2.2.6.5 Spectral Clustering

Spectral clustering methods are a class of algorithms that make use of the top few eigenvectors of a matrix representation of the graph to divide the vertices into groups. A good overview of spectral clustering techniques is provided by Kannan et al. [KVV04] and Luxburg [vL07].

The first spectral graph clustering algorithm has been proposed by Donath and Hoffman [DH73]. Their method works with the eigenvectors of the adjacency matrix $\mathbf{A}$ of the graph $G = (V, E)$. For unweighted graphs the element a_{ij} of $\mathbf{A}$ is 1 if there is an edge between the vertices i and j in E and otherwise a_{ij} is 0 (2.35). In the case of a weighted graph a_{ij} is the weight of the edge connecting the vertices i and j.

$$a_{i,j} := \begin{cases} 1 & \text{if } i \neq j \text{ and } (v_x, v_y) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.35}$$

Other methods use instead of the adjacency matrix the Laplacian matrix $L = (\ell_{i,j})$ with

$$\ell_{i,j} := \begin{cases} d(v_x) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } (v_x, v_y) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.36}$$

where $d(v_x)$ is the degree of vertex v_x.

Several clustering methods based on the eigenvectors of one of the above matrix representations of a graph have been proposed. Unnormalized spectral clustering uses the Laplacian matrix to discover the clusters if the number k of clusters is known. The $n \times k$ matrix $\mathbf{V}$ is created from eigenvectors corresponding to the k smallest eigenvalues. The n rows of $\mathbf{V}$ are handled as the feature vectors for clustering in the Euclidean space

and an algorithm like k-means (see Section 2.1.1) is used to retrieve a partition.

The main idea of the algorithm is to assign each vertex a point in $\mathbb{R}^k$ so that the k-means algorithm can be employed to identify the clusters. The eigenvectors corresponding to the k smallest eigenvalues contain the required information. Each of the k eigenvectors represents one dimension of the $\mathbb{R}^k$.

2.2.6.6 Label Propagation Algorithms

Label propagation is in its broadest sense an iterative method where labels are passed along between the members of a (partially) labeled set of data points.

Raghavan et al. [RAK07] proposed a label propagation algorithm for community detection in large networks. The algorithm initially assigns each vertex an unique label. Then, the labels get iteratively propagated through the network. In each iteration a random order of the vertices is determined. Then, the labels of all vertices get updated one by one. A vertex gets the label the maximum of its neighbors have. In case of a tie, one of the labels with the maximum number of occurrences is randomly selected. The iterative process stops when all vertices have a label that at least half of their respective neighbors have. The partition is determined by the final labels. All vertices that have the same label are regarded to be in the same cluster and vertices with different labels are in separate clusters.

A major advantage of this algorithm in comparison with most other graph clustering algorithms is that it totally relies on local information that can be quickly computed. The fast inner loop of this approach and the fast convergence result in a near linear time complexity (with respect to the number of edges). Furthermore, Raghavan et al. report that their label propagation algorithm clusters homogeneous random graphs into exactly one cluster while many other algorithm return several clusters which is false. However, with regard to other quality measures the algorithm performs less good. The modularity (see Section 3) of the identified clusters is low - especially in comparison to algorithms that optimize this quality measure explicitly.

Similar approaches to that of Raghavan et al. have been analyzed e.g. by Frey and Dueck [FD07]. While Raghavan et al. use binary labeling (a vertex has a specific label or it does not have it) Frey and Dueck described a method where the affinity of one vertex to another vertex gets iteratively

updated based on two different types of real valued messages that are passed between vertices.

2.2.6.7 Infomap Algorithm

The viewpoint that clustering a graph is a kind of size reduction of the coding of the information on the graph's structure is the starting point of the algorithm of Rosvall and Bergstrom [RB07]. They assume an encoding-decoding process, where a graph is encoded in a signal Y and decoded again to the graph Z. The challenge is to determine a signal Y so that the information loss is minimal and the decoded graph Z is as similar to the original graph X as possible.

The description of a community structure is given by the tuple

$$Y = \left\{ \mathbf{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}, M = \begin{pmatrix} l_{11} & \dots & l_{1m} \\ \vdots & \ddots & \vdots \\ l_{m1} & \dots & l_{mm} \end{pmatrix} \right\}, \tag{2.37}$$

where $\mathbf{a}$ is the cluster assignment vector, M is the adjacency matrix of the clusters described in $\mathbf{a}$, m is the number of clusters and l_{ij} is the number of edges connecting vertices in cluster i with vertices in cluster j.

The information necessary to describe the graph X given that the signal Y is known is

$$H(X|Y) = \log \left[\prod_{i=1}^{m} \binom{n_i(n_i-1)/2}{l_{ii}} \prod_{i>j} \binom{n_i n_j}{l_{ij}} \right], \tag{2.38}$$

where n_i the number of vertices in cluster i. Minimizing $H(X|Y)$ means finding a signal Y so that the additional information necessary to reconstruct X is minimal. Rosvall and Bergstrom used a simulated annealing approach to minimize $H(X|Y)$ and, as a consequence, the clustering is slow. However, other algorithms could be employed, too.

2.2.7 Clustering Quality

In Section 2.2.6 a number of methods to identify communities in networks has been described. Inevitably the question arises which of the methods finds the best partition. Unfortunately, there is no answer to this question, as there is no universally valid definition of a good partition.

Therefore, authors who compare community identification algorithms (e.g. [LLM10, SCY+10]) avoid trying to generate quality rankings of clustering methods and limit their analysis to properties of the clustering methods like cluster sizes. However, one approach of assessing clustering quality could be to use the classification of clusters in the strong, weak and weakest sense as discussed in Section 2.2.4. A clustering with predominantly strong clusters will be probably a good clustering in any definition one might come up with as it has very strict requirements regarding the intra-cluster density. Unfortunately, this classification is of little help in practice as most clusters identified by many graph clustering methods are at best clusters in the weakest sense. So, this classification scheme is not able to distinguish the quality of many algorithms.

Trying to compare the results of clustering methods means to compare the clusterings they calculate. In this context, the question arises how to measure the similarity of partitions. The comprehensive discussion of similarity metrics conducted by Meilă [Mei07] (see also [For10]) will be summarized in the following. Let $\mathcal{C} = (C_1, C_2, ..., C_K)$ and $\mathcal{C}' = (C'_1, C'_2, ..., C'_{K'})$ be two partitions. The number of pairs of patterns the partitions agree or disagree on is the basis for several measures. Let $c(\mathcal{C}, u)$ denote the cluster of u in partition $\mathcal{C}$, n_k the number of patterns in cluster $\mathcal{C}_k$ and $n = \sum_{k=1}^{K} n_k$ the number of patterns in $\mathcal{C}$. Likewise, n'_k and n' are the according values for the partition $\mathcal{C}'$. In the four sums

- $N_{11} = |\{\{u, v\} | c(\mathcal{C}, u) = c(\mathcal{C}, v) \wedge c(\mathcal{C}', u) = c(\mathcal{C}', v)\}|$
- $N_{00} = |\{\{u, v\} | c(\mathcal{C}, u) \neq c(\mathcal{C}, v) \wedge c(\mathcal{C}', u) \neq c(\mathcal{C}', v)\}|$
- $N_{10} = |\{\{u, v\} | c(\mathcal{C}, u) = c(\mathcal{C}, v) \wedge c(\mathcal{C}', u) \neq c(\mathcal{C}', v)\}|$
- $N_{01} = |\{\{u, v\} | c(\mathcal{C}, u) \neq c(\mathcal{C}, v) \wedge c(\mathcal{C}', u) = c(\mathcal{C}', v)\}|$

each pair of patterns is counted once. Therefore, $N_{11} + N_{00} + N_{10} + N_{01} = n(n-1)/2$.

Wallace [Wal83] proposed the two measures

$$\mathcal{W}_{\mathcal{I}}(\mathcal{C}, \mathcal{C}') = \frac{N_{11}}{\sum_k n_k(n_k - 1)/2} \tag{2.39}$$

and

$$\mathcal{W}_{\mathcal{II}}(\mathcal{C}, \mathcal{C}') = \frac{N_{11}}{\sum_{k'} n'_{k'}(n'_{k'} - 1)/2} \tag{2.40}$$

that give the probability that a pair of patterns that is in the same cluster in $\mathcal{C}$ is also in the same cluster in $\mathcal{C}'$ and that a pair of patterns that is in the same cluster in $\mathcal{C}'$ is also in the same cluster in $\mathcal{C}$, respectively. Folkes and Mallows [FM83] proposed to use the geometric mean of $\mathcal{W}_{\mathcal{I}}$ and $\mathcal{W}_{\mathcal{II}}$

$$\mathcal{F}(\mathcal{C}, \mathcal{C}') = \sqrt{\mathcal{W}_{\mathcal{I}}(\mathcal{C}, \mathcal{C}')\mathcal{W}_{\mathcal{II}}(\mathcal{C}, \mathcal{C}')} \tag{2.41}$$

Other measures are the Rand index

$$\mathcal{R}(\mathcal{C}, \mathcal{C}') = \frac{N_{11}}{N_{11} + N_{00} + N_{01} + N_{10}} \tag{2.42}$$

and the Jaccard index

$$\mathcal{J}(\mathcal{C}, \mathcal{C}') = \frac{N_{11}}{N_{11} + N_{01} + N_{10}}. \tag{2.43}$$

These measures have problems like not ranging over the interval $[0, 1]$. Normalizations to solve those issues have e.g. led to the proposal of an adjusted rand index that normalizes with the expected value of the Rand index [HA85]. For advanced measures that go beyond pairwise pattern assignment counting see [Mei07].

Measuring the similarity of partitions is also of interest for assessing the stability of a clustering method. If a small change in the data causes a dramatic change in the partition, a clustering approach is instable. This in turn poses the question, whether the identified graph structures are significant or merely a result of chance.

3 Modularity Clustering

3.1 Introduction

The popular modularity function introduced in [NG04] is a measure for the quality of a partition of an undirected, unweighted graph. Modularity measures the difference between the empirical observed number of links within clusters to the expected number. The expected value is calculated for a random graph with the same degree sequence as the observed graph.

Let $G = (V, E)$ be an undirected graph and $C = \{C_1, \ldots, C_p\}$ a partition, i.e. a clustering of the vertices of the graph into groups C_i so that $\forall i, j : i \neq j \Rightarrow C_i \cap C_j = \emptyset$ and $\cup_i C_i = V$. We denote the adjacency matrix of G as W and the element of W in the x-th row and y-th column as w_{xy}, where $w_{xy} = w_{yx} = 1$ if $\{v_x, v_y\} \in E$ and otherwise $w_{xy} = w_{yx} = 0$. The modularity Q of the partition C is

$$Q(C) = \sum_i (e_{ii} - a_i^2) \tag{3.1}$$

with

$$e_{ij} = \frac{\sum_{v_x \in C_i} \sum_{v_y \in C_j} w_{xy}}{\sum_{v_x \in V} \sum_{v_y \in V} w_{xy}} \tag{3.2}$$

and

$$a_i = \sum_j e_{ij} \tag{3.3}$$

e_{ij} is the fraction of edge endpoints belonging to an edge connecting C_i with C_j. The a_i are the row sums of the matrix spanned by the e_{ij} and give the total fraction of edge endpoints belonging to C_i. The difference $e_{ii} - a_i^2$ is a result of the following model: Let $\mathcal{G}$ be the set of all labeled graphs with the same degree sequence as G. Then, e_{ii} is the empirical fraction of edge endpoints belonging to edges that connect vertices in the

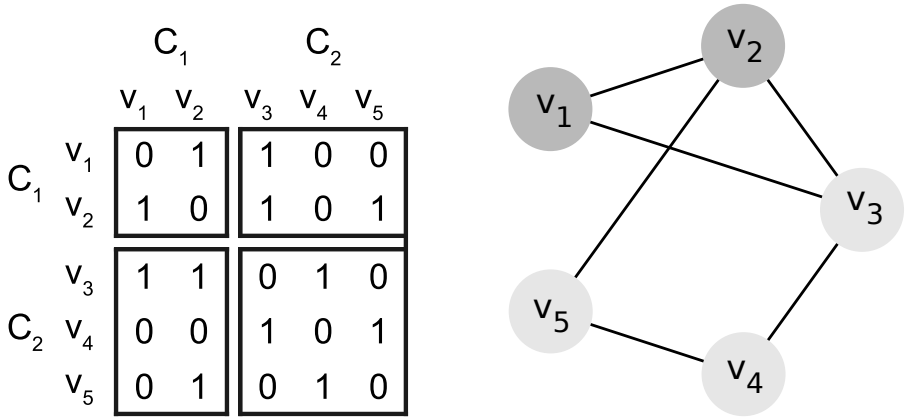

Figure 3.1: Modularity computation example [OGSS10].

group C_i and a_i^2 is the expected fraction of edge endpoints belonging to edges that connect vertices in C_i for a graph in $\mathcal{G}$.

As a_i is the fraction of the degrees in cluster C_i, the probability that a random edge $\{v_x, v_y\}$ of a graph in $\mathcal{G}$ connects two vertices in C_i is $P((v_x \in C_i) \wedge (v_y \in C_i)) = P(v_x \in C_i)P(v_y \in C_i) = a_i a_i = a_i^2$ when start vertex v_x and end vertex v_y are chosen independently. This means, modularity measures the non-randomness of a partition.

To illustrate the calculation of the modularity of a partition Figure 3.1 shows a small example. For a clustering of the example graph with 5 vertices into the clusters $C_1 = \{v_1, v_2\}, C_2 = \{v_3, v_4, v_5\}$ the values of e_{ij} are the sums of the matrix elements belonging to a pair of C_i and C_j divided by the total sum of all matrix elements: $e_{11} = \frac{2}{12}$, $e_{12} = e_{21} = \frac{3}{12}$, $e_{22} = \frac{4}{12}$. The modularity of the partition of the example graph into the two clusters is $Q = (e_{11} - a_1^2) + (e_{22} - a_2^2) = (\frac{2}{12} - \frac{5}{12}^2) + (\frac{4}{12} - \frac{7}{12}^2) = -\frac{1}{72}$. The negative value of Q obviously indicates a suboptimal partition. Assigning the vertex v_3 to C_1 improves Q to $\frac{1}{9}$.

While the notation of modularity in Equation 3.4 helps to understand the principle idea of the quality measure, this notation brings disadvantages when extending the formula and incorporating further concepts. A more practical notation of modularity is

$$Q = \frac{1}{2m} \sum_{x,y} \left(w_{xy} - \frac{s_x s_y}{2m} \right) \delta(C(x), C(y)) \tag{3.4}$$

where m is the number of edges in the graph, s_x is the total degree

of vertex v_x, $C(x)$ denotes the cluster of v_x and the Kronecker symbol $\delta(C(x), C(y)) = 1$ when v_x and v_y belong to the same cluster and $\delta(C(x), C(y)) = 0$ otherwise. In this notation the modularity contributions of vertex pairs rather than the contributions of whole clusters are summed.

3.1.1 Optimization Problem

The higher the modularity of a partition is, the less random it is. Modularity clustering means clustering by means of maximizing the modularity of a partition. According to the general definition of the graph clustering problem in Equation 2.23, let Ω be the set of partitions of G. The modularity clustering problem is to find a partition $C^* \in \Omega$ with

$$Q(C^*, G) = max\{Q(C, G)|C \in \Omega\}. \tag{3.5}$$

In contrast to many other graph clustering approaches without an explicit measure for the quality of a partition (e.g the linkage algorithms discussed in Section 2.2.6.1), modularity clustering allows to compute graph partitions by optimizing an objective function. This way common metaheuristics can be employed to find good partitions (see section 3.2.2). Brandes et al. [BDG+08] proved that finding a partition with maximal modularity is NP-hard. As a consequence, employing heuristics to find high modular partitions is the only practicable way to find a partition based on the modularity measure for large graphs.

3.1.1.1 Modularity Delta ΔQ

The change of modularity when a partition is altered is an important measure. This change will be denoted as ΔQ. While there are several operations on partitions, only the ΔQ of the merge of two clusters will be discussed now as it is required throughout this thesis. The ΔQ of all other operations will be discussed when needed.

Merging two clusters C_i and C_j of a partition C means that C_i and C_j get replaced by $C_i \cup C_j$. The number of clusters in the resulting partition C' is $|C'| = |C| - 1$. In the following $\Delta Q(C_i, C_j)$ or short $\Delta Q(i, j)$ denotes the modularity difference between the partitions C' and C, which is:

$$
\begin{aligned}
\Delta Q(i,j) =& Q(C') - Q(C) \\
=& \sum_{C_a \in C'} Q(C_a) - \sum_{C_a \in C} Q(C_a) \\
=& \sum_{C_a \in C \setminus \{C_i, C_j\}} Q(C_a) + Q(C_i \cup C_j) \\
& - \left(\sum_{C_a \in C \setminus \{C_i, C_j\}} Q(C_a) + Q(C_i) + Q(C_j) \right) \\
=& Q(C_i \cup C_j) - Q(C_i) - Q(C_j) \\
=& \left((e_{ii} + e_{jj} + e_{ij} + e_{ji}) - (a_i + a_j)^2 \right) - (e_{ii} - a_i^2) - (e_{jj} - a_j^2) \\
=& e_{ii} + e_{jj} + e_{ij} + e_{ji} - a_i^2 - 2a_i a_j - a_j^2 - e_{ii} + a_i^2 - e_{jj} + a_j^2 \\
=& e_{ij} + e_{ji} - 2a_i a_j \\
=& 2(e_{ij} - a_i a_j) \qquad (3.6)
\end{aligned}
$$

3.1.2 Clustering Quality of Modularity

As mentioned before, clustering is the classification of patterns into groups where similar patterns should be classified into the same group. This definition is still very vague. The lack of clarity arises from the fact that there is no generally accepted definition of what a cluster, respectively a good partition is. Therefore, evaluating the clustering quality an algorithm produces is a difficult task. Roughly speaking, a partition should resemble the structure of a network. That means, among others, if the network has no community structure at all, the clustering algorithm should not find several communities. Furthermore, if the community structure is hierarchical than the algorithm should reproduce this hierarchical decomposition.

While all those different properties for good partitions are postulated, these properties are often only tested by the postulating author for his newly presented clustering algorithm. Usually, the clustering quality of algorithms is only evaluated with respect to a single measure. The most common measure is the accuracy of resembling a known community structure. Because no general standard of comparison is available, one way clustering algorithms have been evaluated are by comparing their results with the 'known' cluster structure of special randomly created networks. One attempt was to create a set of densely connected graphs and then randomly and sparsely connect these graphs. The quality of algorithms is

then judged by their ability to retrieve the densely connected subgraphs. One major drawback of this approach is that it does not state, what a good partition is, but implicitly assumes that the set of densely connected graphs is the optimal partition.

With their introduction of the modularity function, Newman and Girvan [NG04] evaluated their new measure through this comparison with a 'ground truth'. This 'ground truth' is an observed partition of the network, e.g. the partition resulting from a split up of a club after a dispute (see Karate club data set description in 4.6.1). Furthermore, they created random networks with a known community structure by creating densely connected groups of vertices with z_{in} links going from each vertex to a randomly selected vertex from within the respective group. Then, the groups get connected by connecting each vertex with z_{out} vertices from a different group.

The use of the 'known community structure' of natural networks for evaluations has some methodological problems. The communities that could be identified in the natural networks by observations are supposed to be the result of a natural group building process on the basis of the social relation that has been modeled in the network. As the process by which the entities of the network determine their group membership is unknown and probably different for every network, measuring the clustering quality of algorithms by the degree they resemble the random interaction process explained above is a kind of an anecdotal evidence. There is no indication that those natural networks an author has select for evaluating his or her community detection method are a good representation of for any class of natural networks. Also, the noise in the natural grouping process is not considered. This makes the evaluation of algorithms by comparing their clustering results with a few small natural networks with a known community structure problematic.

The technique used by Newman and Girvan to create random networks with known community structures has some methodological flaws, too:

- Only small networks with 128 vertices have been analyzed.
- The sizes of all communities are equal.
- The properties (e.g. density, degree distribution) of the random networks are different from natural networks.
- The random networks do not have a hierarchical community structure, i.e. natural communities that consists of other natural communities on a higher level of detail.

- To compare the known community structure with the algorithmic results, the number of 'correctly' identified vertices have been counted, which taken alone is a problematic measure.

Especially the fact that the networks used for the evaluation are not representative for natural networks does not allow for a generalization of the evaluation results. The naturally hierarchical community structures are very important e.g. in sociology. This can be seen from the way sociology is divided into sub-fields by the level in the hierarchical structure of interaction networks they analyze: microsociology, mesosociology, macrosociology and global sociology [Sme97].

Lancichinetti et al. [LFR08, LF09a] proposed with the LFR benchmark (Lancichinetti-Fortunato-Radicchi benchmark) a generalization of the benchmark of Newman and Girvan which is a random graph with a more realistic community structure as the vertex degrees as well as the community sizes have a power-law distribution. Lancichinetti and Fortunato [LF09b] applied this benchmark to assess the clustering quality of different graph clustering methods including algorithms that are based on modularity maximization and algorithms based on other approaches.

Every comparison of partitions with the known community structure of an artificial graphs suffers from the problem that the generator of the graph is based on some assumptions what a 'good' cluster or a 'natural group' is. But as there is no common agreement on this matter, all comparative quality assessments suffer from a selection bias.

Overall, modularity can be regarded as suitable objective function for graph clustering, even though no generally valid quality assessment is available. Although modularity seems to be useful for cluster analysis, this quality function has some deficiencies which will be discussed in conjunction with a general review of its properties in Section 3.1.3.

3.1.3 Problems of Modularity Clustering

In Section 3.1.2 it has been discussed that an absolute judgment on a cluster algorithm's clustering quality is problematic. However, the analysis of properties of clustering algorithms is helpful. In this section, properties of the modularity measure will be discussed which cause three deficiencies of the modularity clustering algorithms, namely the resolution limit, the cluster size bias, and the identification of non-significant communities.

3.1.3.1 Resolution Limit

Fortunato and Barthélemy [FB07] could show that algorithms that maximize modularity cannot find communities smaller than a resolution limit which directly results from the maximization of the objective function. Their proof goes as follows: Let l_{AB} be the number of edges connecting two clusters A and B and let K_X be the the total degree of cluster X. The modularity of a partition increases, when two clusters A and B with $\Delta Q(A, B) > 0$ are merged (see Section 3.1.1.1). In a more detailed formulation Equation 3.6 becomes $\Delta Q(A, B) = l_{AB}/m - K_A K_B/2m^2$.

Assume that the two clusters A and B are about the same size and have both a total degree of approximately K. Even when the clusters A and B are only very weakly connected by a single edge ($l_{AB} = 1$), ΔQ_{AB} might still be positive:

A merge increases modularity when

$$\frac{l_{AB}}{m} - \frac{K_A K_B}{2m^2} > 0$$

when l_{AB} is 1 follows

$$\frac{1}{m} > \frac{K_A K_B}{2m^2}$$

under the assumption that K_A and K_B are approximately K follows

$$\begin{aligned} & \frac{1}{m} > \frac{K^2}{2m^2} \\ \Longleftrightarrow & 2m > K^2 \\ \Longleftrightarrow & K < \sqrt{2m} \end{aligned}$$

This means, two connected clusters A and B will always be merged if their size K is below $\sqrt{2m}$.

Fortunato [For10] gives an intuitive explanation for this fact: If two subgraphs are connected with more than the expected edges, they are supposed to have a strong topological correlation. However, when the subgraphs are small compared to the total graph size (with respect to their degrees), the expected number of edges can be smaller than 1. In this case, any connection between the clusters will keep them together - even if both are cliques connected by a single edge.

This means that clusters with less than about $\sqrt{2m}$ edges can be identified by maximizing modularity only, when they are not connected to

another cluster below the resolution limit. How far the resolution limit actually effects the results of modularity clustering for real-world datasets, has not been analyzed so far. It will depend on the connection structure of the 'true' clusters. If small clusters are only at the periphery of large graphs, the theoretical resolution limit will not effect the clustering result. If small clusters tend to cluster together, they are not identifiable by maximizing modularity.

Several strategies to deal with the resolution limit have been discussed. Ruan and Zhang [RZ08] proposed to recursively cluster a network. The clusters identified by optimizing the modularity of the source graph (level 1) are partitioned by clustering the subgraphs induced by the clusters (level 2). This method is applied as long as the clustering of a subgraph yields more than one cluster.

