Karlsruhe Institute of Technology

Complete Hierarchical Cut-Clustering: An Analysis of Guarantee and Quality

Bachelor Thesis of

Michael Hamann

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer:	Prof. Dr. Dorothea Wagner
Second reviewer:	Prof. Dr. Peter Sanders
Advisor:	Dipl.-Inform./Dipl.-Math. Tanja Hartmann

Duration: 01. August 2011 – 11. November 2011

Danksagung

Ich möchte mich bei Prof. Wagner dafür bedanken, dass ich an ihrem Lehrstuhl meine Bachelorarbeit schreiben durfte. Außerdem möchte ich meiner Betreuerin Tanja Hartmann ganz herzlich für die hervorragende Betreuung dieser Arbeit danken. Von der ursprünglichen Definition des Themas bis hin zu immer wieder neuen Ideen und Wendungen nahm sie sich stets Zeit für ausführliche Besprechungen um auf meine Vorschläge einzugehen. Durch ihre hilfreichen Hinweise und Korrekturen habe ich während meiner Bachelorarbeit viel gelernt.

Selbständigkeitserklärung

Hiermit versichere ich, dass die vorliegende Arbeit sowie der verwendete Quelltext meine eigene Arbeit sind. Arbeit anderer wurde durch Quellenverweise oder Nennung des Autors (Quelltext) eindeutig kenntlich gemacht.

Statement of Authorship

I hereby declare that this document and the used code have been composed by myself and describe my own work, unless otherwise acknowledged in the text.

Karlsruhe, 11. November 2011

Abstract

There are many algorithms for dividing a graph into parts, so-called clusters. An essential question is how dense these clusters are. This can be measured by the intra-cluster expansion. The cut-clustering algorithm as presented by Flake et al. [FTT04] provides a theoretical guarantee on the intra-cluster expansion, which for example greedy clustering approaches can not give, as calculating the intra-cluster expansion of a cluster is NP-hard. This guarantee depends on a parameter value. A sequence of parameter values yields a clustering hierarchy.

In the first part of this work we will present two algorithms for finding different clusterings. In particular the second approach, which we have developed, does guarantee that all possible clusterings are found and that the intervals of the parameter values for which a certain clustering is returned by the algorithm are exact. This is possible with not more than twice as many executions of the original cut-clustering algorithm as there are different clusterings in the hierarchy.

In the second part of this work we examine the hierarchies that are defined by these clusterings and also compare the cut clusterings to clusterings calculated by a greedy algorithm based on modularity, a popular measure for the quality of clusterings. For most of the graphs in our test set of 304 graphs, the guarantee that the cut-clustering algorithm gives was better than a trivial lower bound of the intra-cluster expansion. We show that there is a tendency that the clusterings of the cut-clustering algorithm have a higher intra-cluster expansion and that these clusters are not, like the clusters of the modularity algorithm, of almost equal size but do have very different sizes. Some of the cut clusterings do still have a modularity value that almost reaches the modularity value of the modularity clusterings.

Deutsche Zusammenfassung

Der Cut-Cluster-Algorithmus ist ein von Flake et al. [FTT04] vorgestellter Algorithmus zur Clusterung von gewichteten, ungerichteten Graphen, d.h. zur Aufteilung von Graphen in knoteninduzierte Teilgraphen, die im Vergleich zu den Kanten zwischen den Clustern möglichst dicht sind. Eine Möglichkeit, diese Dichte zu messen, ist die sogenannte Intra-Cluster Expansion, die definiert ist durch das Minimum der Gewichte aller Schnitte in dem jeweiligen Subgraphen geteilt durch die kleinere Schnittseite. Da es NP-schwer ist, die Intra-Cluster Expansion eines Clusters zu berechnen, ist der Cut-Cluster-Algorithmus besonders interessant: Er gibt durch einen Parameter-Wert eine untere Schranke für die Intra-Cluster Expansion der zu berechnenden Clusterung an. Die Clusterungen für verschiedene Parameterwerte sind ineinander geschachtelt und bilden deshalb eine Hierarchie.

In dieser Arbeit werden im ersten Teil zwei Algorithmen vorgestellt, die es ermöglichen, Intervalle von Parameterwerten zu ermitteln, für die der Cut-Cluster-Algorithmus jeweils eine andere Clusterung berechnet. Das erste vorgestellte Verfahren basiert auf binärer Suche, der zweite, in dieser Arbeit neu entwickelte Ansatz nutzt Erkenntnisse aus einem parametrischen maximalen s-t-Fluss-Algorithmus und kann garantieren, dass alle möglichen Clusterungen mit exakten Parameter-Intervallen gefunden werden. Hierfür müssen lediglich maximal doppelt so viele Clusterungen berechnet werden, wie tatsächlich unterschiedliche Clusterungen existieren.

In einem zweiten Teil wird untersucht, wie gut die Garantie, die der Cut-Cluster-Algorithmus liefert, in der Praxis verglichen mit einer trivialen unteren Schranke für die Intra-Cluster Expansion abschneidet. Für die untersuchten 304 Graphen wurde dabei festgestellt, dass die Garantie in den meisten Fällen deutlich besser ist als die triviale untere Schranke.

Ein weiteres, weit verbreitetes Maß für die Güte einer Clusterung ist Modularity. In dieser Arbeit wurde der Cut-Cluster-Algorithmus mit einem Algorithmus verglichen, der Modularity lokal optimiert. Die Clusterungen des Modularity-Algorithmus hatten dabei stets bessere Modularity-Werte als die des Cut-Cluster-Algorithmus, für einige Graphen kamen die Modularity-Werte des Cut-Cluster-Algorithmus allerdings nahe an die des Modularity-Algorithmus heran, obwohl der Cut-Cluster-Algorithmus nicht versucht, Modularity zu optimieren. In 32 Fällen hat der Cut-Cluster-Algorithmus eine Clusterung berechnet, die eine höhere Intra-Cluster Expansion hatte als die Clusterung des Modularity-Algorithmus. Für über 90% der Clusterungen des Cut-Cluster-Algorithmus kann außerdem eine höhere untere Schranke garantiert werden als für die entsprechenden Clusterungen des Modularity-Algorithmus.

Beim Vergleich der Clusterungen der verschiedenen Algorithmen wurde außerdem festgestellt, dass die Clusterungen des Cut-Cluster-Algorithmus Cluster sehr unterschiedlicher Größe aufweisen, darunter auch viele einzelne Knoten, während der Modularity-Algorithmus eher gleich große Cluster berechnet. Interessanterweise hat sich gezeigt, dass durch die Vereinigung von Clustern des Cut-Cluster-Algorithmus eine Clusterung berechnet werden kann, die eine Modularity-aufweist, die in vielen Fällen der des Modularity-Algorithmus sehr nahe kommt und in wenigen Fällen diese sogar übertrifft. Dies könnte bedeuten, dass Clusterungen des Cut-Cluster-Algorithmus sich gut als Basis für andere Cluster-Algorithmen eignen.

Contents

1 Introduction **1**
- 1.1 Preliminaries . . . 2
  - 1.1.1 Clusterings and Cuts . . . 2
  - 1.1.2 Quality Measures and Other Indices . . . 3

2 The Hierarchical Cut-Clustering Algorithm **5**
- 2.1 The Algorithm Step-by-Step . . . 6
- 2.2 Finding all Different Hierarchy Levels . . . 7
  - 2.2.1 Binary Search . . . 8
  - 2.2.2 Intersection Points and Clustering Levels . . . 9

3 Implementation **15**
- 3.1 File Formats . . . 16
- 3.2 Structure of the Calculation Process . . . 17
- 3.3 Handling Edge Weights and Parameter Values . . . 17

4 Experiments **19**
- 4.1 Test Instances . . . 19
- 4.2 Running Time . . . 20
- 4.3 Theoretical Guarantee vs Practical Results . . . 22
  - 4.3.1 Analysis of Individual Cluster Hierarchies . . . 22
  - 4.3.2 Comparative Analysis of all Graphs . . . 23
- 4.4 Comparison with Modularity-Based Clusterings . . . 23
  - 4.4.1 Overview of Cut Clusterings and Modularity Clusterings . . . 27
  - 4.4.2 Intra-Cluster Expansion of Modularity Clusterings . . . 30

5 Conclusion **33**

Bibliography **35**

1. Introduction

Dividing a graph into different parts, so-called clusters, is a good way to understand the structure of a graph. The intuition says that we want to have many edges inside these clusters and few between them. A simple solution for this problem are the connected components of a graph. However, this trivial decomposition does not provide any interesting structural information. So we need to formulate the problem differently: In a connected graph we want to find significantly dense subgraphs. The density of such subgraphs is the more significant, the sparser they are linked between each other. In some graphs such a structure is quite obvious, there are clusters you look at and think that they are good, for other graphs there just exists no clustering you could call "good" from looking at it.

There are many algorithms for clustering graphs, some work on a local level and try to greedily analyze the neighbourhood of a node, others work on a global level and try to take into account the whole graph at once. The cut-clustering algorithm, on which we will focus in this work, works on a global level and in contrast to many other clustering algorithms it also provides a theoretical guarantee on the quality of the result.

Given an undirected graph the cut-clustering algorithm introduced by Flake et al. [FTT04] uses minimum s-t-cuts in order to construct a clustering. The result of the algorithm is influenced by a parameter α and the resulting clustering fulfills a quality guarantee depending on this parameter value. On the one hand this quality guarantee limits the strength of the connections between the clusters, on the other hand it gives a minimum of the intra-cluster expansion, which describes the degree of connectivity inside a cluster. The latter is especially interesting as the intra-cluster expansion is NP-hard to compute but a nice measure for the quality of a cluster, since it says that cutting off a part of the cluster is quite expensive in proportion to the size of this part.

This theoretical guarantee of the quality is an advantage of the cut-clustering algorithm compared to other clustering algorithms. We will try to see how good this theoretical guarantee is in practice.

Flake et al. further showed that a sequence of parameter-values gives a hierarchy of clusterings. We will present two approaches for calculating a clustering hierarchy, a first approach that is based on binary search but can not guarantee that all possible clusterings in the hierarchy are found, and in contrast, a second approach that allows for the exact calculation of the whole hierarchy with exact boundaries of the intervals in the parameter range which correspond to the different levels. This second approach is also faster.

Regarding the whole hierarchy gives us the possibility to see the whole spectrum of clusterings the algorithm can calculate for a given graph.

The aim of this work is to analyze the behavior of the hierarchical cut-clustering algorithm in practice. We have implemented the algorithm and tested it on different instances. We will describe our observations and try to explain at least a part of them also from the theoretical side. In addition to that we have applied an algorithm to our data set that tries to optimize modularity, a popular quality measure for clusterings and compared the results of this algorithm to the results of the cut-clustering algorithm.

After the preliminaries in the following section we present the cut-clustering algorithm and the different approaches for calculating cluster hierarchies in Section 2. In Section 3 we give an overview of the implementation we used for the experiments that are described in Section 4. We end this work with a conclusion in Section 5.

1.1 Preliminaries

Unless denoted otherwise, a graph $G = (V, E, \omega)$ is an undirected, simple graph with a non-empty set of nodes V and a non-empty set of edges E and an edge-weight function $\omega : E \to \mathbb{R}_0^+$. If not differently defined $n := |V|$ and $m := |E|$.

1.1.1 Clusterings and Cuts

Definition 1.1. *A* clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ *of a graph* $G = (V, E)$ *is a partition of* V *into non-empty clusters* $C_1, C_2, \ldots, C_k$ *with* $\cup_{C \in \mathcal{C}} C = V$.

We call a cluster trivial if it is either a single node (a singleton) or a connected component of the graph. A clustering is trivial if it contains only singletons or only connected components.

In our analysis we will consider edge weights not only on single edges but also on the adjacencies of a node, inside clusters and with respect to cuts. In order to avoid long and unreadable expressions we will therefore introduce certain shorter notations for these terms.

