Technische Universität
Kaiserslautern

Fernstudium Software Engineering for
Embedded Systems

Masterarbeit

Community Identification in International Weblogs

Autor: Christoph Rueger

Betreuer:
Prof. Dr. Prof. h.c. Andreas Dengel
Dipl.-Inf. Darko Obradovic

December 23, 2010

Ich versichere, dass ich diese Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum

Unterschrift

Abstract

In this thesis, we evaluate and categorize algorithms and methods to identify communities within graphs and networks purely based on structural properties. The goal is to describe the theory of the problems and solutions related to the identification of cohesive groups within graphs and networks. We apply the discussed algorithms to a random generated graph and a dataset of of weblogs in six different languages (German, English, French, Italian, Portuguese and Spanish) and we will analyze and interpret the results.

Based on the results which are gained by pure structural analysis of the network we try to draw conclusions from analysis of the textual content of websites within a single group, to reason about the origin of the network structure. We also try to evaluate if this can be used as some kind of quality criterion for the found groups in addition to existing quality metrics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of the topics that are the subjects of this thesis. It also presents the motivation and the ideas driving the research scope of this work as well as the goals and expectations for the results.

1.1 Background

Weblogs (or Blogs) are very popular publishing media in the Web 2.0[1]. They are used by personal authors or companies as personal diaries, as news journals, as corporate communication channels, etc. worldwide. The basis of blogs are the articles which are published as HTML[2] web pages. In these articles, blog-authors can place links to other websites and weblogs while their blogs and articles can also be linked from other blogs. This creates a network of blogs with links between each other. The reason for placing a link to other blogs are different: Blog authors can be friends and know each other, or they cite another blog, or they just recommend each other because of similar topics. It could also be that the blog platform provider automatically places links between blogs e.g. for marketing purposes, to increase traffic on their blogs. The structure of this link network of blogs is the basis for this thesis.

1.2 Motivation

We have data sets of popular blogs from six different languages, which have been already analyzed with respect to popularity [37]. In this thesis, we will analyze these data with respect to communities (sub-groups) using So-

[1] http://oreilly.com/web2/archive/what-is-web-20.html
[2] http://www.w3.org/MarkUp/

cial Network Analysis (SNA). This includes research, implementation and comparison of different community identification algorithms, assessment of the results and crosschecking the different languages, searching for cultural differences.

1.3 Goals

The goal of this thesis is to give an overview of the current state of available algorithms in order to identify communities within a web-graph of weblogs based on structural properties of the network.

Goal 1: Define the basic terms and theoretical background about the field of community identification.

Goal 2: Categorize, analyze and describe a selection of available methods and algorithms for community identification.

Goal 3: Apply some of those algorithms to available datasets which consist of a generated random graph with pre-defined community structure (see Section 4.8), the complete web-graph of the weblogs in six different languages, and the single language subgraphs and evaluate the results.

Goal 4: Define metrics to measure quality of identified communities.

Goal 5: Based on the results, gained by pure analysis of the link structure between weblogs, we try to prove our hypothesis that there exists a connection between the identified structural communities and the content of the weblogs inside a community. We claim that the textual content and the topics of weblogs are directly related to the communities and with this thesis. We try to prove that and make it visible.

Basically we try to answer the following questions:

Question 1: Can the available algorithms for community identification find communities based on the link-structure of weblogs and the datasets available to us?

Question 2: Can we measure and visualize that?

Question 3: Will that work in practice and does it scale for large real world datasets?

Question 4: Are structural communities and the content of the contained weblogs directly related to each other?

Chapter 2

Theoretical Basics

This chapter provides an overview of the topics that are the subjects of this thesis. It also presents the motivation and the ideas driving the research scope of this work as well as the goals and expectations for the results.

2.1 Community Identification

Many areas of our daily life like society, politics, biology, physics, computer science and economics have formed terms like friends, friendships, villages, towns, political parties and most recently the world wide web where we find social networks like Facebook[1], or forums or web-rings. All these terms refer to some kind of cohesive subgroup within a larger group of data points, where the data points can be persons, biological cells, atoms, computers in a computer network or web-pages in the world wide web.

The points and their connections can be modeled as a *graph*, which contains *vertices* and *edges*. Vertices represent the the points and the edges represent the connections between the points. The edges can be *directed* or *undirected* and can also be *weighted* or *unweighted*. In our example data set the weblogs represent the vertices and the links from a weblog to another weblog represents the edges, which make up a directed graph. The edge-weight could represent e.g. a cost or a distance between two vertices. Figure 2.1 shows an example of a directed, unweighted graph.

You can see that the term community is not clearly defined [14, p 8] and in literature you find terms such as *clusters, groups, subgraphs, subgroups, sub networks and partitions*, which are often used as a synonym. In this thesis we try to define a community as a cohesive sub-group within a network. This can be a group of vertices sharing some common properties or play

[1] http://facebook.com

similar roles within the graph [14, p 2]. Furthermore, we use the measure of modularity and conductance to define the term *community*, which means that a community has more edges within the community than edges to their outside world. In this thesis we will use the terms *community and cluster* to refer to the same thing, but we clearly distinguish between the latter end the term *partition* as described in Section 2.4.5.

Methods of graph theory can now be applied to identify these clusters / communities within graphs.

The reasons to identify communities vary between different areas of interest. For a telecom provider e.g. it can be useful to identify certain groups of users to target marketing campaigns at based on their calling patterns [21, p 9]. In biology it can be useful to "group proteins which have the same specific function within the cell" [14, p 2]. And in our case of weblogs we could determine communities which represent related websites which could be useful e.g. to target specific banner campaigns to websites within a certain community. It is also possible to derive sociological conclusions from the communities of different countries for example to visualize the importance of different topics like *cooking*, *music* or *politics* by identifying clusters of blogs writing about such topics and to compare their properties (e.g. size, density of links etc.). It is also possible to derive current *trending* topics from identifying clusters over a period of time and how the clusters change over time. For example in times of an election it is very likely that many blogs are writing about political topics and also link to each other, and identifying those clusters is a way to visualize that *politics* is a *trending* topic.

After this background information about community identification, the next Section will give an overview about the basic terms related to graph theory to form the basis of the later chapters.

2.2 Graph Basics

This section will give a short overview over basic definitions and terms which are relevant when characterizing graphs and networks. Those terms and definitions will be used throughout this thesis, thus they are explained here first.

Graph A graph is a mathematical representation of a network. It consists of a set of n vertices V and a set of m edges E where n and m represent the numbers of nodes and edges respectively. Nodes are connected by edges.

Vertex A vertex is a single point in the graph, also called node.

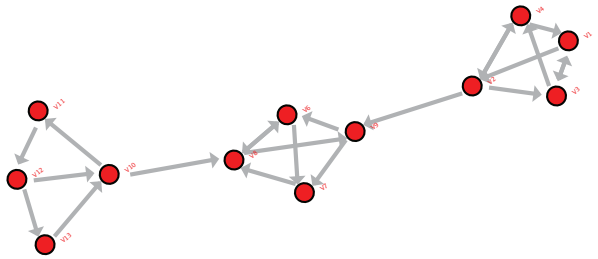


Figure 2.1: Example of a graph with 3 communities

Edge An edge is a connection between two vertices, which can be undirected, directed, weighted and unweighted. Directed means the connection between two vertices only exists in one way (like a one-way street, from A to B, but not B to A). Undirected means that the direction does not matter. The weight e.g. could represent a distance. In unweighted graphs the weight of every edge is 1.

Neighbor A neighbor is a vertex which is adjacent to another vertex thus they are connected by an edge. A vertex can have multiple neighbors.

Adjacency Matrix A graph can be represented with the help of an adjacency matrix. This is a $n \cdot n$ matrix containing all vertex-pairs of the graph where the value equals 1 if both vertices are connected by an edge and 0 if the two vertices are not connected by an edge.

Random Graph A random graph is a graph which was artificially generated by an algorithm.

Walk A walk is a sequence of edges in the graph (e.g. A to B to C to A to B to D). The edges do not need to be distinct. The length of a walk is the number of edges of the walk.

Random Walk A random walk is a set of vertices which is selected that you start on a vertex A, then select a neighbor of this vertex at random (B), then select a vertex out of the neighbors of B (C) and so on. The resulting sequence of edges is called a Random Walk.

Path A path is a sequence of edges in the graph where each line is distinct / unique (e.g. A to B to C to D). This is a stricter definition than a walk. The length of a path is the number of edges of the walk.

Shortest Path The shortest path between two vertices of a graph is the path with the least number of edges inside an unweighted graph where as it is the lowest sum of edge-weights inside a weighted graph.

Cycle A cycle is a path that returns to its own starting point, and can be of any length (like a path).

k-Cycles A k-cycle is a cycle where k defines the length of the cycle (the number of edges which make up the cycle).

Bridge A bridge is a line which does **not** lie on a cycle but connects two or more cycles.

Diameter The diameter of a graph "is the longest shortest-path between any two vertices in the graph" [41, p 95].

Density Density is the number of edges in a graph divided by the maximum possible number of edges (max. possible number of edges $= n \cdot (n-1)$ for undirected graphs and $\frac{n \cdot (n-1)}{2}$). This metric is only useful to compare graphs of the same size [41, p 71].

Inclusiveness Inclusiveness "is the number of connected points in a graph (all nodes minus the isolated nodes) expressed as a proportion to the total number of points" [41, p 70]. It is a metric for how well connected the graph is. Example: If the graph contains 20-points and 5 isolated vertices (points with no connection to other points), then the inclusiveness is 0.75.

2.3 Centrality

This metric tells how well a vertex or an edge *connects* the network. Degree, Closeness and Betweenness are all measures of centrality.

Degree The degree of a vertex is the number of directly connected neighbor-vertices. For example in an undirected graph, if a vertex has 3 directly connected neighbors, then it has a degree of 3. In directed graphs one distinguishes between in-degree and out-degree. E.g a vertex has an in-degree of 3, if there are 3 incoming edges from 3 direct neighbors. It has an out-degree of 3 if there are 3 outgoing edges from the vertex to its neighbors. The degree distribution of a whole graph can be visualized by a histogram, which shows the degree from 0 - n on the x-axis, and the number of vertices having that degree on the y-axis. This is a useful tool for initial inspection of graph properties. For example a typical real-world web-graph appears to have a so called power-law degree distribution with many vertices having a low degree and few vertices with very high degree. An example of a histogram showing a power-law degree distribution can be found in Figure 4.5.

Closeness / Global Point Centrality Closeness is the degree a particular vertex is near all other individual vertices. The vertex with the minimum total distance to all other vertices has the highest closeness. It can be determined by calculating the sum of the vertex's distance to all other vertices. The distance here is the *geodesic* (shortest path). The smaller this value the better / more important, because the vertex is closer to everybody than any other vertex. This can also be a set of vertices with the same closeness value. For directed graphs there is also the *in-centrality / in-closeness* and the *out-centrality / out-closeness*. In other words: *Closeness* is the inverse of the sum of the shortest distances between each individual vertex and every other vertex in the network [41, p 83].

Betweenness Betweenness is the "extend to which a particular vertex or edge lies *between* the various other points in the graph" [41, p 86]. We distinguish between three different kinds of vertex and edge betweenness. A vertex or edge with a high betweenness value can be considered as *important*. For example a single road between two cities has a very high betweenness value as all traffic has to pass through it.

- **Geodesic betweenness** is the betweenness calculated based on the number of all-vertex-pair-shortest-paths running through the vertex or edge under observation [31, p 2].

- **Random-walk betweenness** is the betweenness calculated based on the number of times an all-vertex-pair-random-walker passes through the vertex or edge under observation.
- **Current-flow betweenness** "is defined by considering the graph a resistor network, with edges having unit resistance. If a voltage difference is applied between any two vertices, each edge carries some amount of current, that can be calculated by solving Kirchoff's equations. The procedure is repeated for all possible vertex pairs: the current-flow betweenness of an edge is the average value of the current carried by the edge." [14, p 24]

The basic idea behind the Random-walk and Current-flow approach compared to the Geodesic betweenness is that in real-world networks we do not know if information would always choose the shortest-path, which would imply that the information has the knowledge about the shortest optimal path [31, p 3]. Figure 2.2 shows an example of a graph with geodesic betweenness values on the edges.

2.4 Cohesive Group Concepts

In the area of network analysis and graph theory, there are various concepts of cohesive groups of vertices, for example components, cores, cliques, clusters and partitions. The following definitions should help to understand the terms and their relation to community identification although some of the terms especially related to cliques and plexes are not used later in this thesis, as we are more focused on clusters and communities as described in Section 2.4.4.

2.4.1 Components

A component is the simplest of the various sub-graph concepts, which is the "maximal connected sub-graph". All of the points of the sub-graph "are linked to another through path" [41, p 101]. All points can "reach one another through one or more path" [41, p 101] and there are no path running to points outside the component. There are several types of components, which we will describe here.

Strong Component A strong component is a sub-graph where all the edges "that make up the paths are aligned in a continuos chain without any change of direction" [41, p 103].

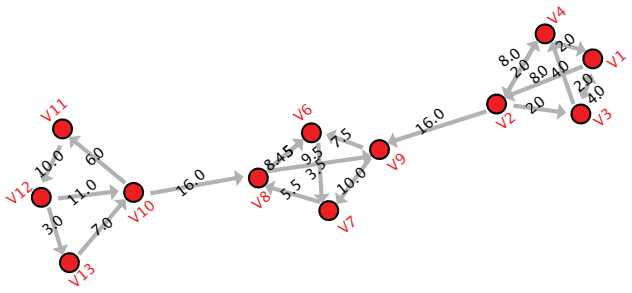


Figure 2.2: Graph with edge-betweenness values

Weak Component A weak component is a sub-graph where the direction of the edges is disregarded and "simply the presence or absence of a connection is taken into account" [41, p 104].

Cyclic Component A cyclic component "is a set of intersection cycles, connected by those edges or points that they have in common" [41, p 105].

2.4.2 Cores

Core A core is a maximal sub-graph where every vertex is connected to every vertex in the sub-graph.

k-Core A k-core is a sub-graph based around the degree of vertices. It is a maximal sub-graph where every vertex is connected to at least k other vertices. All vertices have a minimum degree of k.

m-Core A m-core is based around the multiplicity of edges. It is a maximal sub-graph in which each line has a minimum multiplicity of m [41, p 113]

Core Collapse Sequence The Core collapse sequence is the number of vertices, which will "disappear" in k+1 (when k is increased by 1). This is also given as a proportion to the total number of vertices, e.g. in a *6*-vertex sub-graph, where *2* vertices would "disappear" when we increase k=2 to k=3, the core collapse sequence would be $2/6 = 0.333$ [41, p 111]. The same can also be applied to m-cores where m is increased as *edges* will be *removed* and the points, which were connected through those edges "disappear" [41, p 113].

2.4.3 Cliques and Plexes

A clique "is a sub-set of points in which every possible pair of points is directly connected by a line and the clique is not contained in any other clique" [41, p 114].

Diameter The diameter of a clique is the path distance between its most distant members.

Strong Clique A strong clique in a directed graph is a sub-set of points where all edges between the points are reciprocated edges (edges, which have an arrow at both sides).

Weak Clique In a weak clique all edges are treated as if they were reciprocated.

n-Clique A n-clique is a maximal and complete sub-graph in which all pairs of points are directly connected at maximum distance n. For example a 2-clique is one in which members are connected either directly (at distance 1) or indirectly at distance 2. n-cliques can be detected by multiplying the adjacency matrix by itself or using a more efficient back-tracking algorithm.

n-Clan A n-clan is like the n-clique, but more restrictive. It requires that the maximum path distance is n but also the diameter of the clique is no greater than n.

k-Plex A k-plex is a sub-graph where every vertex is connected to at least n - k other vertices. All vertices have a minimum degree of k. e.g. in a 6 vertices graph, a 3-plex would be sub-graph where every vertex is connected with at least 3 other vertices [41, p 118].

Circle A set of overlapping cliques can be aggregated into *circles* if they have more than a certain proportion of their members in common [41, p 119].

2.4.4 Clusters

A cluster is different than a clique. It is "an area of relatively high density in a graph" [41, pp 126-127]. In other words, a cluster is a group of vertices, which have more edges to other vertices within the cluster than to vertices in other clusters. *Clustering* is another term, which refers to a set of multiple clusters. The result of an algorithm is usually a set of multiple clusters, which then will be referred to as a *clustering*.

2.4.5 Partitions

A partition is a set of nodes of a graph and a graph can be partitioned into many partitions. Partitions are characterized by the fact that they are *disjoint* (each vertex can only be part of a single partition) and optimally have *equal size*. The terms *cluster*, *community* and *partition* are often used interchangeably and often cause confusion. In this thesis we clearly differentiate between *partitions* and the other two terms *cluster* and *community*. We refer to partitions only in the context of partitioning of e.g. a network of PCs, which should be grouped into partitions so that the traffic between the different partitions is minimized. In contrast to that, when we are talking about a community or cluster, we do not have the goal to partition the graph into disjoint groups of equal size, but we are looking for *good* communities within the graph. A *good* community can even be a small group of vertices, with the properties of clusters as described earlier. We can possibly identify multiple communities inside a graph but still have lots of vertices, which are not part of any identified community.

2.5 Cluster Quality Metrics

Once identified, we need some measures to judge the quality of the identified communities. Some of the metrics described here will be used in the chapter 4 to evaluate the communities we have identified when we applied some of the algorithms of chapter 3.2 to our random and real-world datasets.