Lai et al. [LLN10] proposed a procedure that reweights the edge weights of intra-cluster and inter-clusters differently. A random-walk preprocessing is used to iteratively put heavier weights on possible intra-community edges than on inter-community edges.

In the recent literature, some authors mixed the resolution limit problem with the problem of identifying clusters at the same level of granularity, which occurs, when a network has a cluster hierarchy as discussed in Section 2.2.5. However, resolution limit and cluster granularity are two distinct problems. Due to the resolution limit not only clusters of the same super-cluster may not be separable, but clusters may be aggregated that would only meet at the root of the hierarchical cluster tree.

While Arenas et al. [AFG08] do this mix-up, too, the parameterized variant of modularity they propose is still worth looking at. Their idea is to make the importance of a structure dependent on the number of vertices. So, Arenes et al. propose to identify communities $C = \{C_1, \ldots, C_k\}$ of a graph $G = (V, E)$ at a scale r by maximizing a parametrized variant of modularity Q_r:

$$Q_r(C, G) = \sum_{i=1}^{k} \left(\frac{2w_{ii} + |C_i|r}{2w + nr} - \left(\frac{w_i + |C_i|r}{2w + nr} \right)^2 \right) \tag{3.7}$$

where n is the number of vertices of the graph, $|C_i|$ is the number of vertices in cluster C_i, w is the total weight of all edges, w_i is the total weight of all edges in cluster i and w_{ii} is the weight of all intra-cluster edges of cluster i. What Arenas et al. actually do is nothing else than introducing a loop of weight r to every vertex. The loops increase the total weight

of the intra-cluster edges. So, if r is set to a sufficiently high value, the intra-cluster strength will make the inter-cluster connections relatively less unlikely in the null model. Therefore, for appropriate $r > 0$ substructures get identifiable and for appropriate $r < 0$ the superstructures can be identified. There is, however, no rule how to determine the appropriate value of r for a desired result.

3.1.3.2 Cluster Size Bias

A second weakness of the modularity measure is its bias towards clusters of equal size [Bra08]. The square term a_i^2 in the modularity function $\sum_i (e_{ii} - a_i^2)$ gives the expected value. The sum of these terms $\sum_i a_i^2$ is minimized when all a_i are of equal size. As a result, higher values of Q can be reached when all a_i are roughly the same. As equal sized clusters are always favored but the natural cluster size distribution might be totally different, this property of modularity could prevent finding the natural community structure.

3.1.3.3 Non-significant Communities

Reichardt and Bornholdt [RB06] showed that clustering by maximizing the modularity function identifies clusters even in random networks. Obviously, it makes no sense to partition random graphs in groups as these groups cannot be significant. Reichardt and Bornholdt could also show, that the expected modularity of random graphs is positively correlated with the sparsity of the graph and argue that it is, therefore, difficult to detect true modularity in sparse graphs.

As a consequence of the notion that positive values of modularity do not imply that partitions are significant, Kashtan and Alon [KA05] proposed to normalize modularity by the modularity of a randomized network:

$$Q_m = \frac{Q_{real} - Q_{rand}}{Q_{max} - Q_{rand}} \tag{3.8}$$

where Q_{real} is the standard modularity, Q_{rand} the average Q of a randomized network and Q_{max} the maximal modularity of a graph with the same degree sequence.

3.1.4 Generalizations of Modularity Clustering

The original modularity definition by Newman and Girvan [NG04] is for all undirected, unweighted graphs, but not necessarily simple, loop-free

graphs. The concept of modularity can be generalized to directed and/or weighted graphs while the null model remains a configuration model graph [New10]. Of course, then, the configuration model includes directions and weights. The generalized modularity measures return the same results as the original modularity definitions for weighted graphs where all edge weights are equal, respectively for directed graphs, where all edges have a reciprocal edge. Other variations of the objective function will be discussed in Section 3.1.5.

3.1.4.1 Modularity of Weighted Graphs

The simplest and most straightforward generalization of the modularity measure is the extension for weighted graphs. The definition of e_{ij} in Equation 3.2 has to be adapted so that instead of counting the fraction of edges between two clusters, the fraction of edge weights of all edges connecting two clusters has to be calculated. For weighted graphs the term e_{ij} is replaced by

$$e_{ij}^{w} = \frac{\sum_{v_x \in C_i} \sum_{v_y \in C_j} \omega_{xy}}{\sum_{v_x \in V} \sum_{v_y \in V} \omega_{xy}} \tag{3.9}$$

where ω is the edge weight function and ω_{xy} is the weight of the edge connection the vertices v_x and v_y.

Then, the term $a_i^w a_j^w$ (the product of the row sums of (e_{ij}^w)) gives the expected total strength of the edges between the clusters C_i and C_j instead of the expected number of edges. Correspondingly, the weighted modularity is then defined as $Q^w = \sum_i (e_{ii}^w - (a_i^w)^2)$.

3.1.4.2 Modularity of Directed Graphs

Arenas et al. [ADFG07] proposed a straightforward generalization of the standard modularity measure for directed graphs. The modularity variant for directed graphs still calculates the difference between the fraction of intra-cluster edges to the expected fraction of intra-cluster edges. But the expected number of edges between two vertices changes when the null model shall remain a randomly rewired graph with the same degree distribution. Because of the edge directions a vertex in a directed graph has an indegree and an outdegree distribution. As a consequence, for both possible edge directions, there are distinct expected numbers of edges.

Let $G = (V, E)$ be a directed graph. The generalized definition of Equation 3.4 for directed graphs is

$$Q_d = \frac{1}{m} \sum_{i,j} \left(w_{ij} - \frac{d_i^{in} d_j^{out}}{m} \right) \delta(C_i, C_j) \tag{3.10}$$

where d_i^{in} and d_j^{out} give the indegree and the outdegree and $\delta(C_i, C_j)$ is the Kronecker symbol. In contrast to the undirected variant the normalization is now by m and not by $2m$ because each edge is now considered only once and not twice (once in each direction).

While Arenas et al. primarily introduced the generalized modularity for their discussion of a modularity-preserving size reduction technique for graphs, Leicht and Newman [LN08] discussed this generalized measures' ability to incorporate edge directions in the discovery of communities in directed networks. By example they showed that the directed modularity variant is able to identify the known true community structure of a randomly created network and a football-competition network in contrast to the undirected original formulation. They concluded that the directed modularity makes good use of the direction information of the edges. However, this evaluation has some weaknesses: the analyzed networks are small, the networks do not have the usual properties (e.g. degree distribution) of self-organized real-world networks and the differences might be an artefact of the optimization algorithm rather than the objective function.

Kim et al. [KSJ10] showed that the assumption of Newman and Leicht about the generalized modularity's consideration of edge directions is partly wrong. They showed that a flaw of this generalization is that the edge direction between two vertices or clusters has no influence on the ΔQ of their join (see Figure 3.2) when the total in- and outdegrees remain identical.

Kim et al. [KSJ10] proposed therefore a different approach to identify clusters in directed networks. Their idea is to replace the modularity function, informally defined as

$$\begin{aligned} Q^{ud} =& (\text{fraction of links within communities}) \\ & - (\text{expected value of this fraction}) \end{aligned} \tag{3.11}$$

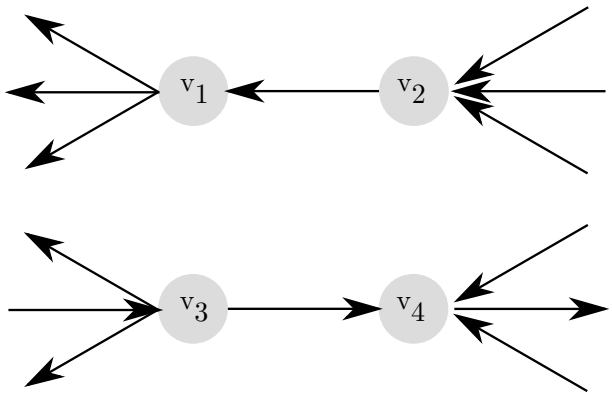

Figure 3.2: Example of Kim et al. [KSJ10] for a problem of the directed modularity variant. The vertices v_1 and v_3 respectively the vertices v_2 and v_4 have identical in- and outdegrees. Therefore, directed modularity can not distinguish between both example situations with different flow directions.

by the new difference

$$\begin{aligned} Q^{lr} =& (\text{fraction of time spent walking within communities} \\ & \text{by a random walker}) \\ & - (\text{expected value of this fraction}). \end{aligned} \tag{3.12}$$

This formulation is able to exploit edge directions for clustering as the edge directions determine how the walker can continue the random walk. The edge direction between two arbitrary vertices v_x and v_y can determine whether v_x and v_y are part of a short cycle (high return probability) or whether v_x and v_y are part of no or only long cycles (low return probability). So, the probability that v_x and v_y are in the same community depends on the edge direction.

As the generalized modularity measures are only slightly different to the originally proposed definition, so far developed algorithms for modularity clustering can be adapted with little effort. For example, Leicht and Newman [LN08] presented a spectral algorithm (see Section 3.2.4.1) for their study of the generalized modularity.

3.1.5 Variations of Modularity Clustering

While the generalizations discussed in Section 3.1.4 are equivalent to the original formulation of modularity when directed or weighted networks

are appropriately simplified, the variations presented in this section take the idea of modularity and modify the measure to produce different partitions. Gaertler et al. [GGW07] call modularity an instance of significance-driven graph clustering as modularity measures the non-randomness of a partition. The null-model used by modularity for calculating the non-randomness of a partition is the random wired configuration model of a graph with the same degree sequence than the considered graph. However, other null-models could be used as well. In the following, those alternative null-models will be discussed.

3.1.5.1 Modularity Variation for Simple Graphs

In its original form, modularity is defined for multigraphs (i.e. a graph where multiple edges between two vertices are allowed) with loops. Many natural networks do not allow multiple edges or loops. Consider, for example, the friendship network of a social network site. Neither can an user be a friend of some other user twice, nor can an user be his own friend. For community detection in this type of network, a network model which allows multiple edges and loops is not appropriate.

As Massen and Doye [MD05] noted, clustering simple networks (networks that do not allow multiple edges) by maximizing modularity has a significant drawback. The expected fraction of edges between a vertex pair v_x, v_y, which is $\frac{a_x}{2m}\frac{a_y}{2m}$, is not limited to the range $[0, 1/m]$. That means, in some cases the expected number of edges according to the network model behind the modularity measure is > 1 and, therefore, the configuration model expects more edges than the semantic of the network allows.

To overcome this problem, Massen and Doye proposed to create an ensemble $\mathcal{G}'$ of random networks with the same degree sequence where multiple edges and loops are forbidden and to calculate the probability of an edge between two vertices from this ensemble. The new quality function Q' is

$$Q' = \sum_i (e_{ii} - f_i) \tag{3.13}$$

where f_i is the estimated fraction of intra-cluster edges in cluster C_i computed from averaging the graphs in the ensemble $\mathcal{G}'$. The term f_i replaces the square of the marginal distribution (a_i^2) used by modularity (see Equation 3.4).

The simulation approach of Massen and Doye is an unsatisfying method as it requires $\mathcal{G}'$ to be large to provide precise probabilities. Therefore, Cafieri et al. [CHL10] worked on an algebraic solution and proposed alternative null models for graphs with loops but no multiple edges, graphs with multiple edges but no loops and graphs without multiple edges nor loops.

Adapting the null model to reflect the changed expected values of the fraction of edges that connect two vertices means in the notation introduced in Section 3.1 that the term a_i^2 (see Equation 3.4) has to be replaced.

The modularity variant Q_{nl} for graphs without loops is [CHL10]

$$Q_{nl} = \sum_{u,v \in V : u > v} \left[\frac{A_{uv}}{m} - \frac{k_u k_v}{2m} \left(\frac{1}{2m - k_u} + \frac{1}{2m - k_v} \right) \right] \delta(c_u, c_v) \quad (3.14)$$

where c_u and c_v denote the cluster of the vertices u and v and δ is the Kronecker symbol.

The relevance of this variation decreases with the size of graph under consideration. The alteration for graphs without loop is in the term $\frac{1}{2m-k_u} + \frac{1}{2m-k_v}$. For large natural graph it seems fair to assume that the outdegree of a vertex v_x will grow sub-linear to the total number of edges so that $\lim_{m\to\infty}(2m - k_u) = 2m$.

For graphs without multiple edges, Cafieri et al. do not give a closed term but show a method to redistribute excess probabilities from vertex pairs where the expected value is greater than 1. The redistribution algorithm applies to graphs without loops as well as to graphs with loops.

3.1.5.2 Alternative Reference Graphs

The principle of the modularity measure is to test whether a given set of vertices forms a dense subgraph by comparing it to an expected inner connectivity. The thought behind modularity is, how many intra-cluster edges would one expect in a randomly chosen group of vertices? The expected value is clearly related to the number of vertices in the group. Furthermore, the expected value will be higher if the vertices in the group have high degrees. Newman and Girvan proposed to compare the number of intra-cluster edges to the expected number of a group of vertices with the same degrees in an arbitrary graph with the same degree sequence as the analyzed graph.

But there is no compelling reason to define the measure of comparison in this way. A first attempt to define a different, better null model that

is not just a generalization of the original one, has been conducted by Chang et al. [CPHL10]. In their model the expected number of edges between two vertices depends not only on the two vertices' degrees, but on the degrees of all vertices of the graph. Chang et al. provide only a brief evaluation of their quality criterion, but their work is a first step in a previously neglected direction of research. If specific knowledge on the topology of a graph or its generating process is known in advance, it seems plausible that an adapted null model is able to provide better results.

3.1.5.3 Overlapping Clusters

Many networks have structures, where a clear assignment of a vertex to a cluster is not possible [LLDM08] or is for some reason not desirable. In fact, many natural networks have an overlapping community structure as e.g. Palla et al. [PDFV05] could show for scientific collaboration and protein interaction networks. Consider a phone call network where a link between two vertices means the denoted persons had a phone conversation. Usually, people will be part of several natural groups. Families will be densely linked as well as groups of friends or co-workers. If a clustering algorithm assigns a vertex to exactly one cluster, the multitude of social embeddings can not be represented. However, clusterings with overlapping clusters, where vertices can belong to more than one cluster, can represent this structures.

Several modularity variations for overlapping clusters have been proposed recently. In the following the EAGLE algorithm and the algorithm of Nicosia et al. will be discussed. While the first algorithm assigns vertices to one or more clusters, the second algorithm assigns vertices for each cluster a degree of affinity.

3.1.5.4 The EAGLE Algorithm

Shen et al. [SCCH09] developed the algorithm EAGLE (agglomerativE hierarchicAl clusterinG based on maximaL cliquE) for the detection of hierarchical overlapping community structures. First, their algorithm searches for maximal cliques (cliques that are not part of a larger clique). From this set of maximal cliques, those cliques that consist only of vertices that are also part of another larger maximal clique are removed. Maximal cliques smaller than a threshold are removed, too. An overlapping clustering of the network is given by the set of maximal cliques and singleton clusters that are created from all vertices that are not part of any maximal clique.

From this initial overlapping clustering a dendrogram is created by joining in every step those two communities C_1 and C_2 which have the largest value of the similarity function S, which is defined as

$$S(C_1, C_2) = \frac{1}{2|E|} \sum_{v_x \in C_1, v_y \in C_2, x \neq y} \left[m_{xy} - \frac{d(v_x)d(v_y)}{2|E|} \right] \tag{3.15}$$

where $|E|$ is the number of edges of the graph, $d(v_x)$ is the degree of vertex v_x and $m_{xy} = 1$ when the vertices v_x and v_y are connected by an edge and otherwise $m_{xy} = 0$. In this phase overlapping clusters are merged to new, larger overlapping clusters. The similarity function S sums (analogous to the ΔQ for non-overlapping cluster mergers) the difference between the fraction of new intra-cluster edges and the expected fraction of new intra-cluster edges. As before, the fractions are calculated to the total number of edges in the graph.

From the dendrogram the best overlapping clustering is selected with the extended modularity measure

$$EQ = \frac{1}{2|E|} \sum_{i} \sum_{v_x \in C_i, v_y \in C_i} \frac{1}{\gamma(v_x)\gamma(v_y)} \left[m_{xy} - \frac{d(v_x)d(v_y)}{2|E|} \right] \tag{3.16}$$

where $\gamma(v_x)$ is the number of communities the vertex v_x belongs to. The EQ measures extends the original formulation by the term $\frac{1}{\gamma(v_x)\gamma(v_y)}$ which has been introduced to normalize the impact of a vertex.

3.1.5.5 The Algorithm of Nicosia et al.

Nicosia et al. [NMCM09] proposed a variation of modularity for overlapping communities in directed graphs that assigns to each vertex a vector of 'belonging factors'. For a partition C with $|C|$ clusters the belonging factors of vertex v_x are $[\alpha_{x,1}, \cdots, \alpha_{x,|C|}]$. The factors are computed by optimizing an overlap variation of modularity and normalized so that $\sum_{c=1}^{|C|} \alpha_{x,c} = 1$.

For a directed graph $G = (V, E)$ the overlap variation of modularity Q_{ov} is defined as follows:

$$Q_{ov} = \frac{1}{|E|} \sum_{C_i \in C} \sum_{v_x, v_y \in V} \left[\beta_{l(x,y),C_i} m_{xy} - \frac{\beta^{out}_{l(x,y),C_i} k^{out}_x \beta^{in}_{l(x,y),C_i} k^{in}_y}{|E|} \right] \tag{3.17}$$

where $\beta^{out}_{l(x,y),C_i}$ is

$$\beta^{out}_{l(x,y),C_i} = \frac{\sum_{v_y \in V} \mathcal{F}(\alpha_{x,C_i}, \alpha_{y,C_i})}{|V|} \tag{3.18}$$

and

$$\beta^{in}_{l(x,y),C_i} = \frac{\sum_{v_x \in V} \mathcal{F}(\alpha_{x,C_i}, \alpha_{y,C_i})}{|V|} \tag{3.19}$$

where the function $\mathcal{F}$ determines the way an edge contributes to modularity of community C_i. For example, $\mathcal{F}$ could be set to $\mathcal{F} = (\alpha_{i,x} + \alpha_{j,x})/2$. That would mean, the degree to which an edge is regarded as an intra-cluster edge of C_i is the average of the edge's both endpoints' degrees of belonging to C_i.

While the approach of Shen et al. (see Section 3.1.5.4) describes an algorithm, the approach of Nicosia et al. proposes a quality measure that can be optimized by other types of algorithms than the genetic algorithm Nicosia et al. originally used.

3.1.5.6 Localized Modularity

For the null model of modularity the assumption is made that connections between every pair of vertices are equally probable. Muff et al. [MRC05] presented a measure to cluster networks like protein folding networks or metabolic networks, where this assumption does not apply. Their localized modularity measure for a partition C is defined as

$$LQ(C) = \sum_{C_i \in C} \left(\frac{L_i}{L_{i_N}} - \frac{(L_i)_{in}(L_i)_{out}}{(L_{i_N})^2} \right) \tag{3.20}$$

with L_i the number of intra-cluster edges of cluster C_i, $(L_i)_{in}$ and $(L_i)_{out}$ the number of edges ending, respectively starting at a vertex in C_i and L_{i_N} the number of edges in the neighborhood of C_i, i.e. all edges adjacent to a vertex in C_i or adjacent to a cluster C_i is connected to.

In contrast to the modularity measure, the localized modularity measure has not the total number of edges in the denominator, but just the number of edges in the neighborhood of a community. Muff et al. regard their measure as complementary to modularity, as it gives a second view on a network on a more detailed level.

As the LQ measure does not take the total number of links in a network into account, it will not suffer likewise from a resolution limit (compare Section 3.1.3.1).

3.2 Modularity Maximization

A large number of algorithms have been developed to optimize modularity since Newman [New04] proposed the first algorithm. Table 3.1 gives an overview on proposed modularity maximization strategies. In a first step, there are exact algorithms whose approaches guarantee to find the optimal solution and there are heuristics that are designed to find good solutions, but most probably do not find the optimum. The heuristics can be subdivided into non-hierarchical, divisive hierarchical and agglomerative hierarchical algorithms. The discussion of modularity maximization algorithms is restricted to representatives of each of these classes of algorithms.

3.2.1 Excact Algorithms

Maximizing modularity is a discrete optimization problem and the number of distinct partitions is finite. Theoretically, the modularity of all partitions of a graph could be calculated to find the optimal solution (exhaustive search). However, in practice this will not be possible due to the required computational effort.

The formulation of modularity maximization as an integer linear program (ILP) allows finding a partition with maximal modularity without considering all possible partitions. Different ILPs have been proposed by Agarwal and Kempe [AK08] and Brandes et al. [BDG+07]. Again, solving these ILPs is only possible for very small graphs. But the ILPs are of interest for another reasons. Agarwal and Kempe relaxed the ILP to a linear program (LP) by dropping the integer constraint. The solution of the LP relaxation can be used to generate a partition in an additional rounding phase. More interesting is that the solution of the LP relaxation provides an upper bound for the maximally reachable modularity. Later on, this upper bound will be used for the evaluation of heuristic modularity maximization algorithms.

Modularity optimization can also be formulated as a mixed integer quadratic program (MIQP) as Xu et al. [XTP07] showed. Transforming modularity optimization to an IP or MIQP is convenient in so far as fast software packages (e.g. IBM ILOG CPLEX Optimization Studio) exist that can be used to solve the transformed problems.

Exact Algorithms		Integer Linear Program	[BDG+07, AK08]
		Mixed Integer Quadratic Program	[XTP07]
		Column Generation Algorithm	[ACC+10]
Heuristic Algorithms	Non-Hierarchical	Simulated Annealing	[MAnD05]
		Genetic Algorithm	[THB07, JHLB10]
		Label Propagation	[BC09, LM10, JLYL09]
		Contraction-Dilation Algorithms	[MHS+09]
		Mean Field Annealing	[LH07a]
	Devisive Hierarchical	Spectral Algorithms	[WS05, New06b, RZ07, RZ08]
		Extremal Optimization	[DA05]
		Iterated Tabu Search	[LH09]
		Minimum Weighted Cut Formulation	[Dji08]
		Locally Optimal Devisive	[CHL11]
	Agglomerative Hierarchical	Plain Greedy	[New04, CNM04]
		Multistep Greedy	[SC08a]
		MOME	[ZWM+08]
		Unfolding (BGLL)	[BGLL08]
		Adaptive Algorithm	[YHY08]
		Balanced Greedy	[WT07]
		PKM	[DFLJ07]
		XCZ	[XCZ09]

Table 3.1: Modularity maximization algorithms

3.2.2 Implementations of Metaheuristics

A metaheuristic is a general approach that describes a strategy how to optimize an objective function. Metaheuristics have been developed to deal with general problems of heuristic search problems like realizing a wide coverage of the search space or preventing to get stuck in local optima. For the maximization of the modularity of a partition, implementations of several metaheuristics have been developed and tested. In the following, the two non-hierarchical metaheuristics simulated annealing (Section 3.2.2.1) and generic algorithms (Section 3.2.2.2) are presented. Simulated annealing [Haj88] as well as genetic algorithms [GS92] converge to the optimum in probability (P(optimum reached) = 1 as search steps $\rightarrow \infty$) when properly applied. Hierarchical implementations of metaheuristics are discussed in Section 3.2.3 and Section 3.2.4.

3.2.2.1 Simulated Annealing

Simulated annealing is a greedy heuristic which is designed to be able to leave local optima. A simulated annealing algorithm starts from a random initial state. Then, its way through the search space is regulated by an acceptance function. If a randomly selected state in the neighborhood of the current state is worse than the current state it is accepted with a probability given by the acceptance function. This function has a cooling schedule that constantly lowers the probability of accepting a worse state.

Based on this optimization schema, Medus et al. [MAnD05] designed their algorithm. Starting from an initial partition, one vertex at a time is moved from its current cluster to an empty cluster or a non-empty cluster which is connected to the selected vertex. The acceptance of a move is determined by the induced modularity difference and the annealing schedule. When more than an upper bound of successive move tries are not accepted, the algorithm terminates. Medus et al. reported that the simulated annealing algorithm is ten times slower than the edge betweenness algorithm (see 2.2.6.3) which has a complexity of $O(mn^2)$.

A related metaheuristic, called mean field annealing, has been applied by Lehmann and Hansen [LH07b] to the maximization of modularity. In contrast to simulated annealing, mean field annealing uses a deterministic rather than a probabilistic approach to determine whether to accept a next state that has a lower value of the quality function than the current state has. Run times for this algorithm have not been reported but the reported quality is low.