First of all, we define the neighbourhood of a node $v \in V$ as $N(v) := \{u \in V \mid \{u, v\} \in E\}$. Then the weighted degree of a node $v \in V$ is defined as

$$\omega(v) := \sum_{u \in N(v)} \omega(\{u, v\})$$

For a cluster $C \in \mathcal{C}$ let $\text{in}(C) := \{\{u, v\} \in E \mid u, v \in C\}$ denote the set of edges inside C. We define the weight of a cluster $C \in \mathcal{C}$ as the weight of all intra-cluster edges:

$$\omega(C) := \sum_{e \in \text{in}(C)} \omega(e)$$

We define the weight of a clustering $\mathcal{C}$ as the sum of the weights of all clusters inside the clustering:

$$\omega(\mathcal{C}) := \sum_{C \in \mathcal{C}} \omega(C)$$

Regarding the whole graph G as a single cluster i.e. all edges are considered as intra-cluster edges, we define the weight analogous to to the weight of a cluster:

$$\omega(G) := \sum_{e \in E} \omega(e)$$

Definition 1.2. *A* cut *in a vertex-induced subgraph $V' \subseteq V$ with $|V'| \geq 2$ is defined as $(S, V' \setminus S)$ with $\emptyset \neq S \subsetneq V'$. The* cut edges *are the edges between S and $V' \setminus S$:*

$$\mathrm{E}(S, V' \setminus S) := \big\{\{u, v\} \in E \,|\, u \in S, v \in V' \setminus S \text{ or } v \in S, u \in V' \setminus S\big\}$$

A cut in the graph G is a cut with $V' = V$.

A cluster C induces the cut $(C, V \setminus C)$ in G.

The weight of a cut is defined as

$$\omega(S, V' \setminus S) := \sum_{e \in \mathrm{E}(S, V' \setminus S)} \omega(e)$$

1.1.2 Quality Measures and Other Indices

In order to analyze the characteristics of clusterings we will introduce quality measures of clusterings and some further indices.

A widely used quality measure for clusterings that is close to human intuition of cluster quality is *modularity*, introduced by Newman and Girvan [NG04]:

Definition 1.3. *Let $\mathcal{C}$ denote a clustering of an undirected graph $G = (V, E, \omega)$. Then using the definitions of the previous section the* modularity *of the clustering $\mathcal{C}$ is defined as (see also [Gö10], Section 1.2.2):*

$$\mathrm{mod}_{\omega}(\mathcal{C}) := \frac{\omega(\mathcal{C})}{\omega(G)} - \frac{1}{4 \cdot \omega(G)^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \omega(v) \right)^2$$

The first term in Definition 1.3 is the fraction of edges covered by clusters, the so-called coverage, the second term is the expected value of the coverage of the clustering, i.e. the coverage of the same clusters with randomly distributed edges. This means modularity measures how much the distribution of the edges differs from randomness and thus how good the clustering matches the distribution of the edge weights. The value can be positive or negative, the range of modularity is $[-0.5, 1]$ (see also [Gö10], Lemma 2.2.1). The maximum is reached for unconnected cliques, the minimum for a complete bipartite graph clustered by the two sides.

Another possibility to measure the quality of a clustering is to look at the cuts inside clusters and the cuts that separate a cluster from the rest of the graph. The fundamental measure we will use for this purpose is *expansion*, which is the weight of the cut in relation to the size of the cut sides:

Definition 1.4. *The* expansion *of a cut $(S, V \setminus S)$ in G is:*

$$\psi(S) = \frac{\omega(S, V \setminus S)}{\min\{|S|, |V \setminus S|\}}$$

We define the *intra-cluster expansion* of a cluster $C \in \mathcal{C}$ as the minimum expansion of all cuts in the subgraph induced by the nodes of the cluster:

$$\text{intraExp}(C) := \min \left\{ \frac{\omega(S, C \setminus S)}{\min\{|S|, |C \setminus S|\}} \middle| S \subsetneq C, S \neq \emptyset \right\}$$

As this definition does not involve singletons we define the intra-cluster expansion of a singleton as $+\infty$.

The intra-cluster expansion thus gives a guarantee on the minimum weight of a cut in a cluster in proportion to the size of the parts of the cluster that are separated by the cut. The intra-cluster expansion of a clustering $\mathcal{C}$ is the minimum intra-cluster expansion of all clusters $C \in \mathcal{C}$: $\text{intraExp}(\mathcal{C}) := \min\{\text{intraExp}(C) \mid C \in \mathcal{C}\}$.

Even for one cluster C the intra-cluster expansion is NP-hard to compute. However, it is possible to calculate a trivial lower and upper bound. Calculating a cut of minimum weight in C, which is possible in polynomial time, and dividing the cut weight by the half of the cluster size $|C|/2$ rounded down yields a lower bound, while dividing by the size of the smaller cut side yields an upper bound. As there is no cheaper cut than the minimum cut and no minimum cut side is larger than the half of the cluster size, the lower bound holds. As an upper bound the expansion of any existing cut in C can be used.

The *inter-cluster expansion* $\text{interExp}(C) := \psi(C)$ of a cluster C is the expansion of the cut induced by C. The inter-cluster expansion of a clustering $\mathcal{C}$ is the maximum expansion of all clusters $C \in \mathcal{C}$: $\text{interExp}(\mathcal{C}) := \max\{\text{interExp}(C) \mid C \in \mathcal{C}\}$. Given a clustering the inter-cluster expansion can be easily calculated in linear time.

A simplified form of the inter-cluster expansion is the *inter-cluster expansion** of a cluster C which is defined as $\text{interExp}^*(C) := \omega(C, V \setminus C)/|V \setminus C|$. Like the inter-cluster expansion, inter-cluster expansion* can also be defined for a clustering as the maximum of the inter-cluster expansion* of all clusters. For a cluster C with $|C| \geq |V|/2$ it holds that $\text{interExp}(C) = \text{interExp}^*(C)$.

A good clustering should have a high intra-cluster expansion and a low inter-cluster expansion in order to guarantee that the nodes inside the clusters are well connected while the clusters are less connected. While the intra-cluster expansion is best for singletons the inter-cluster expansion is best for connected components as clusters. This means that inter- and intra-cluster expansion are conflicting measures and an optimal clustering should provide a compromise between them. However directly optimizing these properties does not work, also not optimizing them in a greedy algorithm, since intra-cluster expansion can not be calculated efficiently and the trivial lower bound is not precise enough. There is no guarantee how good this lower bound is. In fact, it can become arbitrarily bad in large clusters with very unbalanced minimum cuts.

While the modularity of a clustering is not influenced by scaling the edge weights, all expansion values scale linearly with the edge weights. In this work we will often scale edge weights such that the maximum edge weight is one which is consistent to unweighted graphs where we assume an edge weight of one for each existing edge. It is still difficult to compare weighted and unweighted graphs as for weighted graphs not only the structure of the graph but also the distribution of the edge weights decides on the quality of a clustering. However, the scaling of the maximum edge weight to one gives at least a limit of the maximum weight, and thus, a guarantee that the weight of a cut will never exceed the number of edges in the cut. This is, the expansion of a cut in a weighted graph will never be larger than the expansion of the same cut in the same graph without edge weights.

2. The Hierarchical Cut-Clustering Algorithm

In this work we analyze the behavior of the cut-clustering algorithm as introduced by Flake et al. [FTT04]. Given an undirected, weighted graph $G = (V, E, \omega)$ and a parameter $\alpha \in \mathbb{R}_0^+$ it calculates a clustering $\mathcal{C}_\alpha$ of the graph G.

The algorithm is interesting because any clustering $\mathcal{C}$ that is calculated by the algorithm holds a quality guarantee in terms of expansion. This guarantee is defined by the following inequality for any $C \in \mathcal{C}$ (see also [FTT04], Theorem 3.3):

$$\text{interExp}^*(C) \leq \alpha \leq \text{intraExp}(C) \tag{2.1}$$

This means that α acts as an upper bound for the inter-cluster expansion* as well as a lower bound for the intra-cluster expansion of the clustering. One question we will answer in this work is how this lower bound for the intra-cluster expansion behaves compared to the trivial lower intra-cluster expansion bound.

We will call clusterings that are calculated using the cut-clustering algorithm *cut clusterings*. Cut clusterings regarding the same graph but different parameter values are nested, i.e., can be sorted such that for two consecutive clusterings it holds that the clusters of the previous clustering result from merging clusters of the latter one. Lower parameter values yield larger clusters, higher parameter values smaller clusters. Given an decreasing sequence of parameter values $\alpha_1 > \alpha_2 > \cdots > \alpha_k$ we thus get a hierarchy of clusterings $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$. In Figure 2.1 we have depicted an example for such a hierarchy of five clusterings of a graph consisting of eight nodes.

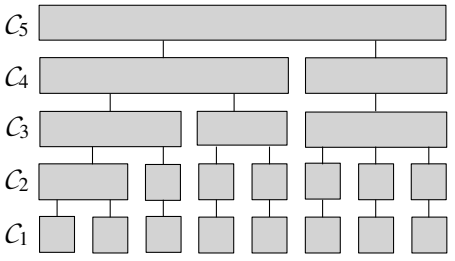


Figure 2.1: Example for a simple clustering hierarchy.

In the following we will use the term *clustering hierarchy* for the tree that is induced by such a hierarchy of clusterings where the clusters correspond to nodes and a node of level $i + 1$ is connected to a node of level i if the corresponding clusters are nested. If not existing yet we add an artificial clustering level as a root of the hierarchy that consists of exactly one cluster containing all nodes of the graph. Following this metaphor of a rooted

tree we call the levels of larger parameter values and smaller clusters *lower* levels and the levels of smaller parameter values and larger clusters *higher* levels.

The algorithm for constructing a single cut clustering, i.e., a single hierarchy level is based on minimum s-t-cuts, or more precisely, s-t community-cuts which are special maximum s-t-flows and can be calculated using the preflow-push maximum-flow algorithm introduced by Goldberg and Tarjan [GT88].

Definition 2.1. *The s-t* community-cut *is the cut of all minimum s-t-cuts for which the cut-side that contains s is smallest. The* community *of a node $s \in V$ with respect to a node $t \in V$ is the side of the s-t community-cut that contains s. We call the node s the representative of its community with respect to t.*

The preflow-push maximum-flow algorithm has a running time of $O(n^3)$. It was optimized, for example by Cheriyan and Mehlhorn who presented a variant of the algorithm with a running time of $O(n^2\sqrt{n})$ [CM99]. This is also the running time of the implementation we have used in our experiments.

We will now present the cut-clustering algorithm for calculating a single cut clustering with respect to a given parameter value and a graph $G = (V, E, \omega)$. In the hierarchical cut-clustering algorithm we will use this cut-clustering algorithm for calculating the individual levels of the hierarchy.

2.1 The Algorithm Step-by-Step

The cut-clustering algorithm as introduced by Flake et al. is quite simple.

For constructing a cut clustering of a graph G, the input graph G is augmented to a graph $G_\alpha = (V_\alpha, E_\alpha, \omega_\alpha)$ by an artificial node t that is connected to each node in G with an edge weighted by α (line 1, Algorithm 1).

Algorithm 1: Cut-Clustering Algorithm

Input: Graph $G = (V, E, \omega)$, $\alpha \in \mathbb{R}_0^+$
Output: Cut clustering $\mathcal{C}$

1 $G_\alpha = (V_\alpha, E_\alpha, \omega_\alpha) \leftarrow (V \cup \{t\}, E \cup (V \times t), \omega_\alpha)$,
$\omega_\alpha : E_\alpha \to \mathbb{R}_0^+, e \mapsto \begin{cases} \omega(e), & e \in E \\ \alpha, & \text{otherwise} \end{cases}$;
2 $V_s \leftarrow V$ sorted by $\omega(v)$ in decreasing order;
3 $\mathcal{C}, R \leftarrow \{\}$;
4 **foreach** $v \in V_s$ **do**
5 **if** $v \notin \cup_{C \in \mathcal{C}} C$ **then**
6 $U \leftarrow$ community of v with respect to t in G_α;
7 **foreach** $r \in R$ **do**
8 **if** $r \in U$ **then**
9 $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C \in \mathcal{C} \mid r \in C\}$;
10 $R \leftarrow R \setminus \{r\}$;
11 $\mathcal{C} \leftarrow \mathcal{C} \cup \{U\}$;
12 $R \leftarrow R \cup \{v\}$;

The algorithm computes the clusters iteratively by considering each node as potential representative of a community in G_α with respect to t (line 6). This is, the clustering

is built by calculating communities of the nodes in V. In the pseudocode the current temporary clustering is denoted by $\mathcal{C}$, the set of representatives is R. Flake et al. pointed out (see Lemma 3.9, [FTT04]) that for two nodes u, v the communities with respect to t are either disjoint or subset of each other. This means that whenever we calculate a community U of a node v with respect to t we can simply check for all other, already calculated representatives if they are inside the new community U (line 8). If this is the case, we delete these representatives (line 10) and their clusters (line 9) from the temporary clustering $\mathcal{C}$.