Modularity A very important measure for measuring community-quality is modularity. It measures the number of community-internal edges

relative to a so called *null-model*, which can be a random graph with the same degree distribution [28, p. 15] or in other words a sub-graph is a community if the number of internal edges is greater than the expected number of edges that the same sub-graph had in a *null-model* [14, p 12]. Modularity can be considered *better* the higher this value is as this means there are more edges than expected inside the cluster and more edges means higher density, which is a property of a good cluster according to our definition of clusters in Section 2.4.4. Modularity is used throughout this whole thesis, and it is also a central part of many algorithms for community identification. The basic definition of modularity assumes an undirected graph. But as we are dealing with directed web-graphs in this thesis (also abbreviated as Mod.), we have used a modularity definition for directed graphs by [4, 27] found in [14, p 34 Equation 37], which is shown in Equation 2.1.

$$Q_d = \frac{1}{m} \sum_{ij} A_{ij} - \left(\frac{{k_i}^{out} {k_j}^{in}}{m} \right) \delta(C_i, C_j) \tag{2.1}$$

In Equation 2.1 Q_d is the modularity for a *directed* graph, m is the total number of edges in the whole graph. A_{ij} is an entry from the graph's adjacency matrix (1 if the vertex pair is connected by an edge, 0 if not connected), ${k_i}^{out}$ is the out-degree of vertex i, ${k_j}^{in}$ is the in-degree of vertex j and $\delta(C_i, C_j)$ has the value 1 if vertices i and j are inside the same community, otherwise it has the value 0.

Conductance The conductance c shown in Equation 2.2 of a community is the number of cut edges (c_c) if you cut the community out the whole graph divided by the number of edges inside the community (e_c) [28, p. 3]. The community can be considered *better* the smaller this value is, which means the community has very few edges to the rest of the graph and many internal edges. Conductance also plays a role in the Network Community Profile Plot (NCP-Plot) (see [28, p. 17] and Section 4.2) and we will abbreviate in tables with "Cond.".

$$c = \frac{c_c}{e_c} \tag{2.2}$$

Fraction of correctly classified vertices "A vertex is correctly classified if it is in the same cluster with at least half of its *natural partners*" [14, p 77]. We will use the short-form $\#\ CCV$ in this thesis.

Rand Index The Rand Index r "is the ratio of the number of vertex pairs correctly classified in both partitions (i.e. either in the same or in different clusters), by the total number of pairs." [14, p 78] This metric is useful for comparing the results of two different algorithms e.g. the clusters identified by the Algorithm by Girvan and Newman 3.3.2 with the clusters identified by the algorithm by Blondel et. al 3.3.4. The Rand Index is a value between 0 and 1, where 0 means that the clusterings do not match at all and a value of 1 means than the resulting clusters of both algorithms are equal. A similar metric is the Jaccard index [14, p 78].

$$r = \frac{a_{11} + a_{00}}{a_{11} + a_{01} + a_{10} + a_{00}} \tag{2.3}$$

Equation 2.3 shows the calculation of the Rand Index R where a_{11} is the number of pairs of nodes that are in the same cluster in algorithm-1 and in the same cluster in algorithm-2. a_{00} is the number of pairs of nodes that are in different clusters in algorithm-1 and in different clusters in algorithm-2. a_{01} is the number of pairs of nodes that are in different clusters in algorithm-1 and in the same clusters in algorithm-2. a_{10} is the opposite of the latter, the number of pairs of nodes that are in the same cluster in algorithm-1 but in different clusters in algorithm-2 [45] and [14, p 78].

Chapter 3

Cluster Algorithmic Theory

3.1 Algorithmic Families

In this section we want to give an overview over several families of algorithms. Some of the algorithms presented here will be explained in more detail in the following chapter. For a much more detailed and fine grained classification of algorithms we recommend the paper by Santo Fortunato [14].

3.1.1 Agglomerative Clustering

The family of hierarchical clustering methods is divided into *agglomerative* and *divisive* clustering methods. Agglomerative methods can be seen as bottom-up approaches starting from a single vertex, each representing a single cluster on their own, which are then iteratively aggregated with other vertices into larger and larger clusters [41, pp 126-130]. The points are compared for their *similarity* or *distance*. Most of the agglomerative clustering algorithms differ in the different definitions of similarity. An example of similarity is in interlocking directorships: Enterprises might be merged into a cluster on the basis of the number of directors they have in common.

The first step of every agglomerative algorithm is to compute a $n \cdot n$ matrix X that is called the *similarity matrix*, containing the similarities of all vertex pairs. Based on that similarity matrix the algorithm iteratively merges points into clusters until some kind of stopping criterion is met e.g. when a given number of clusters has been identified or a quality function like modularity is optimized [14, p 19]. The results can be visualized in a dendrogram[1].

[1] http://en.wikipedia.org/wiki/Dendrogram

3.1.2 Divisive Clustering

Divisive clustering methods also belong to the family of hierarchical algorithms. They are a top-down approach and in contrast to *agglomerative clustering* we start with the graph as a whole representing a single cluster, which will be split into smaller sub-sets by removing edges connecting vertices with low similarity (vertices of different clusters). The results are also hierarchical, can be represented by a dendrogram, and also have the same disadvantage as other hierarchical methods, which is to determine the hierarchy level that represents the community structure of the graph. As we can see later, the algorithm of *Girvan and Newman* provides a solution to this problem and will be described in Section 3.3.2.

3.1.3 Partitioning Methods

The purpose of graph partitioning is to divide the graph into groups of vertices of pre-defined size with the goal that the cut-size (the number of edges between the groups) is minimal, which is in most cases an NP-hard problem [14, p 17]. In practice this often is a problem in parallel computing where the number of interaction between the separate CPUs is tried to be minimized. The partitioning is often achieved by bisecting the graph into two parts. To separate the graph into more than two partitions this process is repeated recursively by bisecting each of the two parts again and so on.

There are various graph and data partitioning algorithms like K-Means, k-clustering, Minimum K-Clustering, k-center and k-median, Spectral Bi-Section [14], Kernighan-Lin algorithm [25], and all these algorithms require to pre-define the number of clusters k. For the purpose of community identification k is not known upfront so graph partitioning methods are not a focus of this thesis although we will explore Spectral Bi-Section in Section 3.3.3 and METIS in Section 3.3.5 for recursively bisecting the graph to identify communities.

3.1.4 Spectral Methods

Most methods of spectral clustering were "developed in computer science and generally focus on data clustering" but can be used for graph clustering as well [14, pp 41-42]. Spectral clustering is based around the idea of relations between objects and their similarities and makes heavy use of eigenvectors of the Laplacian matrix [14, pp 41-42]. The goal is to find groups of *similar* objects . Objects inside the same group are similar and objects in different groups are dissimilar [43, p 2]. It is well known that an ideal partition of a

graph can be found by solving the minimum cut problem (min-cut) or the normalized cut (NCut) problem, which is a partition which passes through those edges of the graph whose weights sum to a minimum [12, p 2]. The problem is that finding the min-cut or NCut is a NP-hard problem and spectral clustering is an approximation solution with much lower complexity while still providing good results.

The main idea behind this approximation is to find the eigenvectors and eigenvalues of the graph Laplacian. Clustering information can then be obtained from the eigenvectors. More details about this process can be found in Section 3.3.3.

3.1.5 Multilevel Algorithms

Multilevel Graph partitioning algorithms like METIS are a new class of algorithms with moderate complexity and excellent graph partitions, which are also faster than e.g. spectral bi-section [23, p 1]. Although this algorithm is a graph-partitioning algorithm and we said in Section 3.1.3 we will not evaluate those algorithms, we will present the METIS algorithm here, because we use it in combination with a recursive bi-section approach, which is able to identify communities of varying size without pre-defining k. What all of the multilevel algorithms have in common are the 3 phases *Coarsening Phase*, *Partitioning Phase* and *Uncoarsening Phase*. The idea is to first coarsen a large graph down to a graph with only a small subset of nodes (e.g. down to a few hundred nodes) by combining vertices into multi-nodes, because it is much easier to partition a small graph. Then for this small graph a bi-section is calculated. For multiple partitions the bi-section is recursively repeated. After the partitions are created the partitions need to be projected back to the original phase in the *Uncoarsening Phase*. In this thesis we have used the METIS algorithm, which we will describe in Section 3.3.5.

3.2 Analysis of Algorithms

This chapter will first describe how we will analyze the algorithms and what properties we will examine. We will distinguish between algorithmic properties and properties of the results. After this introductory part we will start describing the algorithms based on the properties we have described. Some of the algorithms will be implemented or applied to our example dataset and we will analyze and interpret the results. We will try to answer the question whether the properties of the algorithms found in literature also apply for the example dataset.

3.2.1 Algorithmic Properties

Algorithmic properties are all properties of the algorithm before and while it produces its results.

Type of graphs The different algorithms under analysis do not work on all types of graphs. For example, there are algorithms that only work on undirected graphs, but do not work on directed graphs. Some algorithms take the edge-weight into account while others do not take the weight into account and lead to different results on a weighted graph. Furthermore, some algorithms have the property that they perform very well on large graphs but only if the graphs are sparse and they would fail or have very bad performance characteristics on very dense graphs. For each algorithm we will test in this thesis we try to account for this property.

Pre-defined k or dynamic k Some algorithms need the number of clusters or groups to find as an input parameter k. For example the well known k-Means algorithm needs the number of partitions to be identified in advance in order to work. Those kinds of algorithms are not well suited for our purpose of community identification because the normal case is that the number of communities is not known in advance. So in this thesis we will focus more on algorithms, which determine the number of communities automatically. Algorithms with pre-defined k are generally suited better for graph partitioning, a use-case where k is known upfront.

Basic idea of the method Every algorithm is usually based on a single or multiple basic idea, e.g. some algorithms make heavy use of eigenvector calculations while other algorithms are based on the idea of random processes like random walks. We will try to examine each algorithm and find out whether or not we can identify such basic idea. We will then describe the basic steps of each algorithm and how it works in a more or less high level manner and we will refer to the accordant literature for details.

Order of computational and memory complexity The computational complexity and memory complexity of algorithms is usually given for the worst-case in the Big-O notation. But the algorithms have different complexity based on the properties of the graph e.g. complexity can be much lower if the graph is very sparse (less dense) while complexity is near worst-case for very dense graphs. We will try to highlight under which circumstances the algorithms will perform better or not.

Limitations this is basically a summary of all the other points above, where we try to summarize where the limitations of different algorithms are and where their use is limited.

Related algorithms Many algorithms are not the first ones of their kind but are based on the idea of another algorithm. If that is the case we will mention the related algorithms and also try to mention extensions of the algorithm.

3.2.2 Properties of the Results

After having described the algorithmic properties we will describe the properties of the results. We will continue to use the term cluster for those kinds of cohesive subgraphs.

disjoint or overlapping clusters Most algorithms will produce disjoint clusters, which means that every vertex is part of exactly one group. Other algorithms also produce overlapping clusters where a vertex can be part of one or more cluster. Algorithms producing overlapping clusters do exist but there are problems with the visualization of the results because e.g. a dendrogram is not able to display overlapping clusters.

hierarchical or non-hierarchical While some algorithms will segregate the graph into disjoint partitions where every vertex is part of a single partition, it is also possible that the resulting clusters are nested in a kind of hierarchy. One can imagine this from real world groups where e.g. a group of close friends are all members of the same class in the courses, while all courses are held at the same university. So in this scenario all students of the university form one community. This community also has sub-communities, the courses and the close friends. Visually this can be imagined like "zooming into a community" [14, p 90]. states that this can also be a problem when interpreting the results, because it is hard to find the relevant levels in the hierarchy where the identified communities "make sense". The following quote by [14, p 90] describes that well: "If we consider the human body, we cannot say that the organization in tissues of the cells is more important than the organization in organs."

size of the clusters The clusters identified by the algorithms can have equal size or varying size.

3.3 Algorithms

3.3.1 Agglomerative Linkage Methods

As described in Section 3.1.1 the agglomerative methods start at the single vertex, each vertex representing a single cluster, and iteratively merge vertices together into larger cluster. Traditionally there are three basic approaches, Single Linkage, Complete Linkage and Average Linkage as described next.

Method

In *Single Linkage Clustering* points are merged into a cluster with their nearest neighbors. Initially the two closest points are merged into a cluster and later steps merge successively more distant points into clusters. The Single Linkage method often leads to points being merged into existing clusters and "it is less likely that one will find as homogenous and compact clusters as it would have been possible with *Complete Linkage* method" [41, p 129], which will be described next.

The general approach of the *Complete Linkage* is the same as for *Single Linkage*, but it measures the similarity of the two clusters by their most distant members as opposed to the the closest members, thus it is more likely that it initializes new clusters at early stages in the analysis and it finds very homogenous and compact clusters [41, p 130].

In contrast to *Single Linkage* and *Complete Linkage* for the *Average Linking* method it is the average instead of minimum or maximum similarity [14, p 19] that is used. Average Linkage is a compromise between Single Linkage and Complete Linkage, which is not as sensitive to outliers as Complete Linkage.

Algorithmic properties

Types of graphs The algorithm can be applied to any type of graph. The similarity measure chosen impacts the results of the algorithms and is the central element.

Pre-defined k or dynamic k The number of clusters k is dynamic, which is one of the advantages of hierarchical clustering but also one of its limitations, because it is hard to decide wheter the identified clusters are good ones.

Complexity $O(n^2)$ for Single Linkage and $O(n^2 \cdot log(n))$ for complete and average linkage [14, p 19]. It mainly depends on how complex the computation of the similarity measure is.

Limitations It is hard to differentiate between the many partitions found by the procedure [14, p 19]. We do not know which hierarchy level represents the "best" community structure of the graph. Stopping criteria as mentioned above could help in this situation to determine the quality of clusters in the different hierarchy levels. Furthermore, the results depend on the specific similarity measure. The fact that vertices with only a single beighbor are often assigned to different clusters, which is also mentioned in [14, p 19], does not always make sense. The major weakness is its bad scalability [14, p 19].

Properties of the results

Disjoint or overlapping clusters within a single hierarchy, clusters are disjoint, and every vertex belongs to a single cluster.

Hierarchical or non-hierarchical clusters are hierarchical.

Size of the clusters clusters have different sizes.

Related algorithms

Line-link, Adjusted Complete-link and Mahalanobis-link (see [22, p 6]).

3.3.2 Algorithm of Girvan and Newman

The most popular divisive algorithm according to [14, p 23] is found in [16] and [34]. Here the edges of the graph are selected based on the measure of *edge centrality* or *betweenness*, which is the number of shortest paths between all vertex pairs that run along the edge. This measure is an "importance" measure. E.g. in a street map the edges would represent streets. A street that is very central and is used very frequently is important and traffic would collapse if you would remove such an important street. One advantage of this algorithm is the fact, that a refined version of the algorithm by Girvan and Newman in 2004 [34] automatically detects the partition with the largest value of *modularity*, which helps to answer the question "...how do we know when the communities found by the algorithm are good ones?"[34, p 7].

As shown in Section 2.3 betweenness can be distinguished into three different kinds: *geodesic edge betweenness*, *random-walk edge betweenness* and current-flow edge betweenness.

Method

As shown in [14, p 23] the algorithm works as follows:

1. Compute the centrality for all edges.
2. Remove edge with largest centrality. In case of ties remove random edge.
3. Recalculate of centralities on the new graph.
4. Repeat from step 2 until there are no edges to remove any more.

During this process the network will split up into different sets of communities. Thus the results are hierarchical and can be visualized in a dendrogram. In [3] you can find an interactive animation of this algorithm.

Algorithmic properties

Types of graphs The algorithm can be applied to all types of graphs. Originally the algorithm works for unweighted graphs, but the calculation of edge betweenness can be extended so that the edge-weight is taken into account. This can be done by dividing the edge betweenness by the weight of the edge.

Pre-defined k or dynamic k In this algorithm k is dynamic, which means the number of clusters does not need to be pre-defined.

Complexity The complexity depends on the kind of betweenness measure chosen. Calculation of *geodesic edge betweenness* can be calculated in $O(m \cdot n)$ or $O(n^2)$ for sparse graphs with techniques based on breadth-first search [7] and [32] mentioned in [14, pp 23-24]. *Random-walk edge betweenness* and *current-flow edge betweenness* in $O[(m + n) \cdot n^2]$ or $O(n^3)$ for sparse graphs [14, pp 24]. Besides those exact techniques, there are attempts for approximation of the betweenness centrality measure by [5] and [15] to reduce the complexity below the algorithm by [7]. In written in [15, p 99] "a good approximation takes almost as much time as an exact calculation" and also [5, p 11] state that "Approximating the centrality of all vertices in time less than $O(n \cdot m)$ for unweighted graphs and $O(n \cdot m + n2log(n))$ for weighted graphs is a challenging open problem."

Limitations The algorithm is quite slow (mainly because of the recalculation in step 3) and is applicable for sparse graphs up to n ˜ 10k vertices according to [14, p 24]. This algorithm finds communities if the communities are connected only with very few edges. For very dense graphs the algorithm tends to produce super-communities containing most of the vertices (see [6, p 3] and our own results in Section 4.8.4).

Properties of the results

Disjoint or overlapping clusters Originally the algorithm does not find overlapping clusters [14, p 25], but modified versions of the algorithm exist, which can identify overlapping clusters e.g. the algorithm by [38] mentioned in [14, p 25] or the approach named CONGA (Cluster overlap Newman-Girvan Algorithm) by [20] mentioned in [14, p 25].

Hierarchical or non-hierarchical The communities are hierarchical and can be represented in a dendrogram.

Size of the clusters Within a hierarchy level, the clusters can have different sizes.

Related algorithms

A modified version by Tyler et al. shows a way to improve speed of calculation [14, p 24]. Rattigang et al. ([40] mentioned in [14, p 25]) have developed another fast version of Girvan-Newman that uses an approximation of edge betweenness values "by using a *network structure index*" [14, p 25], which reduces the complexity to $O(m)$.