3.2.2.2 Genetic Algorithms

The genetic algorithm is a metaheuristic inspired by biologic evolution. This class of algorithms mimics the evolutionary process with mutations, cross-overs and selection. A genetic algorithm starts with a random set of solutions. The search space is scanned by producing new candidate solutions through recombinations (crossovers) and slight changes (mutations) of existing solutions. The probability that a solution of the current generation is used to create a solution of the next generation depends on its quality which is determined by a fitness function.

Tasgin et al. [THB07] used this metaheuristic to maximize the modularity of a graph clustering. Because modularity should be maximized it is used as the fitness function. A solution is encoded as a vector where the i-th position of the vector has the identifier of the cluster the i-th vertex belongs to. Because of this encoding exchanging parts of the vectors of two solutions to simulate cross-overs is not a good idea. The transfer of parts of the vectors would not exchange the information encoded in this parts as the cluster identifiers of two solutions do not necessarily match. Therefore, a crossover is realized by selecting two solutions k_{src} and k_{dest} and applying the following procedure to the cluster assignments in k_{dest}: A set S of vertices from k_{src} is randomly selected. The vertex assignments in k_{dest} are changed so that a vertex that is in the same cluster as a vertex in S in k_{src} is also in the same cluster in k_{dest}. Mutation is realized by randomly selecting two vertices and moving the second vertex to the cluster of the first vertex.

3.2.3 Hierarchical Agglomerative Algorithms

3.2.3.1 Plain Greedy Algorithm

The first algorithm that was proposed for clustering by optimizing modularity [New04] was based on a steepest gradient search (plain greedy). This algorithm calculates for every step of the algorithm the modularity change (ΔQ) upon the join of every pair of connected clusters. The pair of vertices with the largest modularity increase is joined. If several pairs have the same maximal modularity increase, one of these pairs is selected randomly. This procedure is extremely slow and Clauset et al. [CNM04] proposed an algorithmic improvement (usually referred to as the CNM algorithm) that saves time by caching the ΔQ for every adjacent pair of vertices. This saves a lot of time, as in each step the ΔQs change only for a small fraction of all vertex pairs. This modification provides a speed-up

without affecting the results. We denote this heuristic in the following as the PG (plain greedy) algorithm independently of its first slow [New04] or later improved implementation [CNM04].

The pseudocode of PG without caching ΔQs is given in Algorithm 1. The algorithm first initializes the matrix e that stores the fractions of edges between any two clusters and the vector a of the row sums of e. The two data structures are necessary to calculate the ΔQ of a merger and are updated in every *join* operation. Retrieving the modularity maximal partition from the full dendrogram can be realized by maintaining a list of join operations. This list is then used in the procedure *extractClustersFromJoins* to get the desired partition. To identify the optimal partition, in the dendrogram building phase the modularity of the partitions after every join can be stored as well. Then, the extraction method can identify the join after that the partition with the maximal modularity is found.

Clauset et al. estimate the complexity of the CNM implementation to be in $O(md \log n)$ for a network with n vertices, m edges and a depth d of the resulting cluster hierarchy. They argue that for sparse networks where $m \sim n$ and a roughly balanced dendrogram (i.e. the depth of all branches of the dendrogram is roughly identical) with $d \sim \log n$ the run-time is nearly linear with $O(n \log^2 n)$.

However, for real-world networks there is no indication for $d \sim \log n$. Wakita and Tsurumi [WT07] showed that the assumption of balancedness of the dendrogram does not hold for their large friendship network from the Japanese social network site mixi [Mix]. They presented metrics (consolidation ratios) and proposed to select a join not solely on the maximal ΔQ but on the product $\Delta Q(C_i, C_j) \cdot ratio(C_i, C_j)$. The $ratio(C_i, C_j)$ could e.g. be set to $min(|C_i|/|C_j|, |C_j|/|C_i|)$ so that equally sized clusters are preferred. However, the analysis of the algorithm of Wakita and Tsurumi indicates that the improvement of their algorithm is not due to a skillful choice of the metrics but a result of the randomization effect of the metrics (compare discussion of balancedness in Section 4.2).

3.2.3.2 Multi-Step Greedy (MSG)

Another well performing algorithm has been published by Schuetz and Caflisch [SC08a]. Their multi-step greedy algorithm performs several joins in every step. All joins between connected clusters are grouped by their modularity delta. All joins in the top l groups will be performed. The pseudocode of the MSG algorithm in Algorithm 2 shows the similarity of

Algorithm 1 Plain greedy algorithm

Data: undirected, connected graph g, constant k

▼ **Initialize**

```
forall the v ∈ V do
    forall the neighbors n of v do
        e[v, n] ← 1/(2 * edgecount)
    end
    a[v] ← rowsum(e[v])
end
```

▼ **Build Dendrogram (Greedy)**

```
for i = 1 to rank(e)-1 do
    maxDeltaQ ← −∞
    forall the connected clusters c1, c2 do
        deltaQ ← 2(e[c1, c2] − a[c1] * a[c2])
        if deltaQ > maxDeltaQ then
            maxDeltaQ ← deltaQ
            nextjoin ← (c1, c2)
        end
    end
    join(nextjoin)
    joinList ← joinList + nextjoin
end
clusters ← extractClustersFromJoins(joinList)
```

MSG to PG.

The MSG algorithm has a complexity of $O(md \log n)$. With the same argumentation as [CNM04] ($d \sim \log n$) the authors estimate the time complexity to be $O(n \log^2 n)$ [SC08b]. While the multi-step approach is likely to result in a less unbalanced merging process than the plain greedy algorithm, there is no good reason for the assumption of $d \sim \log n$. A drawback of this algorithm is that the clustering quality depends on the parameter l that needs to be guessed. To achieve their published competitive algorithm runtimes, the optimal value of this parameter has to be known in advance. Otherwise, six independent runs of the algorithm are needed to receive a result near to the result with the optimal parameter l [SC08b].

3.2.3.3 Modularity-based Multilevel Graph Clustering (MOME)

One of the so-far best performing algorithms in terms of speed and quality has been developed by Zhu et al. [ZWM+08]. Their algorithm is based on the multilevel paradigm introduced by Bui and Jones in their influential paper [BJ93]. The idea of the multi-level paradigm is to create a series of more and more coarsed problem representations, find a solution for the most coarsed problem and refine this solution while projecting the solution step by step back to the initial problem. The algorithm of Zhu et al. could be seen as an instance of the multilevel refinement scheme for modularity clustering published by Noack and Rotta [NR08] in parallel. See the discussion on that scheme in Section 3.2.6.

In a first phase, the coarsening phase, the algorithm recursively creates a set of graphs. Starting with the input graph, each vertex of the graph will be merged with the neighbor that yields the maximal increase in modularity. If the modularity delta is negative for all neighbors, the vertex will be left as it is. The resulting graph will be recursively processed until the graph can not be contracted any more. Subsequently, in the uncoarsening phase, the set of successively collapsed graphs will be expanded while the communities get refined by moving vertices between neighboring communities. For several real-world networks this algorithm found the best so-far published clustering quality. However, while an explicit analysis of the run-time complexity is missing, the results from the evaluation indicate that it is by far not linear.

Algorithm 2 MSG algorithm

Data: undirected, connected graph g, constant k

▼ **Initialize**

```
forall the v ∈ V do
    forall the neighbors n of v do
        e[v, n] ← 1/(2 ∗ edgecount)
    end
    a[v] ← rowsum(e[v])
end
```

▼ **Build Dendrogram (Multi-Step Greedy)**

```
while (i, j) with ΔQ(i, j) > 0 exists do
    deltaQsSet ← {}
    forall the connected clusters c1, c2 do
        deltaQ ← 2(e[c1, c2] − (a[c1] ∗ a[c2]))
        deltaQsSet ← deltaQsSet + {(c1, c2, deltaQ)}
    end
    touched ← {} // list of already joined clusters
    forall the (c1, c2, deltaQ) ∈ deltaQsSet do
        if deltaQ in top l values ∧ c1, c2 ∉ touched then
            join(c1,c2)
            joinList ← joinList + (c1,c2)
            touched ← touched + {c1, c2}
        end
    end
end
clusters ← extractClustersFromJoins(joinList)
```

Algorithm 3 MOME algorithm

Data: undirected, connected graph G

▼ **Coarsing**

```
coarsedGraphs ← {G}
repeat
    coarsed ← False
    forall the v ∈ V in random order do
        w ← arg max_{u∈V} ΔQ(v, u)
        if ΔQ(v, w) > 0 then
            merge(G', v, w)
            coarsed ← True
        end
    end
    if coarsed then
        coarsedGraphs ← coarsedGraphs ∪ G'
    end
until coarsed == False;
```

▼ **Uncoarsing and Refinement**

```
forall the G ∈ coarsedGraphs do
    // borderline = vertices with inter-cluster edges
    borderline ← getBorderline(G)
    forall the v ∈ borderline do
        C_n ← arg max_{C_i∈C} ΔQ(v, C_i)
        if ΔQ(v, C_n) > 0 then
            move(v, C_i)
            borderline ← updateBorderline(G, borderline)
        end
    end
end
```

3.2.3.4 Unfolding Algorithm

A very fast agglomerative hierarchical algorithm has been developed by Blondel et al. [BGLL08]. The algorithm starts with singleton clusters. Every step of the algorithm consists of two phases. At first, all vertices are sequentially and iteratively moved between their current and a neighboring cluster, if this increases the modularity. In the case that several moves have a positive ΔQ, the one with the highest ΔQ is chosen. To speed up this process a threshold is introduced to determine when to stop the first phase based on the relative increase in modularity. The pseudocode of the algorithm is shown in Algorithm 4.

Although Blondel et al. do not refer to it, the algorithm's first phase uses the local optimization procedure first proposed in [New06b]. The only difference is the introduction of the threshold that determines when to stop the local optimization. However, the introduction of a threshold is important for the good results in terms of run-time. The most modularity gain is achieved in the first iterations. Later iterations provide very little improvement and omitting them saves a lot of time without significantly decreasing the achievable modularity.

In the second phase of each step, the result of the first phase is used to create a new graph, where all vertices that have been assigned to the same cluster in the first phase are represented by one vertex. The edge weights between the original vertices are summed up as the new edge weights between the new vertices. Then, the algorithm returns to the first phase and moves the new vertices between clusters.

Blondel et al.'s reference implementation of this algorithm showed to be very resource efficient. A graph with about 118 millions vertices and one billion edges can be processed on a machine with 24GB RAM.

3.2.3.5 Adaptive Algorithm

A principle idea of Ye et al.'s [YHY08] adaptive clustering algorithm is to define new measures that shall reflect the forces with which a vertex is attracted to a cluster. If the force from a cluster ($F_{out}^{(t)}$) on a vertex is greater than the force of its current cluster ($F_{in}^{(s)}$) then the vertex is moved. The forces are defined as follow:

$$F_{in}^{(s)}(v) = e_{in}^{(s)}(v) - \frac{k(v)(d_{in}^{s}(v) - k(v))}{2|E|} \tag{3.21}$$

Algorithm 4 Unfolding algorithm (BGLL algorithm)

```
Data: undirected, connected, weighted graph inputGraph, threshold t
g ← inputGraph
while g can be contracted do
    // Phase 1: refine intermediate solution by moving vertices between
    clusters
    forall the vertices v_x do
        c[v_x] ← i  // all vertices are placed in their own cluster
    end
    while totalIncrease > t do
        forall the vertices v do
            best ← v
            maxDeltaQ ← 0
            totalIncrease ← 0
            forall the neighbors w of v do
                deltaQ ← deltaQ from move of c from c[v] to c[w]
                if deltaQ > maxDeltaQ then
                    maxDeltaQ ← deltaQ
                    best ← w
                end
            end
            c[v] ← c[best]
            totalIncrease ← totalIncrease + maxDeltaQ
        end
    end
    // Phase 2: contract graph as induced by partition from phase 1
    g ← contract(g,c[])
end
```

$$F_{out}^{(t)}(v) = \max_{t \in \{ngbs\}} \left\{ e_{out}^{(t)}(v) - \frac{k(v) d_{out}^{t}}{2|E|} \right\} \tag{3.22}$$

where $e_{in}^{(c)}(v)$ is the number of edges a vertex v has to vertices in cluster c, $d_{in}^{(c)}$ is the total degree of all vertices in cluster c, $k(v)$ is the degree of vertex v and $\{ngbs\}$ is the set of neighboring clusters of the current cluster of vertex v.

The algorithm has two alternating phases. In the first phase in repeated sweeps, the vertices are moved, if there is a $F_{out} > F_{in}$ that pulls a vertex into another cluster. If this process converges, the algorithm proceeds to the second phase and all pairs of clusters with a maximal value of ΔQ (i.e. all pairs in the equivalence class of mergers with the highest ΔQ) are merged. Then the process returns to the first phase. After this procedure generated a complete dendrogram, the cut with maximal modularity is extracted.

This algorithm is similar to the PG algorithm or the MSG algorithm. But while those algorithms are truly agglomerative hierarchical and once joined vertices never get separated later on, the adaptive algorithm moves vertices between clusters after every step of mergers.

Ye et al. [YHY08] estimate the runtime for their algorithm to be quadratic in the number of vertices, but do not report runtimes from experiments to verify this estimation. The number of iterations depends on how fast the adaptive moves reduce the number of clusters. This will clearly depend on the processed dataset. Every adaptive move phase is expensive, as the calculation of all forces F_{in} and F_{out} requires a super-linear number of operations in the number of vertices. The partitions identified by the adaptive algorithm have a high quality in terms of modularity.

3.2.4 Hierarchical Divisive Algorithms

Divisive hierarchical algorithms recursively split a graph into subgraphs. This means, in the first step all vertices are put into the same cluster which then is split into two (bi-sectioning) or more subclusters. Then, the subclusters are recursively split by applying the same split procedure as before on the subgraphs induced by the clusters.

Algorithm 5 Adaptive algorithm

```
Data: undirected, connected graph g
while |V| > 2 do
    // Phase 1: adaptive moves
    forall the vertices v ∈ V do
        in ← F_in^(s)(v)
        forall the neighboring clusters t do
            out ← F_out^(t)(v)
            bestT = s
            maxOut ← −∞
            if out > maxOut then
                maxOut ← out
                bestT = t
            end
        end
        move(v,bestT)
    end
    // Phase 2: agglomerative hierarchical join step
    maxDeltaQ ← −∞
    forall the connected clusters c1, c2 do
        if deltaQ(c1, c2) > maxDeltaQ then
            maxDeltaQ ← deltaQ
            nextjoins ← {(c1, c2)}
        end
        if deltaQ(c1, c2) = maxDeltaQ then
            nextjoins ← nextjoins ⋃ {(c1, c2)}
        end
    end
    join(nextjoins)
end
partition ← extractBestCutFromDendrogram()
```

3.2.4.1 Spectral Algorithms

Various spectral graph clustering algorithms have been developed (see Section 2.2.6.5). As the eigensystem of a graph's adjacency matrix does not reflect the modularity, finding highly modular structures by spectral analysis requires the combination of techniques, respectively the use of scoring methods to differentiate cuts identified through spectral analysis with their modularity.

To maximize modularity several spectral algorithms have been developed [WS05, New06b, RZ07, RZ08]. All of them suffer from a high time complexity. The maximal graph size they are able to process in reasonable time is in the order of some ten thousands of vertices.

Well-performing spectral modularity clustering algorithms have been proposed by Ruan and Zhang [RZ07], who combined ideas of existing spectral clustering methods to the algorithm KCUT. This algorithm recursively partitions a graph in k subgraphs, where k is between 2 and an upper limit l. In each step, for all k the candidate partition is calculated and the partition with maximal modularity is selected. KCUT splits the graph with a method similar to the unnormalized spectral clustering described in Section 2.2.6.5. For the given upper limit l, the eigenvectors corresponding to the l largest eigenvalues of the normalized Laplacian matrix are used to create a $n \times l$ matrix $\mathbf{V}$. For each $k \in \{2, \ldots, l\}$ the first k columns of $\mathbf{V}$ are taken and clustered with a k-means algorithm.

In a consecutive work, Ruan and Zhang [RZ08] improved KCUT. The new QCUT algorithm alternates between a partitioning phase executing cuts by means of the KCUT algorithm and a refinement phase. QCUT alternates between the two phases until neither of them is able to increase modularity. In the refinement phase, vertices are moved between clusters and clusters are merged. A matrix $\mathbf{T} = (t_{vj})_{n \times k}$ is computed, where t_{vj} is the modularity change that results from a move of vertex v to cluster j. Furthermore, a matrix $\mathbf{S} = (s_{ij})_{k \times k}$ is computed where s_{ij} is the the ΔQ when the clusters i and j are merged. From both matrices that operation is executed that produces the highest modularity gain. After each move or merger operation the required fields in both tables are updated and the procedure is continued as long as an operation with a positive ΔQ exists.

3.2.4.2 Extremal Optimization

The divisive hierarchical algorithm by Duch and Arenas [DA05] is an algorithm that uses the concept of extremal optimization [BP99] to determine

the recursive splits of the clusters. Extremal optimization is an approach for finding good solutions of combinatorial problems by locally refining a solution. The local refinement is realized by using a local fitness function. Instead of using the global function (for which a near-optimal solution is searched), the refinement alters a solution by employing a second local fitness function. The component of the solution with the least local fitness is altered.

The algorithm of Duch and Arenas uses the extremal optimization approach to find the cuts for recursive bi-sectioning. The vertices of a cluster are randomly split into two groups. Then, iteratively the vertex with the least fitness is moved from its current group to the other group. The fitness of the vertex v_x of cluster C_i is

$$q_x = \kappa_x - d_{v_x} a_i \tag{3.23}$$

where κ_x is the number of intra-cluster links and d_{v_x} the degree of vertex v_x. The term a_i is - as before - the row sum (marginal distribution) of the i-th row of the matrix (e_{ij}).

When the split of a cluster has been determined, all edges between vertices of different groups are deleted. The groups are regarded as the clusters and they get split themselves until further bi-sectioning does not increase the global modularity.

Fortunato [For10] gives a time complexity of $O(n^2 \log n)$ for this extremal optimization algorithm. As the costs are more than quadratic the algorithm of Duch and Arenas is not capable of processing large networks.

3.2.4.3 Iterated Tabu Search

Iterated tabu search (ITS) [MLR06] is (like standard tabu search [Glo86]) a heuristic for combinatorial optimization problems. Tabu Search is an iterative metaheuristic. While tabu search explores the search space and examines one solution after the other, it maintains a tabu list of not-allowed operations to determine the next solution. In every step the best solution in the neighborhood of the current solution is selected that is not prohibited to be selected by a rule in the tabu list. With appropriate tabu rules, tabu search can leave local optima but tends to stick to one region of the search space. To overcome this limitation iterated tabu search randomly alters previously found optima to find new start points to continue the tabu search.

This metaheuristic has been applied to maximizing modularity by Lü and Huang [LH09]. Their ITS for modularity maximization recursively bi-

partitions a graph. A cluster is split into two by starting with a random division into two subclusters that then get optimized. Getting trapped in cyclic move operations is prevented by appropriate tabu rules. To escape one region of the search space the clusters are perturbed by moving critical vertices, i.e. vertices whose move has an impact on the partitions modularity.

For small graphs (datasets up to and including the Email dataset, see Section 4.6.1) the ITS algorithm of Lü and Huang [LH09] finds very good solutions. However, for larger graphs (with tens of thousands of vertices and more) the identified partitions are of comparable low quality and the search time was in the range of hours. Therefore, the algorithm is not qualified to be applied on large graphs with millions of vertices.

3.2.5 Other Non-Hierarchical Algorithms

3.2.5.1 Contraction-Dilation Algorithms

Mei et al.'s [MHS+09] contraction-dilation algorithm is centered around a local vertex moving procedure (and with this is very similar to the unfolding algorithm of Blondel et al. discussed in Section 3.2.3.4). The algorithm alternates between a contraction phase and a dilation phase. In the contraction phase every vertex is moved from its current cluster to the cluster where ΔQ is maximal. The algorithm tries to move every vertex of the list of vertices until in one round no vertex could be moved. Mei et al. called this phase contraction phase because some clusters will be empty after all of their vertices have been moved to other clusters. After the contraction phase the dilation phase starts and a perturbation is conducted to escape local maxima. The perturbation is realized by moving every vertex with a perturbation rate p to a random cluster. Mei et al.'s experiments showed that a perturbation rate between 0.4 and 0.8 is best for most of their test networks. The algorithm starts with a random assignment of the vertices to c_{max} clusters, where c_{max} is a parameter. Then, the alternating contraction and dilation phases are iterated for a given number of times.

The idea of this algorithm is quite simple and for smaller networks competitive quality could be achieved. The biggest advantage of this algorithm is its easy implementation and low memory consumption. However, the vertex movement procedure is costly, when a partition (after each perturbation) is bad and several rounds are needed to find a local maximum.

3.2.5.2 Label Propagation Algorithms

In Section 2.2.6.6 a label propagation clustering algorithm has been discussed that has been proposed as a fast alternative to maximizing the modularity of a partition. Based on that work, label propagation algorithms that explicitly optimize a partition's modularity have been developed.

Barber and Clark [BC09] presented the algorithm LPAm (Label Propagation Algorithm for Modularity) that constrains the label propagation to maximize modularity. The original label update rule [RAK07] for a vertex v_x was

$$l'_x = \underset{l}{\operatorname{argmax}}(\sum_{v_y \in V} m_{xy}\delta(l_y, l)) \tag{3.24}$$

where l is an abritary label, l_x is the label of the vertex v_x and δ is the Kronecker symbol, i.e. $\delta(x, y) = 1$ for $x = y$ and otherwise 0.

The adapted update rule for modularity maximization replaces the adjacency matrix M with the modularity matrix $B = (b_{ij})$, where

$$b_{ij} = m_{ij} - \frac{a_i a_j}{2m}. \tag{3.25}$$

That means, while M is the matrix of the existing edges, B is the matrix of the number of existing edges minus the number of expected edges. The label update rule for LPAm is then

$$l'_x = \underset{l}{\operatorname{argmax}}(\sum_{v_y \in V} b_{xy}\delta(l_y, l)) \tag{3.26}$$

Liu and Murata [LM10] proposed an extension of LPAm, LPAm+, that solves the problem that LPAm gets stuck in local maxima, because it favors clusters of the same size. LPAm+ uses LPAm to find a partition. Then, an agglomerative hierarchical algorithm (in their implementation the MSG algorithm, see Section 3.2.3.2) can be used to merge communities and to escape a local maximum. After the run of the hierarchical algorithm, LPAm is used again. The alternating use of the two algorithms is stopped, when no further improvement of modularity can be achieved. The LPAm+ approach needs significantly more computational time than LPAm, but the average performance in terms of modularity could be increased significantly, as well. The partitions found by LPAm+ are for all test datasets better than those of LPAm and MSG with respect to modularity.

3.2.6 Refinement Techniques

Refinement techniques are algorithms that optimize a clustering identified by a separate algorithm. A clear differentiation of refinement techniques from *main clustering algorithms* by the way the algorithms work is not possible. The differentiation is based on the way the algorithms are employed.

The influential algorithm of Kernighan and Lin [KL70] (see Section 2.2.6) has inspired the first proposed refinement technique for modularity clustering by Newman [New06b] and is an essential part of the MOME algorithm (Section 3.2.3.3), the unfolding algorithm (Section 3.2.3.4) and the contraction-dilation algorithm (Section 3.2.5.1).

A comprehensive discussion of refinement strategies has been conducted by Noack and Rotta [NR08, NR09]. Sticking to their terminology, these strategies are called Complete Greedy (Algorithm 6), Fast Greedy (Algorithm 7) and Adapted Kernighan-Lin (Algorithm 8).

The complete greedy refinement repeatedly calculates the globally best move of a single vertex. If the best move has a positive impact on modularity, it is executed, otherwise the procedure stops. This approach is costly in terms of the number of required operations as in every step for all vertices the ΔQ for each potential move to a neighboring cluster has to be computed.

Algorithm 6 Complete greedy refinement (in notation of [NR08])

Input: graph, partition
Output: partition
repeat
  // determine globally best move of any vertex v
  // to any target cluster D
  $(v, D) \leftarrow \arg\max_{(v,D)} \Delta Q(v, D)$
  if $\Delta Q(v, D) > 0$ **then**
    move vertex v to cluster D
  end
until $\Delta Q(v, D) \leq 0$;

The fast greedy algorithm is computationally less demanding. It iteratively sweeps the set of vertices and moves a vertex to the neighboring cluster that causes the maximal positive modularity increase. The algorithm stops, when in one sweep no vertex move could increase the overall modularity.