Note that all nodes in a community U have communities inside this community, otherwise, if there was a node $w \in U$ with a community $W \supsetneq U$ the cut of W would be cheaper than U and thus U would not be a community. We can thus skip any node that is already contained in a community U (line 5).

Finally, the inclusion-maximal communities form the clusters.

Flake et al. state that sorting the nodes by the weighted node degree (line 2) reduces the number of calculated communities in practice to almost the number of clusters.

2.2 Finding all Different Hierarchy Levels

So far we have assumed a given sequence of parameter values for constructing a clustering hierarchy. However, constructing such a sequence of parameter values is not trivial.

In this section whenever we consider a clustering hierarchy we assume that no two levels are the same unless differently noted. If there exists no further parameter value for which the cut-clustering algorithm calculates a clustering that is not contained in the hierarchy yet, we call the hierarchy a *complete clustering hierarchy.*

In a complete clustering hierarchy we can select a suitable clustering out of all cut clusterings. In a complete hierarchy the parameter values for which the different clusterings are valid decompose $\mathbb{R}_0^+$ into intervals, each assigned to one level respectively clustering. This is, if we knew the interval boundaries of the parameter values, instead of any value, for which a certain clustering is returned by the cut-clustering algorithm, we would know a possibly much better upper bound for the inter-cluster expansion* and also a possibly much better lower bound for the intra-cluster expansion.

Note that for $\alpha = 0$ the cut-clustering algorithm returns all connectivity components as clusters. On the other hand, if α is greater or equal than the largest edge weight in G, the cut-clustering algorithm returns only singletons. This is, a complete hierarchy is bounded by trivial clusterings. All non-trivial clusterings result from parameter values between these extremes.

The first question we want to address is one that has already been answered by Flake et al.: How many hierarchy levels exist at maximum? The answer is that there are at most n different levels. As in the lowest hierarchy level there are (at most) n clusters and in each level in the hierarchy at least two clusters are merged, the number of clusters is reduced by one in each level.

There also exists a family of graphs that has exactly n levels in the clustering hierarchy. Stars with one node in the center and an arbitrary number of nodes around, connected to the center by edges of different weights. Then from the lower to the higher levels, in each level one further node is merged into the cluster of the center. The node connected by the cheapest edge is added last. As the leaves are not connected among each other they remain singletons till they are merged.

The second question we want to address is the question how we can find the interval boundaries in the parameter range, i.e., all different clusterings in the hierarchy. Flake et al. suggest to use binary search for finding all different hierarchy levels and they also mention that it is possible to use the parametric maximum-flow algorithm introduced by Gallo et al. [GGT89] for finding all different communities of a node when the parameter value changes. Based on the idea of binary search we have first developed a simple algorithm for calculating a clustering hierarchy. However with this algorithm we can not guarantee that all different hierarchy levels are found and do not get exact interval boundaries. We then had a closer look at the parametric maximum flow algorithm. We used the ideas presented there in order to construct a much better algorithm that is faster than binary search, both in theory and practice, and guarantees that the calculated hierarchy is complete and the boundaries are exact. Not that we do not use the parametric maximum-flow algorithm itself.

We will first present the approach based on binary search and then our new algorithm. Both approaches start from an unweighted, undirected graph $G = (V, E, \omega)$. The result is a list that stores the clusterings together with the corresponding lower boundary of the parameter interval as calculated by the respective approach.

2.2.1 Binary Search

In order to apply binary search we need to convert the continuous range of parameter values into discrete values. This is also the major disadvantage of this approach as this means we possibly miss certain parameter values that are fractions of these discrete values.

Algorithm 2: binSearchRecursion

Input: Graph G, `int` α_u, Clustering $\mathcal{C}_u$, `int` α_l, Clustering $\mathcal{C}_l$, map hierarchy

```
1  while α_u < α_l − 1 do
2      α_m ← (α_u + α_l)/2;
3      C_m ← Cut-Clustering(G, α_m);
4      if |C_u| == |C_m| then
5          α_u ← α_m;
6      else if |C_m| == |C_l| then
7          α_l ← α_m;
8      else
9          binSearchRecursion(G, α_u, C_u, α_m, C_m, hierarchy);
10         C_u ← C_m;
11         α_u ← α_m;
12 hierarchy[α_l] ← C_l;
```

The main component of this approach is a recursive function named `binSearchRecursion` (see Algorithm 2). This function takes the graph G and a pair of an upper and lower clustering ($\mathcal{C}_u$ and $\mathcal{C}_l$) with the respective parameter values α_u and α_l as input parameters. For storing the clusterings, an initially empty map from parameter values to clusterings is passed to the function, too.

The `binSearchRecursion` function uses a binary search in a while loop in order to find the lower boundary of each interval with respect to the discretization. The binary search calculates the parameter value α_m between α_u and α_l. If the cut clustering $\mathcal{C}_m$ for α_m is equal to $\mathcal{C}_l$ or $\mathcal{C}_u$, the respective parameter value α_l or α_u is set to α_m and the loop continues the binary search.

If $\mathcal{C}_m$ is different from $\mathcal{C}_l$ and $\mathcal{C}_u$, then the previous interval is split. For the new clustering $\mathcal{C}_m$ a new call to the function is made with $\mathcal{C}_m$ as lower clustering and the old $\mathcal{C}_u$ as upper

clustering. Furthermore, the while loop continues the search for the lower boundary for the interval of $\mathcal{C}_l$, now using $\mathcal{C}_m$ as upper clustering and the parameter value α_m as α_u. Note that the lower boundary is the boundary to the next higher level.

Initially, Algorithm 2 is called with the lower and upper trivial clustering and the extremes of the parameter values as described before.

As in each iteration of the while loop the interval is cut in halves and in each iteration one clustering is calculated, the number of clusterings calculated in one recursive call is $\log_2(\alpha_l - \alpha_u)$. Each new clustering in the hierarchy causes only one recursive call. Let a denote the number of discrete steps in the parameter range, r the running time of the cut-clustering algorithm and h the number of levels in the resulting clustering hierarchy, which is at most n. Then the running time of the binary-search algorithm is $O(h \cdot \log(a) \cdot r)$.

As an optimization of this algorithm we have also added contractions as already suggested by Flake et al. Instead of calculating a clustering based on the original graph we contract the clusters of the next lower level we already know and use the resulting graph. This is, we contract the clustering $\mathcal{C}_l$ at the beginning of each recursive call and calculate all clusterings in that call based on this contraction.

Due to the nesting property of the clusters on different hierarchy levels, using contractions does not change the resulting clustering hierarchy.

2.2.2 Intersection Points and Clustering Levels

A fast parametric maximum flow algorithm including an algorithm for finding all communities of a node was published by Gallo et al. [GGT89]. The way the communities can be found can be directly applied to the hierarchical cut-clustering algorithm. Flake et al. [FTT04] already proposed the use of the parametric maximum flow algorithm for calculating clusterings but without giving a concrete algorithm. In this work we apply the ideas of Gallo et al. in order to find all levels in the clustering hierarchy. This leads to an algorithm that is able to calculate a complete clustering hierarchy with at most twice as many clustering calculations as there are levels in the hierarchy. Let h be again the number of levels in the clustering hierarchy and r the running time of the cut-clustering algorithm. Then the running time of the hierarchical clustering algorithm using our new approach is $O(h \cdot r)$.

In order to calculate the interval boundaries of the clustering hierarchy we use the following cut-weight function:

Definition 2.2. *Let $G = (V, E, \omega)$ be a weighted, undirected graph and $S \subseteq V$. Then the cut-weight function for the cut of S in G_α for a variable α is defined as*

$$\begin{aligned} \omega_s \colon \mathbb{R}_0^+ &\to [\omega(S, V \setminus S), \infty) \\ \alpha &\mapsto \omega(S, V \setminus S) + \alpha \cdot |S| \end{aligned}$$

Obviously it is $\omega(S, V_\alpha \setminus S) = \omega_S(\alpha)$.

The slope of the cut-weight function is $|S|$, i.e., if S denotes a cluster this corresponds to the size of the cluster. This is, the slope of the cut-weight function for a cluster C_u is higher than for a cluster $C_l \subsetneq C_u$. The maximum slope is n, the minimum slope is 1.

If we consider two clusters $C_l \subsetneq C_u$ for two parameter values $\alpha_l > \alpha_u$, then the slope of the cut-weight functions of these two clusters is different, this is, the two straight

lines induced by the linear cut-weight functions have an intersection point at a parameter value α_m for which $\omega_{C_l}(\alpha_m) = \omega_{C_u}(\alpha_m)$ holds. As we show in Lemma 2.3 it holds that $\alpha_l \geq \alpha_m > \alpha_u$. The boundaries of the intervals of the parameter values for each level in the hierarchy are such intersection points, however, it is possible that for each level we need to consider a different pair of clusters for intersection. Finding these pairs of clusters is the main challenge of the algorithm. We will present the algorithm first and then prove its correctness.

Algorithm 3: intersectionPointRecursion

Input: Graph G, α_u, Clustering $\mathcal{C}_u$, α_l, Clustering $\mathcal{C}_l$, map hierarchy

```
1  M ← { max { (ω(C_l, V\C_l) − ω(C_u, V\C_u)) / (|C_u| − |C_l|)  |  C_l ∈ 𝒞_l, C_l ⊊ C_u } | C_u ∈ 𝒞_u } \ {α_l};
2  if M ≠ ∅ then
3      α_m ← min(M);
4      𝒞_m ← Cut-Clustering(G, α_m);
5      if |𝒞_m| == |𝒞_l| then
6          hierarchy[α_m] ← 𝒞_l;
7      else
8          intersectionPointRecursion(G, α_u, 𝒞_u, α_m, 𝒞_m, hierarchy);
9          intersectionPointRecursion(G, α_m, 𝒞_m, α_l, 𝒞_l, hierarchy);
10 else
11     hierarchy[α_l] ← 𝒞_l;
```

The structure of the algorithm is similar to the binary search algorithm. There is a recursive function named `intersectionPointRecursion` (see Algorithm 3) that takes the same parameters as the binary search algorithm and the initial call of it is done with the same trivial clusterings and parameter values.

At the beginning, the function calculates the set M that contains for each parent cluster $C_u \in \mathcal{C}_u$ the maximum of the intersection points of its cut weight function and the cut weight function of its children in the current intermediate hierarchy. As we are only interested in parameter values that are smaller than α_l we exclude α_l from M.

It is possible that M is empty, in this case α_l is already the lowest parameter value for which $\mathcal{C}_l$ is returned by the cut-clustering algorithm and we can add $\mathcal{C}_l$ to the result with α_l as parameter value.

Otherwise, we assign the minimum of M to α_m and calculate the corresponding clustering $\mathcal{C}_m$. If $\mathcal{C}_m$ and $\mathcal{C}_l$ are equal, we know by Theorem 2.5 that α_m is the lower boundary for the parameter interval of $\mathcal{C}_m$ and we can add $\mathcal{C}_m$ to the resulting hierarchy with α_m as parameter value. As there is no parameter value lower than α_m in M and for all parameter values higher than α_m the clustering $\mathcal{C}_l$ is returned, we are finished with the interval $[\alpha_u, \alpha_l]$.

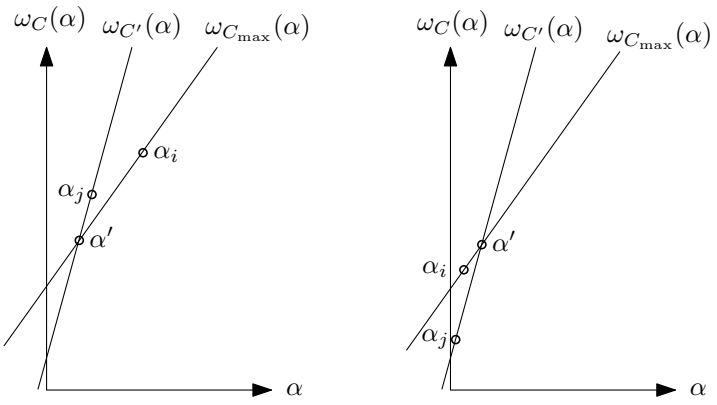
In the case that $\mathcal{C}_m$ is not equal to $\mathcal{C}_l$ we split the interval at α_m and call the function again for both halves.