3.3.3 Spectral Clustering

As described in Section 3.1.4, Spectral methods are all based on the properties of the Laplacian matrix of the graph and clustering information is obtained from the eigenvectors and eigenvalues of this matrix. The graph Laplacian matrix contains information about the pairwise similarities of all vertices in the graph and is show in Equation 3.1.

$$L = D - W \tag{3.1}$$

In Equation 3.1 D is the *Degree Matrix* (degree of vertices are the diagonal, all other values are 0) and W is the *similarity matrix*. There are different kinds of similarity measures represented by the matrix W. In our example of a web graph a similarity metric could be the geodesic distance between each vertex pair, or just the information about adjacency with optional weight information of the edges. More information about similarity measures can be found in [43, ch 2].

The result of the clustering depends on the chosen measure of similarity.

Method

1. In case there is not a graph, first *convert* the data points into a graph (see [43, ch 2.2] for the types of graph e.g. ϵ-neighborhood graph, k-nearest neighbor graph, fully connected graph).

2. Create the graph Laplacian matrix $L = D - W$.

3. Calculate eigenvalues and eigenvectors of L.

4. The smallest eigenvector has eigenvalue = 0.

5. The second smallest eigenvector has values between -1 and 1. *Negative* values could be one partition and *positive* values the other partition. This vector is called the Fiedler Vector.

6. variant #1

   (a) bisect the Fiedler vector into two partitions. Either divide the vector into positive and negative values or use the median of the values for the bisection (also called the median cut) [9, p 1]. This is what we

   (b) repeat the bi-section recursively with each partition until a stopping criterion is met.

7. variant #2

   (a) Create a new matrix X "by stacking the eigenvectors of L in columns" [35, p 2].

   (b) (optionally) create new matrix Y by normalizing X so that each row of X has unit length [35, p 2].

   (c) Run k-means over Y the k-smallest eigenvectors AFTER the smallest eigenvector, corresponding to the k smallest eigenvalues.

   (d) The result are k clusters.

   (e) One can now continue and recursively apply the algorithm to the found clusters [12, p 3] until a stopping criterion is met. A stopping criterion could be e.g. less than p nodes per partition [17, p 12].

Random Walks perspective

Spectral clustering can also be described based on the idea of random walks on the similarity graph [43, p 14]. In this approach, we do not create the graph Laplacian, but we create a matrix P that contains probability information of a random walker over the vertices.

$$P = D^{-1} \cdot W \tag{3.2}$$

P is a matrix derived by dividing the similarity matrix by the degree matrix. Each data point is the probability of a random walker, currently being on vertex i to jump to vertex j. The idea is that a random walker stays longer within a cluster and seldom jumps between clusters [43, p 14]. The described procedure of finding the eigenvectors is the same as above, with one exception: If we use matrix P instead of L, we look for the k **largest** eigenvectors of **P** to identify the cluster. That means in contrast to the description of spectral clustering earlier, we look for the k *largest* eigenvectors of P to identify the cluster, instead of the k smallest eigenvectors smallest eigenvectors of **L**.

Algorithmic properties

Types of graphs Spectral clustering can be applied to both, undirected and directed graphs, as well as unweighted and weighted graphs (see [14, p 43] and [43, p 3]). But as mentioned in [29, p 1] the algorithm assumes and undirected graph thus a directed graph needs to be converted to a undirected graph first. But this conversion also has a

disadvantage described in [29, p 1]: "These methods have the disadvantage that in many cases a clustering that is present in the original asymmetric matrix becomes partially or completely invisible after symmetrization...". How to to approach spectral clustering with directed graphs is discussed in [29].

Pre-defined k or dynamic k k needs to be predefined upfront when using k-means clustering at the end of the process. For the recursive bisection approach k does not need to be predefined.

Complexity There are multiple algorithms and derivations falling into the category of spectral clustering. Each of them have different computational complexity. The algorithm's complexity ranges between $O(n^3)$ (Alves) , $O[K \cdot t \cdot (n \cdot log(n) + m)]$ to $O(n * log(n))$ (power method by Golub and Loan, 1989) mentioned in [14, p 43].

Limitations spectral clustering algorithms tend to have problems to cluster datasets with different scales of size and density [30].

Properties of the results

Disjoint or overlapping clusters Clusters are disjoint. Each vertex belongs to one cluster.

Hierarchical or non-hierarchical It depends on whether the algorithms are applied recursively or not. Normally, the graph is partitioned into disjoint clusters, but it is possible to recursively apply the algorithm again on each cluster until some stopping criterion is met.

Size of the clusters The identified clusters have varying size.

Related algorithms

There are similar algorithms, which differ e.g. in the way the eigenvectors are calculated.

1. Unnormalized spectral clustering [42]
2. Normalized spectral clustering [35]
3. Power Method [14]
4. Krylov subspace technique [14]
5. Lanczos method [14]

3.3.4 Algorithm by Blondel et al.

A very fast algorithm, based on the optimization of the modularity was proposed by [6]. As this algorithm starts at the single vertex level it belongs to the family of agglomerative clustering. This algorithm has been applied to to a data set with 2.6 billion nodes of a mobile phone provider and to a web graph with 118 million nodes and more than one billion links. In the latter case identification of the communities of these 18 million nodes took about 152 minutes on a big-opteron 2.2k CPU with 24GB of memory [6, p 11]. According to [6, p 3] it does not have the disadvantages of similar greedy methods like the one by [34], which tend to produce unbalanced super-communities that contain most of the vertices even if the network has no significant community structure and are also inapplicable for networks with more than one million nodes [6, p 3]. An implementation of the algorithm is available[1].

Method

First Phase

1. each node i gets assigned a different community
2. for each neighbor j of i we evaluate the gain of modularity that would take place if i was moved into the community of j.
   (a) Node i is then placed into the neighbor's community where this modularity is maximal and positive.
   (b) If i is negative, i is left in its original community.
   (c) This process is repeated for every node until all nodes have been processed.

"First phase stops when local maxima of modularity is attained. No further move can improve modularity" [6, p 4]

Second Phase

1. "build a new network whose nodes are the communities found in the first phase"
2. "links between new nodes are the sum of the weights of the corresponding two communities". Links between nodes inside the community lead to self loops.

[1] http://sites.google.com/site/findcommunities/

When second phase is completed, start again with **First Phase** on the new network just built. At each *pass* (a combination of **First Phase** and **Second Phase**) the number of *meta-communities* decreases. The pass-stop criterion for the process is when there are no more changes (communities have become stable) and local maxima of modularity is attained.

Algorithmic properties

Types of graphs The algorithms works on all kinds of graphs.

Pre-defined k or dynamic k The number of clusters k is dynamic and does not need to be pre-defined.

Complexity O(m) according to a comparative analysis of [26] mentioned in [14, p 80]). "118 million nodes took 152 minutes" [6, p 4] on a CPU Opteron 2.2k, RAM 24GB. Another article by [19, p 10] states "The time to run the underlying Blondel algorithm ranged from a few milliseconds up to ˜ 40 seconds for the million node graph"

Limitations According to [6, p 4] the main limitations is memory capacity of RAM rather than computational complexity.

Properties of the results

Disjoint or overlapping clusters The clusters within one hierarchy-level are disjoint.

Hierarchical or non-hierarchical The clusters are hierarchical. The number of levels depend on the size of the graph until communities become stable. In our tests we experienced a maximum of three levels.

Size of the clusters According to [6, p 3], they have introduced "tricks in order to balance the size of the communities being merged", which leads to the assumption that the communities are balanced. That does not mean that the communities have equal size but as mentioned earlier the algorithm avoids producing super-communities containing most of the vertices [6, p 3].

Related algorithms

Algorithms by [11], [39] and [44]

3.3.5 METIS - Multilevel Graph Partitioning

As shown in Section 3.1.5 multilevel algorithms like METIS consist of the 3 phases *Coarsening Phase*, *Partitioning Phase* and *Uncoarsening Phase*, which will be described next.

Method

Coarsening Phase Graph is coarsened down to a few vertices by combining vertices into "multinodes". the new edge weights are the sum of the combined original edges. intermediate results are a sequences of smaller graphs G1, G2...Gm

Partitioning Phase then the bisection of the coarsened graph is computed, where each partition contains half of the original graph. This step can be repeated recursively by dividing every bi-partition again into a bi-partition until the amount of desired partitions is obtained [24, p 2].

Uncoarsening Phase typically consists of two steps: Projection of the partition of the coarser graph towards the original graph and then a refinement (or relaxation) process [10, p 6]. The purpose of refinement is "to select two subsets of vertices, one from each part such that when swapped the resulting partition has smaller edge-cut." [24, p 5]. There are different refinement algorithms such as Kernighan-Lin Refinedment (KLR), Greedy Refinement (GR), Boundary Refinement (BR) (mentioned in [24, p 5] or the Fiduccia-Mattheyes (FM) Refinement (by [1] mentioned in [10, p 7])

One property of the method is that the "edge-cut of the partition in the coarser graph will be equal to the edge-cut of the same partition in the finer graph" [24, p 3].

Regarding the coarsening phase there are different coarsening schemes. *Strict* and *weighted aggregations* [10, p 5] SAG and WAG respectively. SAG is simpler as each node belongs to one aggregate and the aggregates are disjoint. WAG is more complex as "each node can be divided into *fractions* and different *fractions* belong to different aggregates" [10, p 5] which makes the uncoarsening phase more difficult.

The collapsing idea of the coarsening phase can be described in terms of so called matchings. "A matching of a graph, is a set of edges, no two of which are incident on the same vertex" [24, p 3]. A matching can be constructed by different methods which are *Random Matching (RM)*, *Heavy Edge Matching*

(HEM), *Light Edge Matching (LEM)* or *Heavy Clique Matching (HCM)* as described in [24, pp 3-4].

The following describes the coarsening phase using Random Matching and Heavy Edge Matching.

Coarsening phase using RM Algorithm similar as described in [24, p 3]

1. Vertices are visited in random order.
2. If a vertex v has not been matched yet, randomly select one of its adjacent vertices u.
3. If u exists then the edge (v,u) is included in the matching and mark v and u as being matched
4. Iteratively collapse each pair inside the matching into one.

Coarsening phase using HEM Algorithm similar as described in [24, p 4]

1. Vertices are visited in random order.
2. If a vertex v has not been matched yet, select the edge with the maximum weight between v and its adjacent vertices u.
3. If u exists then the edge (v,u) is included in the matching and mark v and u as being matched.
4. Iteratively collapse each pair inside the matching into one.

Algorithmic properties

Types of graphs We could not find any hint in the papers available if those methods also work on directed graphs directly, but in [2, p 10] the algorithm was applied on a web-graph (which is directed by nature), but they first converted it into an undirected graph.

Pre-defined k or dynamic k k needs to be predefined upfront. But in case the algorithm is used for recursively bisecting the graph, k can be fixed ($k = 2$).

Complexity Linear complexity $O(E)$ according to [8, p 18] and [23, p 5].

Limitations A weakness mentioned by [10, ch 5.1] is that the results of classical matching-based coarsening schemes are quite unpredictable, characterized by high standard deviation of the edge-cuts, undesirable sensitivity to the parameters and other factors.

Properties of the results

Disjoint or overlapping clusters The resulting partitions are disjoint. The coarsening scheme WAG (Weighted Aggregation) mentioned in [10, p 4, Fig. 3.] shows that the coarsened subsets are not disjoint, as they contain fractions of vertices in intersections. But this fact will be taken care of in the uncoarsening phase where those fractions are used. It can be seen as an intermediate implementation detail. The resulting partitions are still disjoint.

Hierarchical or non-hierarchical The final results are non-hierarchical, but the intermediate steps during the coarsening phase are hierarchical.

Size of the clusters The goal of all k-way partitioning algorithms is to find clusters of nearly equal size. We have not found any hints that the resulting partitions are very unbalanced. Also in our results discussed in Section 4.9.2 the algorithm produced partitions of equal size.

Related algorithms

As mentioned in [10, p 8] Multilevel algorithms consist of many algorithmic parts and they all have the three phases of coarsening, partitioning and uncoarsening / refinement in common, but we have not found explicit names of those algorithmic families. Thus, we refer to the papers of [24], [23], [2] and [10].

3.3.6 Conclusion

The algorithms presented above were a subset of different algorithms available for community identification and we have tried to pick one algorithm out of each algorithmic family. We have described basic properties of the underlying method and the results. For a much more complete description of many more algorithms we refer to the work by [14] especially the summary tables about the algorithmic algorithm's computational complexity in [14, p 90].

While in this chapter pure theoretical information has been presented, we have applied some of the presented algorithms to example datasets in order to get a more practical view in the results in the following Chapter 4.

Chapter 4

Evaluation

In the following chapter we will evaluate different methodologies we have applied to analyze the results of different community identification algorithms on our data sets available.

4.1 Visual Evaluation of Clustering

A first method of evaluating the identified clusters and their quality is a, what we call Clustering-Link-Matrix-Plot (CLM-Plot). The idea for this plot came from a presentation (PDF) by Christos Faloutsos [13, p. 50] about Co-Clustering, but we have not found an existing name for it.

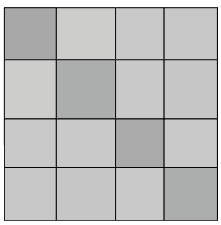


Figure 4.1: Example of a CLM-Plot

Figure 4.1 shows an $n \cdot n$ matrix containing information about the 4 clusters of the random graph described in Section 4.7. Each row and column represents a cluster. Thus, the diagonal entries represent information about the cluster itself while all non-diagonal entries describe the links of that cluster to the other cluster. Each rectangle shows the cluster quality in form of a grey value. The grey value is calculated by the Equation 4.1:

$$d = m_{xy}/(n_x \cdot n_y) \tag{4.1}$$

In Equation 4.1, m_{xy} is the number of edges running between two clusters, n_x is the number of nodes in cluster-X and n_y the number of nodes in cluster-Y. For the diagonal entries m_{xy} is simply the number of edges within the cluster. The resulting grey value is calculated by projecting the square root of d onto a scale of 255 grey values.

While the first CLM-Plot in Figure 4.1 shows 4 nearly "perfect" clusters, which are separated very well from each other, Figure 4.2 shows a CLM-Plot of a set of clusters, which are not separated that well.

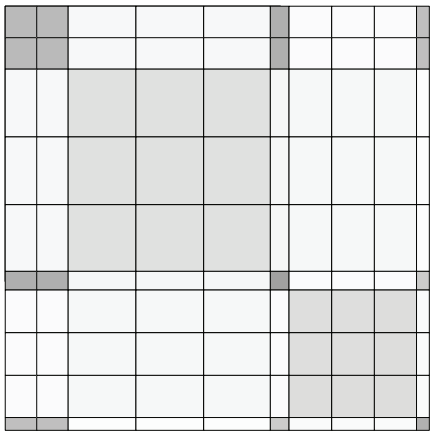


Figure 4.2: CLM-Plot of a more real-world clustering

Figure 4.2 shows 10 Clusters, which were the result of Blondel Clustering on the EN-Graph. For example, take the cluster in the lower right corner. Although it looks dark-grey, we can see that it has many links to the first two clusters, which can be seen by looking at the lower left and upper right corner of the plot, which are also dark-grey. This effect shows that these clusters could actually be merged with the first two clusters as there are lots of links running between these clusters. It could also be sign for hierarchy within the clustering. Also the two light-grey areas from upper left to lower right show clusters with lots of links between each other, which is a sign that these clusters actually belong together, but they were split up by the algorithm.

This plot is a powerful tool showing the quality of the clusters identified by an algorithm and we will use it in the following sections to compare the clustering results with each other.

4.2 Network Community Profile Plot

The Network Community Profile Plot (NCP-Plot) shows the cluster-size on the x-axis and the conductance on the y-axis (see Section 2.5). As conductance is the better the lower its value is, this plot can show at which cluster-size the clusters are good by determining the lowest values on the y-axis.

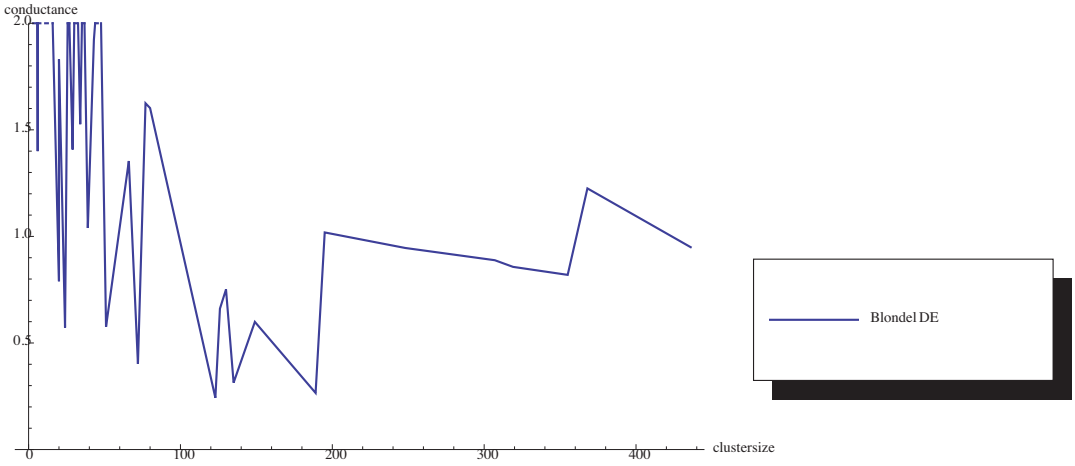


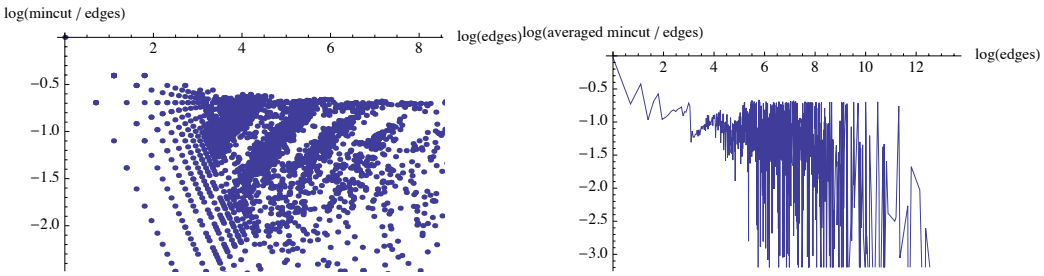
Figure 4.3: NCP-Plot of the DE-Graph

Figure 4.3 shows the NCP-Plot for the Blondel clustering of the DE graph. We can see that at a size of 120 vertices and 190 vertices (the lowest points in the plot) the conductance is best. That means we need to look at the communities at that given sizes to find good communities. This plot can help to get a quick understanding of the results of clustering algorithm.