Algorithm 7 Fast greedy refinement (in notation of [NR08])

Input: graph, partition
Output: partition
repeat
  foreach *vertex v* **do**
    $D \leftarrow \arg\max_D \Delta Q(v, D)$ // determine for v best target cluster D
    if $\Delta Q(v, D) > 0$ **then**
      move vertex v to cluster D
    end
  end
until *no improved partition found*;

The adapted Kernighan-Lin refinement is a more complex algorithm than the two previously described refinement techniques. Similar to the global greedy algorithm, the adapted Kernighan-Lin refinement executes the globally best move, but in contrast even, when this move decreases modularity. An additional restriction is that every vertex may only be moved once. After every move operation the resulting partition is stored, if it has a higher modularity than the best previously found partition (peak modularity). Further moves are executed, as long as there is a vertex that has not been moved before and as long as the peak modularity has been increased in the last k steps. When this move procedure stops, the algorithm returns to the best previously found partition and starts from the beginning. The complete algorithm ends, when the partition could not be improved in the last move procedure.

Sun et al. [SDJB09] proposed a different adaptation of the Kernighan-Lin algorithm. Their algorithm considers also moves of vertices to new clusters (singletons) and it has no break condition of quitting when no improved partition has been found in the last k moves. No evaluation of the effect of including empty clusters as a move target has been conducted. However, a positive impact is doubtful for the same reason why the Kernighan-Lin refinement is unlikely to escape a local maximum: To form a new dense cluster from a singleton, probably several modularity-decreasing moves have to be made in a row.

Noack and Rotta conducted tests to evaluate the refinement quality and the runtime of the three algorithms. The complete greedy as well as the Kernighan-Lin refinement method scale much worse than the fast greedy refinement. The complete greedy approach achieves with the increased computation effort only a slight advantage over the fast greedy algorithm.

Algorithm 8 Adapted Kernighan-Lin refinement (in notation of [NR08])

```
Input: graph, partition
Output: partition
repeat
    peak ← partition
    mark all vertices as unmoved
    while unmoved vertex v exists do
        D ← arg max_D ΔQ(v, D) // determine for v best target cluster D
        move v to cluster D, mark v as moved
        if Q(partition) > Q(peak) then
            peak ← partition
        end
        if k moves since last peak then
            break
        end
        partition ← peak
    end
until no improved partition found;
```

The Kernighan-Lin method resulted in minor advantages compared to the fast greedy algorithm and the complete greedy algorithm, but is prohibitively expensive.

The fast greedy refinement algorithm provides the best trade-off between time and quality. This refinement technique does not necessarily stop at a local maximum, but could also stop at a saddle point. However, in all test runs, where the fast greedy algorithm optimized a solution found by the RG algorithm (see Chapter 4) the procedure stopped at a local maximum. It seems that the modularity function usually has no plateaus (at least in the datasets used in Section 4.6).

The three discussed refinement algorithms achieve only minor improvements of modularity. The only method that could theoretically escape a local maximum is the adapted Kernighan-Lin refinement, because this is the only approach that allows moves that decrease modularity. However, as Noack and Rotta argue, an escape is not very likely as that would usually require a series of sharply modularity-decreasing moves.

Therefore, Noack and Rotta [NR08] propose a multi-level refinement algorithm (Algorithm 9). The refinement requires that first, in a coarsening phase, a set of increasingly coarser graphs is created by an agglomerative hierarchical algorithm (*coarsener*). In the following refinement phase, the

hierarchical information encoded in the set of coarsed graphs is used to receive better refinement results with e.g. one of the refinement algorithms (*refiner*) shown above. The refinement starts with refining the coarsest graph. Then the cluster assignments of this graph are projected to the next less coarse graph and the refinement takes place once again. While other refinement techniques only move single vertices, the multi-level refinement tries to move groups of vertices: in the first step whole clusters, then smaller groups and only in the last step single vertices.

Algorithm 9 Multi-Level refinement [NR08]

Input: graph, reduction factor
Output: partition

```
▼ Coarsening Phase
  level[1] ← graph
  for l = 1 ... n do
      level[l+1] ← coarsener(level[l],reduction factor)
      if no clusters merged then
          break
      end
  end

▼ Refinement Phase
  for l = l_max − 1 ... 1 do
      project clustering from level[l+1] to level[l]
      partition ← refiner(level[l], partition)
  end
```

3.2.7 Pre-Processing

Some algorithms that have been proposed as main clustering algorithms can also be used to refine partitions. Especially those algorithms that do extensive local searches and have a high complexity and, therefore, are not able to cluster large networks in a reasonable amount of time, could be still of use to optimize already good partitions. For example, a simulated annealing algorithm like the one of Medus et al. [MAnD05] (see Section 3.2.2.1) could be used. In this case, it depends on the point of view, whether to say the main algorithm is accompanied by a post-processing

refinement or a pre-processing is applied to find a good starting point for the main algorithm.

In the following, two pure pre-processing approaches are presented that are not able to find good partitions on their own: the neighborhood-similarity approach of Wang et al. [WSWA08] and the random-walk pre-processing of Pujol et al. [PBD06].

Wang et al. [WSWA08] combine the plain greedy algorithm (see Section 3.2.3.1) in the implementation of Clauset et al. [CNM04], respectively the algorithm of Wakita and Tsurumi [WT07], with a pre-processing that generates clusters on the base of vertex similarity. Let $N(v) = \{w \in V | (v, w) \in E\}$ be the set of neighbors of a vertex v. Wang et al. [WSWA08] measure the similarity of a pair of vertices v and w by the Jaccard index of the neighborhood of the vertices:

$$S_{jaccard}(v, w) = \frac{|N(v) \bigcap N(w)|}{|N(v) \bigcup N(w)|}. \tag{3.27}$$

Furthermore, they define *near neighbors* as those neighbors that have a higher similarity than the average

$$N_{near}(v) = \left\{ w | (v, w) \in E, S_{jaccard}(v, w) > \frac{\sum_{(a,b)\in E} S_{jaccard}(a, b)}{|E|} \right\}, \tag{3.28}$$

and *good neighbors* as those neighbors that have a higher similarity than the average near neighbor

$$N_{good}(v) = \left\{ w | (v, w) \in E, S_{jaccard}(v, w) > \frac{\sum_{a\in N_{near}(v)} S_{jaccard}(a, v)}{|N_{near}(v)|} \right\}. \tag{3.29}$$

Finally, Wang et al. define *friends* as those vertices v and w that are reciprocally *good neighbors* of each other

$$Friend(v, w) \Leftrightarrow (v \in N_{good}(w)) \wedge (w \in N_{good}(v)). \tag{3.30}$$

The initial partition of a graph consists of clusters, where all vertices are directly or indirectly *friends*. This partition is then clustered with a

plain greedy algorithm. The similarity-based pre-processing significantly improved the clustering quality and the overall run-time.

Pujol et al. [PBD06] presented another pre-processing method that is intended to reduce the dimensionality of the clustering problem in a computationally cheap way. Once the problem dimension is reduced, it does not matter that the following greedy agglomerative hierarchical algorithm is rather expensive in terms of time.

The pre-processing idea of Pujol et al. is based on random walks. Let A be the adjacency matrix of a graph G and I the identity matrix. Furthermore, the diagonal matrix $D = (d_{ij})$ is defined as $d_{ij} = 1 + \sum_j a_{ij}$. Then, the matrix

$$M = (A + I)D^{-1} \tag{3.31}$$

gives the transition probabilities for random walks on the graph G. m_{ij} is the probability of a random walker to go from vertex i to vertex j. Let G^t_{ij} be the probability distribution that the j-th random walker is at vertex i at time t. One random step of all random walkers changes the probability distribution as follows

$$G^{t+1} = M'G^t. \tag{3.32}$$

The idea of Pujol et al. is not to calculate the stationary state but the transient state after three steps, i.e G^3. Then, a vertex i is assigned to that group j where g_{ij} is maximal. That means, all vertices in one group have been visited most often by the same walker. Choosing the transient state after only e.g. three steps ensures that the random walkers stay in the neighborhood of the initial position.

The random-walk pre-processing step was able to improve the runtime as well as the quality in terms of modularity compared to Newman's plain greedy algorithm.

Both pre-processing methods could also be promising approaches to improve other algorithms. Especially, other agglomerative hierarchical algorithms like the MSG algorithm (see Section 3.2.3.2) could benefit from a pre-processing. PG as well as MSG suffer from imbalanced merger processes (see Section 4.2) which a pre-processing is likely to prevent or at least to reduce.

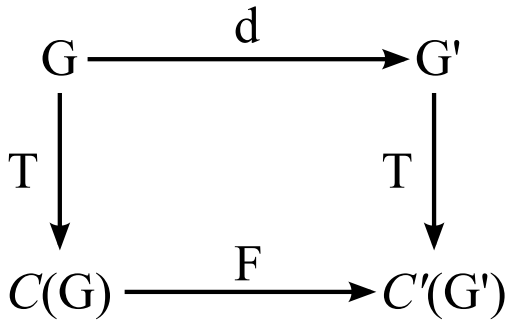

Figure 3.3: Dynamic update problem [GMSW10b]. A function F is required that updates the partition $C(G)$ to $C'(G')$ when the graph G is updated to G'.

3.2.8 Dynamic Update Algorithms

Often, real-world networks actually represent snapshots of the relations between the modeled entities at one point in time while the relations constantly change. In friendship networks new friendships are formed and other friendships are broken off. In co-citation networks, with every new paper new links between co-authors emerge. If one wants to keep community information up-to-date, the clustering process has to be redone with every change of the graph or an update algorithm has to be used that incorporates the changes in the graph in the existing partition.

The generic dynamic update problem is depicted in Figure 3.3. When a graph G is updated and becomes G' two approaches to update the partition $C(G)$ to $C'(G')$ are possible: apply the same produce T to G' that has been used to cluster G or use an incremental update procedure F that calculates $C'(G')$ on basis of $C(G)$ and the changes of the graph d. A general introduction to incremental update algorithms for graph clustering methods is out of the scope of this thesis. A good overview on this topic provides [Fra07].

The incremental update algorithms presented below are heuristics with a procedure F that does not guarantee to find the partition a reclustering would identify. If the update procedures are called many times, update errors (the differences to $C'(G')$) can accumulate. So far, the only incremental update algorithm for graph clustering that does not suffer from this problem is the algorithm of Franke and Geyer-Schulz [FGS09] for updating restricted random walk clusterings.

Incremental update algorithms for graph clustering based on modularity

have been developed by Görke et al. [GMSW10a, GMSW10b] and Nguyen et al. [NXDT10]. Both methods work with local update heuristics to incorporate the changes in the graph structure in the partition. Görke et al. showed that updating a modularity optimal partition of G to a modularity optimal partition of the altered graph G' is NP-hard. As a consequence, update algorithms for modularity-based partitions have to be heuristic to be of practical use.

The Quick Community Adaptation (QCA) algorithm of Nguyen et al. is based on a case differentiation of simple graph change events: adding a vertex (including adjacent edges), deleting a vertex (including adjacent edges), adding an edge, deleting an edge. When a cluster update is considered as necessary from the nature of the graph alteration, a local update procedure based on the forces idea of Ye et al. [YHY08] (see Section 3.2.3.5) is employed. Nguyen et al. compared the QCA algorithm with the static Unfolding algorithm (see Section 3.2.3.4) on three datasets. While the static algorithm requires more and more time with the growth of the analyzed graphs, the runtime of the QCA update procedure remains constant. The QCA algorithm showed bad results for one dataset (the modularity dropped behind the one of the partitions identified by the Unfolding algorithm; the number of communities strongly increases while the number of communities slightly decreases for the Unfolding algorithm) but the results for two other datasets were comparable to the results of the static Unfolding algorithm.

The update algorithm by Görke et al. uses a more sophisticated strategy. Assuming that the changes in the graph cause only local changes in the partition, the idea is to break up parts of the partition $C(G)$ receiving a *preclustering* $\tilde{C}(G)$. That means, vertices assumed to be affected by the graph update ($\tilde{V}$) are moved from their cluster in $C(G)$ to a singleton. Then, agglomerative hierarchical algorithms are started from this initial partition (instead of starting from singletons) to compute $C(G')$. The unfolding algorithm of Blondel et al. (see Section 3.2.3.4) as well as the plain greedy algorithm (see Section 3.2.3.1) have been evaluated for this task.

Several strategies to break up the relevant regions of a graph have been proposed by Goerke et al. Three subset strategies calculate the set $\tilde{V}$ of vertices that will be freed, i.e. removed from their current cluster and put into a singleton, based on their relation to the vertices involved in an update. If the edge (v, w) gets changed in an update of the graph, then $\tilde{V}$ gets calculated as follows:

- breakup strategy: $\tilde{V} = C(v) \bigcup C(w)$, where $C(u)$ is the cluster of vertex u
- neighborhood strategy: $\tilde{V} = N_d(v) \bigcup N_d(w)$, where $N_d(v)$ is the d-hop neighborhood of vertex v, i.e. all vertices is a maximal distance of d starting from v
- bounded neighborhood strategy: the first s vertices found by a breadth-first search started at v and w

A fourth strategy is based on searching in the previous partition's dendrogram for those vertices that are (depending on the graph change operation) supposed to be candidates for the different cluster assignment in the updated partition.

Goerke et al. [GMSW10b] evaluated the performance of the four strategies in combination with the PG or the Unfolding algorithm. The performance of a combination of break-up strategy and an algorithm to start from the partition resulting from the break-up procedure (*preclustering*) depended on the network used for evaluation.

An insight from a later on discussed property of agglomerative hierarchical algorithms (see Section 4.2) is of interest for the discussion of the dynamic modularity clustering algorithms presented above: The huge differences in the cluster sizes (newly freed singletons on the one hand and large non-altered clusters on the other hand) of the partition used as a starting point for the agglomerative hierarchical clustering are a bias for merger decisions. So, the breaking up of clusters can produce a problematic situation for agglomerative hierarchical algorithms.

4 Randomized Greedy Algorithms

In this chapter the randomized greedy (RG) algorithm for modularity clustering and its extension RG+ will be presented. RG has been published first in [OGSS10] and RG+ has been introduced in [OGS10].

First the idea of the RG algorithm and its implementation will be discussed. Then an important property for agglomerative hierarchical algorithms like RG, the balancedness of the merge process, will be discussed. The analysis of the balancedness provides the idea for the advanced RG+ algorithm, which will be presented next. Afterwards, implementation details, a discussion of time and memory complexity and evaluation results for both algorithms follow.

4.1 RG Algorithm

Randomized algorithms have been successfully employed for a broad range of problems. This class of algorithms differs from deterministic algorithms in so far, as they make random choices during execution. For a good generic introduction to randomized algorithms, the different ideas behind randomization and examples for various types of problems, see the text book by Motwani and Raghavan [MR95].

Random choices are used in many modularity clustering algorithms discussed in Section 3.2 for tie breaking. If an algorithm is not able to decide which of several options is best (e.g. because the ΔQ of all options is equal) a random choice is used as a tie break. In contrast to those algorithms, two new algorithms will be presented, for which randomization is a fundamental principle. Next, it will be discussed, how randomization can be used to decrease complexity, and how it can increase optimization quality.

4.1.1 Idea

The first greedy algorithm (Algorithm 1) developed by Newman [New04] follows a quite simple approach. As every agglomerative hierarchical algorithm does, this algorithm starts with putting each vertex of the processed graph in a separate cluster (singleton). Then, in each step of the algorithm those two clusters are merged that result in the highest ΔQ. This way, the complete dendrogram is created in $n-1$ steps. That level of the dendrogram is considered as the final clustering, that yields the highest value of modularity. The decision variable of this algorithm is the delta of the quality function upon a join (ΔQ). Recall the derivation of ΔQ for merge operations (Equation 3.6). The ΔQ of the partition C' resulting from the join of the clusters C_i and C_j of the partition C is:

$$
\begin{aligned}
\Delta Q(i,j) =& Q(C') - Q(C) \\
=& \sum_{C_a \in C'} Q(C_a) - \sum_{C_a \in C} Q(C_a) \\
=& \sum_{C_a \in C \setminus \{C_i, C_j\}} Q(C_a) + Q(C_i \cup C_j) \\
& - \left(\sum_{C_a \in C \setminus \{C_i, C_j\}} Q(C_a) + Q(C_i) + Q(C_j) \right) \\
=& Q(C_i \cup C_j) - Q(C_i) - Q(C_j) \\
=& \big((e_{ii} + e_{jj} + e_{ij} + e_{ji}) - (a_i + a_j)^2 \big) - (e_{ii} - a_i^2) - (e_{jj} - a_j^2) \\
=& e_{ii} + e_{jj} + e_{ij} + e_{ji} - a_i^2 - 2a_i a_j - a_j^2 - e_{ii} + a_i^2 - e_{jj} + a_j^2 \\
=& e_{ij} + e_{ji} - 2a_i a_j \\
=& 2(e_{ij} - a_i a_j)
\end{aligned}
$$

where $e_{ij} + e_{ji} = 2e_{ij}$ because of the symmetry of the adjacency matrix.

Newman [New04] noted that merging two unconnected clusters will always result in a decrease of modularity. This is intuitive as the observed fraction of edges between unconnected clusters is 0 and the expected value for connected graphs has to be greater than 0. The formal analysis goes as follow: If two clusters C_i and C_j are not connected $e_{ij} = e_{ji}$ is 0. As only connected graphs are considered, each vertex has a degree of at least 1 and, therefore, every cluster has a degree of at least 1. So, $a_i > 0$ for every $C_i \in C$. Consequently, the modularity change is

$$\begin{aligned}\Delta Q(i,j) =& 2\left(\underbrace{e_{ij}}_{=0} - \underbrace{a_i a_j}_{>0}\right)\\ <& 0\end{aligned}$$

for unconnected clusters C_i, C_j.

As a join of two clusters can only increase the modularity if those two clusters are connected, unconnected clusters do not have to be considered for a join.

The modularity change $\Delta Q(i,j)$ depends just on the local properties of the two involved clusters. The terms a_i and a_j are the sums of the degrees of the vertices in cluster C_i respectively C_j divided by 2 times the total number of edges in the graph. The term e_{ij} is the number of edges connecting vertices in cluster C_i and C_j once again divided by 2 times the total number of edges in the graph. As $\Delta Q(i,j)$ depends only on the total degrees of the merged clusters and the number of edges connecting both, the distribution of the vertex degrees has a huge influence on the number of different values $\Delta Q(i,j)$ can take.

Especially in larger graphs many possible cluster joins will be equivalent in terms of modularity change. For example, the dataset of collaborations of authors [New01] consists of 116181 edges. That means, in step one (i.e. with the start configuration into singletons) there are 116181 possible joins. But there are only 3475 classes of equivalent joins in terms of modularity. For this reason, considering all potential joins is unnecessary as many of them will be structurally identical. For each class of equivalent joins (with respect to the modularity metric) only one member needs to be considered to find the join that increases modularity most. Of course, picking one join of each equivalent class without calculating the ΔQ of each join is not possible as the equivalence classes are unknown. But anyway joining that pair of clusters with maximal ΔQ in each step does not necessarily lead to a global maximum. Therefore, it can be assumed that considering only a sample of all pairs of vertices for a join performs as well as considering all pairs of vertices. Especially, because of the low number of equivalence classes of joins, there is a good probability of having a pair of vertices in the sample with a near maximal ΔQ.

4.1.2 Core Algorithm

Based on the observation that natural networks show a high redundancy of those structures determining the modularity delta upon a join of two clusters and the knowledge that choosing the local optimum in every step of the algorithm is not necessary as it does not lead to the global optimum, the randomized greedy (RG) algorithm has been developed. The pseudocode of RG is shown in Algorithm 10. There is only a slight difference of the randomized approach compared to the plain greedy algorithm. But while the conceptual change is minor, it takes major changes of the algorithm's implementation to exploit the full speed-up potential as will be discussed later.

The RG algorithm generates a set S of k randomly selected vertices. For every pair of vertices consisting of at least one vertex $v \in S$ the ΔQ is calculated and that join with the highest value of ΔQ is executed. I.e. instead of considering every edge for a join as the previous greedy algorithm does, just a small sample of edges is considered. All other parts of the RG algorithm are the same as for the plain greedy algorithm. As PG does, RG creates a full dendrogram and selects that cut from the dendrogram with the highest modularity as the result.

The approach of RG has an additional advantage compared to PG. Randomizing the path through the search space offers the possibility to start over from the beginning and to find another solution. As a single run of the randomized algorithm is extremely fast, it does not matter that it occasionally might find comparably bad solutions, as the best solution out of several runs can be selected as the final result. $RG(r)$ denotes the RG algorithm with a parameter r, where r gives the number of runs from which the best result is taken. In Figure 4.1 the influence of r on the results of the RG algorithm is shown by test runs on the yeast dataset (see Section 4.6.1). With increasing r the modularity converges towards the best identifiable solution.

The core of the randomized greedy algorithm (Algorithm 10) can be improved by integrating the following 2 ideas.

1. Tuning of the sample size for searching for an improvement of ΔQ in the greedy phase (section 4.1.3).

2. Postprocessing to guarantee a local optimum (section 4.1.4).

Algorithm 11 shows the pseudocode of the RG algorithm including these improvements. This pseudocode shows that version of the RG algorithm

Algorithm 10 Generic randomized greedy algorithm

Input: undirected, connected graph g, constant k
Output: clustering

▼ **Initialize (same as for Plain Greedy Algorithm)**

```
forall the v ∈ V do
    forall the neighbors n of v do
        e[v, n] ← 1/(2 ∗ edgecount)
    end
    a[v] ← rowsum(e[v])
end
```

▼ **Build Dendrogram (Randomized Greedy)**

```
for i = 1 to rank(e)-1 do
    maxDeltaQ ← −∞
    for j = 1 to k do //search among k communities for best join
        c1 ← random community
        for all communities c2 connected to c1 do
            deltaQ ← 2(e[c1, c2] − (a[c1] ∗ a[c2]))
            if deltaQ > maxDeltaQ then
                maxDeltaQ ← deltaQ
                nextjoin ← (c1, c2)
            end
        end
    end
    join(nextjoin)
    joinList ← joinList + nextjoin
end
clusters ← extractClustersFromJoins(joinList)
```

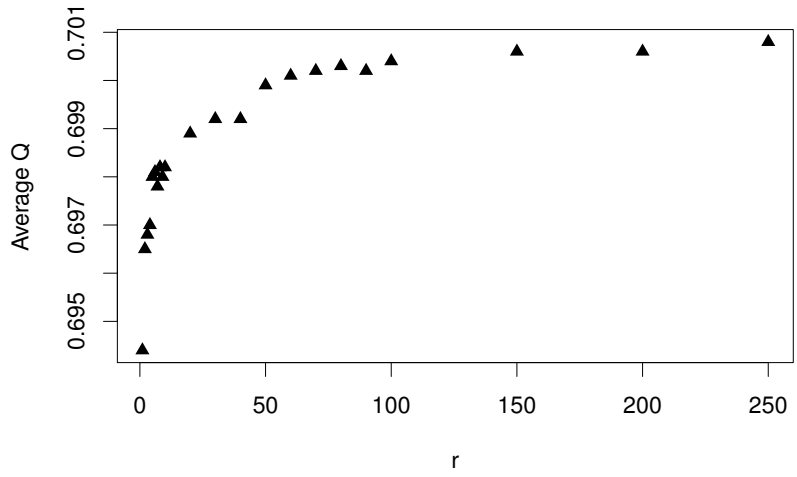

Figure 4.1: Analysis of the effect of parameter r. The results show the averages of 100 test runs of RG on the yeast dataset (see Section 4.6.1) [OGSS10].

that has been implemented for the evaluation in Section 4.6. Details of the improvements are discussed in the following subsections.

4.1.3 Tuning of k

Tests have shown that the best results are achieved when k is not fixed but gets adjusted during the cluster calculation. At the beginning k should be set to 1, because in only slightly contracted graphs a high number of cluster pairs with the same ΔQ exist. About half of the $n-1$ necessary joins to build the complete dendrogram should use this value for k. During the second half of the joins – when the size of the equivalence classes is much lower than in the beginning – a choice of $k = 2$ is best. Only on rare occasions a higher value of k led to better results while it always took more time. For none of the real-world datasets used to evaluate performance and quality a value of $k \geq 10$ increased the quality, but usually lowered it. The decline in quality for higher values of k is probably the result of a less balanced merge process (compare Section 4.2). Within a range between 1 and 10 the run-time impact of any choice of k is very low. The adjustment of k usually improves the quality by less than 1%.