We will now prove the claims we made in the description of this algorithm and show that this algorithm returns a complete clustering hierarchy with exact boundaries. In Section 3 we will show how this algorithm can be implemented such that these guarantees (completeness, exact boundaries) can be kept in practice.

Lemma 2.3. *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ denote two different clusterings in a clustering hierarchy $\mathcal{C}_1, \ldots, \mathcal{C}_k$, $1 \leq i < j \leq k$ and α_i, α_j the corresponding parameter values that induce $\mathcal{C}_i$ and $\mathcal{C}_j$ respectively (i.e. $\alpha_i > \alpha_j$).*

Let $C_1, \dots, C_l \in \mathcal{C}_i$ be the children of a cluster $C' \in \mathcal{C}_j$ with $C_1 \cup \dots \cup C_l = C'$ and $l > 1$. Let $\alpha' := \max_{y=1,\dots,l}\{\alpha \mid \omega_{C_y}(\alpha) = \omega_{C'}(\alpha)\}$ denote the maximum intersection point and $C_{\max} \in \mathcal{C}_i$ the corresponding child.

Then it holds $\alpha_j < \alpha' \leq \alpha_i$.



(a) First assumption (b) Second assumption

Figure 2.2: Illustration of the assumptions in Lemma 2.3.

Proof. Assume α' is smaller or equal to α_j as shown in Figure 2.2(a). This means that in G_{α_j} the cluster $C_{\max}$ induces a cut at most as expensive as the cut induced by C'. However, $C_{\max}$ has already the maximum point of intersection with the cut-weight function of C'. This means that the cluster $C_r \in \{C_1, \dots, C_l\}$ that contains the representative of C' with respect to α_j has a point of intersection of at most α' and thus for α_j its cut is also cheaper or equal to the cut of C'. This is a contradiction to C' being a community of its representative as C_r in G_{α_j} has a cheaper or equal cut weight compared to C' and $C_r \subsetneq C'$.

Now let us assume that α' is larger than α_i as shown in Figure 2.2(b). This means that in G_{α_i} the cut weight of C' is cheaper than of $C_{\max}$ as the slope of the cut-weight function of C' is higher than the slope of the cut-weight function of $C_{\max}$. This is a contradiction to $C_{\max}$ being a community of its representative in G_{α_i}.

□

The following lemma is only needed in the proof of Theorem 2.5.

Lemma 2.4. *Consider the same situation as in Lemma 2.3. If α' does not induce a cluster $\hat{C}$ such that $C_{\max} \subsetneq \hat{C} \subsetneq C'$ then the representative of $C_{\max}$ in G_{α_i} is also a representative for C' in G_{α_j}.*

Proof. Let r be the representative of $C_{\max}$ in G_{α_i}. We need to show that the community of r in G_{α_j} is C'. Recall that communities of nodes in a community are subsets of this community. This is, the community S of r in G_{α_j} is a subset of C', i.e. $C_{\max} \subseteq S \subseteq C'$.

We can exclude that $C_{\max} = S$ as the cut of C' in G_{α_j} is cheaper than the one of $C_{\max}$ and the cut of S must be cheaper or equal to the cut of C' as otherwise S would not be a minimum r-t-cut for α_j. Further ω_S has a smaller or equal slope compared to $\omega_{C'}$ and it holds $\omega_S(\alpha') \leq \omega_{C'}(\alpha') = \omega_{C_{\max}}(\alpha')$. If $\omega_S(\alpha') < \omega_{C_{\max}}(\alpha') = \omega_{C'}(\alpha')$ was true as shown in Figure 2.3, neither $C_{\max}$ nor C' would be a valid cluster at α'. However our initial situation is that either C' or $C_{\max}$ is a valid cluster for α', as otherwise $\hat{C} = S$ would exist. This means that $\omega_S(\alpha') = \omega_{C'}(\alpha')$. As $S \neq C_{\max}$ and for α' ω_S has a smaller or equal

slope compared to C', this is only possible if the slope of ω_S equals the one of $\omega_{C'}$ and $\omega_{C'}(\alpha_j) = \omega_S(\alpha_j)$ which implies that $S = C'$. This means that C' is the community of r with respect to t and thus r is a representative of C' as we have claimed.

□

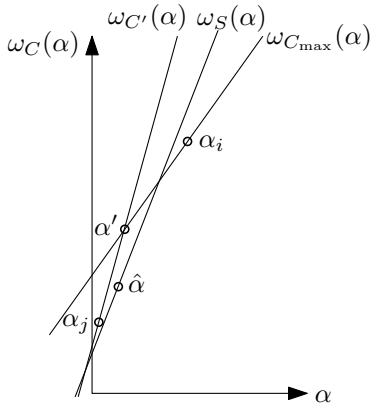


Figure 2.3: The additional community S in Lemma 2.4 and Theorem 2.5.

Theorem 2.5. *Assume α' as in Lemma 2.4. All clusterings for $\alpha \in [\alpha', \alpha_i]$ contain $C_{\max}$ as cluster and all clusterings for $\alpha \in [\alpha_j, \alpha')$ contain C' as cluster.*

Proof. First consider the clustering for α'. According to the supposed situation it either contains C' or $C_{\max}$. However, if C' was a valid cluster for α' it would induce a cheaper cut than the cluster in $\mathcal{C}_i$ that contains the representative of C' for α'. This is not possible as the cluster C' has the same cut weight as the cluster in $\mathcal{C}_i$ with the maximum cut weight for α' and thus C' can not be a valid cluster. This means that $C_{\max}$ is contained in the clustering for α' and thus in all clusterings for $\alpha \in [\alpha', \alpha_i]$.

Now we need to show that C' is a valid clustering for $\alpha \in [\alpha_j, \alpha')$. We already know that it is a valid clustering for α_j. Now take an arbitrary $\hat{\alpha} \in [\alpha_j, \alpha')$. Let $\hat{C}$ be the cluster the algorithm returns for $\hat{\alpha}$ with $C_{\max} \subseteq \hat{C}$ which is possible because of the nesting property of communities for different parameter values. Now we have the situation $C_{\max} \subseteq \hat{C} \subseteq C'$. Note that $\hat{C} = C_{\max}$ is not possible as for all parameter values smaller than α' C' has a cheaper cut than $C_{\max}$ and thus $C_{\max} = \hat{C}$ can not induce a minimum cut.

We know from Lemma 2.4 that the representative r of $C_{\max}$ for α_j is also a representative of C' for α_i. As r is also in $\hat{C}$ it has a community S in $G_{\hat{\alpha}}$ that is subset of $\hat{C} \subseteq C'$. This community S is also larger or equal compared to $C_{\max}$ because of the nesting property of communities for different parameter values.

For $\hat{\alpha}$, the cluster $\hat{C}$ has a cut weight that is lower or equal to the cut induced by C'. The community S has also a cut weight that is smaller or equal to the cut weight of C' for $\hat{\alpha}$ as shown in Figure 2.3.

If this community S was smaller than C' or had a cut weight smaller than the weight of the cut induced by C' at $\hat{\alpha}$, it would have a smaller cut weight than C' and $C_{\max}$ at α' and thus $C_{\max}$ would not be a valid cluster at α' which is a contradiction to what we have proved before. This means that $S = \hat{C} = C'$.

□

This shows the correctness of the algorithm: Whenever we look at the maximum intersection point of a parent clustering C' and its children as in Lemma 2.3, we either find

a new cluster between C' and $C_{\max}$, or we find the lower boundary of the interval of parameter values for which $C_{\max}$ is returned according to Theorem 2.5. This shows that our algorithm returns a complete clustering hierarchy.

This also shows the running time we have claimed: For each parameter value we either calculate a new different clustering or we find a new a lower boundary of a level. This means that for each level we need to calculate at most two cut clusterings.

Contractions can be added to this approach in the same way as we have used them in the approach based on binary search by simply contracting the lower clustering at the beginning of each recursive call.

3. Implementation

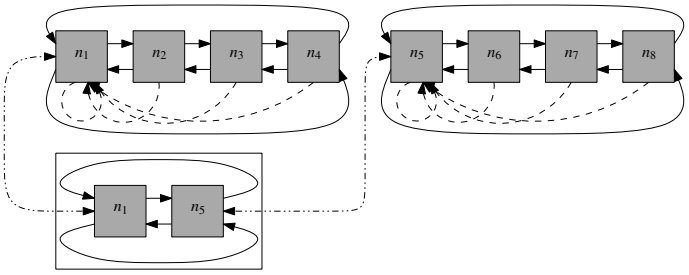
The implementation of the hierarchical cut-clustering algorithms and analysis tools is based on LEMON [LEMb] version 1.2.1. LEMON is short for Library for Efficient Modeling and Optimization in Networks, a C++ template library for optimization tasks with a focus on graphs and networks. It provides an efficient implementation of a preflow-push maximum flow algorithm, its worst case time complexity is $O(n^2\sqrt{m})$[LEMa]. We have used this maximum flow algorithm as the basis of the cut-clustering algorithm. The library also provides graph data structures like maps that allow $O(1)$ access to values associated with nodes or edges.

Based on the LEMON library we have created a set of data structures to represent and analyse the various aspects of clusterings. This includes a clustering class that provides functions for calculating various indices and other properties describing the clustering.

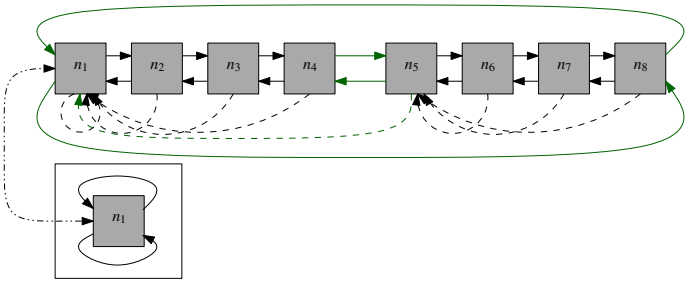
The representation of an individual clustering uses both a linked list for iterating over the nodes of a cluster as well as a union find data structure for finding the representative of a cluster.

Figure 3.1 shows an example of a merge operation in the data structure for two clusters. The data structure is shown first before the merge operation (Figure 3.1(a)) and then after the merge operation (Figure 3.1(b)). The items that are shown in the upper part of the figures are not the actual node objects, but rather simple container objects that are stored in a node map. Before the merge operation, there are two clusters containing nodes $n_1, \ldots, n_4$ and $n_5, \ldots, n_8$ respectively. The solid lines show the linked list structure we use to provide an iterator class for the nodes inside a cluster. The nodes n_1 and n_5 are the representatives of the two clusters, they are referenced using a union find data structure that is denoted by the dashed lines. The representatives point to themselves in the union find data structure so they can be easily recognized. The representatives are additionally stored in a linked list that is indicated in the box below the clustering. The representatives in the cluster structure contain additional pointers to the items in the linked list in order to be able to quickly delete them.

For merging these two clusters, the linked lists are connected which is indicated by the solid green arrows in Figure 3.1(b) and in the union find data structure the pointer of the representative of the one cluster is changed to point to the representative of the other cluster. Then the representative is deleted from the linked list using the pointer to the item in the linked list. This means that merge operations as used in the cut-clustering algorithm can be done in $O(1)$. Adding nodes to a cluster or creating a new cluster can be



(a) Before the merge operation.



(b) After the merge operation.

Figure 3.1: Representation of a clustering.

done in $O(1)$ as well. The complexity of the check if two nodes are in the same cluster is also very fast but depends on the union find data structure, our implementation uses only a very simple path compression. Deleting clusters or nodes from a cluster C is in $O(|C|)$ as the union find data structure needs to be updated for all nodes in the deleted cluster.

The clustering hierarchy is stored in a simple map using the parameter value as key and the clustering as value. For generating a hierarchy tree we have also implemented a class that generates a (LEMON) graph of the hierarchy. Each cluster is represented by a pair of its representative and the parameter value associated to the level of the cluster.