4.3 Min-Cut Plots

Min-Cut plots are a tool we read about in [8, p. 5] for analyzing a graph and its characteristic properties. The plot is created by recursively bisecting the graph into equal sized partitions (we have used the METIS algorithm). For each cut we save the number of edges of the graph / partition before the cut, and the edge-cut, which was necessary to bisect the graph or partition. Then, we create a plot where the x-axis shows the number of edges of each partition and the y-axis shows the ratio of edge-cut to number of edges. This leads to a plot as shown in Figure 4.4a. In order to get to the plot shown in Figure 4.4b we average over all points having the same x-coordinate as described in [8, p. 6].

[8, p. 6] describes Min-Cut-Plots as a tool to compare synthetic graphs to real-world graphs. For example, the min-cut plot of a random graph would result in a straight horizontal line while a real-world graph has more a structure of our plot in Figure 4.4 although our min-cut plots do not have the "lip" described in [8, p. 11].



(a) Min-Cut Plot all (b) averaged all

Figure 4.4: Min-Cut plots ALL-Graph

This plot can be used to analyze how good a graph can be partitioned by looking for local minima in the plot, which are points of "good" cuts. By looking at the corresponding number of edges you could derive the related clusters (clusters, which have the accordant number of edges). This plot can basically help to get an understanding of the graph under observation. Although described here, we have not used this min-cut plots any further to evaluate the cluster results despite our initial intention to do so, as we have found NCP-Plots to be more useful for this purpose.

4.4 Top-10 Tables

Top10 tables are another tool we have developed and use in this thesis to evaluate the results of the algorithms. They show the clusters sorted by modularity, conductance and number of correctly classified vertices. Those tables are especially helpful dealing with hierarchical clustering results, because the tables can show the best 10 results over multiple hierarchy levels. This has helped to answer the question of "What are the best communities of the algorithm under observation?" and also "At which level are the best communities in a hierarchical clustering?". It also helps to compare the results of different clustering algorithms as it enables us to compare the best results of the algorithms and say "Algorithm A best clusters have much better modularity than the best clusters of algorithm B".

4.5 Rand Index Matrix

We have already described the Rand Index as a metric for comparing two different clusterings which could be the clusters of two different algorithms on a single hierarchy level. Our goal was to get an understanding how similar the clustering algorithms are to each other and we tried to use the Rand Index for that purpose. We did this by simply creating an $n \cdot n$ matrix where rows and columns consist of the algorithms under observation with the Rand Index as the value. It is important to note that the Rand Index only makes sense to compare disjoint clusters on a single hierarchy level where all vertices of a graph are part of at least one cluster.

4.6 Cluster Similarity Matrix

The cluster similarity matrix is a way to compare a set of clusters with the results of different algorithms and to figure out the algorithm producing the most similar cluster to the cluster under observation. Most similar is the clusters, which has most of the blogs in common with the cluster it is being compared with. The driving question behind is: "Which other algorithm produces similar cluster as the current algorithm under observation?"

The matrix has the following anatomy: The rows contain the clusters (including number of blogs) of the algorithm under observation. The columns contain the other algorithms. The process then iterates through every pair of *row-cluster* to *algorithm* and tries to identify the best-matching clusters out of the clusters belonging to the algorithms in the columns. Each cell contains three values: *cb* / *r* / *size*

- r - ratio of fractions of common blogs
- cb - number of common blogs
- n_1 - size (number of blogs) of the cluster under observation (row)
- n_2 - size of the best matching cluster (column)

The r-value is calculated with the Equation 4.2.

$$r = \frac{cb}{n_1} \cdot \frac{cb}{n_2} \tag{4.2}$$

The value of r lies between 0 and 1. A value of 1 means that there is a similar cluster, which is 100% identical (contains the same blogs and is of the same size). 0 means that a cluster does not match any of the blogs.

	Blondel	**Spectral**	**METIS**
politics economy news (137)	80/0.107/436	81/0.052/918	**74/0.174/230**
cooking food recipes (24)	**9/0.027/123**	6/0.013/115	8/0.023/115

Table 4.1: Example Cluster Similarity Matrix

Table 4.1 shows an example of a Cluster Similarity Matrix for two clusters of a clustering compared to 3 algorithms. The table can be read as follows: For the cluster *politics economy news* containing 137 nodes there exists a similar cluster found with the Blondel algorithm, which has 80 blogs in common of 436 blogs. There is also a similar cluster found with Spectral Clustering having 81 common blogs, but is much larger with 918 blogs. And there is a third cluster identified with METIS which has 74 blogs in common of 230 blogs. In terms of the *fraction of common clusters over the size of the similar clusters* the METIS clusters is the "best" cluster here, as r has the highest value of the row (0.174 = (74/137) * (74/230)). For the second row there is a most-matching cluster found with the Blondel algorithm.

4.7 Description of the Datasets

In this section we will show how we have applied our algorithms to different datasets that were available to us in order to evaluate the algorithms.

1. A Random Network of 180 Nodes with 4 pre-defined communities (random).
2. Blog-Network of 6 languages (ALL-graph) with every language being a pre-defined community.
3. Blog-Network of the German blogs (DE-Graph), which is a subgraph of the ALL-graph.

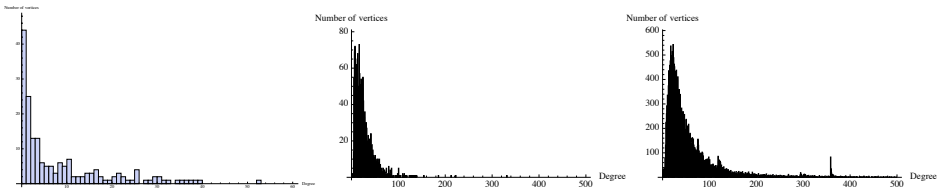
While the first two datasets have pre-defined community structure and serve as a Ground Truth or Gold standard for our clustering algorithms, the DE-Graph has no predefined community structure and represents a real-world use case. In order to verify even the results of the DE-Graph we have undertaken some more efforts to be able to compare the results against something. This will be part of Section 4.10.

Table 4.2 shows some basic graph theoretical properties of the available data sets.

	en	**es**	**de**	**fr**	**it**	**pt**	**all**	**random**
Blogs	8991	5372	1836	3401	2773	3777	26150	180
Links	480577	102837	24031	90472	75421	94968	880499	1448
density	0.005	0.003	0.007	0.007	0.009	0.006	0.001	0.0056
average degree	106.89	38.28	26.15	53.20	54.39	50.28	67.33	16
max. Degree	1983	310	331	506	1271	1198	1984	80
isolated blogs	1	7	0	13	2	14	31	0

Table 4.2: Network Comparison

All of the networks show a power-law degree distribution, which is typical for web-graphs. That means that there is a high number of blogs with very low degree and and very small number of blogs having a very high degree [14, p. 2]. Figure 4.5 shows the degree-distributions of our 3 networks.



(a) random (b) de (c) all

Figure 4.5: Degree Distributions of the data sets.

Another interesting aspect especially about the ALL-graph was the question which language has how many links into the other language. Table 4.3 shows an adjacency matrix where you can see that e.g. there are 1228 links from German blogs to English blogs but only 190 links in the other direction from English blogs to German blogs. This general pattern is clearly visible with the exception of France, which is about twice as big as the German network but has only 550 links to English blogs. This could reflect the reality of the strong attitude of the French towards the preservation of their own language. This matrix also gives us a rough idea about how well the clustering algorithms should be able to separate the language space of this graph as our assumption is that the algorithm should recognize communities more exact the sparser connected the communities are.

Table 4.3 basically shows the same information being basis for our CLM-Plot, which also shows information about the links between different clusters as shown in Figure 4.6.

	de	en	fr	it	es	pt
de	24031	1228	75	20	41	1
en	190	480577	73	657	184	100
fr	74	550	90472	24	158	56
it	14	1142	50	75424	71	221
es	40	2195	771	65	102838	582
pt	43	2449	285	49	787	94968

Table 4.3: Adjacency matrix of links between different language blogs

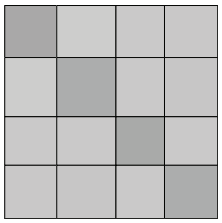


Figure 4.6: Example of a CLM-Plot

4.8 Algorithms on Random Graph

A first test case for the algorithms was our generated 180-node random graph with pre-defined community structure of 4 communities. The method we have used to generate the graph is based on the *Configuration Model* by [33, p 22] but with a modification by my mentor Darko Obradovic. The 4 communities are assigned to each blog with equal probability and have been given the labels A,B,C,D. Then he introduced a so called community factor f. In contrast to the configuration model by [33, p 22] the endpoints are not chosen with equal probability but endpoints within the same community have f times higher probability than endpoints outside the cluster. This means the higher the community factor is, the less links are running between clusters but more inside the cluster. For our example we have used a community factor $f = 7$

The result is a random graph with 4 communities it also has a power-law degree-distribution as shown in Figure 4.5 and the following community properties shown in Table 4.4.

The CLM-Plot of the 4 communities is shown in Figure 4.7. There we can see that the 4 clusters shown on the diagonal entries are darker than the rest, which means that they have a high density. But we can also see that the other non-diagonal fields are more grey than white, which means that the

ID	Cluster	Mod.	Cond.	# CCV	# blogs
429	A	0.098	1.126	41 (93.18 %)	44
430	B	0.108	1.055	44 (88 %)	50
431	C	0.086	1.302	34 (82.93 %)	41
432	D	0.091	1.293	40 (88.89 %)	45

Table 4.4: Properties of communities in the random graph.

clusters still have lots of links between them. Later in this chapter, we will see that this can be problematic for some algorithms e.g. Edge-Betweenness Clustering in Section 4.8.4.

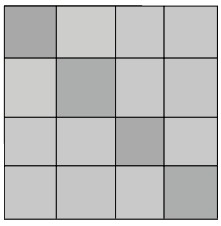


Figure 4.7: CLM-Plot of pre-defined clusters of the Random Graph

In the following sub sections we are going to apply the algorithms to the random graph and will discuss the results.

4.8.1 Blondel

First we have applied the algorithm by [6] as described in Section 3.3.4. with generally good results. The algorithm has correctly identified the 4 pre-defined communities almost absolutely correct. As the algorithm produces hierarchical results until the clusters converge against an optimal modularity value the final and best clusters could be found at Level 2 (Level 0 -2) shown in Table 4.5. The last column shows how many vertices of each pre-defined community (A,B,C,D) are inside each cluster as well as the metrics modularity, conductance and Number of correctly classified vertices.

We can see that the metrics are very close to the metrics of our Ground Truth shown in Table 4.4.

Figure 4.8 shows the CLM-Plot, which - to no surprise - almost equals the plot of the ground truth shown in Figure 4.7.

We have also tried to run the algorithm with a lower community factor $f = 3$, $f = 4$ and $f = 5$ and the results were worse the lower the community

ID	Cluster	Mod.	Cond.	# CCV	# blogs	Cluster Distribution
24	C1	0.087	1.290	36 (85.71 %)	42	A:1 B:1 C:40 D:0
27	C2	0.094	1.238	42 (97.67 %)	43	A:0 B:2 C:1 D:40
26	C3	0.099	1.096	44 (100 %)	44	A:42 B:0 C:0 D:2
25	C4	0.111	1.014	48 (94.12 %)	51	A:1 B:47 C:0 D:3

Table 4.5: Blondel Clusters in Level 2

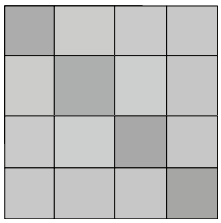


Figure 4.8: CLM-Plot of Blondel clustering on the random 4-community random graph.

factor was because more edges are running between clusters the lower the factor f is. This proves our assumption that the algorithms work better the fewer edges are running between clusters at least for this algorithm.

4.8.2 METIS

The next algorithm we have applied was METIS as described in Section 3.3.5. This was done using the METIS library and the *pmetis* executable. We applied the algorithm by using it to bisect the graph recursively with $k = 2$. We have split the graph in two parts and each bi-section again until we reach a stopping criterion. In our case we stopped recursion when a bi-section had fewer than 10 edges. The results were satisfying as all 4 pre-defined communities were identified almost absolutely correct as you can see in Table 4.6 and Figure 4.9. As it was also the case for the Blondel algorithm in Section 4.8.1 METIS did not recognize the communities as good when the community factor f was smaller.

4.8.3 Recursive Spectral Clustering

The next algorithm we are going to apply to our random data set is Spectral Clustering as described in Section 3.3.3 to bi-sect the graph at the median of the Fiedler Vector of the Laplacian Matrix of the graph and repeat the

ID	Cluster	Mod.	Cond.	# CCV	# blogs	Cluster Distribution
341	C00 #0	0.099	1.096	44 (97.78 %)	45	A:43 B:0 C:0 D:2
342	C00 #1	0.094	1.236	43 (97.73 %)	44	A:0 B:1 C:3 D:40
375	C01 #0	0.086	1.353	35 (77.78 %)	45	A:1 B:5 C:38 D:1
376	C01 #1	0.106	1.066	45 (97.83 %)	46	A:0 B:44 C:0 D:2

Table 4.6: Clusters in Level 1 (after 2 bi-sections).

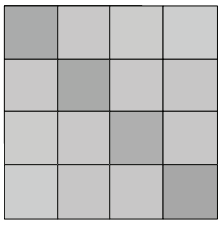
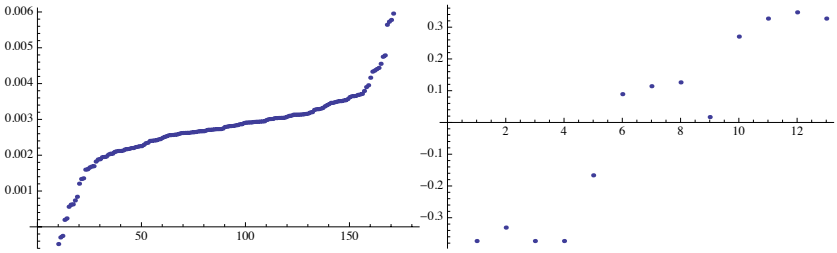


Figure 4.9: CLM-Plot of METIS Bi-Section on the random 4-community random graph.

process recursively until we meet a stopping criterion. In this case we have stopped after 10 bi-sections or when a bi-section had <= 10 nodes.



(a) random (b) toy graph

Figure 4.10: Spectral Plot of random Graph and Toy-Graph

Figure 4.10 shows the plot of the Fiedler Vector of the Laplacian matrix of the random graph and for the toy-graph shown earlier in Figure 2.1. If we compare two plots we notice that in the toy-graph plot 4.10b shows three clearly plateaus where each plateau represents a community and bisecting this plot would reveal those three communities after 2 bi-sections. In contrast to that, in the plot of the random graph 4.10a we do not see such obvious plateaus. This is also the reason why the results are not as good as with the Blondel algorithm in Section 4.8.1 or METIS in Section 4.8.2. Bisecting this plot just cuts it into half but as the cut is the median we do

not cut at an optimal point so that both partitions contain nodes of the other community. This effect can be made visible in Table 4.7, which shows that each identified cluster after 2 bi-sections contains multiple nodes of more than one community, e.g. Cluster C2 #0 contains 10 nodes of Community A, 16 nodes of community C, and 10 nodes of community D.

Cluster	**Mod.**	**Cond.**	**# CCV**	**#blogs**	**Cluster Distribution**
C1 #0	0.168	2.074	26 (57.78 %)	45	A:6 B:23 C:3 D:13 E:0
C1 #1	0.062	5.072	4 (8.89 %)	45	A:21 B:8 C:4 D:12 E:0
C2 #0	0.036	6.909	5 (11.9 %)	42	A:10 B:6 C:16 D:10 E:0
C2 #1	0.036	6.527	2 (4.17 %)	48	A:7 B:13 C:18 D:10 E:0

Table 4.7: Spectral Clustering: 4 Communities at level 1 (after two bi-sections)

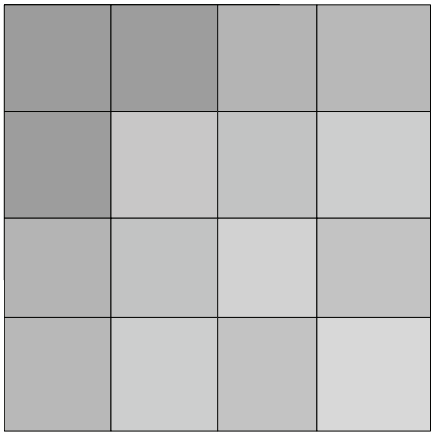


Figure 4.11: CLM-Plot Random graph at level 1

Figure 4.11 also makes those poor results visible as the diagonal entries are clearly brighter than the other entries, which means that the communities itself are very sparse and not very cohesive but have lots of links to other communities. There is one interesting aspect though: Cluster C1 #0 has a modularity value of 0.168. This value is clearly higher than the modularity of any of the pre-defined communities shown in Table 4.4. This is surprising as Cluster C1 #0 clearly consists of nodes of more than one community but according the modularity it is even more cohesive than any single community. We can only guess at this time about the cause of this effect. One idea would be to run a k-core analysis over the random graph and see if one of the k-cores matches Cluster C1 #0.