Algorithm 11 Randomized greedy algorithm with refinement and heuristic for parameter k

Input: undirected, connected graph g
Output: clustering
▶ **Initialize (same as for Plain Greedy Algorithm)**

▼ **Build Dendrogram (Randomized Greedy)**

```
for i = 1 to rank(e)-1 do
    maxDeltaQ ← −∞
    if i < rank(e)/2 then
        k ← 1
    end
    else
        k ← 2
    end
    for j = 1 to k do //search among k communities for best join
        c1 ← random community
        for all communities c2 connected to c1 do
            deltaQ ← 2(e[c1, c2] − (a[c1] ∗ a[c2]))
            if deltaQ > maxDeltaQ then
                maxDeltaQ ← deltaQ
                nextjoin ← (c1, c2)
            end
        end
    end
    join(nextjoin)
    joinList ← joinList + nextjoin
end
clusters ← extractClustersFromJoins(joinList)
```

▶ **Fast greedy refinement (see Algorithm 12)**

4.1.4 Postprocessing: Fast Greedy Refinement

Once a clustering has been calculated, the quality of the result can usually be increased by a local optimization postprocessing step as discussed in Section 3.2.6. After the extraction of the best clustering from the dendrogram, the RG algorithm uses a fast greedy postprocessing. An increase of modularity is tried to be achieved by moving vertices to neighboring clusters. The list of vertices is walked through and each vertex is moved to a connected cluster, if this increases the modularity of the partition. When in one pass through the list of vertices no vertex move with positive ΔQ was found, the procedures stops. Then, either a local maximum or a saddle point has been reached. Surprisingly, a series of test runs on the set of real-world networks (see Section 4.6.1) showed that all partitions found are local optima. Having reached a saddle point would imply that there is a vertex move with $\Delta Q = 0$. It seems to be very rare in natural networks that a vertex is equally strongly connected to two (or more) clusters.

In the average case, the fast greedy refinement process is able to improve the modularity of a clustering found by the RG algorithm by 1-2 %. The time complexity of this refinement step depends on the number of vertices and the number of clusters, and scales worse than the randomized greedy clustering. For small clusters with a few thousand vertices the run-time is low compared to the main clustering phase. However, for large graphs with more than 4-5 million vertices the run-time of this step approaches the one of the randomized greedy clustering. A stop rule for the iterative sweeps over the set of vertices can improve the runtime with only slight effects on the quality. The number of sweeps could be set to a fixed number or be set in relation to the graph's size. Or a probably even better way would be to stop the iterations, if the modularity gain in an iteration falls below a given threshold.

Algorithm 12 shows the pseudocode of the fast greedy refinement.

4.1.5 Analysis of Equivalence Class Sizes

In Section 4.1 it has already been mentioned that the idea for the RG algorithm is based on the observation that in natural networks many joins are equal with respect to the induced modularity change ΔQ and that, anyway, choosing in every step of the algorithm the join with the highest value of ΔQ is no guarantee to reach the global optimum.

In the following the behavior of the algorithm will be discussed in detail. Figure 4.2 shows the distribution of the sizes of the equivalence classes at

Algorithm 12 Fast greedy refinement

Input: undirected, connected graph g, partition p
Output: clustering

```
▼ Fast greedy refinement
    change ← true
    while Change do
        change ← false
        forall the v ∈ V do
            C_c ← currentCluster(v, p)
            bestDeltaQ ← 0
            forall the neighboring clusters C_n of v as given by p do
                deltaQ ← moveDeltaQ(v, C_c, C_n)
                if deltaQ > bestDeltaQ then
                    bestDeltaQ ← deltaQ
                    C* ← C_n
                end
            end
            if bestDeltaQ > 0 then
                move(v, C_c, C*)
                change ← true
            end
        end
    end
```

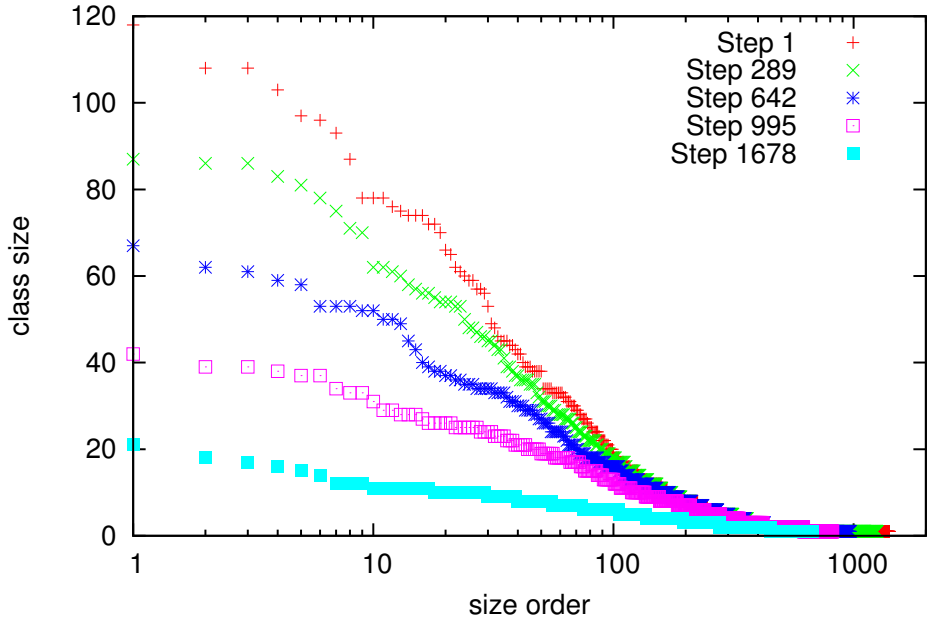

Figure 4.2: Distribution of the sizes of the equivalence classes at five steps of the RG algorithm during a sample run on the yeast dataset (see Section 4.6.1). Step 1 shows the initial situation before the first join. The other four steps are those steps where the average class size is for the first time lower or equal to 8,6,4 respectively 2. The x-axis has a logarithmic scale.

five distinct steps. At the beginning there are many large classes. During the run of the algorithm the class sizes decrease. One reason is that the number of clusters decreases, but it is more significant that through the joins of clusters the number of similar structures decreases.

Because the change of modularity caused by a merger of two clusters C_i and C_j is $\Delta Q(C_i, C_j) = 2(e_{ij} - a_i a_j)$, only the number of edges connecting both clusters and both sums of outdegrees influence ΔQ. That means, when initially all clusters are singletons and therefore e_{ij} is either 0 or $1/(2m)$, only the degrees of the singletons determine the ΔQ. As most natural networks have a power-law degree distribution, most vertices have a low degree. As a result of that, many pairs of vertices (= possible joins) have the same ΔQ.

Equivalent joins are defined as follows:

$$j_1 \equiv j_2 \Leftrightarrow \Delta Q(j_1) = \Delta Q(j_2) \tag{4.1}$$

Table 4.1: Overview of the number of equivalence classes of joins and average sizes of equivalence classes before the first join for several networks (for a dataset description see Section 4.6.1).

Name	Edges	Equiv. classes	Avg. class size
Karate	78	36	2.17
Jazz	2742	900	3.05
Email	5451	677	8.05
Yeast	7031	675	10.42
PGP	24340	1616	15.06
Condensed Matter	116181	3475	33.43
WWW	1090108	9820	111.01

with $\Delta Q(j)$ the modularity delta when executing the join j. Table 4.1 shows the initial average size of equivalence classes of mergers for several networks.

The randomization strategy exploits that in natural networks many joins are identical with respect to ΔQ. The larger a network is, the larger the equivalence classes of identical joins are. This shows that the randomization strategy is especially suitable for large networks. The quality and run-time evaluation discussed in Section 4.6 supports this conclusion.

A sample run of the RG algorithm on the yeast dataset shows the extent of the local similarities. Figure 4.3 shows the number of possible joins and the equivalence classes of joins. The number of possible joins decreases constantly, because every merge operation decreases the number of clusters and with it the number of possible joins. The number of equivalence classes increases in the beginning, because the number of distinct cluster degrees increases and the number of distinct connection strengths of clusters increases. Later on, the number of equivalence classes decreases when the effect of decreasing numbers of possible joins exceeds the effect of diversified cluster degrees and cluster connection strengths.

Figure 4.4 shows the resulting average size of the equivalence classes at each step. The graph has a convex shape. The average class size decreases especially fast in the beginning and looses momentum to the end.

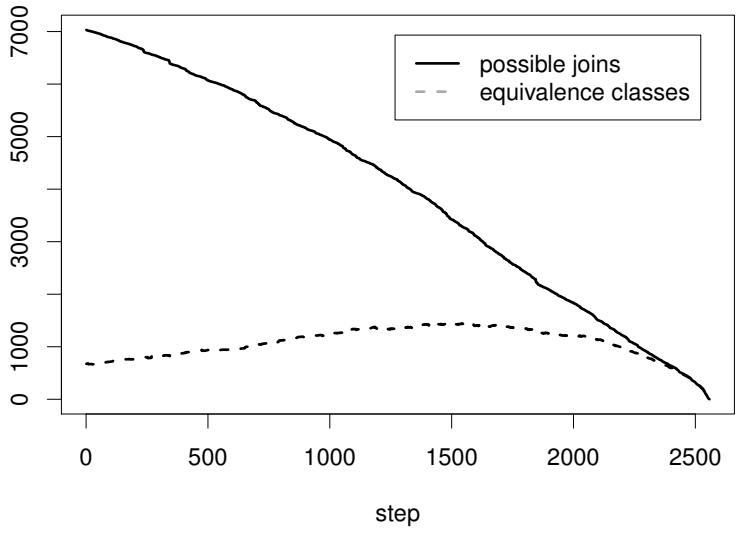

Figure 4.3: Number of equivalence classes and number of possible joins (= number of edges) during a sample run of the RG algorithm on the yeast dataset.

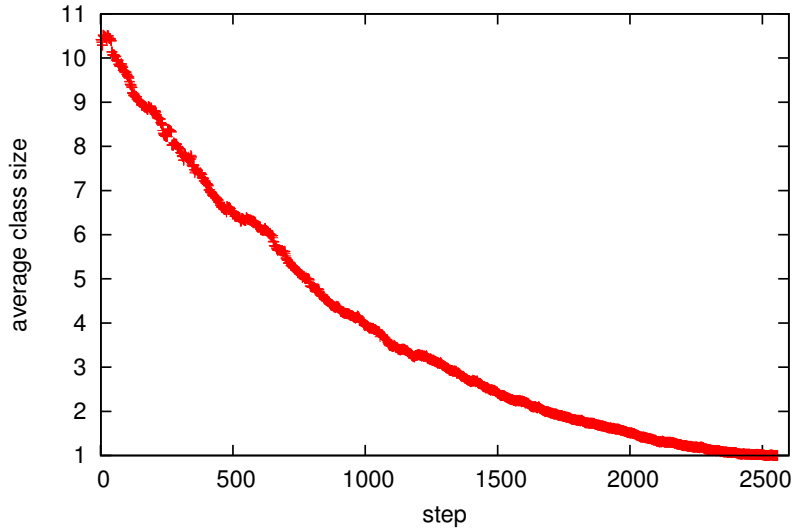

Figure 4.4: Average class size of the equivalence classes during a sample run of the RG algorithm on the yeast dataset. (Note, that the minimum average is always 1 at the end of the run.)

4.2 Balancedness

By definition, agglomerative hierarchical algorithms start with a partition of the graph into singleton clusters. The clusters get merged step by step and a complete dendrogram is created. From this dendrogram the clustering with the highest modularity will be extracted. Most agglomerative hierarchical algorithms for modularity clustering use the modularity difference upon a join (ΔQ) as the decision variable. One or more mergers with the highest values of ΔQ picked either from all possible mergers (PG, MSG) or from a sample (RG) are executed.

A critical problem of agglomerative hierarchical clustering algorithms for modularity optimization is what can be denoted as the locality problem. The ΔQ of the merger of two clusters depends on how strongly their respective neighborhoods are already contracted. Figure 4.5 shows the graph with which the dependence of the merge process on prior mergers by a greedy selection process will be illustrated. Under the assumption that the shown vertices are a subgraph of a graph with 100 vertices and 400 edges the following example is discussed. All neighboring vertices of v_1 have the same degree. Therefore, $\Delta Q(v_1, v_y)$ is the same for y = 2,3,4 and 5. If v_2 and v_3 are already merged to the cluster $\{v_2, v_3\}$, $\Delta Q(\{v_1\}, \{v_2, v_3\}) > \Delta Q(\{v_1\}, \{v_4\}) = \Delta Q(\{v_1\}, \{v_5\})$. However, if v_4 and v_5 are already merged to the cluster $\{v_4, v_5\}$, $\Delta Q(\{v_1\}, \{v_4, v_5\}) > \Delta Q(\{v_1\}, \{v_2\}) = \Delta Q(\{v_1\}, \{v_3\})$. The cluster with which v_1 will be merged solely depends on which of v_1's neighbors has been merged before. Let us assume that there is no other pair of vertices with a higher ΔQ than $\Delta Q(\{v_2\}, \{v_3\}) = \Delta Q(\{v_4\}, \{v_5\})$ (e.g. because all not shown vertices have a degree greater than six), then the final cluster assignment of v_1 is only determined by the sequence of the mergers of the vertices.

Because of this locality problem, unbalanced merging processes lead to bad merging decisions. The merging process of an agglomerative hierarchical algorithm is defined as balanced, if all (remaining) clusters grow roughly evenly. That means, the current size of a cluster is not correlated with the cluster's probability to be merged in the next step. A merging process is maximally unbalanced, when a single cluster grows while all other remaining clusters remain singletons.

The problem of an unbalanced merging process is shown by a comparison of the three algorithms RG, PG and MSG. First, the differences in the balancedness of the merging process are shown. Afterwards an analysis of the impact of balancedness on the merging process follows that will show why the randomized variant achieves superior clustering results in terms

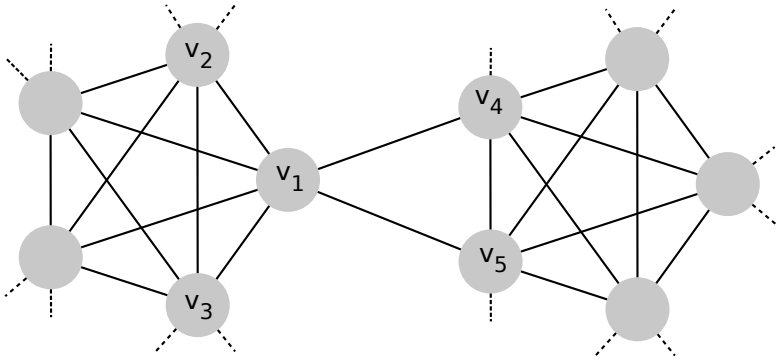

Figure 4.5: Example for influence of prior mergers on merge decision.

of modularity.

Figure 4.6 shows the merging processes of RG, PG and MSG on the yeast dataset. The plain greedy merging process contains many phases where one cluster is merged many times in a row. The largest of these phases consists of 347 joins. In contrast, the clusters grow simultaneously for RG. For the larger PGP dataset the differences in the merge process are even more striking as can be seen from Figure 4.7. Another approach with a more balanced merge process than that of PG is MSG. This algorithm joins in every iteration all clusters within the top l equivalence classes with regard to modularity. Not recalculating the ΔQs after every join, but executing several joins from one calculating step, lets several clusters grow simultaneously. However, the merge process is still not balanced and the few large clusters that grow simultaneously grow extremely fast.

An unbalanced merge process means that there is a considerable size difference between the clusters during the process. While a few clusters are large, most clusters are small or even singletons. This uneven cluster sizes mean that some parts of the network are much stronger contracted than others. The uneven graph contraction directly influences the local search heuristic of a greedy algorithm, as shown in the example motivated in Figure 4.5 and discussed above.

PG merges in every step those two clusters C_i and C_j where $\Delta Q(C_i, C_j)$ is maximal. As a result of this, a large cluster tends to get merged over and over with singletons. From the perspective of the singleton $C_i = \{v_x\}$, large neighboring clusters are a problematic bias for the determination to which cluster the singleton belongs in an optimal clustering.

In Section 3.2.3.1 another problem of unbalanced mergers has already been discussed. When the runtime complexity of an algorithm depends

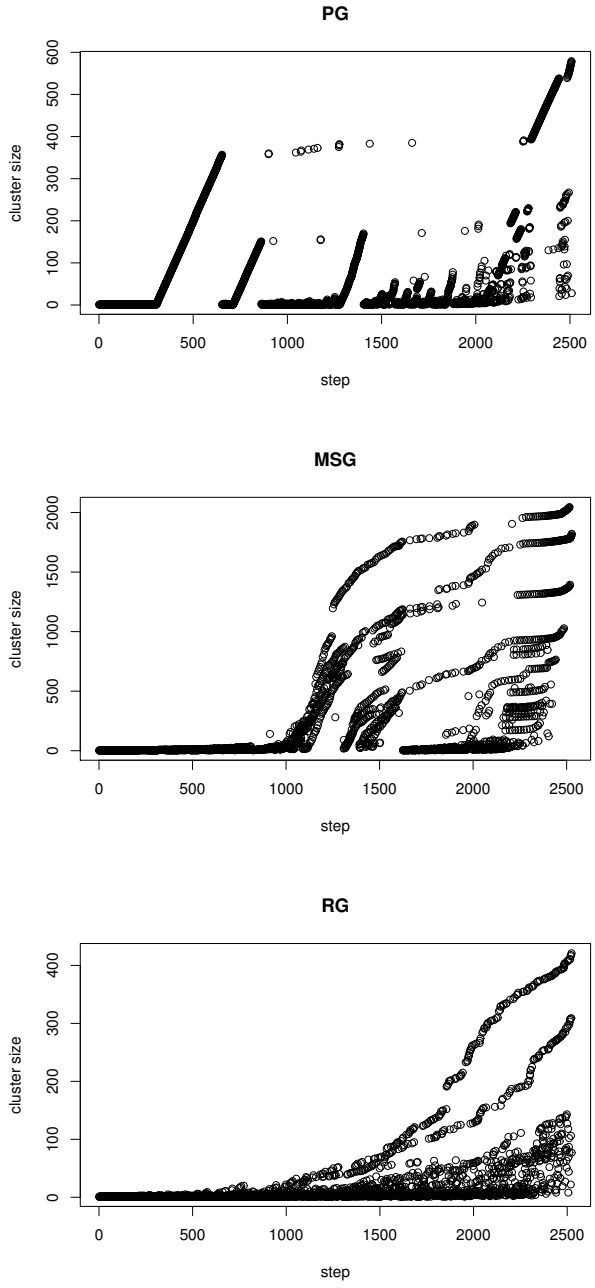

Figure 4.6: Sample merge processes of the PG, the MSG (level parameter $l = 35$), and the RG algorithm on the yeast network from [KCY+06] (in each case without applying refinement procedures). PG achieved a modularity of 0.670, MSG 0.698 and RG 0.680. The depicted cluster size is the size of the larger of both merged clusters. Note the different scales of the y-axis.

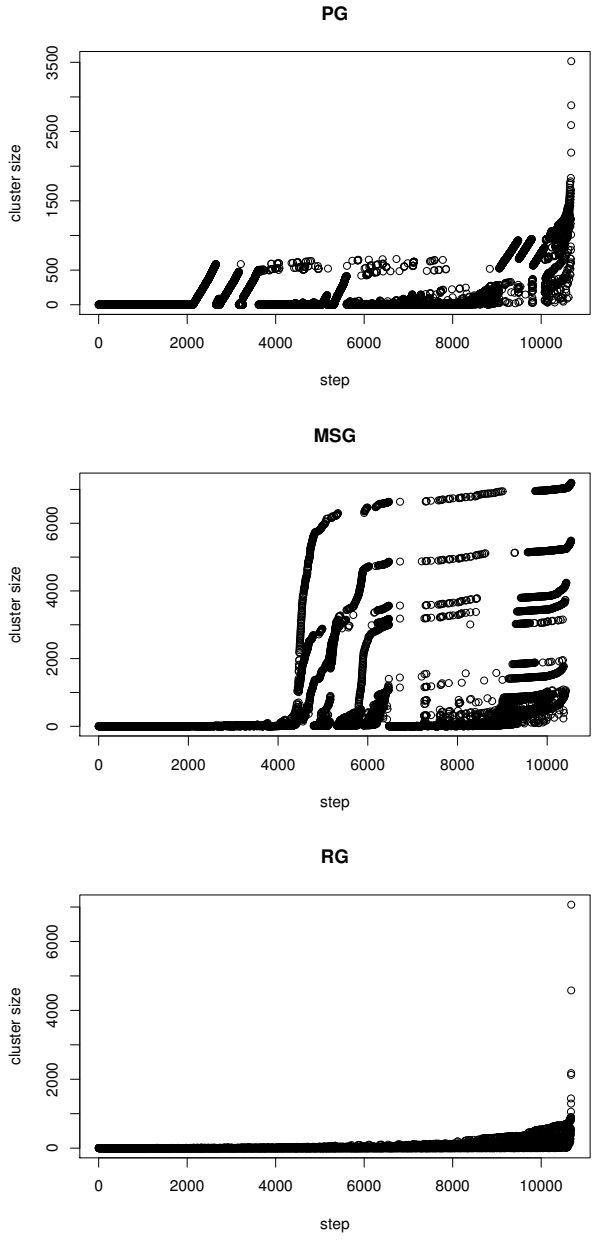

Figure 4.7: Sample merge processes of the PG, the MSG (level parameter $l = 44$), and the RG algorithm on the PGP key signing network from [BnPSDGA04] (in each case without applying refinement procedures). PG achieved a modularity of 0.849, MSG 0.869 and RG 0.874. The depicted cluster size is the size of the larger of both merged clusters. Note the different scales of the y-axis.

Table 4.2: Fractions of vertices of sample clusterings for different datasets that are not connected to any vertex in another cluster.

Network	#Vertices	#Inner Vertices	%Inner Vertices
Karate	34	17	50%
Jazz	198	48	24%
Email	1133	393	35%
PGP	10680	9340	87%
Condensed Matter	27519	15996	58%
WWW	325729	301120	92%
LiveJournal	4843953	2922965	60%

on the depth of the dendrogram, an unbalanced merge process is negative not only for the quality, but also for the time-performance.

4.3 RG+ Algorithm

As shown in Section 4.2 the contraction of the neighborhood of a vertex influences the gradient at the vertex (ΔQ). Furthermore, it has been shown in Section 4.2 that the randomized approach of RG prevents an uneven contraction of the graph. However, prior mergers still influence succeeding merge decisions. Obviously, for some vertices it is harder to decide to which cluster they belong than for others. For example, a vertex with degree 1 always belongs to its only neighbor in an optimal clustering. Making decisions about vertices that are easier to assign to the 'correct' cluster first would provide more information for the harder to make decisions. This is the starting point of our idea for the RG+ algorithm.

A simplified, but illustrative notion for this separation into 'easy' and 'hard' decisions is that some vertices are on the 'border' of clusters and have connections to vertices in other clusters. These vertices are the only ones that can potentially be assigned to a 'wrong' cluster (given that we already know the correct assignment for all other vertices). Vertices that are only connected to other vertices in the same cluster are denoted in the following as inner vertices with respect to a particular clustering. Vertices that have connections to at least one vertex in a different cluster are denoted as boundary vertices. Table 4.2 shows that for many datasets more than 1/2 of all vertices are inner vertices.

But the simplified distinction between inner vertices and boundary vertices is insufficient to separate those vertices that are 'easy' to assign to

the correct clusters and those that are 'hard' to assign correctly. If it was not, it would be sufficient to take the groups of inner vertices as a starting point and then search for the best assignment of the boundary vertices. Unfortunately, the possibly wrong assignments of any of the boundary vertices would have affected the whole merge process and, therefore, the assignment of the inner vertices will be wrong, too. However, the comparison between inner and boundary vertices indicates that there will be a considerable amount of vertices which have a clear cluster assignment.

4.3.1 Idea

In contrast to non-hierarchical heuristics, most hierarchical algorithms can not alter decisions once they are made. Once a hierarchical divisive approach separated two vertices into different clusters they cannot be remerged later on. Likewise, a hierarchical agglomerative approach cannot separate two vertices once they are merged. This means, each decision (whether it is a split or a join) narrows the search space. Of course, hierarchical algorithms could use backtrack strategies to search along several join paths. But this is a very expensive search strategy, as this strategy does not correct errors but actually creates several dendrograms. The limitation of not being able to alter decisions once they are made does not apply to, for example, a simulated annealing algorithm, which can reach every solution in the search space independently of the current position in the search space (with different probabilities, of course).