For the contractions we simply construct a new graph and then maintain a bidirectional mapping between the nodes of the contracted graph and the representatives in the original graph. Thus a clustering of the contracted graph can be expanded to a clustering for the original graph using the iterators and operations the clustering implementation provides.

3.1 File Formats

Our implementation supports two different input formats for the graph: The format used in the DIMACS graph clustering challenge [DIM] which is the METIS format [METa] and a format which is only slightly different from the format used for the DIMACS maximum flow problem description [Max]. We have also written a tool for converting graphs from the DIMACS maximum flow problem format into the latter format.

For the METIS graph file format please refer to the METIS User's Guide, Version 4.0 [METb], our implementation supports both unweighted and weighted edges and ignores node weights.

The second input format begins with a line "p cluster n m" where n and m denote the node and edge count. It can contain comment lines that start with "c" and are ignored by our implementation. Edge definitions, one on each line, have the format "e *source target weight*" where source and target are the node IDs and weight is the (positive, integer) edge weight. Node IDs are the numbers from 1 to n.

As output files we have on the one hand a very simple text format that describes a certain clustering or a clustering hierarchy and on the other hand we are generating GraphML files describing either a single clustering or a whole hierarchy tree that can be read by yEd [yEd].

The simple text format for describing a single clustering begins with a line that contains the number of nodes n. Then n lines follow that represent the nodes in the graph, ordered by node ID. Each line contains the ID of the representative of the cluster the node belongs to.

The format for whole hierarchies begins with a line that contains the maximum edge weight used in the computation. Then the file contains for each clustering a line with the parameter value followed by the same format as used for single clusterings. The maximum edge weight is included in the hierarchy format as the found parameter values are relative to it and change, when the edge weights are scaled. For the analysis of the experiments we need to know the maximum edge weight in order to rescale the parameter values for comparability.

3.2 Structure of the Calculation Process

We have split the calculation process into two steps: In the first step, the hierarchical cut-clustering algorithm is executed and produces the output text-file describing the clustering hierarchy. In a second step GraphML files for individual clusterings or the hierarchy can be generated and also different indices and statistics can be calculated without recalculating the whole clustering hierarchy.

We have split the process as the hierarchical cut-clustering algorithm needs some time to be executed. For the larger graphs in our test sample it took some hours till some days (compare Table 4.1).

3.3 Handling Edge Weights and Parameter Values

The two different approaches for calculating the clustering hierarchy require a different handling of edge weights in practice. For the binary search we have scaled the edge weights before the whole calculation process. For the algorithm based on points of intersection we have used rational numbers for the parameter values and for the actual minimum-cut calculations we have then scaled the edge weights by the denominator of the rational parameter value. We will now explain the different approaches in detail.

For the binary search algorithm on the one hand we wanted to scale the edge weights by a large factor in order to ensure that there is no hierarchy level missed. On the other hand, however, we wanted to keep the running time, which is directly influenced by the scaling, low. We scaled the smallest edge weight to either $1000 \cdot n$ if $n < 1000$ or n^2 otherwise with a limit of $\texttt{max\_long}/(n + 2 * m)$ for the largest edge weight. This limit makes sure that sums of all edge weights including reverse edges and edges to the additional node t can be calculated without overflows.

For the approach based on points of intersection we wanted to make sure that we can exactly calculate and store the parameter values. The denominators of the points of intersection are always the difference of two cluster sizes (compare Algorithm 3, line 1). This means that this value is very limited for a single point of intersection, however for representing all possible values we would need to scale in a dimension of $n!$, which is of course not possible for graphs with over thousand nodes. Thus we have tried to only scale by the actually needed factors by increasing the scale factor every time we encounter a non-integer intersection point but even then 64 bit integers were not enough for representing

the values even for graphs with less than thousand nodes. However for a single execution of the non-hierarchical cut-clustering algorithm this means that we only need to scale by a factor smaller than n so we simply scale the weights for each calculation of a single cut-clustering and reset the edge weights afterwards. The parameter values themselves are then stored as rational numbers using the implementation of rational numbers of the boost library [boo].

We have implemented contractions only for the first approach and not for the second one as implementing contractions for the second approach would require that the contraction class handles the scaling or that the cut-clustering algorithm knows about the contraction which is not easily possible in our implementation. As our experiments have shown (see Table 4.1) the second approach is still faster than the binary search based algorithm with contractions.

4. Experiments

With a set of 304 graphs we tested both hierarchical cut-clustering algorithms, the approach based on binary search and the approach based on intersection points. We start with a brief description of the graphs we used. After this we will have a look at the running times of our implementations. We will then have a closer look at some hierarchies and will select one level of each hierarchy for the further analysis. For this particular clustering we will then see how good the theoretical guarantee in terms of expansion is compared to the trivial expansion bounds.

As a second part of the experiments we compared the cut-clustering algorithm with an algorithm that tries to locally optimize modularity. We will first compare the clusterings directly, then we will have a look at the expansion values.

4.1 Test Instances

In this work we used instances provided by different sources. We used 27 instances of the test bed of the 10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering [DIM] and 275 snapshots of the dynamic network of email communication at the Department of Informatics at Karlsruhe Institute of Technology (KIT) [KIT]. In addition we used a graph describing a protein interaction network published by Jeong et al. [JMBO01] and a graph generated of a snapshot of the wiki pages of `http://www.dokuwiki.org` on November 13, 2010. The latter contains the pages as nodes and internal links as edges, counting links in both directions with a weight of two instead of one. We have also added a small star graph with a center node and six leaves to the test instances.

From the DIMACS Implementation Challenge we decided to use graphs of the category "Clustering Instances" as these graphs have a size the cut-clustering algorithm can handle and are popular benchmarks for clustering algorithms. We have also added the three smallest Delaunay graphs listed in the challenge.

The KIT provides a dataset containing anonymized email communication data between the email accounts of the Department of Informatics. For generating a graph from that data one needs to define which period of time should be covered by the graph. We have decided to print a snapshot every 2000 steps that contains the emails of the last 72 hours. This limit of 72 hours has also been used in previous works dealing with that data [GHW09]. We have also deleted singletons in these graphs. The graphs are named emailgraph550K_X, where X is the number of the snapshot.

In all graphs we have ignored loop edges. For the dokuwiki.org data as well as for the email communication network we implemented our own tools to construct the graphs from the data. The protein interaction network, which is named "bo", almost fit into the used graph format, so we adapted it using simple regular expressions.

4.2 Running Time

Even though our main focus is not on the performance of the algorithm we still want to give an idea how fast the hierarchical cut-clustering algorithm is. All runs have only been done once so it can not be excluded that non-reproducible cache effects, larger delays in file access etc. have influenced the results, so all the listed running times should not be taken as exact numbers.

We have measured the CPU time of the execution of the hierarchical cut-clustering algorithm including IO but without the generation of GraphML output or statistics. All executions have been done on an AMD Opteron^{TM}Processor 252 with 2.6 GHz with 16 GB RAM.

In Table 4.1 we have summarized the running time of a few small graphs and all large graphs sorted by the running time of the second approach based on intersection points. We have excluded most of the smaller graphs as we assume the running time is dominated by IO for small graphs and from the email graphs we have also just selected two examples as we wanted to keep the table clear.

For the largest graphs we have omitted the binary search calculation with contractions as we did not have enough time for it. The missing values have been marked with a "*".

The running times show that using contractions is in most cases an advantage, however the advantage is sometimes very small. This can be explained by the fact that for some graphs many clusterings contain a large number of small clusters and even singletons, and thus, the effect of the contraction is very small. For the graph rgg_n_2_15, the running time with contractions was even a lot higher than without contractions. This can be explained by the large amount of singletons in all hierarchy levels, even in the highest non-trivial level there are 31414 singletons out of 32768 nodes. In the graph cond-mat-2003 that has a similar size, only 16749 nodes were singletons in the highest non-trivial level and at maximum 21352 nodes were singletons in the lowest non-trivial level, and for this graph the contractions are an advantage again. This shows that the efficiency of the implementation of contractions is essential, especially for clusterings that contain mostly singletons.

However, for all instances the running time of the second approach based on intersection points is a lot better than the running time of the first approach with and without contractions. For some graphs the running times differ by even a factor of more than 20. Using contractions could also improve the running time of the second approach but we expect that the impact is not higher than for the first approach, especially as, depending on the implementation, less clusterings will be calculated using the same contraction.

The measured values also indicate that the main part of the running time is the time of the executions of the cut-clustering algorithm. The running time seems to be influenced by the number of levels in the hierarchy, see the column denoted by "h". The factor of 10 or 20 between the first and the second approach can be explained by the scaling factor which is between 2^{10} and 2^{30} and thus leads to around 10 to 30 times more executions of the cut-clustering algorithm. The running time of the first approach using contractions shows that linear factors should not be neglected and can provide a possibility to improve the running time for larger graphs that yield clusterings that contain mostly singletons.

Name of the graph	n	m	h	Bin. search	w. contract.	Inters.
jazz	198	2742	3	0.48s	0.47s	0.06s
celegans_metabolic	453	2025	8	2.51s	2.28s	0.30s
celegansneural	297	2148	17	4.02s	3.51s	0.40s
delaunay_n10	1024	3056	2	4.22s	4.19s	0.47s
emailgraph550K_19	491	853	33	10.68s	9.14s	0.77s
email	1133	5451	4	9.77s	9.44s	1.11s
emailgraph550K_26	527	1046	38	14.71s	12.84s	1.23s
delaunay_n11	2048	6127	2	15.93s	16.58s	1.79s
netscience	1589	2742	38	51.51s	17.37s	4.31s
bo	2114	2277	19	55.74s	26.16s	4.35s
polblogs	1490	16715	7	54.35s	51.93s	4.49s
delaunay_n12	4096	12264	2	1m11.31s	1m18.86s	7.22s
data	2851	15093	4	1m50.68s	1m32.29s	11.50s
dokuwiki_org	4416	12914	18	10m19.97s	8m14.65s	39.81s
power	4941	6594	66	22m29.64s	12m32.16s	1m25.73s
hep-th	8361	15751	56	117m02.37s	47m27.67s	6m26.21s
PGPgiantcompo	10680	24316	94	249m15.52s	86m43.51s	13m25.12s
as-22july06	22963	48436	33	821m26.75s	495m36.43s	39m54.49s
cond-mat	16726	47594	80	1213m	660m08.44s	44m15.31s
astro-ph	16706	121251	60	2443m	2150m	98m25.79s
rgg_n_2_15	32768	160240	46	5539m	8037m	245m25.64s
cond-mat-2003	31163	120029	74	5615m	4910m	268m14.60s
G_n_pin_pout	100000	501198	4	7380m	*	369m29.03s
cond-mat-2005	40421	175691	82	13994m	*	652m32.16s

Table 4.1: Running times of the larger graphs in our data set.

We have not tried to optimize the speed of the implementation by handling special cases separately, for example by skipping the unnecessary calculation of contractions for the singleton clusterings. Thus most probably all these running times can be further improved by optimizing the implementation.

4.3 Theoretical Guarantee vs Practical Results

In this section we have a look at the values the algorithm gives a guarantee for: the intra-cluster expansion and the inter-cluster expansion*. We will first have a look at these guarantees in the different hierarchy levels, then we will give an overview of these values in all graphs.

4.3.1 Analysis of Individual Cluster Hierarchies

In this section we examine the hierarchies of four graphs that are typical for the graphs in our set of graphs. We will have a look at the intra-cluster expansion and inter-cluster expansion* but also the modularity of the different levels in the hierarchy.

In practice, the hierarchies of both hierarchical cut-clustering approaches calculate for our set of graphs the exactly same clustering hierarchies, i.e., not a single hierarchy level is missed by the binary search algorithm. The boundaries of the parameter intervals the binary search algorithm calculates, however, are not exact in most cases. In the following we will always use the results of the second approach based on intersection points.

Figure 4.1 and 4.2 show on the horizontal axis the parameter value with the maximum edge weight scaled to one, on the left vertical axis the modularity value and on the right vertical axis the expansion values of the different levels of the clustering hierarchies of the four selected graphs. The diagonal denotes the guarantee the parameter value gives in terms of expansion, the crosses on it denote the boundaries of the different levels in the clustering hierarchy. Below the diagonal the inter-cluster expansion* is shown, most of the time it is almost on the horizontal axis. Above the diagonal the trivial upper bound of the intra-cluster expansion can be seen. For some values it touches the diagonal which means that for these values the guarantee the cut-clustering algorithm gives is already the exact intra-cluster expansion. In the first two graphs below, in the second two graphs also sometimes above the diagonal there is the trivial lower intra-cluster expansion bound.