4.8.4 Girvan and Newman

As an implementation of the Edgebetweenness-Clustering algorithm by Girvan & Newman we have reused the *EdgeBetweennessClustering* implementation of the JAVA JUNG[1] Framework, which already uses the faster algorithm by [7] to compute betweenness centrality.

The *Edge Betweenness Clustering* by [16] was applied in two different approaches in our implementation:

- Remove edge with highest edge-betweenness from graph → perform Weak-Component-Clustering → repeat both steps until no edges to remove anymore. Each time the Weak-Component-Clustering identifies more than one cluster we persist the cluster to the database.
- Remove edge from initial graph → perform Weak-Component-Clustering until we find more than one cluster. If more than one cluster has been identified recursively repeat the process on each new cluster until there are no edges to remove any more in each cluster.

ID	Cluster	Mod.	Cond.	# CCV	# blogs	Cluster Distribution
175	C1 #51	0.008	0.441	123 (100 %)	123	A:30 B:34 C:31 D:28
149	C2 #25	6.658E-4	16.0	0 (0 %)	2	A:0 B:0 C:0 D:2
+ 54 1-node clusters						

Table 4.8: Girvan and Newman Clustering - Approach #1: Clusters after 1292 removed edges

To our surprise both approaches were not able to separate and recognize the pre-defined communities as shown in Table 4.8. Both approaches have basically identified one large cluster containing most of the nodes, while the rest only isolated nodes. We assume that our random graph is still too dense for the algorithm to produce good results and we suspect that it would have worked better if the communities were much better separated with just a few links between each other. This effect of *super-communities* containing most of the nodes, has also been described by [6, p 3] and highlighted as an advantage of the Blondel et al. algorithm over the algorithm by Girvan & Newman. This can be see in Figure 4.12, which visualizes the links running between the 4 communities where the nodes of each community are plotted

[1] Java Universal Network/Graph Framework - http://jung.sourceforge.net/

at one side of the square. In contrast to that you see that the toy-graph in Figure 4.12b can easily split into 3 communities by removing the edges with betweenness of 48.0 and 32.0.

Another potential problem could be the Weak-Component-Clustering, thee second step in our algorithm. As we have reused the existing *EdgeBetweennessClustering* implementation of the JAVA JUNG Framework this process was given and we did not change it. But maybe Weak-Component-Clustering is too strict, as it expects a real separate component to split up after the edge-removal. In a dense graph, it is not very likely to happen. This calls for some kind of modification of that process. For example, one can think about using a bi-section algorithm in favor of Weak-Component-Clustering, but further research in this direction would go beyond the scope of this thesis.

Notes on performance

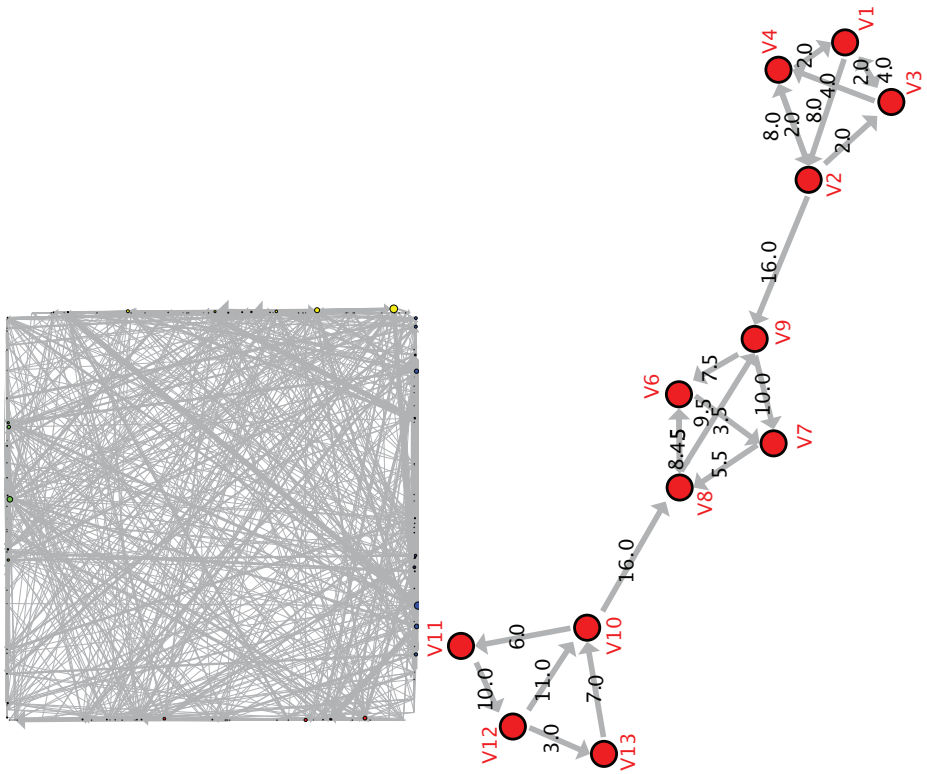
It is also worth mentioning the performance difference of different implementations. Regarding the DE-network and the IT-network we have tried to calculate betweenness centrality in three different languages: JAVA, Perl and C++. For JAVA we have used the JUNG framework, for Perl a framework by Darko Obradovic[1] and for C/C++ the popular tool network analysis toolkit PAJEK[2].

	# Vertices	# Edges	JAVA	Perl	C++
DE	1836	24031	48s	82s	2s
IT	2773	75421	122s	472s	12s

Table 4.9: Performance differences of programming languages for calculation of betweenness centrality

Table 4.9 clearly shows that the choice of the programming language has a crucial impact on the run-time of that algorithm and should be considered when dealing with larger graphs.

[1] http://search.cpan.org/~obradovic/SNA-Network/
[2] http://vlado.fmf.uni-lj.si/pub/networks/pajek/



(a) random links between communities

(b) toy graph with edge-betweenness

Figure 4.12: Demonstration of edges between clusters in the random graph and the toy graph. There is much noise between the random graph clusters.

4.8.5 Summary Algorithms on Random Graph

We can summarize that Blondel and METIS have identified the pre-defined communities almost correctly. The other algorithms were not able to separate the pre-defined communities nearly as good as these two algorithms.

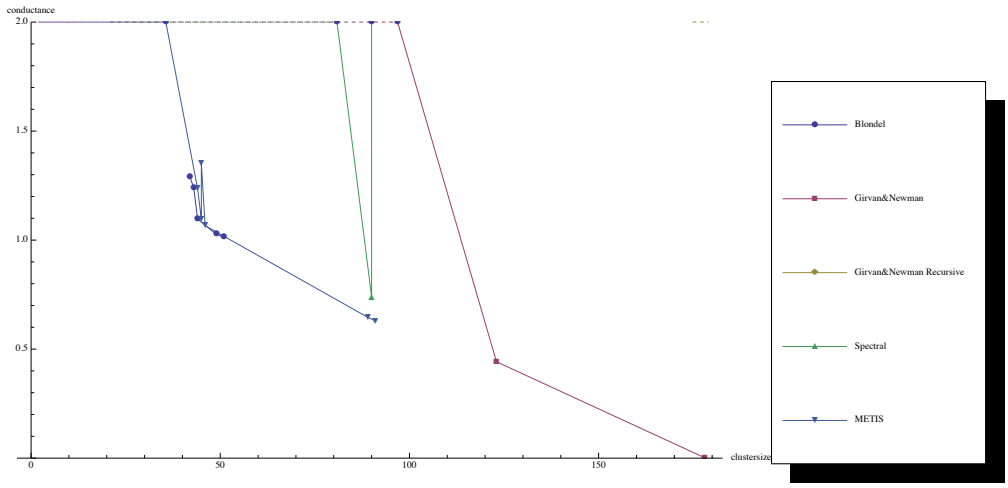


Figure 4.13: NCP-Plot of Random Graph

The NCP-Plot over all clusters and over all hierarchy levels in Figure 4.13 also shows that Blondel and METIS have identified multiple communities with a conductance value of around 1.0 at size of about 50 nodes, which is an indicator that at this size there are good communities in the graph. There are some better communities in terms of conductance at around 100 nodes size identified by METIS and Spectral Bi-Section but those communities are just good communities in terms of the conductance metric but not in terms of our pre-defined Ground Truth.

4.9 Algorithms on the ALL-graph

After having applied the algorithms to our generated mini-random graph with pre-defined community structure we are going to apply the algorithms to our real-world graph but we will focus on the clustering of the ALL-graph, which contains blogs in the six languages. As those 6 languages also represent some kind of pre-defined community structure - each language represents a single community - this use case is also a ground truth, which we use to compare our clustering algorithms with. The goal is to see if the algorithms are able to separate the ALL-graph into the six language spaces. The challenging aspect

in this use-case was the large size of the graph compared to the random graph and to see how the algorithms perform on this real-word example.

4.9.1 Blondel

The advantage of the Blondel algorithm [6] is, that it determines the numbers of clusters automatically (k is dynamic). We have used the tool *Network Workbench*[1] (see[36]) which already includes an implementation of this algorithm. This implementation will generate hierarchical results, which are in average three levels deep and we can see that the number of identified clusters converges already at level three, which means modularity cannot be improved further by an additional iteration. In case the algorithms identify more than six distinct communities, we will check if each of the identified communities contains a majority of blogs belonging to a single language. If most of the clusters are dominated by a single language then this is a proof that the algorithm was able to separate the language spaces.

Results

This algorithm was the fastest out of all algorithms we have tested especially on that large graph and has identified a reasonable number of communities for each language. Table 4.10 shows the number of identified communities per level for each language including the ALL-graph.

	de	**es**	**en**	**fr**	**it**	**pt**	**all**
Level 0	31	37	22	26	26	44	102
Level 1	14	14	15	17	19	30	50
Level 2	13	13	15	16	19	30	49

Table 4.10: Blondel: Number of clusters per language per level

The algorithm has identified 49 clusters inside the ALL-graph at level 2, although we know that we have only six defined language-communities. We have continued to analyze whether or not the 6 language clusters are actually spread across the 49 clusters identified clusters.

As Table 4.10 shows, although the algorithm has identified more than 6 communities, each identified community is dominated by a single language. That means that the algorithm has properly recognized each language-space,

[1] http://nwb.slis.indiana.edu/

Clus.	Mod.	Cond.	# CCV	# Blogs	Cluster Distribution
C1	0.026	0.085	1831 (98.87 %)	1852	de:1832 en:18 fr:1 pt:1
C7	0.158	0.081	2401 (99.17 %)	2421	en:2420 it:1
C3	0.028	0.286	425 (95.29 %)	446	en:446
C6	0.020	0.157	382 (95.98 %)	398	en:395 fr:1 it:1 pt:1
C	0.141	0.171	3680 (99.06 %)	3715	de:1 en:3710 it:4
C0	0.061	0.252	1600 (92.81 %)	1724	de:1 en:1718 fr:4 pt:1
C9	0.091	0.023	3365 (99.73 %)	3374	en:5 fr:3369
C5	0.006	0.142	161 (95.83 %)	168	en:168
C17	0.001	0.158	44 (97.78 %)	45	en:44 es:1
C14	0.067	0.103	3811 (99.48 %)	3831	en:13 it:4 es:3811 pt:3
C13	0.020	0.149	776 (99.49 %)	780	fr:1 es:779
C11	1.532E-4	0.281	14 (93.33 %)	15	fr:15
C15	0.046	0.046	2565 (99.42 %)	2580	en:7 fr:2 it:2570 pt:1
C2	0.035	0.011	183 (99.46 %)	184	fr:1 it:183
C18	0.010	0.108	447 (99.33 %)	450	en:1 es:448 pt:1
C16	0.008	0.110	321 (98.17 %)	327	de:1 en:1 fr:1 es:324
C8	0.066	0.086	2164 (99.45 %)	2176	en:8 fr:1 it:2 es:2 pt:2163
C10	0.026	0.194	1406 (98.25 %)	1431	en:15 fr:1 it:2 es:5 pt:1408
C12	0.006	0.207	197 (97.04 %)	203	en:19 es:2 pt:182

Table 4.11: Blondel: Number of clusters per language at level 2 (sorted by language majority)

but has further subdivided each language into more communities within the same language. We can summarize that at least for the ALL-graph the algorithm has produced a satisfactory result what is also supported by the metrics. The #CCV is $> 90\%$ in every case, which means each vertex is inside the same community with at least half of its natural neighbors which is a proof that the identified vertices are not just randomly assigned to a community but are very close to each other in the original graph as well, which is the characteristic a community should have.

Figure 4.14 also gives a good impression of the quality of the clusters at level 2. The diagonal entries separate from the rest of the entries, which shows that the clusters do not have many links between each other. This is a good property for a compact cluster.

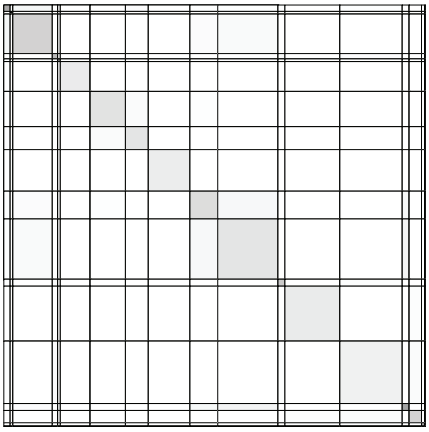


Figure 4.14: Blondel Clustering at Level 2 on the ALL-graph

4.9.2 METIS

The METIS algorithm was applied the same way as it was done for the random graph and our expectation was that it is also able to separate the 6 languages. We can see in Table 4.12 that there are 8 clusters. Almost every cluster has a single language majority except cluster C4, which contains 2078 Spanish and 1182 Portuguese blogs.

Clus.	Mod.	Cond.	# CCV	#blogs	Cluster Distribution
C1	0.056	0.145	99.79%	3266	en:3 es:3261 pt:2
C2	0.178	0.07	99.51%	3266	en:3261 fr:1 it:2 pt:2
C3	0.079	0.11	99.51%	3266	de:913 en:8 it:2 pt:2343
C4	0.067	0.150	97.37%	3265	en:5 **es:2078 pt:1182**
C5	0.086	0.051	98.44%	3264	de:104 en:48 fr:121 it:2745 es:31 pt:215
C6	0.090	0.027	99.54%	3264	en:3 fr:3259 pt:2
C7	0.132	0.207	97.03%	3264	de:3 en:3242 fr:1 it:14 es:1 pt:3
C8	0.094	0.256	92.19%	3264	de:815 en:2418 fr:15 it:4 pt:12

Table 4.12: METIS Bi-Section: Communities at level 2 (after 3 bi-sections)

After another recursion at level 3 with 16 clusters we can see that now the results have improved further and the languages are separated much better across the clusters with each cluster having one majority language. Especially Cluster C4 was split again and resulted in Cluster C5 and C10, which demonstrates that the Spanish blogs were split up from the Portuguese blogs, which now have the majority in cluster C10. The related entries have been highlighted in Table 4.12 and Table 4.13.

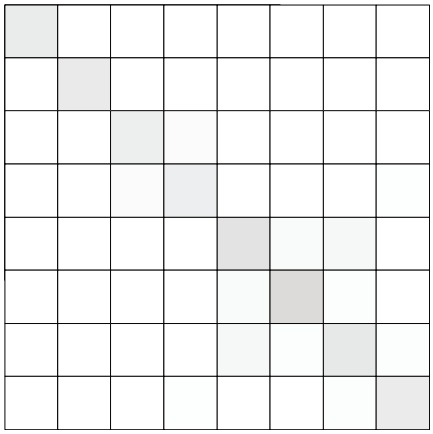


Figure 4.15: METIS Bi-Section at Level 2 on the ALL-graph

The results got even better in the next iteration at level 4 in a sense that the languages are separated better with clusters being dominated by a single language.

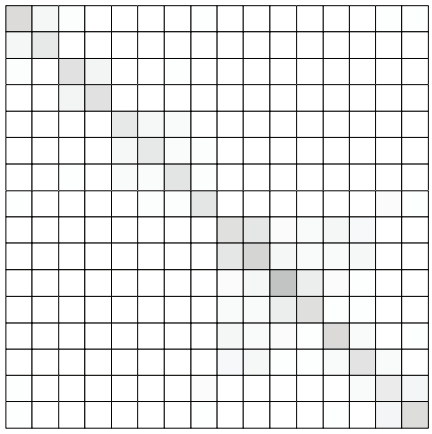


Figure 4.16: METIS Bi-Section at Level 3 on the ALL-graph

METIS Bi-Section was able to separate the language clusters quite well, results improved with every new bi-section. The METIS library for computing the bi-section is very fast and it does not consume very much of memory. Thus, this approach works well on large graphs. The algorithms also seems to work better, the less densely the clusters are connected. A bi-section of the ALL-graph could be calculated in less than 1s.