The heuristic of the randomized greedy algorithm (and heuristics of other greedy algorithms for modularity clustering as well) is very simple and completely local. An influence on the ΔQ, which is the decision variable of the greedy algorithms, have only the two involved clusters of a merge operation. Not even the structure of the subgraph including the vertices in a distance of one is taken into account. Although the randomized greedy algorithm is able to find very good partitions, it makes many bad decisions in the dendrogram building phase that have to be corrected in the refinement phase. However, the refinement procedure is only able to fix some bad assignments of vertices to clusters. If the merge process is led by early bad merge decisions into a section of the search space where all local maxima are low compared to the global maximum, the refinement will not be able to leave this section (because it is local, too).

A different decision variable than ΔQ would be desirable: one that is not totally local. It should take more information into account so that

supposedly good mergers that are bad from a more global view are not realized. A drawback from taking a more global view is the higher time complexity, when not just two clusters (the ones that are candidates for a merger) have to be considered, but also their direct neighbors or, even worse, all clusters within a distance of two. The problem is to find a fast heuristic with a better accuracy (in determining which vertices belong to the same cluster in the clustering with maximum modularity) than the accuracy provided by the ΔQ.

The idea behind the RG+ algorithm is based on the following assumption: Some vertices will belong to the same group in every possible clustering of the graph representing a local modularity optimum. If these vertices could be identified and the respective groups could be built, then the graph can be collapsed to some degree without errors, i.e. without merging two clusters that contain vertices that do not belong together in an optimal clustering. This idea is related to the approaches of e.g. Raghavan et al. [RAK07] and Gfeller et al. [GCDLR05] who do splitting, respectively, measuring the significance of a cluster assignment. Gfeller et al. compute several partitions of a graph where each time random noise is added to the graph. Then, they calculate for all pairs of vertices the probability of being in the same cluster from the partitions. This way, they try to identify instabilities. Raghavan et al. split several solutions into maximal overlaps as they expect their label propagation algorithm not to identify all communities in a single run.

Let $P^* = \{C^{*1}, \ldots, C^{*s}\}$ be the set of all clusterings where $Q(C^{*t})$ is a local optimum. Those pairs of vertices v_x, v_y that are part of the same cluster for all locally optimal clusterings need to be identified, i.e.

$$\forall_t \exists_k : v_x \in C_k^{*t} \wedge v_y \in C_k^{*t}. \tag{4.2}$$

This relation between vertices is transitive. That means,

$$\begin{aligned} &(\forall_t \exists_k : v_x \in C_k^{*t} \wedge v_y \in C_k^{*t}) \\ \wedge &(\forall_t \exists_k : v_y \in C_k^{*t} \wedge v_z \in C_k^{*t}) \\ \Rightarrow &\forall_t \exists_k : v_x \in C_k^{*t} \wedge v_z \in C_k^{*t} \end{aligned}$$

Because of the transitivity, the groups of vertices that pairwise belong together in an optimal clustering, can be easily built. These groups will be denoted as *core groups*.

4.3.2 Implementation

The idea for the advanced clustering algorithm assumes that identifying the core groups is easier than finding the optimal clustering directly. Unfortunately, there is no good reason to believe that identifying pairs of vertices that are always in the same cluster in all local optima is easier than directly finding the clustering with maximal modularity. Vertices with only one neighbor can be easily paired with their respective neighbors. For all vertices with a higher degree, it is not clear that they have any neighbor that is in the same cluster in all locally optimal clusterings. The presented idea is rather an illustrative description than a statement about the topology of the search space.

However, the idea is still helpful. It is unclear, if there are groups at all, but that does not matter as it is impossible to compute them in reasonable time anyway. (Computing the core groups would require to compute P^* completely. Then, the clustering with maximal modularity would also be known and the core groups are not required any more.) Therefore, a heuristic needs to be employed to assess the core groups. If in the core groups vertices are grouped that do not belong to the same core group for all local optima, this does not matter as long as these vertices belong to the same cluster in the globally optimal clustering. Still, this is a heuristic approach.

The implementation of the idea for RG+ into an algorithm consists basically of two phases:

1. Assess core groups.

2. Cluster core groups.

In the first phase, the algorithm tries to assess the core groups. A small sample of z locally optimal clusterings is used to assess the core groups. This approach shows the first significant difference to RG and most other algorithms for modularity maximization. While the ΔQ heuristic of the RG algorithm is purely local, this heuristic to build the core groups of vertices is based on a sample of local maxima and, therefore, a 'more global' one.

Because of its nondeterministic nature, the RG algorithm (see Algorithm 11) creates with a high probability a different clustering every time it is executed. Therefore, and because of its speed, RG is the ideal algorithm to create the set of clusterings needed for the assessment of the core groups.

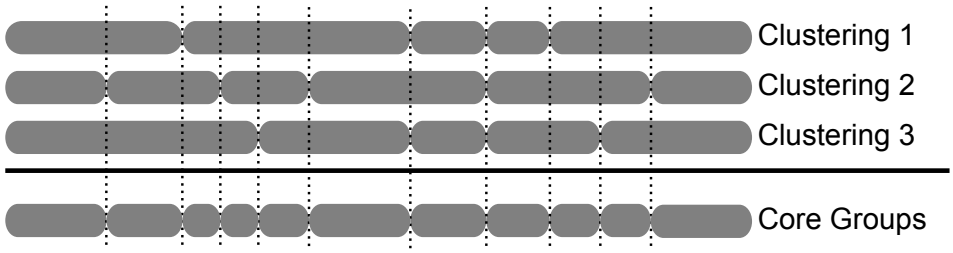

Figure 4.8: Example how core groups are extracted from several clusterings.

Once the initial clusterings are created, the core groups get assessed from this clustering. A schematic example for the extraction of core groups from clusterings is shown in Figure 4.8.

Algorithmically this task is performed as follows. First, one clustering is regarded as the temporary set of core groups. Then, one clustering at a time is used to split groups of the temporary set of core groups whenever vertices are part of the same temporary group but in different clusters in the currently processed clustering. Let $g_{v_x,t}$ denote the core group of which vertex v_x is part of in step t and $C_t(v_x)$ denote the cluster of v_x in the t-th clustering. Whether two vertices v_x and v_y are in the same temporary core group after the t-th split is given by the recursive formulation

$$g_{v_x,t} = g_{v_y,t} \iff g_{v_x,t-1} = g_{v_y,t-1} \wedge C_t(v_x) = C_t(v_y). \tag{4.3}$$

In a second phase, the core groups will be used as the starting point for the main clustering step. The partitions used to extract core groups have been generated starting from a partitioning of the graph into singletons. Now, the RG algorithm is started once again with the core groups as the initial partition.

The pseudo-code for RG+ is given in Algorithm 13 where the function `RG(graph,partition)` contracts the `graph` starting from a given `partition`. The function `split(c,clustering[i])` splits the cluster `c` into subclusters, so that only vertices that are in the same cluster in `clustering[i]`, are in the same subcluster.

4.4 Custom Datatypes of RG and RG+

The time and memory complexity and thus the scalability of a piece of software depends not only on the employed algorithms, but on the way the

Algorithm 13 RG+ algorithm

Input: Graph graph, number of clustering for core group generation z
Output: clustering

▼ **Identify core groups**

```
singletons ← {{v_x}|v_x ∈ V}
for i=1...z do
    clusterings[i] ← RG(graph, singletons)
end
coreGroups ← clustering[1]
for i=2...z do
    forall the clusterc ∈ coreGroups do
        if not all vertices in c in same cluster in clusterings[i] then
            coreGroups ← coreGroups − c + split(c, clusterings[i])
        end
    end
end
```

▼ **Generate clustering**

```
clustering ← RG(graph, coreGroups)
```

algorithms have been implemented. One key factor of the implementations are the utilized data structures. The way, data gets stored best, depends on the operations on the data structure that need to be supported and on the relative frequency of the calls of such operations. A prominent example are the different implementations of a priority queue. A priority queue is a data structure, where elements with associated keys are stored. A fast extraction of the element with the lowest key is sought. A priority queue can be implemented in several ways, e.g. with a linked list or a balanced binary tree [CLR09]. Extracting the element with the lowest key takes $O(n)$ when the priority queue is implemented with a linked list but only $O(\log n)$ when a balanced binary tree has been used. On the contrary, inserting a new element is faster ($O(1)$) with linked lists than with balanced binary trees ($O(\log n)$). This brief comparison is intended to illustrate the difference between the complexity of an algorithm and the complexity of an implementation.

The focus of the implementation of the RG and RG+ algorithm was on scalability. To be able to process huge datasets, resource-efficiency in terms of time and memory is required. For this reason, C++ has been chosen to implement the algorithms as it allows better control over the data storage than most other modern languages (e.g. Java or Python).

The run-time efficiency of the reference implementation of the RG and RG+ algorithms relies on the proficient choice and implementation of data structures. Those data structures that have a major impact on the run-time of the clustering process and are necessary for the complexity analysis conducted in section 4.5 are discussed in the following.

4.4.1 Sparse Matrix

The data structure storing the matrix (e_{ij}) is the most run-time critical one, because most read and write operations of the RG algorithm are on this data structure. The matrix storing the connections between the communities will be huge, but very sparse in the beginning. Its rank decreases with every join of two communities until it is one after the last community join. Concurrently, the once sparse matrix will become denser. Especially in the beginning, when the rank of the matrix is very large while the density is very low, a fast way to join sparse rows and columns is required - that means a procedure to replace two rows, respectively two columns, by their sum.

The matrix (e_{ij}) is stored as an array of hash tables. Only non-zero elements are stored. This allows fast access to the elements of interest.

By using an array of hash tables instead of a matrix two rows can be joined in linear time to the number of non-zero elements as it can be iterated over the key set of a row. The costs of resizing the hash table are almost negligible as it is rarely necessary due to the limited size changes of rows (which again is a result of the balanced growth process).

The used hash tables are instances of the `unordered_map` container from the Boost C++ Libraries [Boo]. For each hash value a linked list (called bucket) stores all associated elements. This hash table implementation has an amortized lookup speed of $O(1)$.

4.4.2 Active Row Set

To be able to select a random cluster which will be considered for a merger, a list of all rows in the matrix (e_{ij}) that are still active, is stored. An active row is one that represents a cluster, whereas inactive rows are those that are no longer in use because they have been merged with other rows. The ActiveRowSet is an encapsulated array of all active rows. Selecting a random cluster means reading a random field of the array. Once a row gets inactive, as the corresponding cluster has been merged into another cluster, the corresponding field in the array is replaced by the value in the last used field of the array and the counter of used fields is decreased by 1. This means, an array of length n is initialized for a graph with n vertices. After 3 arbitrary merge operation, the counter of used fields has the value $n - 3$ and the ids of all still active rows are in the first $n - 3$ fields of the array.

This data structure allows looking up and maintaining the list of active rows in constant time.

4.5 Complexity

In this section the time and memory complexity of the reference implementations of the two algorithms RG and RG+ will be analyzed. The complexity analysis is important to assess the scalability of an algorithm as it describes the algorithms asymptotic requirements of computational resources.

4.5.1 Time Complexity

4.5.1.1 RG

The RG algorithm has to initialize the data structures for the matrix (e_{ij}) and the vector $\mathbf{a}$ before it can start with building the dendrogram. To initialize the matrix (e_{ij}), for each undirected edge two fields of (e_{ij}) have to be written. Simultaneously, the vector $\mathbf{a}$ can be created through summation of the matrix rows. This has a complexity of $O(m)$, with m the number of edges.

RG needs to perform $n-1$ joins to build the complete dendrogram for a graph with n vertices and m edges. For every join, k clusters are randomly selected, and for each of them the ΔQs of a merger with their respective neighbors is calculated. Then, the pair with the highest ΔQ is joined.

The complexity of the dendrogram building phase is roughly estimated with the help of the average cluster size. The degree reduction in (e_{ij}) is not considered. The algorithm performs two tasks in every step. First, it searches for a join and then executes it.

The search for a join consists of calculating the ΔQ for k randomly selected clusters and each of their respective neighbors. A cluster can not have more neighboring clusters than the outdegree sum of the vertices it consists of. The average outdegree of a vertex is $2m/n$. Because RG starts with n singleton clusters and builds the dendrogram by merging two clusters in every step until all vertices belong to one cluster, the average cluster size over all steps is:

$$\begin{aligned} s(n) &= \frac{\sum_{i=1}^{n} \frac{n}{i}}{n} \\ &= \sum_{i=1}^{n} 1/i \end{aligned} \tag{4.4}$$

$s(n)$ is the n-th harmonic number and, therefore, $s(n) \approx \ln(n) + \gamma$ where γ is the Euler-Mascheroni constant (0.5772...) (see p. 75 [Knu00]). Altogether, the complexity of finding the next join is $O(\ln(n) \cdot m/n)$.

To execute this join, two rows of the matrix (e_{ij}) need to be merged. To minimize the costs, the elements of the sparser row are copied into the denser of both rows. This copy procedure takes time linear to the number of non-zero elements of the sparser row. As shown above, an upper limit for the average compound cluster degree is $\ln(n) \cdot m/n$.

Both, finding and executing a join have a complexity of $O(\ln(n) \cdot m/n)$ and these tasks are performed in every one of the $n-1$ steps. Together with the initialization, the overall complexity is $O(m + n(\ln(n) \cdot m/n)) = O(m \ln n)$. Note, that this is just an upper bound for the average case.

Estimating the average case is hard, because of the complex estimation of the average cluster degree. Merging two clusters has two effects. To begin with, the outdegree of the new cluster that emerges from the join has an outdegree of $d(C_1) + d(C_2) - |o(C_1, C_2)| - 2l(C_1, C_2)$, where $d(C_i)$ is the degree of cluster C_i, $o(C_1, C_2)$ is the overlap of neighbors from C_1 and C_2, and $l(C_1, C_2)$ is the number of edges connecting vertices in C_1 with vertices in C_2. Additionally, the outdegree of all clusters in $o(C_1, C_2)$ decreases by 1. Assessing the overlap is the critical problem. The average overlap of two randomly selected neighbors would be a bad estimation. Those pairs of clusters that are selected for a join will probably have many common neighbors. So, for not degenerated real-world datasets the complexity of $O(m \ln n)$ will not be a tight bound.

The complexity of the refinement depends on the number of clusters found by the main algorithm and the quality of the clustering, because the quality influences the number of iterations required. In each iteration, for every vertex the ΔQ for all moves to adjacent clusters has to be calculated. The refinement procedure stops when in one iteration no move with positive ΔQ has been found. For a partition $C = \{C_1, \cdots, C_p\}$, ΔQ is calculated for the move of every vertex to every cluster it is connected to. So, each iteration has a complexity of $O(pn)$ and i iterations take $O(ipn)$. Unfortunately, the number of clusters and the number of required iterations are unknown as they depend on the input data. Table 4.3 shows the results of the test that have been conducted to assess the asymptotic runtime of the refinement procedure. The high time ratio for the refinement on the LiveJournal dataset results from the large number of iterations (between 19 and 50). After 5 iterations usually more than 95% of all vertex moves have been made. A limitation of the number of iterations ensures that the refinement does not scale worse than the dendrogram creation without significantly effecting the result, but this implies that the solution is not guaranteed to be a local optimum.

4.5.1.2 RG+

For RG+, RG needs to be run $\ln n$ times (see Section 4.6.2) and then the core groups need to be extracted from the clusterings. RG without the fast greedy refinement has a complexity of $O(m \ln n)$. RG with an

Table 4.3: Time and modularity results by algorithm phase for RG and a selection of test datasets.

	Dendrogram		KL-Refinement		Time Ratio
Network	time t_d [s]	Q	time t_{kl} [s]	Q	t_{kl}/t_d
Yeast	0.032	0.680	0.018	0.695	0.56
PGP	0.110	0.874	0.050	0.878	0.45
Cond. Mat.	0.552	0.733	0.296	0.749	0.54
WWW	6.056	0.935	2.278	0.937	0.38
LiveJournal	29.921	0.734	20.296	0.760	0.68

iteration restricted refinement does not scale worse.

The core group extraction is done by integrating a clustering into the temporary core groups one at a time. In each integration step for each pair of vertices in a group of the temporary core groups is tested whether it is in the same cluster in the currently processed clustering. For a clustering with c clusters and a maximal cluster size of s this needs $O(cs^2)$. For a low number of clusters and very uneven cluster sizes the complexity could get almost as worse as $O(n^2)$, but in our test the core group extraction required between 0.8% and 1.5% of the total runtime independently of the size of the graph. In part this can be explained by the decreasing maximal size of a core group. Each integration step leads to a finer clustering in the temporary core groups, so that the processing costs per integration step tend to fall except in the very unlikely case that the currently processed clustering is identical with a previously processed clustering.

The final run of RG is started from the core groups, so that a considerably lower number of merge operations needs to be executed to create the full dendrogram than when starting from singletons. But the complexity estimation of $O(m \ln n)$ can be used once again, as this term is dominated by the core cluster creation in any case. Altogether, the time complexity is $O(\ln(n) \cdot m \cdot \ln(n)) = O(m(\ln n)^2)$.

4.5.2 Memory Complexity

The two data structures RG relies on are the matrix (e_{ij}) and the vector $\mathbf{a}$. (e_{ij}) is realized as an array of hash tables. After initialization this matrix has $2m$ elements. The hash overhead is a constant factor and does

not need to be taken into account. As shown for the time complexity, the total number of edges decreases in every step. The vector a_i has initially n elements which decrease by 1 in every step. The generated dendrogram is stored in a list of performed joins, which has $n-1$ entries. So, the compound memory required by RG has a complexity of $O(n+m)$.

The refinement step requires once again the storage of (e_{ij}) and $\mathbf{a}$. Additionally, for each vertex all adjacent clusters are stored. A cluster C_i is adjacent to a vertex v_x if $\exists v_y \in V : \{v_x, v_y\} \in E \wedge v_y \in C_i$. Thus, a vertex cannot be connected to more clusters than vertices and at most $2m$ connections need to be stored. Therefore, the refinement needs $O(n+m)$ memory.

RG+ runs RG several times. Additional to RG, RG+ needs to store the core groups. These groups are stored as an array of arrays and need $O(n)$ memory. To save memory, after each run of RG the generated clustering is used to split the core groups at once and the clustering is deleted before the next run of RG. As the final clustering of the core groups is a run of RG on an already partially contracted graph, it requires no higher amount of memory than in the initial runs. Thus, RG+ has a memory complexity of $O(n+m)$, too.

4.6 Evaluation

In this section, the reference implementations of several modularity optimization algorithms will be compared with regard to their performance in maximizing the modularity measure and with regard to the resource usage (time and memory). For the evaluation, the real-world datasets described in Section 4.6.1 will be used. The quality of the clusterings found is not evaluated. The ability to identify community structures by maximizing the modularity measure has been discussed in Section 3.1.2.

4.6.1 Datasets

For the evaluation of the algorithms publicly available real-world datasets are used. If necessary, these graphs were transformed into connected, undirected graphs by extracting the largest connected component of the symmetrized graph. Self-loops have been removed, when they exist. An overview of the datasets is provided by Table 4.4. The shown vertex and edge numbers are given for the transformed graphs. In the following the semantics of the evaluation networks as well as network transformations

Table 4.4: Real-world test datasets used for algorithm evaluation

Name	Network Type	Vertices	Edges	Reference
Karate	social	34	78	[Zac77]
Football	game plan	115	613	[GN02]
Net.Sci.	co-authorship	379	914	[GN02]
Jazz	musician overlap	198	2742	[GD03]
Email	communication	1133	5451	[GDDG+03]
Yeast	protein interact.	2559	7031	[KCY+06]
PGP	key signing	10680	24316	[BnPSDGA04]
Cond.Mat.	co-authorship	27519	116181	[New01]
WWW	hyperlink	325729	1090108	[AJB99]
LiveJournal	friendship	4843953	42845684	[LLDM08]

performed will be described.

Karate This dataset describes the personal relations between members of a karate club and was created by Zachary [Zac77], who analyzed how the club split up into two new clubs after an internal conflict. Zachary could show that the personal relations were a good indicator for the prediction of which member joined which of the new founded clubs. As this dataset is well-studied, it has been used by several authors to evaluate the quality of cluster methods. A good quality is assumed, when a method splits the network into the groups which have been observed. The partition with maximal modularity consists of four clusters and each observed group is represented of two of them [BDG+08].

No changes to the network were necessary, as it already had all required properties.

Football This network of United States college football represents the game schedule of Division I for the 2000 season. It was collected by Girvan and Newman [GN02]. The vertices in this network are the teams of Division I. Edges indicate that two teams have had a regular-season game against each other. For this network no transformation was necessary.

In contrast to the other networks of the test set of networks, the football network is no self-organized, evolved network. The teams are divided into conferences around 8-12 teams each which primarily influence the opponents of a team. Characteristic properties of the other networks as a

power-law degree distribution are not given for this network.

Network Science This network of co-authorship of researchers working in the field of network science has been compiled by Newman [New06a] from the bibliographies of two review articles on networks. From the network with 1589 scientists and 2742 links, the largest connected component with 379 scientists and 914 links has been extracted.

Jazz The Jazz network describes the network of jazz bands obtained by Gleiser and Danon [GD03] from *The Red Hot Jazz Archive*. It consists of 198 bands that performed between 1912 and 1940. A link between two bands means that they have at least one musician in common. The network is already undirected and connected, so no transformation has been conducted.

Email Guimera et al. [GDDG+03] compiled a dataset from the email interchanges between members of the Rovira i Virgili University (Tarragona, Spain). The network contains an unweighted, directed link from one person to another, when at least one email had been sent in this direction. From the original connected network with 1133 persons and 10902 directed links, a symmetrized network with 5451 links has been created.

Yeast The yeast dataset from Krogan et al. [KCY+06] describes the protein-protein interactions of Saccharomyces cerevisiae. The edge weights give the interaction probability of two proteins, where interaction means that e.g. two proteins carry out a molecular process together. The weighted network has been transformed into an unweighted one for our evaluation.

PGP This network represents the *web of trust* of users of the Pretty-Good-Privacy encryption software. In this network of PGP users, a link means that at least one of both users has signed the public key of the other one. Signing means that an user verifies that a public key belongs to the person stated in the key metadata. The dataset provided by Boguñá et al. [BnPSDGA04] already consisted only of the largest connected component of this network. Probably because of an error, the dataset contained 24 multiple links which have been removed.

Condensed matter This is the network of co-authorship of scientists posting preprint articles on the condensed matter archive at www.arxiv.org. The network is based on the preprints posted between January 1, 1995 and June 30, 2003. The network consists of 31,163 authors and 120,029 undirected co-authorship relations. The largest connected component of this network with 27519 authors and 116181 links is used.

LiveJournal LiveJournal [Liv] is an online community, where members can create blogs. Furthermore, members can create 'friend' lists where they add fellow members without requiring the consent of the added person. The result is a directed, unweighted network. Leskovec et al. [LLDM08] collected a dataset of 4,847,571 LiveJournal members and 68,993,773 directed links. We symmetrized the network and extracted the largest connected component with 4,843,953 members and 42,845,684 undirected links.

4.6.2 Parameter Settings

Extensive tests helped to identify those parameter settings, with which RG and RG+ find clusterings with the highest modularity.

For RG, as already mentioned in Section 4.1.3, the best results are achieved when the parameter k (which specifies the number of communities among which the join with the highest ΔQ is searched) is set to 1 for the first half of the steps and set to 2 for the second half. As shown in Table 4.1, natural graphs have only a few, but large classes of equivalent joins with respect to ΔQ. The behavior of the graphs is as discussed (see Section 4.1.5): With an increasing number of executed join operations the number of equivalence classes increases, too, and, therefore, the average class size decreases. During the first steps, the number of equivalence classes of joins is low and their sizes are large. Then, joining a vertex with that neighbor where ΔQ is maximal, is sufficient. Later on, when the graph is already semi-contracted, selecting the best join from a larger basis is necessary to prevent choosing a bad join.