For all of the four selected graphs the trivial lower and upper intra-cluster expansion bound do not exceed one. This is not for all our test instances the case, there are also graphs for which these values exceed one also with the maximum edge weight scaled to one.

The inter-cluster expansion* is for all four graphs much lower than the guarantee the cut-clustering algorithm gives, this means that the guarantee concerning the inter-cluster expansion* is for these graphs not meaningful.

In the first two graphs one can see that the guarantee the cut-clustering algorithm gives in terms of intra-cluster expansion can be a lot better than the trivial lower intra-cluster expansion bound, in the second two graphs one can see that the trivial intra-cluster expansion bound is sometimes also better than the guarantee the cut-clustering algorithm gives. These differences can also be found in the other graphs in our test set.

In the upper graph in Figure 4.2 one can see that the lowest parameter value for which the trivial clustering is returned does not need to be one but can also be a lot lower even if the edge weights are scaled to one. This is, the lower bound of the intra-cluster expansion the cut-clustering algorithm guarantees at maximum for a non-trivial clustering can also be lower than one.

For lower parameter values also the trivial upper intra-cluster expansion bound is relatively low in most cases which means less uncertainty but also a definitely lower intra-cluster expansion.

For most graphs the level with the best modularity is a level that is pretty close to the upper trivial level that contains the connected components. For higher parameter values, the modularity decreases in general, in some graphs there is also more than one local maxima.

We decided to pick the non-trivial clustering with the best modularity value from each clustering hierarchy in order to further compare them.

4.3.2 Comparative Analysis of all Graphs

In Figure 4.3 we plotted the lower and upper trivial intra-cluster expansion bounds as well as the inter-cluster expansion* and the lower and upper boundary of the parameter interval of the hierarchy level we have selected based on modularity for all graphs in our test set. On the horizontal axis the graphs are listed, each value represents one graph. In order to be able to visually compare the results we have scaled the values such that the upper boundary of the parameter interval is always 100. The expansion values on the vertical axis are all relative to the upper boundary of the parameter interval. We have sorted the graphs by the trivial lower bound of the intra-cluster expansion. Even though the values are discrete points we have plotted lines as we found that lines make it easier to get a visual impression of the relations between the different values.

One can see that in most cases the parameter value (the horizontal line at 100) is a better lower bound than the trivial lower bound. In a few cases, the trivial upper intra-cluster expansion bound (the line above the horizontal line) even equals the parameter value, thus in these cases we know that we have found the true intra-cluster expansion.

For now we can already conclude that for the graphs we have examined the theoretical guarantee of the cut-clustering algorithm gives us in most cases a better lower intra-cluster expansion bound than the trivial one.

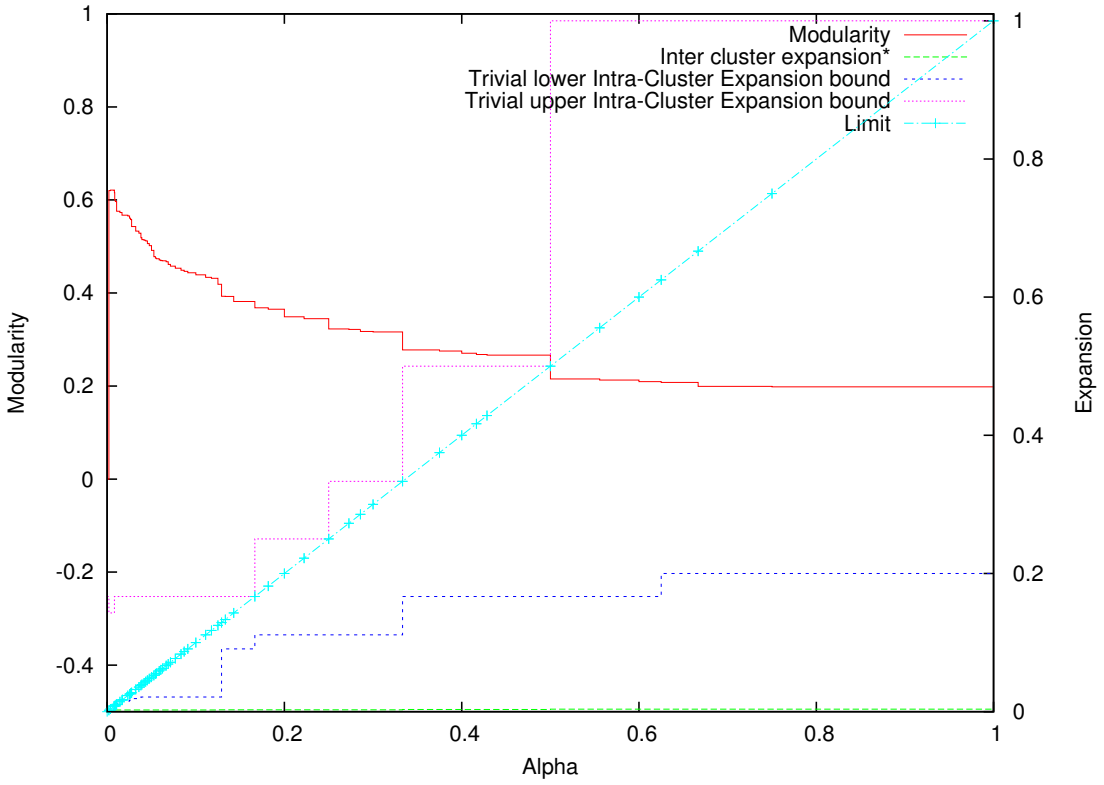
Regarding the inter-cluster expansion* one can see that in most cases the inter-cluster expansion* is better (i.e. lower) than the bound the cut-clustering algorithm gives us.

4.4 Comparison with Modularity-Based Clusterings

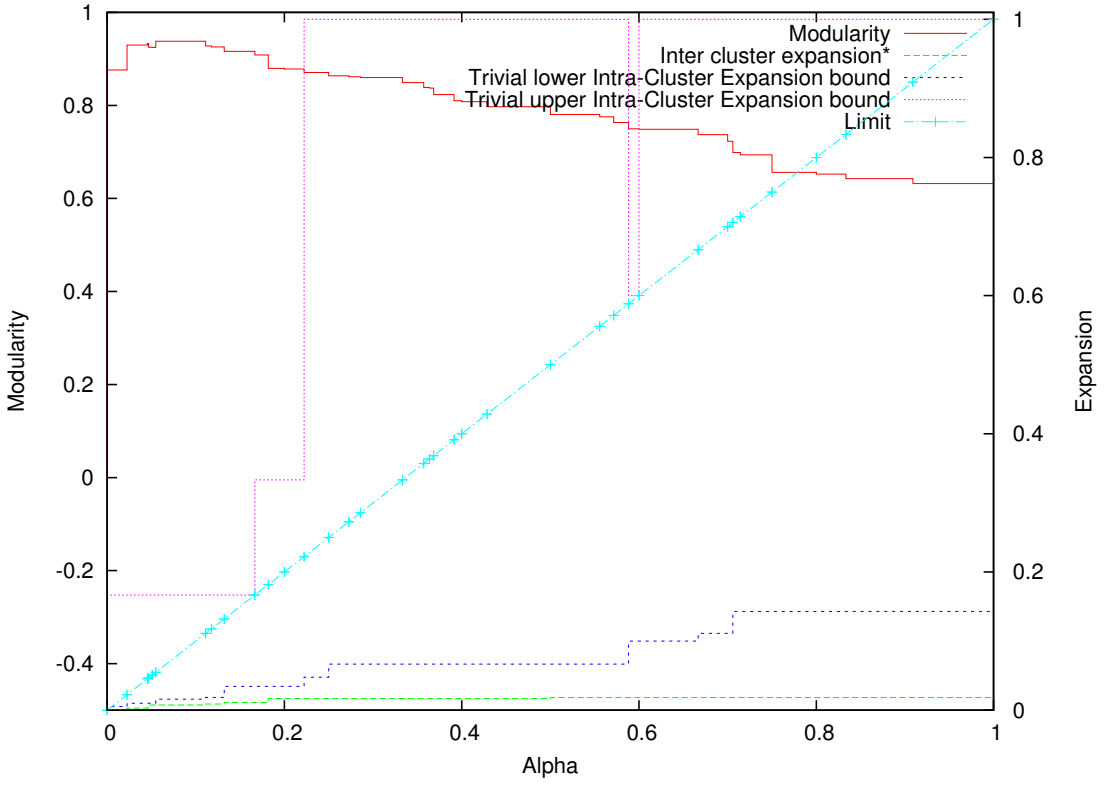
As modularity is a widely used measure for clusterings we want to compare the results of the cut-clustering algorithm with a clustering that is optimized for modularity in order to see how well the cut-clustering algorithm performs in terms of modularity. David Lisowski implemented an algorithm that optimizes modularity of a clustering greedily as part of his diploma thesis and provided the implementation of the algorithm. The algorithm and the implementation are described in detail in his diploma thesis [Lis11]. We will describe the algorithm in short here.

The general problem with optimizing modularity is that finding the modularity-optimal clustering is NP-hard. However it is possible to optimize modularity locally and in general the results of such algorithms is relatively good in terms of modularity.

Here we will describe a multi-level local-greedy algorithm. The algorithm begins with a trivial clustering in which each node forms its own cluster and then moves nodes between clusters whenever it increases the modularity. As soon as no increase in modularity is found anymore, the nodes in each cluster are contracted and the procedure continues on the next level. At the end, the different contractions are expanded again top-down and