Clus.	Mod.	Cond.	# CCV	#blogs	Cluster Distribution
C1	0.027	0.307	97.98%	1634	en:2 es:1630 pt:2
C2	0.055	0.181	99.57%	1634	en:3 it:1 pt:1630
C3	0.027	0.242	100%	1633	en:1 it:1632
C4	0.044	0.212	99.82%	1633	en:1 fr:1632
C5	0.036	0.204	95.65%	1633	en:1 es:1631 pt:1
C6	0.070	0.629	94.73%	1633	en:1624 fr:1 it:4 es:1 pt:3
C7	0.135	0.178	99.51%	1633	en:1632 it:1
C8	0.049	0.531	79.3%	1633	en:1629 fr:1 it:1 pt:2
C9	0.026	0.343	96.2%	1632	en:1 es:1631
C10	0.033	0.157	96.45%	1632	en:4 **es:447 pt:1181**
C11	0.060	0.248	93.69%	1632	de:1 en:1627 fr:2 it:2
C12	0.038	0.465	82.23%	1632	de:814 en:791 fr:13 it:2 pt:12
C13	0.021	0.615	87.62%	1632	de:913 en:5 it:1 pt:713
C14	0.057	0.164	93.99%	1631	de:104 en:47 fr:121 it:1113 es:31 pt:215
C15	0.041	0.265	96.63%	1631	en:2 fr:1627 pt:2
C16	0.044	0.991	82.71%	1631	de:3 en:1618 it:10

Table 4.13: METIS Bi-Section: Partitions at level 3 (after 4 bi-sections)

4.9.3 Recursive Spectral Clustering

Applying spectral clustering was not easy because of its massive memory requirements. We needed a machine with 16GB RAM to be able to compute the eigenvectors and eigenvalues of the Laplacian matrix of the ALL-graph. For all other algorithms a machine with 3GB of RAM was sufficient. That maybe due to the used libraries (JAVA JUNG, JBLAS and COLT) and maybe the boundary could have been lowered by using more efficient methods to compute the eigenvectors and eigenvalues like the Lanczos algorithm [18] but those implementations were not available to us and would have gone beyond the scope of this thesis to implement them manually.

But similar to the application on the random graph the spectral plot of the Fiedler vector shows that there are no clearly visible plateaus, which makes it hard to identify communities, just as the results demonstrate. The algorithm was not able to clearly separate the six languages, not even after 10 recursive bi-sections. During each bi-section *some* good partitions appeared e.g. in terms of modularity as we can see in Table 4.14. For example, the table shows that C0 and C1 are basically the graph cut in two halves and apparently there is a small cut, which results in good conductance values, although the six language clusters are not separated at all (C0 $\rightarrow$ de:883

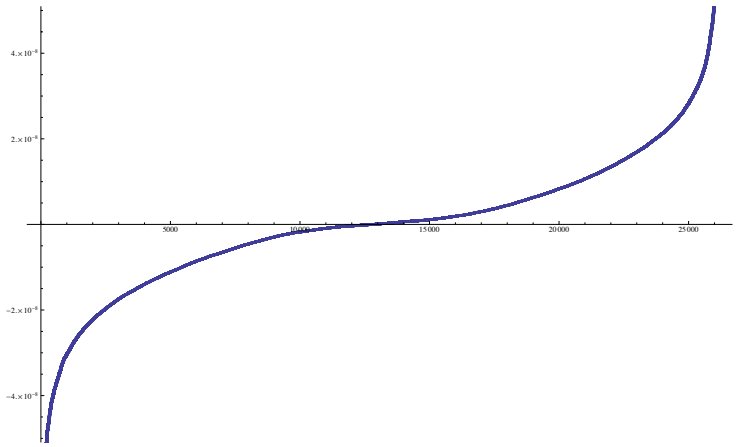


Figure 4.17: Spectrum of the Fiedler Vector - ALL-graph

en:4383 fr:1702 it:1476 es:2710 pt:1921 *and* C1 → de:953 en:4608 fr:1699 it:1297 es:2662 pt:1856). Also at level 5 with Cluster C2 we have a cluster with 409 nodes (C2 → de:6 en:51 fr:14 it:296 es:26 pt:16). But although it is the third best cluster in terms of conductance, the number of correctly classified vertices is very low at 30.56%. This shows the nodes in the cluster appear not very often within the same cluster together with more than half of their natural neighbors. In other words: The nodes in the cluster are not very close together in the original graph.

Cluster	**Mod.**	**Cond.**	**# CCV**	**# blogs**	**Level**
C0	0.209	1.381	8354 (63.89 %)	13075	0
C1	0.206	1.415	7804 (59.69 %)	13075	0
C2	0.016	1.652	125 (30.56 %)	409	5
C3	0.023	2.149	157 (19.22 %)	817	4
C4	0.067	2.820	731 (22.37 %)	3268	2
C5	0.094	2.885	1310 (20.04 %)	6537	1
C6	0.096	2.956	1249 (19.11 %)	6537	1
C7	0.031	3.484	168 (10.28 %)	1634	3
C8	0.049	3.680	454 (13.89 %)	3269	2
C9	0.078	3.704	1169 (17.88 %)	6538	1

Table 4.14: TOP10-Conductance Spectral Bi-Section ALL-graph over all levels

If we look at the spectral plot of the Fiedler Vector of level 5 we notice the two plateaus are more obvious and the first plateau represents C2 in Table 4.14. The same cluster can also easily be spotted in the CLM-Plot shown in Figure 4.18 where the darkest diagonal entry indicates C2 in Table 4.14.

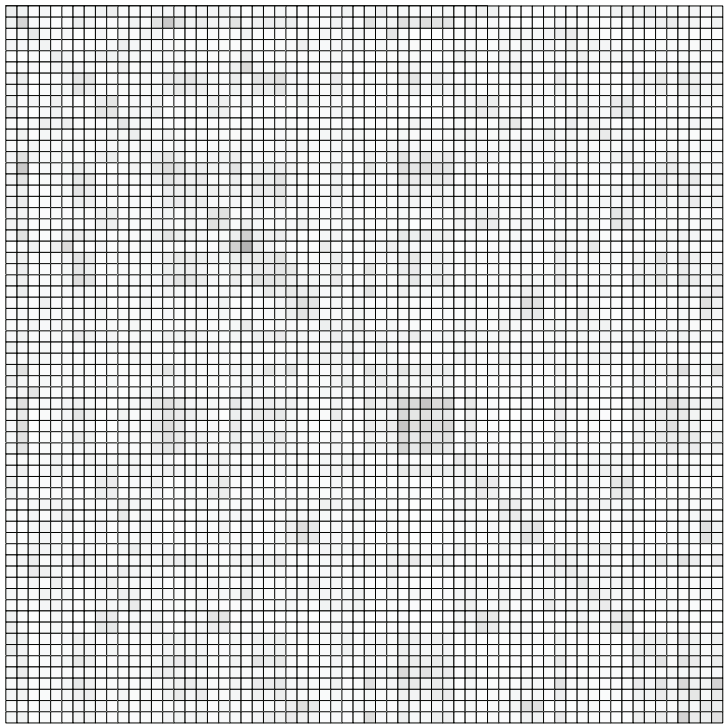


Figure 4.18: CLM-Plot ALL-graph Spectral Clustering Level 5

In summary, Spectral Bi-Section clustering was not able to separate the six languages and it also consumed lots of memory. Thus, the algorithm is not the best choice for very large graphs because of the large matrices, which need to be held in memory. Also, there is lots of room for improvements. For example, instead of the idea of bisecting the graph we could have used k-means clustering on the Fiedler-Vector as described in Section 3.3.3 but this assumes that we define the number of clusters up-front, which is normally not known in advance, as that is something we want to find out dynamically.

4.9.4 Girvan and Newman

Unfortunately, it was not possible to test the Edge-Betweenness clustering of Girvan & Newman [16] on the ALL-graph (26150 nodes, 880499 edges) because of its computational complexity of $O(m \cdot n)$ or $O(n^2)$ for sparse graphs for computing the centrality (*geodesic edge betweenness*), which also

needs to be recomputed after each removed edge leading to a total complexity of $O(m^2 \cdot n)$. This process has taken about 8 hours for the DE-graph (1836 nodes, 24031 edges) so it would have taken several months for the ALL-graph.

4.9.5 Summary

To summarize the application of algorithms on the ALL-graph the results are similar for the random graph in Section 4.8.5. Blondel and METIS were able to separate the six languages in the resulting clusters. In contrast to that, Spectral Clustering was not able to separate the languages and the algorithm of Girvan and Newman could not be applied at all.

4.10 Clustering of the DE-Graph

After we have applied the algorithms to our random and our ALL-graph ground truth data set we applied the algorithms to the German dataset (DE-graph). The difference here is that we do not know what we are actually looking for as we do not have a known community structure. This is a real world use case.

Thus, we have the problem to verify the results to know whether or not the identified communities are good or not. Other instruments are the metrics *modularity*, *conductance* and *number of correctly identified vertices* we have already used in the tables earlier in this chapter.

But in order to be able to compare the identified communities against *something* we have applied an additional method in order to compare the results of the algorithms against *some kind* of ground truth.

4.10.1 Manual tagged clusters

In a labor-intensive process we have manually added tags to each blog in the DE-graph to somehow categorize each blog's content (e.g. web, music, cooking...).

The goal of that process is to find out if there is a connection between content-clusters and structural clusters identified by our algorithms. For example, supposed we have formed a cluster containing all blogs tagged with *cooking* and *food*. If our assumption is true then our clustering algorithms must also have identified a cluster containing all or most of the blogs tagged with the tag *cooking* and *food*.

In the following section we will describe our process in order to answer that question.

Currently we see three ways how to make use of the tags:

1. Analyze frequency of the manually assigned tags in the DE-clusters identified by the algorithms. This gives a good overview over the tag distribution in the clusters and will give a feeling about which tags are used the most. Example: A cluster containing 90% of all blogs tagged as "movies". This process is described in more detail in Section 4.11.

2. Create a Cluster for each tag and calculate modularity, conductance and number of correctly classified vertices. Our assumption is that such a tag-cluster having a good modularity value should also be identified by one of the structural clustering algorithms. This structural cluster partly should contain a high number of blogs of the tag-cluster. This should be partly visible from the previous point. We will describe this process in Section 4.10.1.

3. Grouping of tags. During the manual tagging process there are a lot of tags, that basically mean the same, or tags that are used together in many cases. In some cases it makes sense to group those tags and create clusters containing blogs with those grouped tags. E.g. instead of creating two separated clusters for *food* and *recipes* we can create a single cluster containing all blogs tagged with *food* or *recipes*. This process will be described in Section 4.10.1.

Creating a Cluster for each blog

In this section we form a cluster for each tag assigned to the blogs. E.g. there will be a cluster containing all blogs tagged with the word "food" and another cluster for all blogs tagged with "movies". The resulting clusters will be overlapping, which means they will contain partially the same blogs because there can be blogs tagged with "food" and "movies", which is totally possible in reality. This overlapping is not a problem in our case because our goal is to identify "good" communities rather than partitioning the graph into disjoint clusters.

Then we have calculated modularity, conductance and number of correctly classified vertices for every single-tag cluster and produced Top10 tables for each metric because we are not interested in all clusters but in the "best" clusters in terms of the metrics. Table 4.15 shows the best 10 tag-clusters in terms of their conductance value and Figure 4.19 shows a CLM-Plot of all tag-clusters.

From the plot in Figure 4.19 we can see that most of the clusters on the diagonal are not very dark, which means they are not very cohesive and

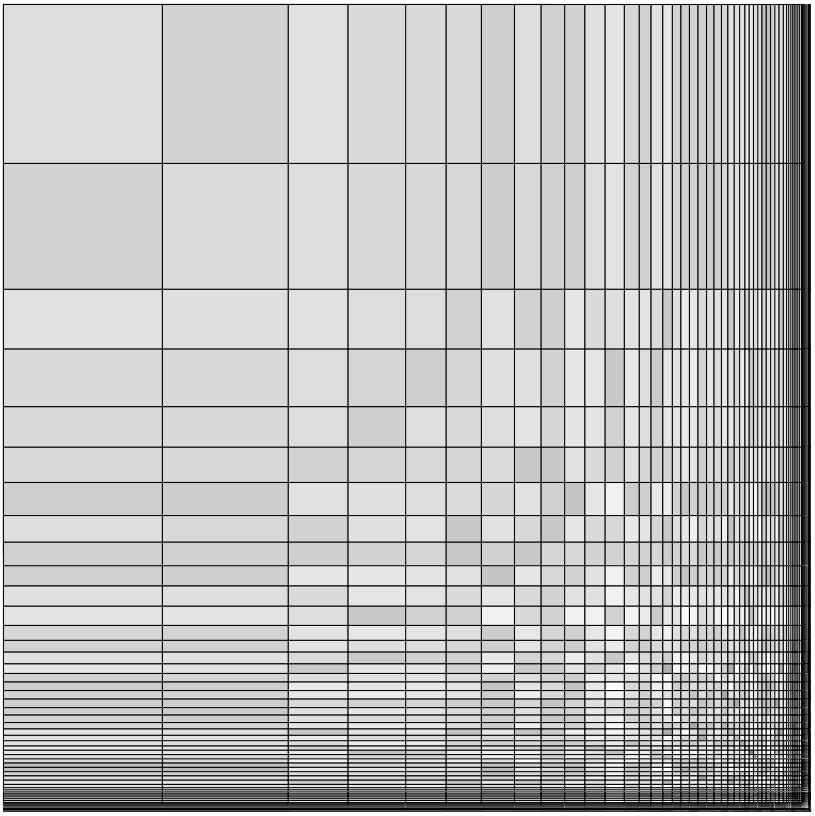


Figure 4.19: CLM-Plot DE-Graph - each tag in a cluster

Cluster	Mod.	Cond.	# CCV	# blogs
personal	0.043	2.815	105 (32.21 %)	326
misc	0.031	3.657	66 (25.48 %)	259
politics	0.010	6.891	11 (9.32 %)	118
web	0.007	8.025	16 (13.01 %)	123
business	0.001	8.076	2 (10 %)	20
swiss	8.939E-4	8.909091	0 (0 %)	10
career	1.658E-4	9.25	0 (0 %)	4
media	0.004	10.226	2 (2.7 %)	74
advertising	8.305E-5	12.0	0 (0 %)	2
socialmedia	0.002	12.694445	1 (1.82 %)	55

Table 4.15: TOP 10 Conductance of single tag clusters DE-graph

dense and the have lots of links to other clusters. Some smaller clusters like the *de_politics* and *de_swiss* are darker than the rest but are also very small. Another interesting effect becomes visible in this plot: For example, the 2 diagonal clusters starting from the upper left corner are brighter than their linking rectangles, which means that those two clusters have many links between each other. That could indicate they can potentially be combined into one cluster. This makes an aspect visible and we are going to investigate this in Section 4.10.1.

Grouping of tags

As we mentioned earlier, there are blogs tagged with "food" and "recipes" and in our previous section this would result in two different single-tag clusters that overlap to a greater extend because both tags are related and are ofter used together. In order to algorithmically determine the tags that qualify to be combined we have tried to come up with the following process.

We created a quadratic co-occurrence matrix for every tag. The value of each cell c is shown in Equation 4.3:

$$c = n_{12}/n_1 \tag{4.3}$$

In Equation 4.3, n_{12} being the number of blogs containing *tag1* and *tag2* together and n_1 is the number of blogs which contain only *tag1*.

c is a value between 0 and 1. This matrix can be asymmetric because e.g. "cooking" might only be used in conjunction with "recipes" but "recipes" can also be used with other tags.

	personal	**misc**	**web**	**politics**	**society**	**media**
personal	0.996	0.584	0.122	0.131	0.113	0.051
misc	0.737	1	0.131	0.069	0.077	0.050
web	0.325	0.276	1	0.105	0.073	0.260
politics	0.364	0.152	0.110	1	0.491	0.067
society	0.435	0.235	0.105	0.682	1	0.058
media	0.226	0.173	0.426	0.106	0.066	0.986

Table 4.16: Subset of a tag-co-occurrence matrix - showing which tags are often used together

After having created this matrix in Figure 4.16, we will filter this matrix to only show values having e.g. a $c > 0.7$. Those tag-pairs are very likely to be combined or grouped. The value of *0.7* was chosen arbitrarily.

From that list of tags and combinable tag-candidates we have created 16 clusters consisting of single or combined tags, which we have used for further analysis. For example, from the co-occurrence matrix we have created a combined cluster for the tags *web* and *socialmedia* as both tags are mostly used together.

This process was executed manually. At one time we have thought about automating this process too: The idea was to see the Tag-Co-occurrence-Matrix in Table 4.16 as a weighted adjacency matrix of tag relationships with the co-occurrence value being the edge weight. Then the idea was to use a clustering algorithm to create clusters of combinable tags. But this was not possible for this thesis because we cannot use a clustering algorithm in a thesis on clustering algorithms in order to verify clustering algorithms. Thus, we performed the grouping manually.

So the result of that process was the following list of 16 cluster in Table 4.17 for further analysis:

Then for some samples of combined-tag-cluster we tried to identify the best matching cluster out of the clusters identified by our algorithms containing most of the blogs of that *tag-cluster* with the help of our *Cluster Similarity Matrix* described in Section 4.6.

Table 4.17 shows three sample clusters we have picked randomly.