For RG+, the best setting for RG, when running as a part of RG+, is determined as follows: First, the parameter k of RG for the initial runs of RG (starting from singletons) and the final run (starting from core groups) is separately determined. The tests showed that the value of k for the initial runs has no influence on the achieved modularity of the final clustering (Figure 4.9). The observed differences in the results are minimal, show no correlation with the value of k and are within the range

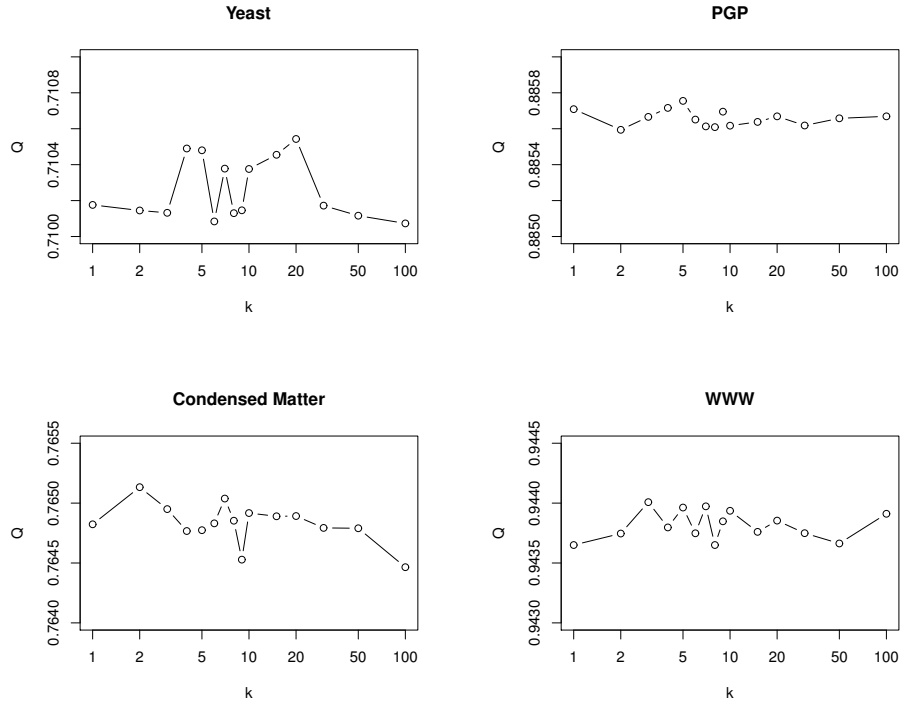

Figure 4.9: Average modularity of 30 test runs of RG+ on four datasets subject to the sample size k of the employed RG algorithm for core group creation phase. The sample size k in the core group clustering phase is set to 1000.

of expected variance. As the value of k does not matter at all, it is set to 1 to save computational costs.

For the core-groups clustering phase, however, the modularity achieved on average increases with increasing k (Figure 4.10). This analysis results fit with the explanation given above for different settings of k in the first and second half of the steps of RG. Here, the grouping of vertices into core groups eliminated the structural similarities in the graph. Therefore, a more exhaustive search is necessary. Setting k to 1000 showed to be sufficient. For smaller graphs, this means a complete search among all possible joins like PG does. For larger graphs this value is sufficient, too, and a higher value of k did not increase the modularity achieved on average.

A third parameter is the number of partitions used to generate the core

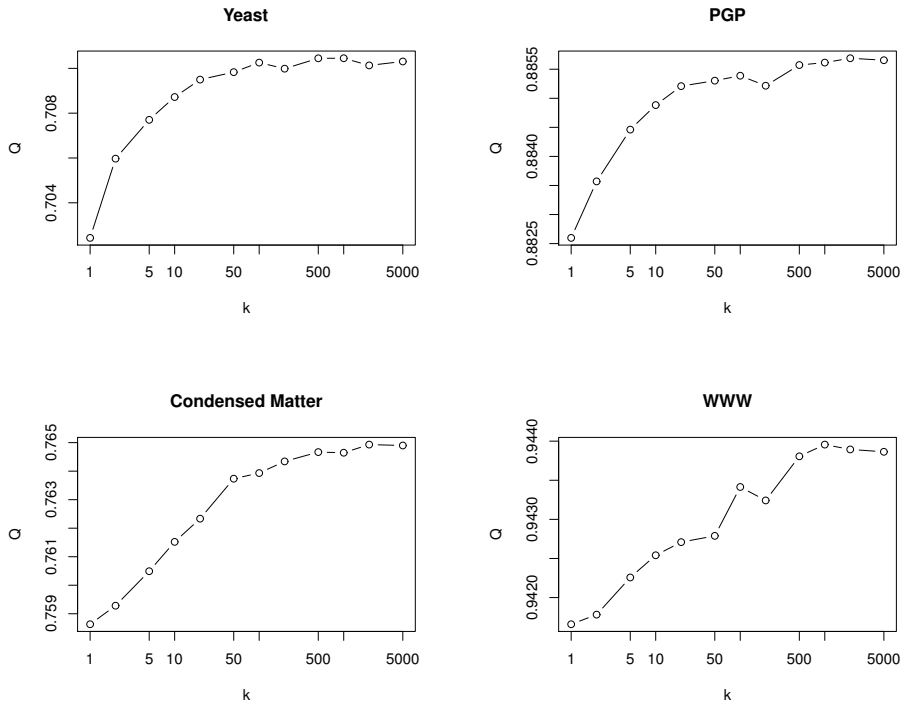

Figure 4.10: Average modularity of 30 test runs of RG+ on four datasets subject to the sample size k of the employed RG algorithm for core group clustering phase. The sample size k in the core group creation phase is set to 1.

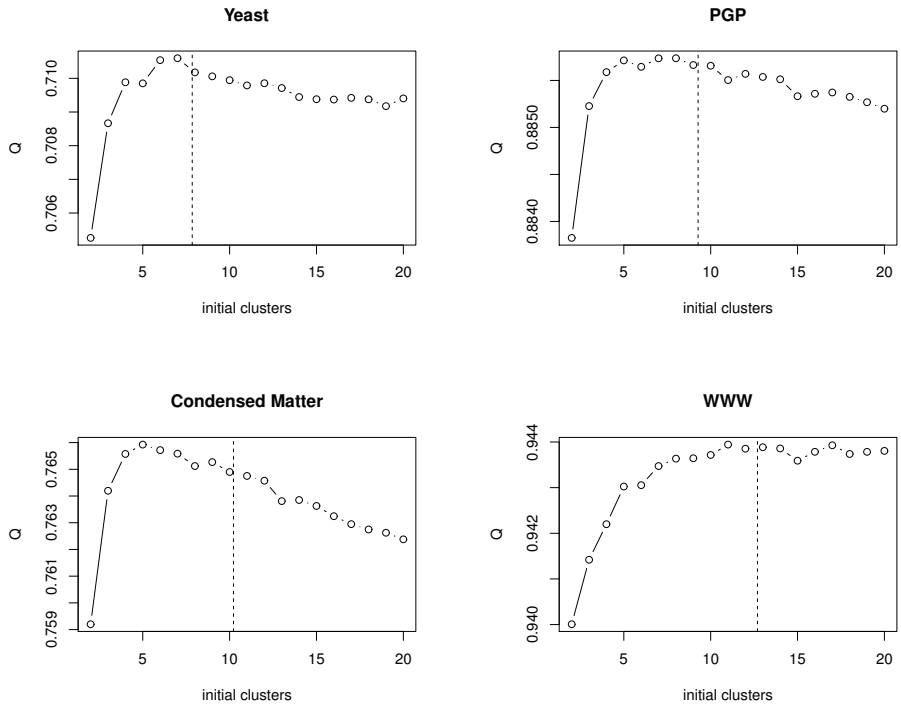

Figure 4.11: Average modularity of 30 test runs of RG+ on four datasets subject to the number of partitions used for core group identification. The dotted vertical line shows the value of $\ln n$ (where n is the number of vertices).

groups. Here, tests have shown that a rough estimation for the best value of z is about $z \sim \ln |V|$ (Figure 4.11). If z is set to a larger value, the average quality decreases again. It seems likely that with the number of clusterings the chance to receive an odd clustering increases and that this leads to a split of good core groups.

4.6.3 Evaluation Results

The algorithms RG and RG+ are evaluated in terms of clustering quality and runtime with help of the natural networks presented in Section 4.6.1. As a measure of comparison the results for the MOME algorithm and the BGLL algorithm are shown as well. Because RG, RG+, MOME and BGLL are non-deterministic, the results are given for the average case in Table 4.5 and for the best case in Table 4.6. The average case results show the modularity and runtime averages of 100 test runs in each case. The best case results show the modularity of the best clustering result achieved in a specified amount of time.

To achieve comparability for the algorithm evaluation, all results have been obtained from test runs on a dedicated test machine with an Intel Xenon 2.66 GHz CPU and 16 GB RAM. The machine runs Gentoo Linux and our C++ implementations of RG and RG+ have been compiled with GNU gcc 4.3.4. The source code of MOME has been kindly provided by Zhu. To be able to run MOME in the same Linux environment as RG and RG+, Visual C++ library calls have been exchanged with their equivalent GNU gcc library calls. The adapted sources have been recompiled with GNU gcc 4.3.4. The source code of the BGLL algorithm has been downloaded from the authors' website at `http://sites.google.com/site/findcommunities/`. Note, that the shown time and memory usage results are actually results for the respective implementations rather than the algorithms as such. Only the modularity results are actually independent from the implementation, if the pseudorandom numbers of the used random number generator are regarded as true random numbers.

In Table 4.6 the best achieved modularity of RG+ is compared to the upper bound and previously published results. The upper bound is calculated with help of the method and software of Agarwal and Kempe [AK08]. Agarwal and Kempe compute the upper bound by solving a linear program (LP) which is the relaxation of the integer linear program that finds a clustering with maximal modularity (see 3.2.1). For large networks, the LP consists of an enormous number of constraints. Even on a high performance computer the available main memory of 144GB limited the ability

Table 4.5: Modularity and runtime averages of 100 runs of RG, RG+, MOME and BGLL. Entries showing "oom" could not be calculated because the program ran out of memory (i.e. > 28 GB memory usage).

Network	RG		RG+		MOME		BGLL	
	Q	time [s]	Q	time [s]	Q	time [s]	Q	time [s]
Karate	0.412	0.004	0.417	0.005	0.420	0.002	0.419	0.000
Football	0.589	0.008	0.605	0.014	0.580	0.007	0.605	0.000
Jazz	0.444	0.015	0.445	0.058	0.444	0.011	0.443	0.000
Network Science	0.840	0.007	0.848	0.040	0.842	0.020	0.848	0.000
Email	0.572	0.038	0.580	0.276	0.573	0.082	0.543	0.000
Yeast	0.695	0.049	0.710	0.525	0.700	0.175	0.696	0.01
PGP	0.880	0.242	0.886	1.813	0.880	0.748	0.883	0.050
Condensed Matter	0.749	0.972	0.765	14.8	0.756	2.588	0.751	0.220
WWW	0.936	10.178	0.943	120.1	0.937	28.68	0.935	6.43
LiveJournal	0.760	481.9	0.771	10417.9	oom	oom	0.728	229.49

Table 4.6: Comparison of the best achieved modularity by RG and RG+ with the upper bound (UB) calculated with the method of [AK08], MOME and BGLL. The computation of the upper bound was not possible for larger graphs due to its computational complexity (denoted with n/a). MOME ran out of memory (oom) when processing the LiveJournal dataset.

Network	UB	Q_{RG}	Q_{RG+}	Q_{MOME}	Q_{BGLL}
Karate	0.420	0.420	0.420	0.420	0.419
Football	0.606	0.604	0.606	0.606	0.605
Jazz	0.446	0.445	0.445	0.445	0.445
Network Science	0.850	0.848	0.849	0.845	0.848
Email	n/a	0.578	0,583	0.575	0.572
Yeast	n/a	0.704	0.713	0.706	0.707
PGP	n/a	0.881	0.886	0.878	0.884
Condensed Matter	n/a	0.751	0.767	0.748	0.756
WWW	n/a	0.936	0.945	0.938	0.936
LiveJournal	n/a	0.763	0.772	oom	0.747

Table 4.7: Main memory usage measured with pmap ("writeable/private" memory). MOME ran out of memory (oom) on the LiveJournal dataset after using 16 GB RAM and 12 GB swap memory. For BGLL the peak memory of both executables required to get a partition is measured.

Name	RG	RG+	MOME	BGLL
PGP	5MB	6MB	25 MB	157 MB
Condensed Matter	16 MB	21 MB	71 MB	157 MB
WWW	171 MB	228 MB	764 MB	167 MB
LiveJournal	5.55 GB	5.78 GB	oom	1.7 GB

to calculate the upper bound. The largest network which could be processed is the Network Science dataset with 379 vertices and 914 edges. For those networks where the calculation of the upper bound was feasible, the results are shown. Note that it is unknown whether there is a clustering whose modularity reaches the upper bound or not. For larger networks, it is not possible to know whether RG+ achieves near-optimal modularity. However, the RG+ algorithm achieves superior modularity to previously published heuristics.

As shown in Table 4.5, RG achieves for most datasets about the same quality as MOME and BGLL with significantly lower computational effort than MOME but more effort than BGLL. However, RG outperforms MOME and BGLL for the two largest datasets in terms of modularity. RG+ is able to find clusterings with a much higher modularity than RG, MOME and BGLL. However, the cost of this higher quality in terms of modularity is a much higher runtime. But as can be seen from Table 4.7, RG+ scales only worse than RG and MOME in terms of runtime. The memory usage of RG+ scales only slightly worse and the total memory usage is only slightly higher compared to RG and much lower compared to MOME. BGLL has a low memory consumption. However, an exact comparison to the other algorithm implementations is difficult because the BGLL implementation allocates about 150 MB RAM independently of the actual requirements.

The results of Table 4.6 show that for most networks the equality of RG, MOME and BGLL in terms of modularity does not only hold for the average case but also the best case. Once again, the results of RG+ are significantly higher than those of RG, MOME and BGLL.

Table 4.8: Comparison of the number of clusters found in real-world datasets in test runs of different algorithms.

Network	RG	RG+	MOME	BGLL
Karate	3	3	4	4
Football	7	10	7	10
Jazz	3	3	3	4
Network Science	18	19	18	18
Email	12	11	10	11
Yeast	25	36	25	36
PGP	84	117	69	102
Condensed Matter	57	77	49	66
WWW	704	921	233	472
LiveJournal	1924	3079	n/a	5471

4.6.4 Cluster Size Distributions

The results presented in Section 4.6.3 compare the performance of algorithms with respect to the objective function. When used in an application context to analyze real-world data to solve a real-world problem, further properties of the clusters identified by a clustering algorithm are of interest. One essential property of partitions is the distribution of the cluster sizes.

The number of clusters of the partitions identified by RG, RG+, MOME and BGLL in test runs on several real-world datasets are shown in Table 4.8.

The analysis of the merge processes of hierarchical clustering algorithms in Section 4.2 explain why the RG+ algorithm usually finds solutions with more clusters than the RG algorithm. Because RG totally relies on the ΔQ measure for selecting joins, and this measure is totally local and is likely to merge at least some vertex pairs where both vertices belong to different clusters in an optimal clustering. Once a bad merge operation groups together vertices from two different "optimal" clusters, this cluster nucleus is likely to be merged with vertices from both "optimal" clusters.

The MOME algorithm finds a considerable smaller number of clusters than RG and RG+ for larger networks. For smaller networks the results are about the same for all algorithms as all find a near optimal, respectively optimal solution.

The number of clusters shows only an incomplete picture of the parti-

tions an algorithm identifies. For the six datasets Network Science, Email, Yeast, PGP, Condensed Matter and WWW Figures 4.12–4.14 show the distribution of cluster sizes for RG, RG+, MOME and BGLL. Figures for the smaller datasets have been omitted as all algorithms find the same near optimal solutions.

The results show that especially MOME identifies larger clusters than RG+. This algorithmic behavior is especially intense for the larger datasets. The RG algorithm shows a less clear behavior. RG produces results with a cluster size distribution very similar to RG+ for some datasets (PGP, WWW), but partitions with cluster size distributions more similar to MOME for other datasets (Yeast, Condensed Matter). In contrast to this difference regarding the small clusters, the three algorithms show no pattern regarding the maximal cluster size.

The explanation for the higher number of small clusters is the same as the explanation for the higher number of total clusters. Because of the core group identification step of RG+, this algorithm is able to separate vertices that both other algorithms may have merged into one cluster, because of the complete reliance on the ΔQ measure.

4.6.5 Evaluation Summary

To wrap up the results of the evaluation of different clustering algorithms, RG and RG+ are two algorithms which are able to identify clusterings with high quality in terms of modularity. The quality improvement RG+ is able to achieve comes with extra computational costs. When choosing one of the algorithms a trade-off between runtime and quality has to be made.

The actual problem in clustering huge datasets is the memory complexity. MOME was not able to cluster the LiveJournal dataset with about 4.8 million vertices and 43 million edges as the algorithm needed more than the 16GB RAM (+Swap) of our test machine. RG and RG+ required less than 6GB RAM for this dataset. But even with a memory complexity of $O(n + m)$ the memory limits of non-high-performance machines will be reached soon. However, telecommunication networks as well as networks on social networks sites can exceed hundreds of millions of vertices. Memory efficiency is a major challenge in processing huge datasets but so far received almost no attention in the modularity clustering literature.

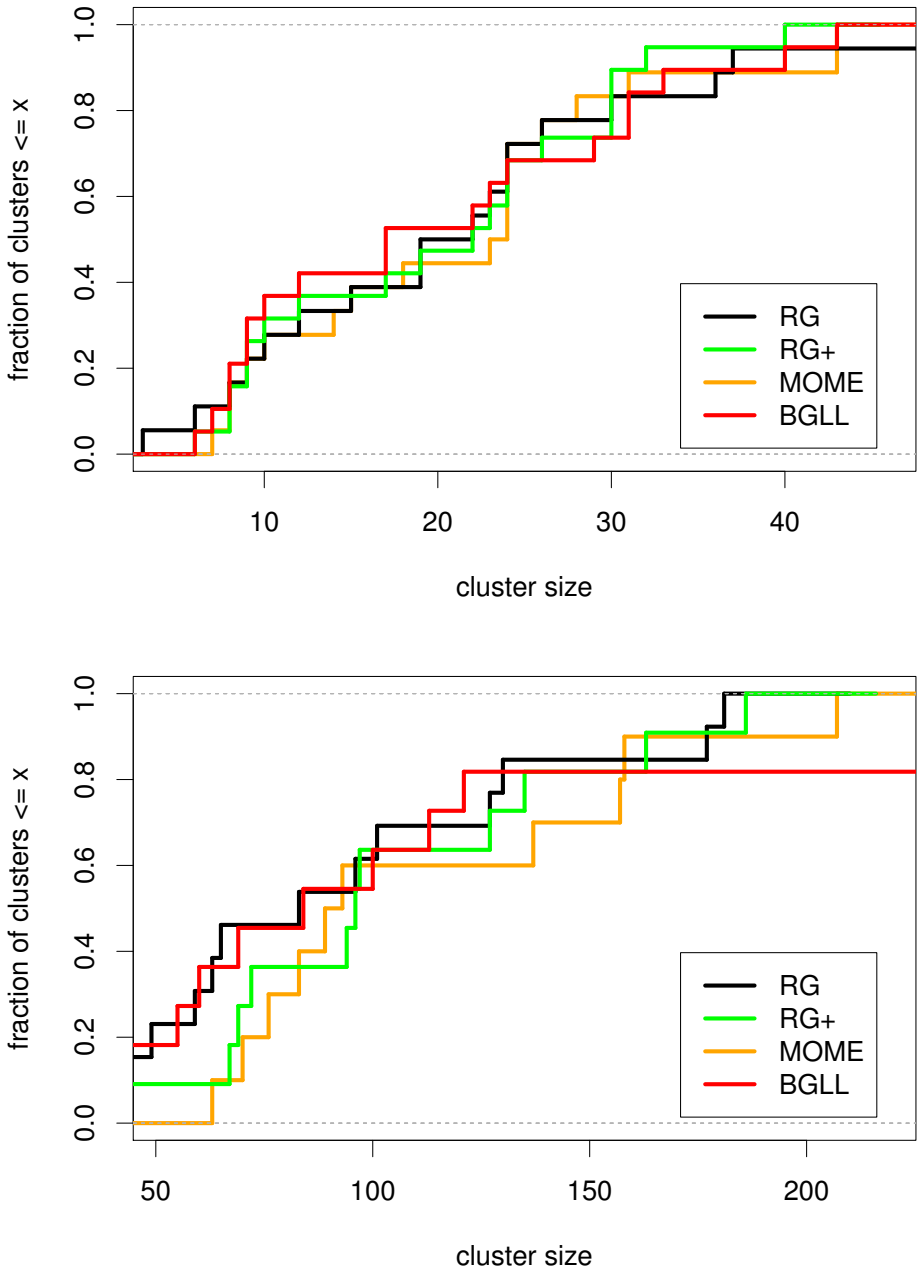

Figure 4.12: Cluster size distribution of test runs of RG, RG+, MOME and BGLL for the datasets Network Science (top) and Email (bottom).

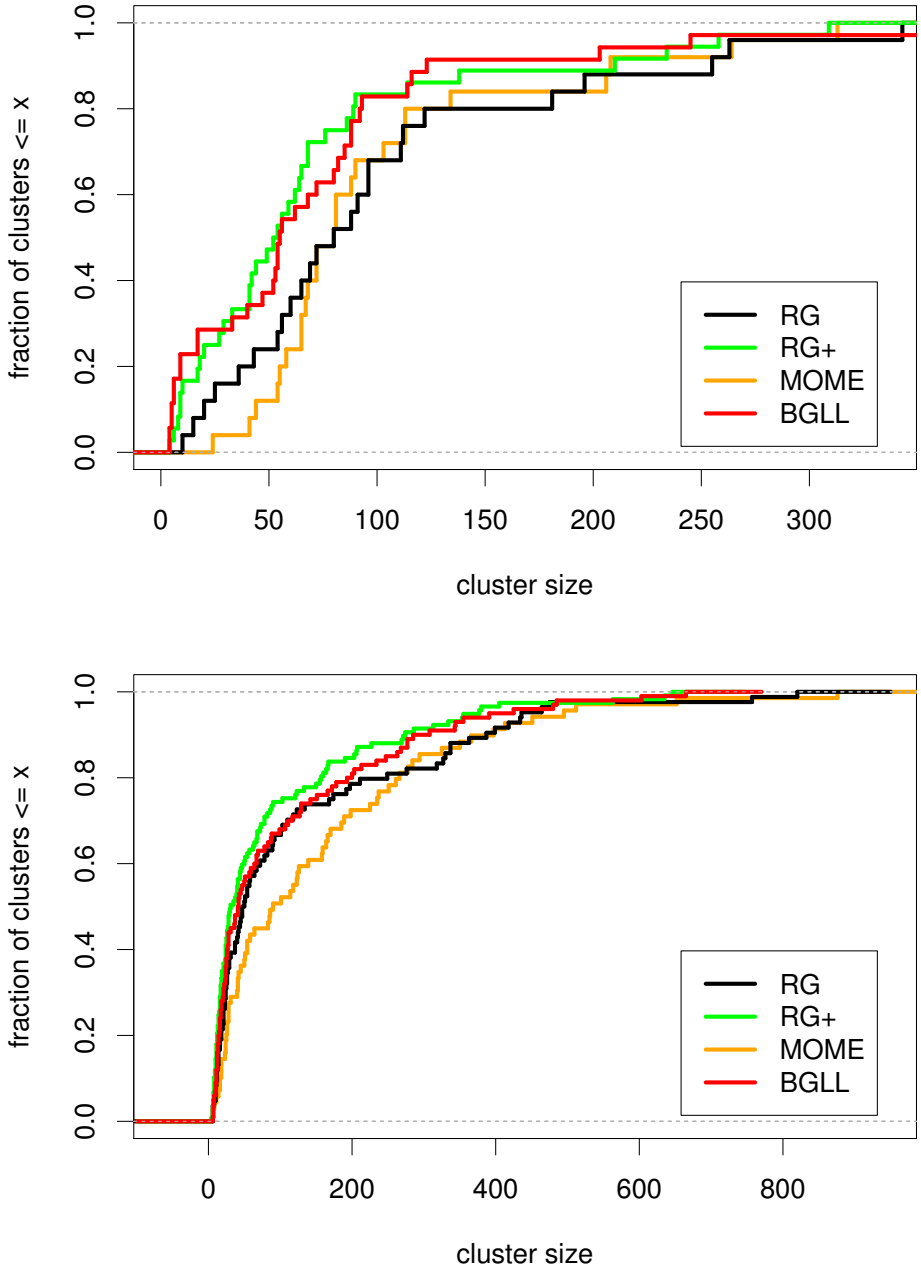

Figure 4.13: Cluster size distribution of test runs of RG, RG+, MOME and BGLL for the datasets Yeast (top) and PGP (bottom).