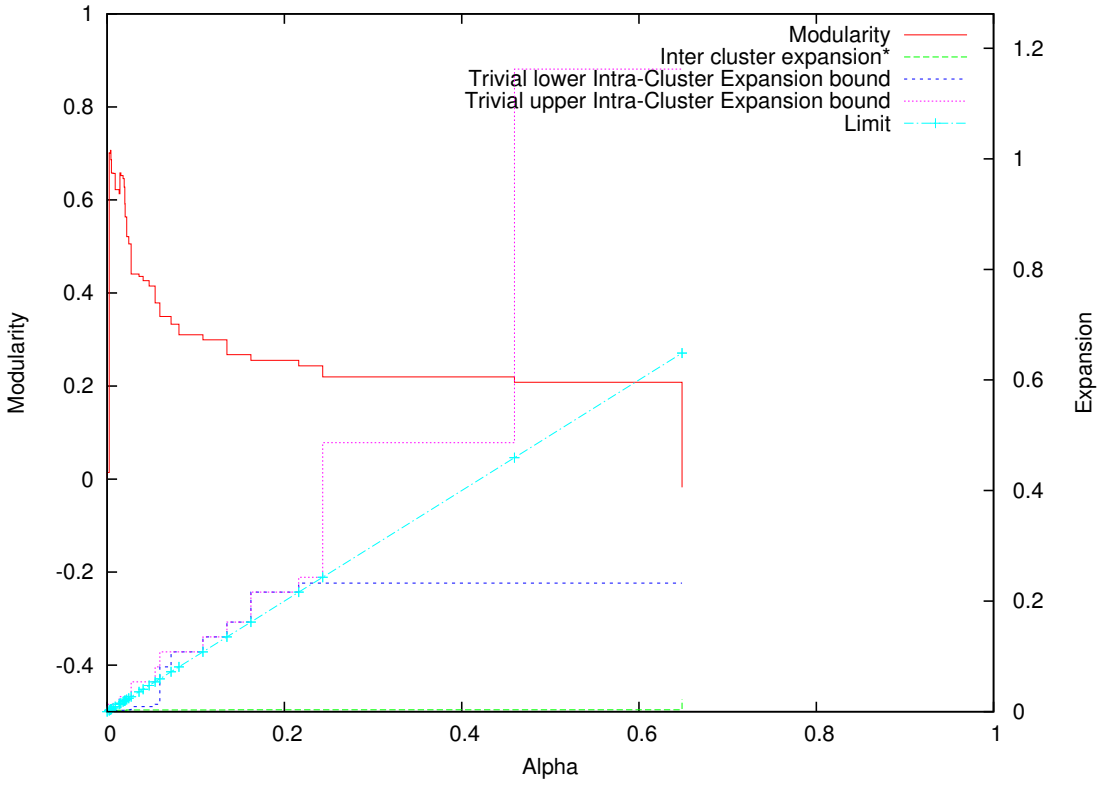


(a) power

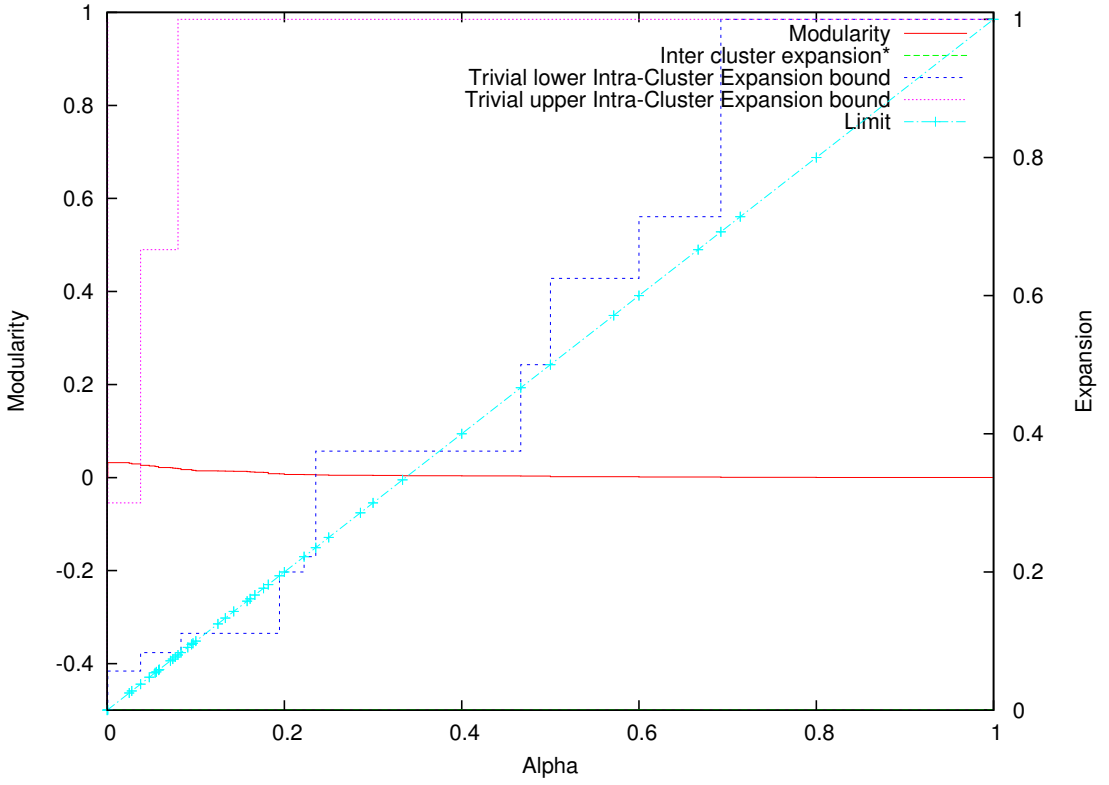


(b) netscience

Figure 4.1: Cluster hierarchy statistics of power and netscience.



(a) emailgraph550K_109



(b) rgg_n_2_15_s0

Figure 4.2: Cluster hierarchy statistics of emailgraph550K_109 and rgg_n_2_15_s0.

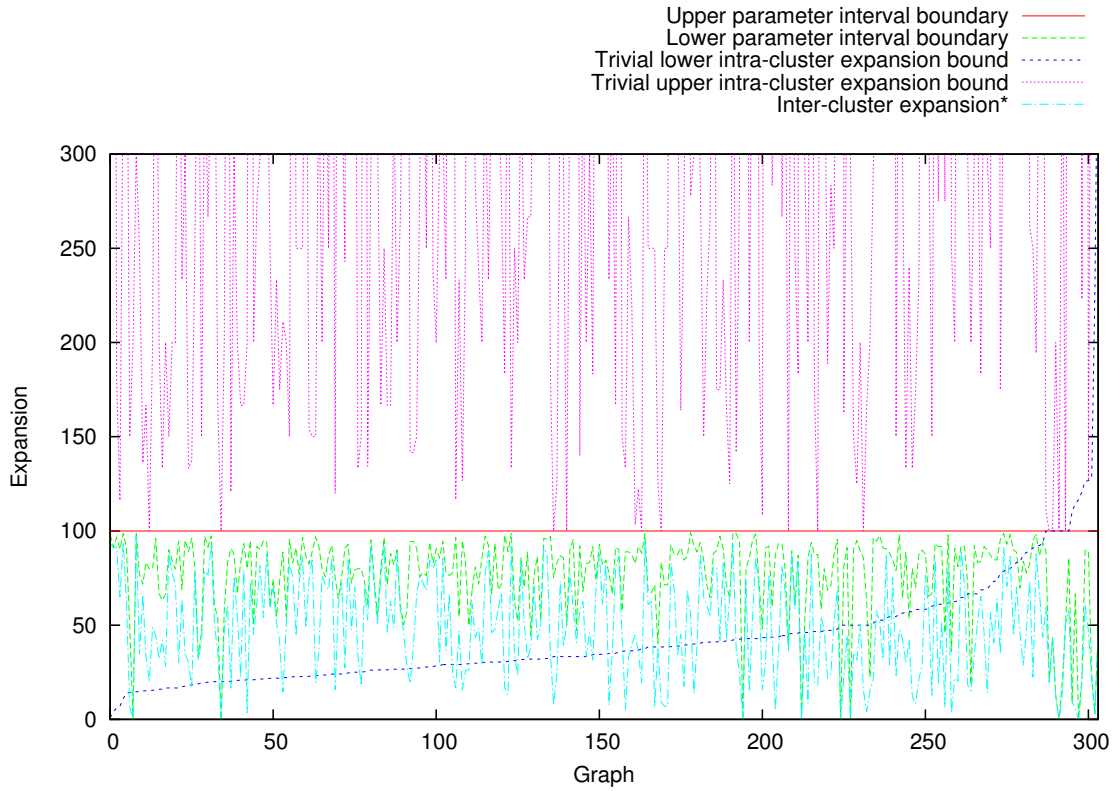


Figure 4.3: Overview of the expansion values in comparison with the parameter value.

after each expansion, the algorithm tries again to optimize modularity by moving single nodes.

We will refer to this algorithm as *modularity algorithm*. We will refer to the clustering calculated by the modularity algorithm as *modularity clustering* in contrast to the *cut clustering* calculated by the cut-clustering algorithm which will always refer to the non-trivial level in the hierarchy tree with the highest modularity value.

We have converted all graphs in the test set into the format of the modularity algorithm and we converted the output back into our own clustering format such that we could use the same analysis tools we have already used for the cut-clustering algorithm.

Unfortunately, the algorithm as described above does not guarantee that the resulting clusters are connected and for our collection of graphs the algorithm actually calculated some unconnected clusters. This can happen if the algorithm gets stuck at a local maximum. As this makes the analysis of intra-cluster expansion impossible, we decided to post-process the clusterings and split unconnected clusters in their connected components. In doing so, the resulting modularity values can only become better. In the following we will always use these post-processed clusterings.

For visualizing clusterings we generated plots of the adjacency matrices of graphs where the node IDs are sorted by the clusters, i.e., nodes inside a cluster have consecutive IDs. Each non-zero entry in the matrix is denoted by a point in the plot. As we did not vary the size of the dots by the edge weight, these plots are only meaningful for unweighted graphs. In order to clearly indicate the clusters we also added a point for each representative on the diagonal, for the modularity clusterings, which do not have special representatives, we selected arbitrary representatives. This means that each cluster is the rectangle defined by two points on the diagonal that is also the diagonal of this rectangle. We sorted the clusters by size, the largest clusters have the highest node IDs, this is, the largest cluster is

in the top-right corner. All points outside these rectangles are, even if they look sometimes as if they had a structure or were even clusters, inter-cluster edges that show a structure that is defined by the order in the input files.

In Table 4.2 we combine different indices of the cut-clustering algorithm and the modularity algorithm. Most of the results presented in the following sections are based on these values.

In the leftmost column are the names of the graphs and below each name some general properties like the number of nodes (n), the number of edges (m) and the number of connected components (k). For each graph there are two rows. The upper row contains indices concerning the cut clustering, referred to as $\mathcal{C}$, and the lower row lists results concerning the modularity clustering, referred to as $\mathcal{C}_{\text{mod}}$. The operator exp refers to intra-cluster expansion and α is the upper boundary the interval of the parameter values for the level of the cut clustering $\mathcal{C}$. In the table we also mention a clustering $\mathcal{C}_m$ which we will explain in the next paragraphs.

In the following we compare the modularity clusterings and the cut clusterings in a general overview. Then we will also have a look at the intra-cluster expansion and see how they can be compared using trivial bounds. In a last step we will select some graphs and compare their clusterings in more detail.

4.4.1 Overview of Cut Clusterings and Modularity Clusterings

The first question we address to is if there are obvious similarities between cut clusterings and modularity clusterings.

Looking at the modularity values of the cut-clustering algorithm and the modularity algorithm in the third column in Table 4.2 one can already see that in all cases the modularity value of the results of the modularity algorithm is better than the modularity of the results of the cut-clustering algorithm, in most cases slightly better, in some cases even significantly better.

Looking by hand at the results of the algorithms for a couple of graphs gives the impression that the result of the cut-clustering algorithm with the best modularity value is in general more fine-grained than the result of the modularity algorithm. Furthermore, in many graphs there are nodes for which the cut-clustering algorithm does not decide to which cluster they belong as they are connected equally well to more than one cluster. Instead the cut-clustering algorithm creates a singleton cluster. Based on these observations we decided to consider a further clustering type which results from merging all clusters in the cut clustering for which the representatives are in the same cluster in the modularity clustering. We refer to this clustering as merged clustering C_m.

In Figure 4.4 we have plotted the modularity of the cut clustering, the modularity clustering and the merged clustering. The horizontal axis is again the list of graphs, sorted by the modularity of the modularity clustering.

Merging these clusters improves the modularity value of the clustering in almost all cases, only for some graphs there is a decrease in modularity as one can see in the figure. The graphs for which the modularity both of the cut clustering and the merged clustering are low are trivial clusterings that contain only the connected components, thus merging clusters has no effect.

We have also added the quotient between the modularity values to the table in column five, upper row. In the lower row in the same column we have noted the quotient of the modularity of the merged clusters and the clustering of the modularity algorithm. One can also see in the table that in many cases the modularity of the merged clusters is close to or even better (for example emailgraph550K_273) than the modularity of the