Observation #1 A best-matching similar cluster for the first cluster seems to be identified in row 1 with the Blondel algorithm even though this best-match contains only 9 out of the 24 blogs and this matching clusters contains a total of 123 blogs. That means that the tagged

Cluster	Mod.	Cond.	# CCV	# blogs
misc personal	0.022	4.507	34 (17.8 %)	191
politics economy news	0.013	5.904	21 (15.33 %)	137
lifestyle art music design	0.005	11.28	4 (4.04 %)	99
tv media advertising	0.004	10.225	2 (2.7 %)	74
music	0.003	15.10	2 (2.9 %)	69
poetry lyric literature journalism	0.002	16.96	0 (0 %)	65
photo	0.001	22.98	0 (0 %)	42
web socialmedia	0.001	19.09	1 (2.5 %)	40
personal photo	0.001	23.72	0 (0 %)	27
cooking food recipes	0.001	23.16	0 (0 %)	24
business career startup	0.001	8.0	2 (9.52 %)	21
traveling	2.753E-4	54.111	0 (0 %)	18
local	5.414E-4	36.058	0 (0 %)	18
sports	3.406E-4	30.666	0 (0 %)	12
women	1.402E-4	65.5	0 (0 %)	10
personal women	7.315E-5	81.5	0 (0 %)	8

Table 4.17: List of combined and single tag clusters

Cluster	Blondel	Spectral	METIS
cooking food recipes (24)	**9/0.027/123**	6/0.013/115	8/0.023/115
poetry lyric literature journalism (65)	26/0.029/355	34/0.019/918	**11/0.062/30**
business career startup (21)	**5/0.17/7**	11/0.006/918	**5/0.17/7**

Table 4.18: Cluster-similarity for 3 random clusters of combined tags

cooking-blogs are only a minor fraction. But at least we gained the knowledge that 9 of 24 of all the blogs we have tagged with *cooking, food or recipes* are found all together in one cluster identified by the structural algorithms. The error could be due to the fact that we have tagged only 731 of 1836 total DE-blogs and maybe if we would have tagged more blogs out of the best-matching cluster with these tags, the results were clearer. A manual check confirms this assumption as the best-matching Blondel clusters mainly contains blogs dealing with the topic *cooking* what we can see by looking at the tag-cloud generated from the blog's RSS-feed. The tags there are *teil,rezept,weinernte,kochen,event,euro,koch,penne,johann,stehen.* This proves that the results of our cluster similarity matrix are useful and

would result in more exact results the better the quality and completeness of the tagging-process is. But even the current result is an indicator for the existence of a connection between the structural clusters and their content.

Observation #2 For the third cluster *business career startup* Blondel and METIS seems to have identified the same similar cluster. Both clusters have a total of 7 blogs where 5 of them are blogs from the *business career startup*-cluster. A manual test confirms that both 7-blog clusters are almost identical with the exception of one blog, which is different in both clusters. The blogs of these two cluster blogs like are: *deutsche-startups.de, connectedmarketing.de,blog.wiwo.de/gruenderraum, e-commerce-blog.de, inside.gruenderszene.de, www.handelskraft.de, webregard.de and http://ecommerce.typepad.com*

4.10.2 Conclusion of tag-clustering

Tag clustering was performed as a try to identify the existence of a connection between the structural clusters and their content. This process - though labor intensive - has definitely helped to get a better understanding about the network structure of the DE-graph and the clusters. Creating the tag-clusters, measuring their modularity and conductance and finally trying to identify similar clusters was a good way to visually compare the clusters against each other by also taking the content into account.

But we clearly have to say that this process has its limitations and is very prone to human errors. First of all: tagging is a manually executed process. People use different tags for the same thing, or do spelling mistakes. Another way to extract keywords from the page content could have been to use external services e.g. like Yahoo's Query Language API[1], which is able to extract keywords from text automatically. All sorts of these kinds of errors can distort the results. Another cause for errors was the fact, that the initial crawl of the blog-network was executed in 2009 (see [37]), which was the basis for the link structure. Unfortunately the website content of that initial crawl was not persisted. Thus, we needed to re-crawl the content again, in order to have content for tagging which we could correlate with the structural clustering results. The problem is there now is a mismatch between the source for the initial structure and the assigned tags as the tags are not for the initially indexed articles, which led to the current link structure, but they are derived from the content of the blogs today.

[1] http://developer.yahoo.com/yql/console/

4.10.3 Automatic Clustering

After having evaluated the tagging-process we have applied the clustering algorithms to our DE-graph to be able to evaluate the results and compare it with the results of the tag-clusters.

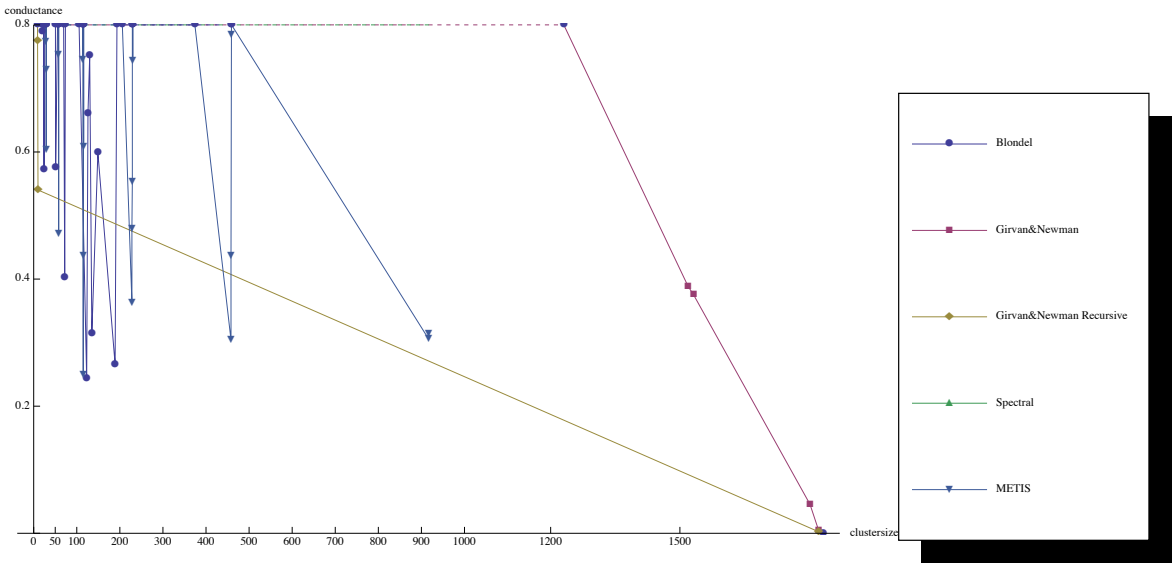


Figure 4.20: NCP-Plot of the clustering of the DE-graph.

The NCP-plot in Figure 4.20 shows that Blondel clusters have best conductance values at a clusters size of about 100 and 200 blogs. Also METIS has identified a good cluster at around 100 vertices and two good clusters at the size of 450 blogs and 900 nodes. This can also be seen if we look at Table 4.19 showing the best cluster in terms of conductance for each algorithm. This also shows that the tag-clusters (PerTag and CombinedTags) are not as good as the clusters identified by the algorithms in terms of conductance.

From the high conductance values for the tag-clusters we can tell that the clusters are not as good as the structural ones. It is also likely a result of the fact that we have not tagged all 1836 German blogs but only 731 of them. The EdgeBetweenness clustering results have repeatedly shown to produce a super-community containing almost all of the vertices, which can be seen in Table 4.19. That makes this algorithm rather unsuitable for our purposes as it will be obviously hard to compare such a large cluster with the tag-clusters, as almost every tag-cluster will be covered inside that large cluster. The results would be meaningless and that is why an Edge-Betweenness Cluster does not appear in Table 4.18.

Cluster	Mod.	Cond.	# CCV	# blogs	level
Blondel	0.061	0.242	119 (96.75 %)	123	2
METIS	0.0594	0.249	112 (97.39 %)	115	3
EdgeBetweenness	0.024	0.375	1425 (93.02 %)	1532	17367
Spectral	0.204	1.535	600 (65.36 %)	918	0
PerTag	0.043	2.815	105 (32.21 %)	326	0
CombinedTags	0.022	4.507	34 (17.8 %)	191	0

Table 4.19: DE: Best clusters by conductance by algorithm

As a last analysis of the results of the clustering algorithms on the DE-graph we have used the Rand Index Matrix as described in Section 4.5. One property of the Rand Index mentioned in this section was that it only makes sense to use the Rand Index if the clusters of the two cluster-sets to be compared are disjoint and all vertices are assigned to a single cluster. Otherwise the counts of the Rand Index are not comparable if you compare clusters with different characteristics. It turned out that this limits the usefulness to use the Rand Index to compare different sets of communities. In Section 2.4.5 we said that community identification is about identifying good communities rather than partitioning the graph into disjoint sets. In this chapter we have seen that these *good* communities can be found in different levels of hierarchical clusterings. When creating the Rand Index Matrix you have decide upfront which levels you want to compare. This decision impacts the Rand Index, as in each level the vertices are potentially assigned to different clusters. For demonstration purposes we will show two Rand Index Matrices of two different scenarios.

	Spectral (Level 2)	Blondel (Level 2)	METIS (Level 2)
Spectral (Level 2)	0.0	0.774	**0.868**
Blondel (Level 2)	0.0	0.0	0.787
METIS (Level 2)	0.0	0.0	0.0

Table 4.20: DE: Rand Index Matrix for 3 clusterings at level 2

Table 4.20 shows that Spectral Clustering and METIS produce similar clusters at level 2 as the Rand Index has the highest value in that table.

Table 4.21 shows the Rand Index for the 3 algorithms for different hierarchy levels. Basically the number of clusters is higher in this table. Blondel

	Blondel (Level 0)	**Spectral (Level 3)**	**METIS (Level 3)**
Blondel (Level 0)	0.0	0.852	0.897
Spectral (Level 3)	0.0	0.0	0.887
METIS (Level 3)	0.0	0.0	0.0

Table 4.21: DE: Rand Index Matrix for 3 clusterings at level 2

has 31 clusters at level 0, and Spectral and METIS have 16 clusters at level 3, which implies the size of the clusters is smaller. Apparently the Rand Index value is higher than in Table 4.20. This time Blondel and METIS seem to have the matching of clusters.

Although the presented facts can reveal interesting facts in general, during our experiments the Rand Index Matrix was of less value for our evaluation than we initially thought. A possible improvement could be to calculate the Rand Index Matrix for all possible permutations of clustering levels and see were the Rand Index is maximal but this was out of scope from a time perspective and would have been computationally very complex.

4.11 Explorative Cluster Analysis

While the previous sections where rather algorithmic and have compared different clusterings with the help of metrics, this section will describe an experimental approach by analyzing the clusters in a more explorative way. During the creation of this thesis we often asked ourselves the question "What are the blogs which form a cluster actually about?". To answer this question we have used the idea of a tag cloud, a very common pattern in the blogosphere. A tag cloud can be used to give an overview over a set of articles by showing the keywords of the articles in a so called "cloud" where some keywords are printed bigger than others to show their importance or frequency. This approach looks reasonable for our purpose as well as it let us "zoom" into a cluster and see more than a couple of numeric metrics only.

Tag cloud based on blog content

In order to be able to render a tag cloud we first got the content from which we extract the keywords. We have indexed the most recent 10 articles (title + content) from every blogs RSS feed and put it inside an index (based on Apache Lucene). Then for every blog in the index we have extracted the 10 keywords having the highest *Term Frequency - Inverted Document*

Frequency (TF-IDF) value (see [46]). In other words we have extracted the 10 most important keywords for every blog in the cluster. The *keyword-importance-value*, the TF-IDF value, was always calculated for each cluster as the corpus with the goal to identify the most important keywords for the cluster. Apache Lucene was a tool, which helped us to gather these metrics.

After we had those important keywords, we have counted the frequency of each keyword in the cluster-blogs. This list sorted by frequency in descending order will be rendered as our tag cloud. The higher the frequency the higher the font-size of the keyword will be, which gives the common look-and-feel of a tag cloud.

The 10 extracted keywords for the DE-graph ordered by frequency are as follows: *heute,stuttgart,woche,gegen,neue,blog,teil,tage,alles,neues*

Those tags for the whole graph are of little value as the whole topic spectrum is too broad. More interesting is to use those keywords to analyze the content of single communities. Examples of various tag-clouds for specific clusters are covered in Table 4.22.

The advantage of this tag-cloud generating process is that it is fully automated. We have created a small web-application performing all the required steps automatically. This tool has helped us during this thesis to explore the clusters manually to get a better understanding of the cluster's anatomy. But there were also problems with that approach, which have impacted the quality of the results. There was the technical problem to automatically identify the RSS feed and parse it. We have reused a software library, which discovers the URL of the RSS feed automatically but apparently there were still a larger number of blogs where that automatic feed discovery resulted in an error so that we couldn't get the content. An alternative would have been to use the whole HTML source code for every blog URL instead of the RSS Feed. Due to time constraints this couldn't be implemented anymore. It is also worth noting that content extraction, more specifically tag-extraction seems to be a whole research topic on its own and we have just used this approach as an experiment to get a rough overview over a cluster and to have a tool at hand to make it "visible" easily.

Tag Cloud from manual tags

As described in Section 4.10.1 we have already tagged the DE-blogs manually. Thus, it is an easy step to create another tag-cloud based on those manually created tags as opposed to the tag-cloud created in an automated process based on the real website content. As those tags were assigned by a person who has manually looked at each website and assigned the tags more carefully we expect those tag tags contain less "noise" than the automati-

cally determined tags. Although this manual process is also prone to errors - especially subjectiveness - we think that those tags give a better impression about the question "what are the blogs inside a cluster writing about", and we expect them to match up with the tag cloud from section 4.11 at least in a sense that if the automatic tag cloud contains the terms *recipes, cooking, food, restaurant or wine*, then the manual tag cloud should also contain similar terms.

personal (327),misc (259),web (123),politics (118),society (85),media (75),music (70),socialmedia (55),blogging (50),photo (42),tech (42),journalism (40),art (32),movies (24),news (24),business (20),literature (18),traveling (18),cooking (18),local (18),books (17),humor (16),food (15),marketing (15),sports (12),design (12),mobile (10),swiss (10),law (10),women (10),berlin (10),poetry (9),lifestyle (9),culture (9),pr (9),apple (7),health (6),science (5),family (5),shopping (5),wordpress (5),it (5),lyric (5),religion (4),economy (4),career (4),recipes (4),videogames (3),podcast (3),startup (3),history (3),thailand (2),languages (2),software (2),animals (2),videos (2),privacy (2),comics (2),advertising (2),mannheim (1),people (1),nature (1),bahn (1),patents (1),wine (1),ecommerce (1),london (1),language (1),spam (1),hardware (1),google (1),events (1),bloging (1),security (1),education (1),brands (1),restaurants (1),leipzig (1),community (1),hamburg (1),asus (1),photos (1),politcs (1),russia (1),basel (1),tv (1),finance (1),garden (1),electronics (1).

This gives a first impression of the DE-graph. We see that *personal* blogs and blogs writing about *web* topics and *politics* are the majority of the blogs tagged manually.

Examples of Cluster Tag-Clouds

In this section we are giving a few random examples of single clusters and their tag clouds.

4.11.1 Content clustering

Another approach we have taken to evaluate the results of the structural algorithms was to cluster the DE-network by content with the help of a clustering algorithm based on Non-negative matrix factorization (NMF) with the help of Rafael Schirru (DFKI). That means in theory after having applied this algorithm over all blogs and their content you get a list of clusters and the

Cluster	# Blogs	Mod.	Cond.
Blondel	24	0.007	0.572
content-tags: vfb, spieltag, frankfurt, bundesliga, vfl, teil, saison, fc, eintracht, schalke			
manual-tags: sports, personal, misc			
Blondel	123	0.061	0.242
content-tags: teil, rezept, weinernte, kochen, event, euro, koch, penne, johann, stehen			
manual-tags: food, cooking, personal, misc, music, wine, nature, recipes, garden, media, restaurants, business			
METIS	458	0.155	0.304
content-tags: stuttgart, israel, berlin, blog, gegen, deutschland, morgen, welt, heute, neues			
manual-tags: politics, society, personal, journalism, misc, news, web, media, swiss, music, tech, socialmedia, religion, local, movies, law, art, economy, women, traveling, lifestyle, humor, privacy, history, russia, basel, berlin, culture			

Table 4.22: Example of 3 random DE-clusters and their content-tags and manual tags

topic they belong to e.g. *Cluster 1 (news and politics)*, *Cluster 2 (food and cooking)* and so on. It is basically the opposite of our structural-clustering algorithms. This content clustering does not take the link-structure into account at all, but uses only the content. The goal was to compare the structural cluster against the content clusters and to see if content clusters are related to structural clusters.

Unfortunately this approach was not successful at all and we have not identified any meaningful clusters. The reasons for this may vary. On the one hand a reason could have been that we did not get enough content for each blog e.g. because the RSS-Feed couldn't be parsed or our automatic process couldn't find one. On the other hand we only used the titles of the 10 recent blog articles. It could be that the information provided in the titles was not enough and we should have taken the article content as well. As this approach was not a central topic of this thesis it was out of scope to perform any further investigations in this direction.

4.11.2 Conclusion of explorative analysis

The goal of this whole Section 4.11 was to have a tool at hand, which let us easily "look" inside a cluster of blogs to see what they are about. Having this possibility had also opened up new ways to compare how the content of the clusters relates to the structural clustering algorithms. A permanent question was: "Why do certain algorithms produce which clusters?" and this explorative tools help to answer that question by also looking at the content of each cluster. It helped to find out if certain clusters are formed because the blog-owners all write about the same topic and tend to link each other because of that common topic or even know each other in person.