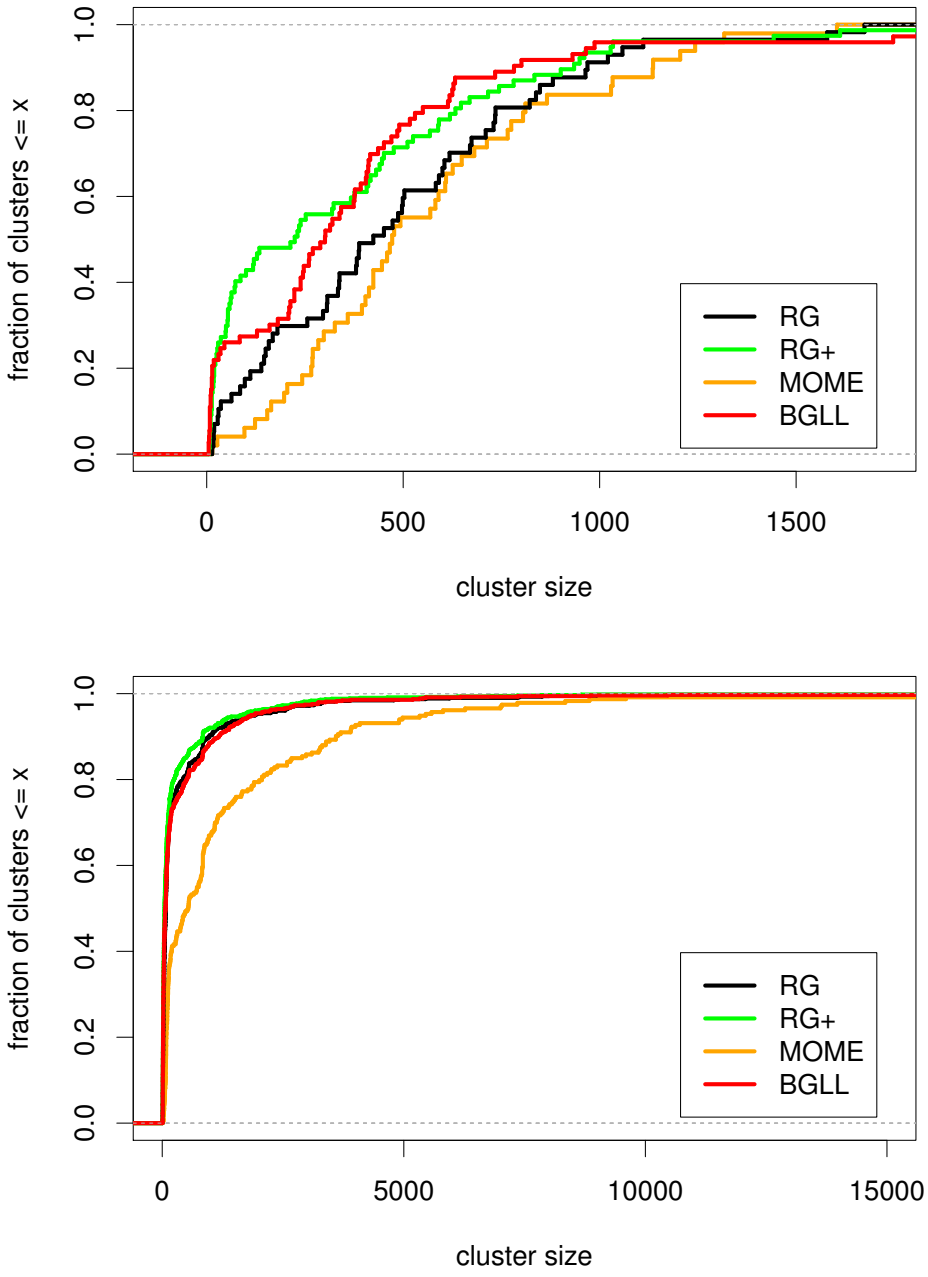

Figure 4.14: Cluster size distribution of test runs of RG, RG+, MOME and BGLL for the datasets Condensed Matter (top) and WWW (bottom).

5 Conclusion and Outlook

In this thesis two algorithms for modularity clustering have been presented. With the fast and scalable agglomerative hierarchical algorithm RG it has been shown that a less exhaustive greedy search can find superior results than a steepest gradient search. The analysis of the merge processes of different agglomerative hierarchical algorithms showed the problem of unbalanced growth that RG is able to avoid. A deeper analysis of the impact of unevenly contracted graph regions on the local measure ΔQ (denoted as the *locality problem*) and thereby on succeeding merge decisions provided the basis for the development of the advanced algorithm RG+. Guided by the objective to find a way to make better cluster assignment decisions for vertices that lie on the border of two natural clusters, the identification of core groups has been approached. In core groups vertices are collected that are very likely to belong to the same cluster. The core groups are expected to be a broadly flawless graph contraction that does not suffer from the locality problem. The succeeding merge process benefits from the initial contraction, as fewer wrong merges in the first contraction phase mislead the decisions in the later steps. The evaluation proves the advantages of RG+. This algorithm finds for many datasets partitions with higher modularity than any previously published algorithm we are aware of.

The RG+ algorithm consumes considerable more time than the RG algorithm as it requires several runs of RG to determine the core groups. However, future work could use two strategies to improve the runtime of RG+: First, the number of clusterings used to build core groups can be adjusted to receive a balance between runtime and modularity that better fits the needs of a specific application scenario. Second, the initial clusterings of RG can be computed in parallel on hardware with multiple processors or multi-core CPUs and sufficient memory. Then, RG+ requires only slightly more runtime than RG.

The work presented in this thesis provides a tool to identify 'natural groups' in very large networks. The main contributions are the analysis of previous algorithms and the derivation of new heuristic ideas for combinatorial optimization algorithms.

Whether the proposed algorithms are of any practical value has not been discussed as the assessment of cluster quality is still an open research problem if the assessment is not bounded to a specific use case. Future work has to show whether the identified communities have those properties required in a specific application context. Therefore, the contributions of this thesis are mainly directed to researchers and algorithm developers that require scalable concepts to solve combinatorial optimization problems.

However, research on clustering methods in general can benefit from the ideas presented in this thesis. The core group identification approach can be seen in an abstract way as a heuristic method to find those points in a search space where the paths to several local optima separate. These points are therefore promising points to (re-)start the search of optimization heuristics. Analyzing how the performance of other algorithms (e.g. genetic algorithms or simulated annealing) improves by starting from the cluster core solution is another task for future research.

Graph clustering will probably be a very active field of research for a long time as many challenges of practical relevance have not been solved, yet. An algorithmic challenge is to write memory efficient and distributed algorithms. A conceptual challenge is to find a way how to determine clustering quality, respectively, what clustering quality means in a specific domain.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.

[ABL10] Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466:761–764, 2010.

[ACC+10] Daniel Aloise, Sonia Cafieri, Gilles Caporossi, Pierre Hansen, Sylvain Perron, and Leo Liberti. Column generation algorithms for exact modularity maximization in networks. *Physical Review E*, 82(4):046112, 2010.

[ADFG07] Alexandre Arenas, Jordi Duch, Alberto Fernández, and Sergio Gómez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176, 2007.

[ADHP09] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 2009.

[AFG08] Alex Arenas, A Fernández, and Sergio Gómez. Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics*, 10(5):053039, 2008.

[AJB99] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[AK08] G. Agarwal and D. Kempe. Modularity-maximizing graph communities via mathematical programming. *European Journal of Physics B*, 66:409–418, 2008.

[BC09] Michael J. Barber and John W. Clark. Detecting network communities by propagating labels under constraints. *Physical Review E*, 80(2):026129, 2009.

[BDG+07] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On finding graph clusterings with maximum modularity. In *Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer, 2007.

[BDG+08] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[BH03] Gary Bader and Christopher Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, 2003.

[BJ93] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[BnPSDGA04] Marián Boguñá, Romualdo Pastor-Satorras, Albert Díaz-Guilera, and Alex Arenas. Models of social networks based on social distance attachment. *Physical Review E*, 70(5):056122, 2004.

[Bol98] Bela Bollobas. *Modern Graph Theory*. Springer, New York, 1998.

[Boo] Boost c++ libraries. `http://www.boost.org/`.

[BP99] Stefan Boettcher and Allon G. Percus. Extremal optimization: Methods derived from co-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 825–832, 1999.

[Bra08] L. Karl Branting. Overcoming resolution limits in MDL community detection. In *Proceedings of the 2nd Workshop on Social Network Mining and Analysis (SNA-KDD'08)*, 2008.

[CHL10] Sonia Cafieri, Pierre Hansen, and Leo Liberti. Loops and multiple edges in modularity maximization of networks. *Physical Review E*, 81(4):046102, 2010.

[CHL11] Sonia Cafieri, Pierre Hansen, and Leo Liberti. Locally optimal heuristic for modularity maximization of networks. *Physical Review E*, 83(5):056105, 2011.

[CLR09] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. The MIT Press, Cambridge, Massachusetts, 3 edition, 2009.

[CMN08] Aaron Clauset, C. Moore, and Mark E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[CNM04] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004.

[CPHL10] Yu-Teng Chang, D. Pantazis, H.B. Hui, and R.M. Leahy. Statistically optimal graph partition method based on modularity. In *2010 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1193 –1196, 2010.

[CS03] Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

[Cze09] J. Czekanowski. Zur Differentialdiagnose der Neandertalgruppe. *Korrespondenzblatt der Deutschen Gesellschaft für Anthropologie, Ethnologie und Urgeschichte*, 40:44–47, 1909.

[DA05] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72(2):027104, 2005.

[DFLJ07] Haifeng Du, Marcus W. Feldman, Shuzhuo Li, and Xiaoyi Jin. An algorithm for detecting community structure of social networks based on prior knowledge and modularity. *Complexity*, 12(3):53–60, 2007.

[DH73] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17:420–425, 1973.

[Dji08] Hristo N. Djidjev. A scalable multilevel algorithm for graph clustering and community structure detection. In *Algorithms and Models for the Web-Graph: Fourth International Workshop, WAW 2006*, pages 117–128. Springer-Verlag, Berlin / Heidelberg, 2008.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[FB07] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences of the United States of America*, 104(1):36 – 41, 2007.

[FD07] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[FGS09] Markus Franke and Andreas Geyer-Schulz. An update algorithm for restricted random walk clustering for dynamic data sets. *Advances in Data Analysis and Classification*, 3(1):63 – 92, 2009.

[FM83] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):pp. 553–569, 1983.

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[Fra07] Markus Franke. *An Update Algorithm for Restricted Random Walk Clusters*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, 2007.

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[Gae05] Marco Gaertler. Clustering. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 178–215. Springer Berlin / Heidelberg, 2005.

[GCDLR05] David Gfeller, Jean-Cédric Chappelier, and Paolo De Los Rios. Finding instabilities in the community structure of complex networks. *Physical Review E*, 72(5):056135, 2005.

[GD03] Pablo M. Gleiser and Leon Danon. Community structure in jazz. *Advances in Complex Systems*, 6(04):565–573, 2003.

[GDDG+03] R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6):065103, 2003.

[GGW07] Marco Gaertler, Robert Görke, and Dorothea Wagner. Significance-driven graph clustering. In *AAIM '07: Proceedings of the 3rd international conference on Algorithmic Aspects in Information and Management*, pages 11 – 26, 2007.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[GMSW10a] R. Görke, P. Maillard, C. Staudt, and D. Wagner. Modularity-driven clustering of dynamic graphs. Technical Report TR 2010-5, Universität Karlsruhe (TH), Informatik, 2010.

[GMSW10b] Robert Görke, Pascal Maillard, Christian Staudt, and Dorothea Wagner. Modularity-driven clustering of dynamic graphs. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 436–448. Springer Berlin / Heidelberg, 2010.

[GN02] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002.

[GS92] Andreas Geyer-Schulz. On learning in a fuzzy rule-based expert system. *Kybernetika*, 28:33–36, 1992.

[GSOS10] Andreas Geyer-Schulz, Michael Ovelgönne, and Andreas Sonnenbichler. Getting help in a crowd – a social emergency alert service. In *Proceedings of the International Conference on e-Business*, Athens, Greece, 2010. Institute for Systems and Technologies of Information, Control and Communication.

[HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985. 10.1007/BF01908075.

[Haj88] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[JHLB10] Di Jin, Dongxiao He, Dayou Liu, and C. Baquero. Genetic algorithm with local search for community mining in complex networks. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, pages 105 –112, 2010.

[JLYL09] Di Jin, Dayou Liu, Bo Yang, and Jie Liu. Fast complex network clustering algorithm using agents. In *DASC '09. Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 615 –619, 2009.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264 – 323, 1999.

[KA05] Nadav Kashtan and Uri Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):13773–13778, 2005.

[KCY+06] N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, T. Punna, J. M. Peregrín-Alvarez, M. Shales, X. Zhang, M. Davey, M. D. Robinson, A. Paccanaro, J. E. Bray, A. Sheung, B. Beattie, D. P. Richards, V. Canadien, A. Lalev, F. Mena, P. Wong, A. Starostine, M. M. Canete, J. Vlasblom, S. Wu, C. Orsi, S. R. Collins, S. Chandran, R. Haw, J. J. Rilstone, K. Gandi, N. J. Thompson, G. Musso, P. St Onge, S. Ghanny, M. H. Y. Lam, G. Butland, A. M. Altaf-Ul, S. Kanaya, A. Shilatifard, E. O'Shea, J. S. Weissman, C. J. Ingles, T. R. Hughes, J. Parkinson, M. Gerstein, S. J. Wodak, A. Emili, and J. F. Greenblatt. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature*, 440:637–643, 2006.

[KL70] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(1):291–307, 1970.

[KMN+02] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:881–892, 2002.

[Knu00] Donald E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, Reading, 3 edition, 2000.

[KSJ10] Youngdo Kim, Seung-Woo Son, and Hawoong Jeong. Finding communities in directed networks. *Physical Review E*, 81(1):016103, 2010.

[Kul27] S. Kulczynski. Die Pflanzenassoziationen der Pieninen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres*, pages 57–203, 1927.

[KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, 2004.

[LAWD04] Lun Li, David Alderson, Walter Willinger, and John Doyle. A first-principles approach to understanding the internet's router-level topology. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 3–14, New York, NY, USA, 2004. ACM.

[LF09a] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.

[LF09b] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5):056117, 2009.

[LFR08] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[LH07a] S. Lehmann and L. K. Hansen. Deterministic modularity optimization. *The European Physical Journal B - Condensed Matter and Complex Systems*, 60:83–88, 2007. 10.1140/epjb/e2007-00313-2.

[LH07b] S. Lehmann and L.K. Hansen. Deterministic modularity optimization. *The European Physical Journal B - Condensed Matter and Complex Systems*, 60(1):83–88, 2007.

[LH09] Zhipeng Lü and Wenqi Huang. Iterated tabu search for identifying community structure in complex networks. *Physical Review E*, 80(2):026130, 2009.

[Liv] Livejournal. `http://www.livejournal.com`.

[LL95] François-Joseph Lapointe and Pierre Legendre. Comparison tests for dendrograms: A comparative evaluation. *Journal of Classification*, 12:265–282, 1995.

[LLDM08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008. `http://arxiv.org/abs/0810.1355`.

[LLM10] Jure Leskovec, Kevin J. Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 631–640, New York, NY, USA, 2010. ACM.

[LLN10] Darong Lai, Hongtao Lu, and Christine Nardini. Enhanced modularity-based community detection by random walk network preprocessing. *Physical Review E*, 81(6):066118, 2010.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[LM10] X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A: Statistical Mechanics and its Applications*, 389(7):1493 – 1500, 2010.

[LN08] Elizabeth A. Leicht and Mark E. J. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11):118703, 2008.

[Mac67] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Sympos. Math. Statist. and Probability (Berkeley, Calif., 1965/66)*, pages Vol. I: Statistics, pp. 281–297. Univ. California Press, Berkeley, Calif., 1967.

[MAnD05] A. Medus, G. Acuña, and C.O. Dorso. Detection of community structures in networks via global optimization. *Physica A: Statistical Mechanics and its Applications*, 358(2-4):593 – 604, 2005.

[MD05] Claire P. Massen and Jonathan P. K. Doye. Identifying communities within energy landscapes. *Physical Review E*, 71(4):046101, 2005.

[Mei07] Marina Meilă. Comparing clusterings – an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.

[MHS+09] Juan Mei, Sheng He, Guiyang Shi, Zhengxiang Wang, and Weijiang Li. Revealing network communities through modularity maximization by a contraction–dilation method. *New Journal of Physics*, 11(4):043025, 2009.

[Mil67] Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[Mix] Mixi. `http://mixi.jp`.

[MLR06] Alfonsas Misevicius, Antanas Lenkevicius, and Dalius Rubliauskas. Iterated tabu search: an improvement to standard tabu search. *Information Technology and Control*, 35:187–197, 2006.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambrige, UK, 1995.

[MRC05] Stefanie Muff, Francesco Rao, and Amedeo Caflisch. Local modularity measure for network clusterizations. *Physical Review E*, 72(5):056107, 2005.

[MS57] Charles D. Michener and Robert R. Sokal. A quantitative approach to a problem in classification. *Evolution*, 11(2):130–162, 1957.

[New01] Mark E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences of the United States of America*, 98(2):404–409, 2001.

[New03] Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[New04] Mark E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.

[New06a] Mark E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006.

[New06b] Mark E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 103(23):8577–8582, 2006.

[New10] Mark E. J. Newman. *Networks: An Introduction*. Oxford University Press, Oxford, 2010.

[NG04] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004.

[NMCM09] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):P03024, 2009.

[NR08] Andreas Noack and Randolf Rotta. Multi-level algorithms for modularity clustering. *CoRR*, abs/0812.4073, 2008. `http://arxiv.org/abs/0812.4073`.

[NR09] Andreas Noack and Randolf Rotta. Multi-level algorithms for modularity clustering. In Jan Vahrenhold, editor, *Experimental Algorithms*, volume 5526 of *Lecture Notes in Computer Science*, pages 257–268. Springer Berlin / Heidelberg, 2009.

[NXDT10] Nam P. Nguyen, Ying Xuan, Thang N. Dinh, and My T. Thai. Adaptive algorithms for detecting community structurein dynamic social networks. Technical Report REP-2010-504, University of Florida, Department of Computer & Information Science & Engineering, 2010.

[OGS10] Michael Ovelgönne and Andreas Geyer-Schulz. Cluster cores and modularity maximization. In *ICDMW '10. IEEE International Conference on Data Mining Workshops*, pages 1204–1213, 2010.

[OGS11] Michael Ovelgönne and Andreas Geyer-Schulz. A comparison of agglomerative hierarchical algorithms for modularity clustering. In *Challenges at the Interface of Data Analysis, Computer Science, and Optimization*, Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg, 2011.

[OGSS10] Michael Ovelgönne, Andreas Geyer-Schulz, and Martin Stein. Randomized greedy modularity optimization for group detection in huge social networks. In *SNA-KDD '10: Proceedings of the 4th Workshop on Social Network Mining and Analysis*, 2010.

[OSGS10] Michael Ovelgönne, Andreas C. Sonnenbichler, and Andreas Geyer-Schulz. Social emergency alert service – a location-based privacy-aware personal safety service. In *Proceedings of the 2010 Fourth International Conference on Next Generation Mobile Applications, Services and Technologies*, Amman, Jordan, 2010.

[PBD06] Josep M. Pujol, Javier Béjar, and Jordi Delgado. Clustering algorithm for determining community structure in large networks. *Physical Review E*, 74(1):016107, 2006.

[PDFV05] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[POM09] Mason A Porter, Jukka-Pekka Onnela, and Peter J Mucha. Communities in networks. *Notices of the American Mathematical Society*, 56(9):1082–1097, 2009.

[RAK07] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[RB06] Jörg Reichardt and Stefan Bornholdt. When are networks truly modular? *Physica D: Nonlinear Phenomena*, 224(1-2):20 – 26, 2006.

[RB07] Martin Rosvall and Carl T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences of the United States of America*, 104(18):7327 – 7331, 2007.

[RCC+04] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.

[RSM+02] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási. Hierarchical Organization of Modularity in Metabolic Networks. *Science*, 297(5586):1551–1555, 2002.

[RZ07] Jianhua Ruan and Weixiong Zhang. An efficient spectral algorithm for network community discovery and its applications to biological and social networks. In *ICDM 2007, Seventh IEEE International Conference on Data Mining*, pages 643–648, 2007.

[RZ08] Jianhua Ruan and Weixiong Zhang. Identifying network communities with a high resolution. *Physical Review E*, 77:016104, 2008.

[SC08a] Philipp Schuetz and Amedeo Caflisch. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Physical Review E*, 77:046112, 2008.

[SC08b] Philipp Schuetz and Amedeo Caflisch. Multistep greedy algorithm identifies community structure in real-world and computer-generated networks. *Physical Review E*, 78(2):026112, 2008.

[SCCH09] Huawei Shen, Xueqi Cheng, Kai Cai, and Mao-Bin Hu. Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications*, 388(8):1706 – 1712, 2009.

[Sch07] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[SCY+10] Chuan Shi, Yanan Cai, Philip S. Yu, Zhenyu Yan, and Bin Wu. A comparison of objective functions in network community detection. In *Proceedings of the IEEE International Conference on Data Mining Workshops*, pages 1234–1241, 2010.

[SDJB09] Y. Sun, B. Danila, K. Josić, and K. E. Bassler. Improved community structure detection using a modified fine-tuning strategy. *EPL (Europhysics Letters)*, 86(2):28004, 2009.

[Sim62] Herbert A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962.

[Sme97] Neil J. Smelser. *Problematics of Sociology: The Georg Simmel Lectures, 1995*. University of California Press, Berkeley / Los Angeles, 1997.

[Sne57] P. H. A. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17(1):201–226, 1957.

[SPGMA07] Marta Sales-Pardo, Roger Guimerà, André A. Moreira, and Luís A. Nunes Amaral. Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences*, 104(39):15224–15229, 2007.

[SS63] Robert R. Sokal and Peter H. A. Sneath. *Principles of numerical Taxonomy*. Freeman and Company, San Francisco, 1963.

[THB07] M. Tasgin, A. Herdagdelen, and H. Bingol. Community Detection in Complex Networks Using Genetic Algorithms. *ArXiv e-prints*, 2007. arXiv:0711.0491v1, `http://arxiv.org/abs/0711.0491`.

[vL07] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[Wal83] David L. Wallace. A method for comparing two hierarchical clusterings: Comment. *Journal of the American Statistical Association*, 78(383):pp. 569–576, 1983.

[WF94] Stanley Wassermann and Katherine Faust. *Social Network Analysis, Methods and Applications*. Cambridge University Press, Cambridge, 1994.

[WS05] S. White and P. Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, pages 274–285. SIAM, 2005.

[WSWA08] Yang Wang, Huaiming Song, Weiping Wang, and Mingyuan An. A microscopic view on community detection in complex networks. In *PIKM '08: Proceeding of the 2nd PhD workshop on Information and knowledge management*, pages 57–64, New York, NY, USA, 2008. ACM.

[WT07] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks. In *IADIS International Conference WWW/Internet 2007*, pages 153–162, 2007.

[XCZ09] Biao Xiang, En-Hong Chen, and Tao Zhou. Finding community structure based on subgraph similarity. In Santo Fortunato, Giuseppe Mangioni, Ronaldo Menezes, and Vincenzo Nicosia, editors, *Complex Networks*, volume 207 of *Studies in Computational Intelligence*, pages 73–81. Springer Berlin / Heidelberg, 2009.

[XOX02] Ying Xu, Victor Olman, and Dong Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536–545, 2002.

[XTP07] G. Xu, S. Tsoka, and L. G. Papageorgiou. Finding community structures in complex networks using mixed integer optimisation. *The European Physical Journal B - Condensed Matter and Complex Systems*, 60:231–239, 2007.

[XW05] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[YHY08] Zhenqing Ye, Songnian Hu, and Jun Yu. Adaptive clustering algorithm for community detection in complex networks. *Physical Review E*, 78(4):046115, 2008.

[Zac77] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[ZWM+08] Zhemin Zhu, Chen Wang, Li Ma, Yue Pan, and Zhiming Ding. Scalable community discovery of large networks. In *WAIM '08: Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management*, pages 381–388, 2008.

[ZWW+09] X. S. Zhang, R. S. Wang, Y. Wang, J. Wang, Y. Qiu, L. Wang, and L. Chen. Modularity optimization in community detection of complex networks. *EPL (Europhysics Letters)*, 87(3):38002, 2009.