Although we have used a lot of tools from the field of textual analysis like TF-IDF values, term co occurrence Matrices, Tag-frequencies and tag clouds, which we have related with structural data to come up with a measure, which we call cluster-similarities, the question cannot be answered in an absolute way. It is true that in some cases there is a connection between content of the blogs and their link structure with the result that the algorithms identify them to be part of the same cluster or community but this was always the case. Good examples have been the "cooking-blogs", which seem to link to each because of the content. We will discuss this phenomenon in Section 4.12 in more detail.

4.12 Clustering of the Six Language Graphs

After we have elaborated on the methodologies how we have evaluated and benchmarked the algorithms on various graphs in the previous sections we have now applied the algorithms on all six language graphs. The result of this computationally intensive process is Table 4.23 and Table 4.24 showing the Top-2 Clusters for every algorithm in every language. Table 4.23 contains results ordered by modularity and Table 4.24 contains results ordered by conductance. The results are returned over multiple hierarchy levels and not just for a specific level. The goal is also to draw conclusions at which level the best clusters can be found which is a common problem with hierarchical clustering algorithms [14, p 90]. In addition to the metrics of modularity and conductance those tables also include the tags collected in Section 4.10.3 to see what are the best clusters writing about to see if we can see a prove for our hypothesis that a connection between structural clusters and their content does exist.

Observation #1: There is a difference between the tags of the two DE-clusters in Table 4.23 and Table 4.24. While there does not seem to be a

connection between the tags in first table, there is clearly a connection visible in the second table. The tags of both DE-clusters in the second table are clearly about the topic cooking. Thus, it seems that the best clusters (in terms of conductance) of Blondel and METIS contain similar blogs.

Observation #2: The same surprising effect seems to happen for the FR-clusters, which are also about the same topic *cooking* in the conductance table. From a sociological point of view one can reason that weblogs writing about cooking seem to link very often to other blogs of the same topic and less often to blogs of other topics. At least for the French and German blogs that seems to be the case as those **cooking-clusters** are the best clusters in both languages out of our dataset. It is also interesting given the different sizes of the two networks and clusters. While the French networks is about twice the size of the German network, the French best-clusters are about 4 times as big as the best German clusters but still share the same topic.

Observation #3: While the EN-clusters in Table 4.23 are rather huge clusters ($>$ 2000 blogs), there is a significant difference in the size of the EN-clusters in Table 4.24 where the METIS cluster with the best conductance value is very large ($>$ 2000 blogs) and the Blondel clusters is rather small (45 blogs). This can be interpreted that the size of the community is no quality criterion for a community. Only when you introduce other metrics as modularity and conductance it is possible to judge the quality of a community as those values seem to be independent of he size of the cluster.

Lang	Algorithm	Mod.	Cond.	# CCV	# blogs	level
DE	Spectral	0.204	1.535	600 (65.36 %)	918	0
stuttgart,heute,berlin,tage,neue,deutschland,gegen,bitte,ach,neues						
DE	METIS	0.182	0.314	911 (99.35 %)	917	0
tage,woche,blog,heute,alles,teil,letzte,bitte,wochenende,neue						
EN	Spectral	0.224	1.003	3719 (83.2 %)	4470	0
day,wednesday,blog,happy,time,love,life,week,review,book						
EN	METIS	0.219	0.091	2241 (99.73 %)	2247	1
day,love,happy,time,wednesday,birthday,review,blog,book,fun						
FR	Spectral	0.201	1.653	1025 (60.29 %)	1700	0
retour,petite,vacances,petits,bonne,noël,jour,rentrée,nouveau,suite						
FR	Blondel	0.165	0.419	923 (98.72 %)	935	2
retour,rentrée,vacances,petite,happy,petits,sac,jour,robe,merci						
IT	METIS	0.240	0.041	316 (91.59 %)	345	2
cinema,film,iphone,ottobre,trailer,parla,puntata,factor,tv,berlusconi						
IT	Spectral	0.206	1.685	962 (69.41 %)	1386	0
ottobre,milano,google,blog,tv,roma,facebook,web,online,libro						
ES	Spectral	0.229	0.847	2178 (81.09 %)	2686	0
nuevo,mundo,internet,españa,otoño,madrid,nueva,feliz,contra,verano						
ES	METIS	0.221	0.102	2646 (98.51 %)	2686	0
españa,nuevo,iphone,madrid,mundo,octubre,cómo,blog,internet,google						
PT	Spectral	0.215	1.443	1341	1888	0
fotos,iphone,blog,poesia,google,mundo,só,portugal,anos,tv						
PT	METIS	0.206	0.150	1784	1878	0
fotos,amor,blog,google,concurso,mundo,iphone,download,natal,poesia						

Table 4.23: DE: Best two clusters by modularity of each algorithm incl. tags

Lang	Algorithm	Mod.	Cond.	# CCV	# blogs	level
DE	Blondel	0.061	0.242	119 (96.75 %)	123	2
teil,rezept,weinernte,kochen,event,euro,koch,penne,johann,stehen						
DE	METIS	0.059	0.249	112 (97.39 %)	115	3
teil,rezept,kochen,weinernte,euro,koch,penne,johann,stehen,riesling						
EN	METIS	0.219	0.091	2241 (99.73 %)	2247	1
day,love,happy,time,wednesday,birthday,review,blog,book,fun						
EN	Blondel	0.002	0.095	44 (97.78 %)	45	0
facebook,vehicle,airtight,series,game,coupon,day,southpeak						
FR	Blondel	0.160	0.152	721 (98.23 %)	734	2
chocolat,gâteau,tarte,pain,pommes,cake,pause,riz,moelleux,macarons						
FR	METIS	0.163	0.157	835 (98.47 %)	848	1
chocolat,gâteau,tarte,pain,pommes,cake,recette,petits,noix,crème						
IT	METIS	0.240	0.041	316 (91.59 %)	345	2
cinema,film,iphone,ottobre,trailer,parla,puntata,factor,tv,berlusconi						
IT	Blondel	0.022	0.116	44 (97.78 %)	45	0
roma,halloween,iphone,trailer,smartphone,hotel,milano,puntata,salone,genova						
ES	Blondel	0.079	0.084	448 (99.78 %)	449	2
boca,fútbol,españa,barça,gol,copa,river,actitud,liga,messi						
ES	METIS	0.221	0.102	2646 (98.51 %)	2686	0
españa,nuevo,iphone,madrid,mundo,octubre,cómo,blog,internet,google						
PT	Blondel	0.164	0.089	1173	1176	2
fotos,google,blog,download,concurso,amor,iphone,poesia,windows,twitter						
PT	METIS	0.206	0.150	1784	1878	0
fotos,amor,blog,google,concurso,mundo,iphone,download,natal,poesia						

Table 4.24: DE: Best two clusters by conductance by algorithm incl. tags

Chapter 5

Conclusion

5.1 Considerations

Before drawing conclusions from the evaluation results there are some things to be considered first:

The list of used algorithms and metrics used in this thesis is far from complete. There are many more algorithms and metrics, which could have been considered but we wanted to show a selection of algorithms of different algorithmic classes.

The goal of this thesis was to give an overview over the topic community identification and show practical use especially with the focus on the available blog dataset. That is why this thesis does not focus on the mathematical details as it is found in many other papers about this topic, which usually have a more specific focus on a single aspect.

5.2 Results

Referring back to the questions we posed in Chapter 1, we come to the following answers:

The algorithms were able to identify communities / clusters in the datasets available to us. The properties of the graph have an impact on the results of the algorithms.

We were able to use available metrics like modularity and conductance to quantify the quality of the identified clusters and we were also able to visualize this with the help of several kinds of plots like CLM-Plots and

NCP-Plots. It was also possible to compare some of the results against a pre-defined gold standard / ground truth.

We could demonstrate the practicability and impracticability of the algorithms under test. Scope and limitations of different algorithms could be demonstrated to a certain extend even with the limited datasets available.

Regarding the correlation between structural clusters and the content of the weblogs we could at least show some examples where this is true. We can also say that content and clusters are not correlated in every case, as there are too many reasons for why different blogs link to each other, but the content and topic of interest of certain blog authors is definitely one reason and we were able to show that in some cases.

5.3 Future Work

The whole area of social network analysis, community identification, clustering, data analysis and visualization is a very active topic in the scientific community, because of the availability of massive datasets and the success of social networks like Facebook and Twitter[1] these days. The amount of scientific papers, the number of different algorithms, metrics and their derivates is too large to be handled in a single thesis. Another fact to consider is the interaction between different disciplines like mathematics, statistics, physics, computer science, programming, software architecture, data mining and graphics. To do social network analysis all areas have to be touched and understood.

Thus there is lots of room for future work to be done here.

As the choice of the programming language and the implementation has a considerable impact on the runtime and thus the practical applicability of certain algorithms, future work could focus on more efficient implementations of the algorithms also in different programming languages.

Also related to the software development aspect is the creation of tools, which make it easy to analyze such data and to apply the algorithms in a more integrated and user-friendly way. Especially the visualization of the data will be a key-factor for the success of social network analysis in the future. Today there are hundreds of different tools for

[1] http://twitter.com

graph analysis, clustering and visualization but many of them are very scientific and need a lot of expert knowledge in order to be used which is due to the complexity of the whole topic. This expert-knowledge is not available in all places outside research-community and in order to be used in practice like companies who want to use the data, research also has to focus on ways how to improve the tools and make them available to the non-scientific users or to scientists of different disciplines like sociology.

The metrics used for quantifying the quality of the identified communities in this thesis were mainly modularity, conductance and number of correctly classified vertices. There are more metrics or variations of the metrics above. Future work could concentrate on those different metrics and evaluate the impact on the results of measuring community quality.

In Section 4.10.1 we have used different techniques to categorize the content of the weblogs. We have used keyword extraction based on TF-IDF and manual tagging. There are more efficient ways to extract keywords from websites, which e.g. handle synonyms etc. Also content clustering we have tried to apply in Section 4.11.1 has failed so future work could concentrate on identifying why it has failed and how it can be improved. This would make it possible to compare the results of the structural algorithms to a different content-based clustering technique to answer the questions of how much the content influences the structural clusters. Another interesting aspect is how information about structure and content can be combined in order to improve the identification of communities.

In this thesis we have always talked about identification of static communities. The next step would be to analyze how we can identify communities and track their evolution over a period of time. There is some work done in this area in [19].

Bibliography

[1] *A Linear-Time Heuristic for Improving Network Partitions*, 1982. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1585498.

[2] Amine Abou-rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs, 2005.

[3] Lada Adamic. Betweenness clustering guess demo. URL http://www-personal.umich.edu/~ladamic/GUESS/betweennessclust.html.

[4] A Arenas, J Duch, A Fernández, and S Gómez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176, 2007. URL http://stacks.iop.org/1367-2630/9/i=6/a=176.

[5] David Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. pages 124–137. 2007. doi: 10.1007/978-3-540-77004-6_10. URL http://dx.doi.org/10.1007/978-3-540-77004-6_10.

[6] Vincent D Blondel, Jean loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks, 2008. URL http://sites.google.com/site/findcommunities/.

[7] U. Brandes. A faster algorithm for betweenness centrality, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.2024.

[8] Deepayan Chakrabarti, Christos Faloutsos, and Yiping Zhan. Visualization of large networks with min-cut plots, a-plots and r-mat ,.

[9] Tony F. Chan, Tony Chan Ciarlet, and W. K. Szeto. On the optimality of the median cut spectral bisection graph partitioning method. *SIAM Journal on Scientific Computing*, 18:943–948, 1997.

[10] Cédric Chevalier and Ilya Safro. Comparison of coarsening schemes for multilevel graph partitioning. pages 191–205, 2009. doi: http://dx.doi.org/10.1007/978-3-642-11169-3_14.

[11] Aaron Clauset, Cristopher Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453 (7191):98–101, May 2008. ISSN 0028-0836. doi: 10.1038/nature06830. URL http://dx.doi.org/10.1038/nature06830.

[12] Pejus Das and Dr. Mathew Beal. Techniques for spectral clustering, 2006. URL \url{http://www.pejusdas.com/documents/spectral.pdf}.

[13] Christos Faloutsos. Talk 3: Graph Mining Tools – Tensors, communities,parallelism, 2010.

[14] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010. ISSN 0370-1573. doi: DOI:10.1016/j.physrep.2009.11.002. URL http://www.sciencedirect.com/science/article/B6TVP-4XPYXF1-1/2/99061fac6435db4343b2374d26e64ac1.

[15] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. 2008. URL http://www.siam.org/proceedings/alenex/2008/alenex08.php.

[16] M. Girvan and M. 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, June 2002. ISSN 0027-8424. doi: 10.1073/pnas.122653799. URL http://dx.doi.org/10.1073/pnas.122653799.

[17] David Gleich. Hierarchical directed spectral graph partitioning, 2006.

[18] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. The Johns Hopkins University Press, 3rd edition, October 1996. ISBN 0801854148. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0801854148.

[19] D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In *Proc. International Conference on Advances in Social Networks Analysis and Mining (ASONAM'10)*, 2010.

[20] Steve Gregory. An Algorithm to Find Overlapping Community Structure in Networks. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, volume 4702 of *Lecture Notes in Computer Science*, chapter 12, pages 91–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74975-2. doi: 10.1007/978-3-540-74976-9_12. URL `http://dx.doi.org/10.1007/978-3-540-74976-9_12`.

[21] Fredrik Hildorsson. Scalable solutions for social network analysis, 2009.

[22] Sepandar D. Kamvar, Dan Klein, and Christopher D. Manning. Interpreting and extending classical agglomerative clustering algorithms using a model-based approach. Technical Report 2002-11, Stanford InfoLab, February 2002. URL `http://ilpubs.stanford.edu:8090/529/`.

[23] GEORGE KARYPIS and VIPIN KUMAR. Analysis of multilevel graph partitioning, 1995.

[24] GEORGE KARYPIS and VIPIN KUMAR. Multilevel graph partitioning schemes, 1995.

[25] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[26] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80(1):016118, Jul 2009. doi: 10.1103/PhysRevE.80.016118.

[27] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11):118703+, Mar 2008. doi: 10.1103/PhysRevLett.100.118703. URL `http://dx.doi.org/10.1103/PhysRevLett.100.118703`.

[28] 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, 2008. URL `http://arxiv.org/abs/0810.1355`. cite arxiv:0810.1355 Comment: 66 pages, a much expanded version of our WWW 2008 paper.

[29] Marina Meila and William Pentney. Clustering by weighted cuts in directed graphs. In *SDM*, 2007.

[30] B. Nadler and M. Galun. Fundamental limitations of spectral clustering. In *Advanced in Neural Information Processing Systems*, volume 19, pages 1017–1024, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.1567.

[31] M. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, January 2005. ISSN 03788733. doi: 10.1016/j.socnet.2004.11.009. URL http://dx.doi.org/10.1016/j.socnet.2004.11.009.

[32] M. E. J. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical Review E*, 64(1):016132+, Jun 2001. doi: 10.1103/PhysRevE.64.016132. URL http://dx.doi.org/10.1103/PhysRevE.64.016132.

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

[34] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113+, Feb 2004. doi: 10.1103/PhysRevE.69.026113. URL http://dx.doi.org/10.1103/PhysRevE.69.026113.

[35] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.8100.

[36] Northeastern University NWB Team Inidiana University and University of Michigan. Network workbench tool, 2006. URL http://nwb.slis.indiana.edu. [Documentation and wiki pages of the NWB - Network Workbench Tool].

[37] Darko Obradovic and Stephan Baumann. A journey to the core of the blogosphere. *Social Network Analysis and Mining, International Conference on Advances in*, 0:1–6, 2009. doi: http://doi.ieeecomputersociety.org/10.1109/ASONAM.2009.29.

[38] J.W. &. Westhead D.R Pinney. Betweenness–based decomposition methods for social and biological networks. In P.D. Baxter K.V.Mardia &. R.E. Walls (Eds.) S. Barber, editor, *Interdisciplinary Statistics and Bioinformatics: Proceedings*. 2006.

[39] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *J. of Graph Alg. and App. bf*, 10:284–293, 2004.

[40] Matthew J. Rattigan, Marc Maier, and David Jensen. Graph clustering with network structure indices. Technical report, 2007.

[41] John Scott. *Social Network Analysis: A Handbook*. Sage Publications, second. edition, 2000. ISBN 0761963391. URL `http://www.amazon.com/Social-Network-Analysis-Professor-Scott/dp/0761963383/ref=sr_1_1?ie=UTF8&s=books&qid=1256622319&sr=1-1`.

[42] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 731, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7822-4.

[43] U. von Luxburg. A tutorial on spectral clustering. Technical Report 149, Max Planck Institute for Biological Cybernetics, 2006.

[44] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks. *CoRR*, abs/cs/0702048, 2007.

[45] Wikipedia. Rand index — wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/w/index.php?title=Rand_index&oldid=401883673`. [Online; accessed 12-December-2010].

[46] Wikipedia. Tf–idf — wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=396377491`. [Online; accessed 13-December-2010].

Acknowledgements

First of all I want to thank Prof. Dr. Andreas Dengel for accepting this thesis and my mentor Darko Obradovic for the very competent and friendly support throughout the duration of this thesis.

Next, I would like to thank Sirko Schneppe, Sascha Sauer and Christof Sander of Ageto GmbH in Jena for their support of my studies during the last two years.

I would also like to thank my family, friends and all persons who were supporting me during this thesis with their patience, knowledge and especially support in proofreading. These are, in no particular order: Samuel Jackisch, Martin Junghanns, Frank Lanitz, Dr. Stephan Baumann, Thasso Griebel, Wolfgang Schlauch, Elisabeth Rueger, Patrick Rueger, Ingrid Rueger and Dajana Dobras.