Graph			$\lvert\mathcal{C}\rvert$	mod $(\mathcal{C})$	$\lvert\mathcal{C}_m\rvert$	$\frac{\text{mod }(\mathcal{C}_m)}{\text{mod }(\mathcal{C})}$	$\exp_{\min}(\mathcal{C})$	$\exp_{\max}(\mathcal{C})$	α	$\frac{\exp_{\min}(\mathcal{C})}{\alpha}$
n	m	k	$\frac{\lvert\mathcal{C}\rvert}{\lvert\mathcal{C}_{\text{mod}}\rvert}$	mod $(\mathcal{C}_{\text{mod}})$	mod $(\mathcal{C}_m)$	$\frac{\text{mod }(\mathcal{C}_m)}{\text{mod }(\mathcal{C}_{\text{mod}})}$	$\exp_{\min}(\mathcal{C}_{\text{mod}})$	$\exp_{\max}(\mathcal{C}_{\text{mod}})$	$\frac{\exp_{\max}(\mathcal{C})}{\alpha}$	$\frac{\exp_{\min}(\mathcal{C}_{\text{mod}})}{\alpha}$
emailgraph550K_38			40	0.8842	34	1.0167	0.0128	0.0333	0.0333	0.3846
222	237	28	35	0.9012	0.8989	0.997456	0.0128	0.0417	1.0000	0.3846
G_n_pin_pout			99947	0.0001	130	5051.7180	1.0000	1.0000	1.0000	1.0000
100000	501198	6	130	0.4786	0.4786	1.000000	0.0002	0.2500	1.0000	0.0002
emailgraph550K_205			38	0.8208	33	1.0106	0.0009	0.0009	0.0007	1.2667
193	204	29	36	0.8506	0.8295	0.975214	0.0013	0.0066	1.2667	1.9000
emailgraph550K_273			56	0.7855	41	1.0348	0.0013	0.0036	0.0024	0.5455
264	287	33	41	0.8109	0.8129	1.002447	0.0011	0.0036	1.5000	0.4615
celegans_metabolic			6	0.0373	5	1.0386	0.0046	1.0000	0.5097	0.0091
453	2025	1	10	0.4395	0.0387	0.088146	0.0294	1.0000	1.9619	0.0577
cond-mat-2005			27569	0.1502	1869	4.8490	0.0417	0.0769	0.0345	1.2083
40421	175691	1798	1892	0.7287	0.7283	0.999437	0.0006	0.0667	2.2308	0.0183
PGPgiantcompo			2169	0.5643	62	1.5612	0.0067	0.0714	0.0244	0.2733
10680	24316	1	107	0.8844	0.8810	0.996147	0.0030	0.2000	2.9286	0.1239
polbooks			103	-0.0068	4	-76.3600	2.0000	2.0000	0.5000	4.0000
105	441	1	4	0.5208	0.5208	1.000000	0.1000	2.0000	4.0000	0.2000
emailgraph550K_187			41	0.7590	22	1.0345	0.0004	0.0119	0.0024	0.1667
339	617	12	25	0.8364	0.7852	0.938740	0.0005	0.0119	5.0000	0.2273
cond-mat-2003			19890	0.1952	1664	3.9159	0.0400	0.1667	0.0313	1.2800
31163	120029	1599	1683	0.7651	0.7646	0.999374	0.0009	0.0400	5.3333	0.0289
as-22july06			13846	0.2052	44	3.2821	0.0041	1.0000	0.1000	0.0415
22963	48436	1	45	0.6736	0.6735	0.999850	0.0004	0.2500	10.0000	0.0039
emailgraph550K_262			38	0.6573	35	1.0057	0.0019	0.0227	0.0022	0.8750
224	256	31	38	0.7775	0.6610	0.850214	0.0025	0.0455	10.5000	1.1667
rgg_n_2_15_s0			31540	0.0320	85	30.2098	0.0566	0.3000	0.0250	2.2642
32768	160240	6	85	0.9653	0.9653	1.000000	0.0026	0.0357	12.0000	0.1031
power			1583	0.6213	35	1.4709	0.0023	0.1429	0.0077	0.2955
4941	6594	1	42	0.9383	0.9138	0.973925	0.0101	0.0625	18.5714	1.3131
football			1	0.0000	1	DIV/0!	0.1228	7.0000	0.1053	1.1667
115	613	1	10	0.6046	0.0000	0.000000	0.3750	0.8333	66.5001	3.5625
data			2504	0.1153	9	6.9051	0.0714	5.0000	0.0647	1.1032
2851	15093	1	9	0.7959	0.7959	1.000000	0.0101	3.0000	77.2222	0.1560
delaunay_n10			1	0.0000	1	DIV/0!	0.0059	3.0000	0.0117	0.4995
1024	3056	1	16	0.8189	0.0000	0.000000	0.0526	2.0000	255.7501	4.4868

Table 4.2: Selected properties of different clusterings of some of our test instances.

28

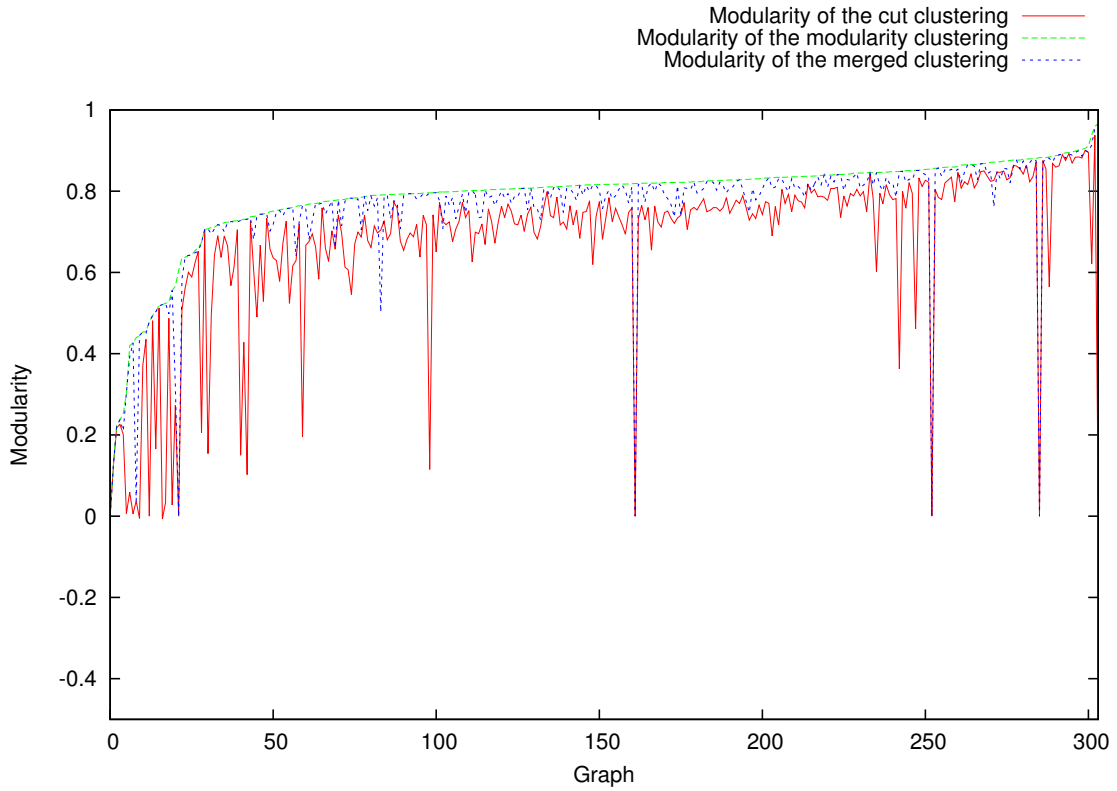


Figure 4.4: Overview of the modularity values of the different clustering algorithms.

modularity clustering. As we have already seen in the figure there are a few cases where the cut-clustering algorithm returned a trivial clustering or less clusters than the modularity algorithm, this is for example for the graph celegans_metabolic and the delaunay graphs the case. In these cases the modularity is not similar to the modularity of the modularity clustering, i.e., it does not improve at all or only very slightly. We will now have a look at these two examples in more detail.

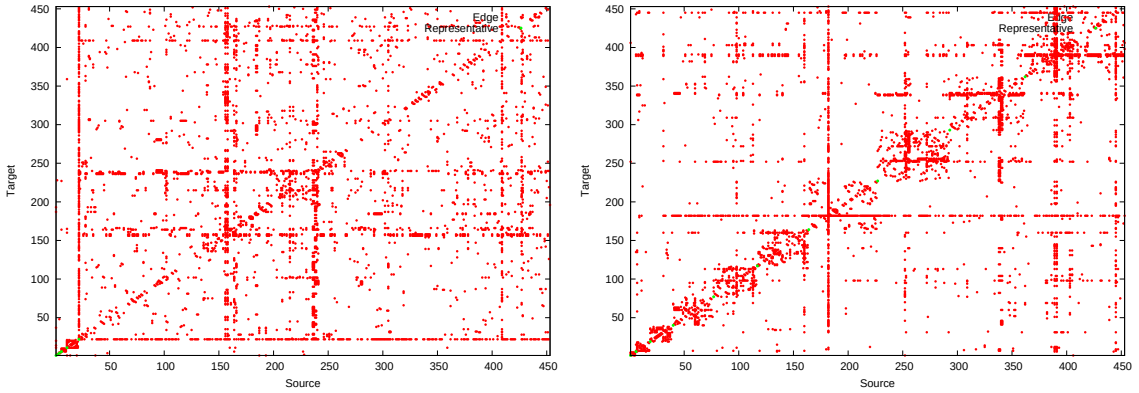


Figure 4.5: Graph "celegans_metabolic" - the cut clustering on the left side, the modularity clustering on the right side.

In Figure 4.5 we have plotted the cut clustering and the modularity clustering of celegans_metabolic. The large upper area in the left plot of the cut clustering is just one cluster while the modularity clustering contains almost equally sized clusters that seem to have relatively strong connections between them. With only six instead of ten clusters the modularity of the cut clustering is with 0.037 very low, but also the modularity algorithm

reaches only a modularity value of 0.439.

A whole family of graphs for which the cut-clustering algorithm has not found any structure are the delaunay graphs, for all three delaunay graphs we have tested only trivial clusterings with a modularity value of zero were returned while the modularity algorithm produced clusterings with a high modularity value (0.8 and higher). One could wonder if these graphs do not have enough structure, but looking at the result of the modularity algorithm in Figure 4.6 (right side) the modularity clustering seems to be a good clustering even though there are some edges between the clusters. The unclustered result of the cut-clustering algorithm looks interesting, too, but this is not the result of the cut-clustering algorithm but rather of the order of the nodes in the input file.

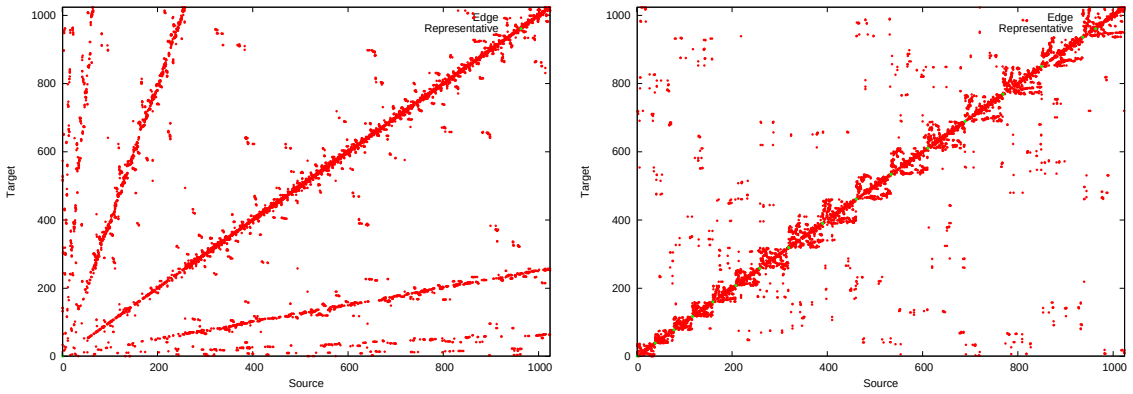


Figure 4.6: Graph "delaunay_n10" - the cut clustering on the left side, the modularity clustering on the right side.

The similarity of modularity values and the similar number of clusters in the merged clusterings and in the modularity clusterings give the impression that in many cases the merged clustering is similar to the modularity clustering but has sometimes better boundaries of the clusters, for example the merged clustering of emailgraph550K_273 has the same number of clusters as the modularity clustering, but a better modularity is achieved.

4.4.2 Intra-Cluster Expansion of Modularity Clusterings

In the previous section we have seen that regarding modularity cut clusterings are for our test set worse than the modularity clusterings, so in this section we want to consider intra-cluster expansion where the guarantee that the cut-clustering algorithm gives is at least in theory an advantage over the modularity algorithm. We want to see if this guarantee is an advantage in practice, too. We will compare the trivial bound of the intra-cluster expansion of the modularity clusterings with the bounds we have already seen for the cut clustering.

In addition to the selected results in Table 4.2 we have plotted an overview of all graphs in Figure 4.7. As in Figure 4.3 we have plotted the lower and upper bound of the intra-cluster expansion in the cut clusterings and also the upper boundary of the interval of the parameter values of the cut clusterings, as in Figure 4.3 scaled to 100. The graphs are again sorted by the trivial lower bound of the intra-cluster expansion of the cut clustering. In addition to that we have added the trivial lower and upper bound of the intra-cluster expansion of the modularity clusterings.

The figure shows that for most graphs the trivial lower bound of the intra-cluster expansion is similar in both clusterings and the trivial lower bound of the intra-cluster expansion of the modularity clustering is below the guarantee the cut clustering algorithm gives. For

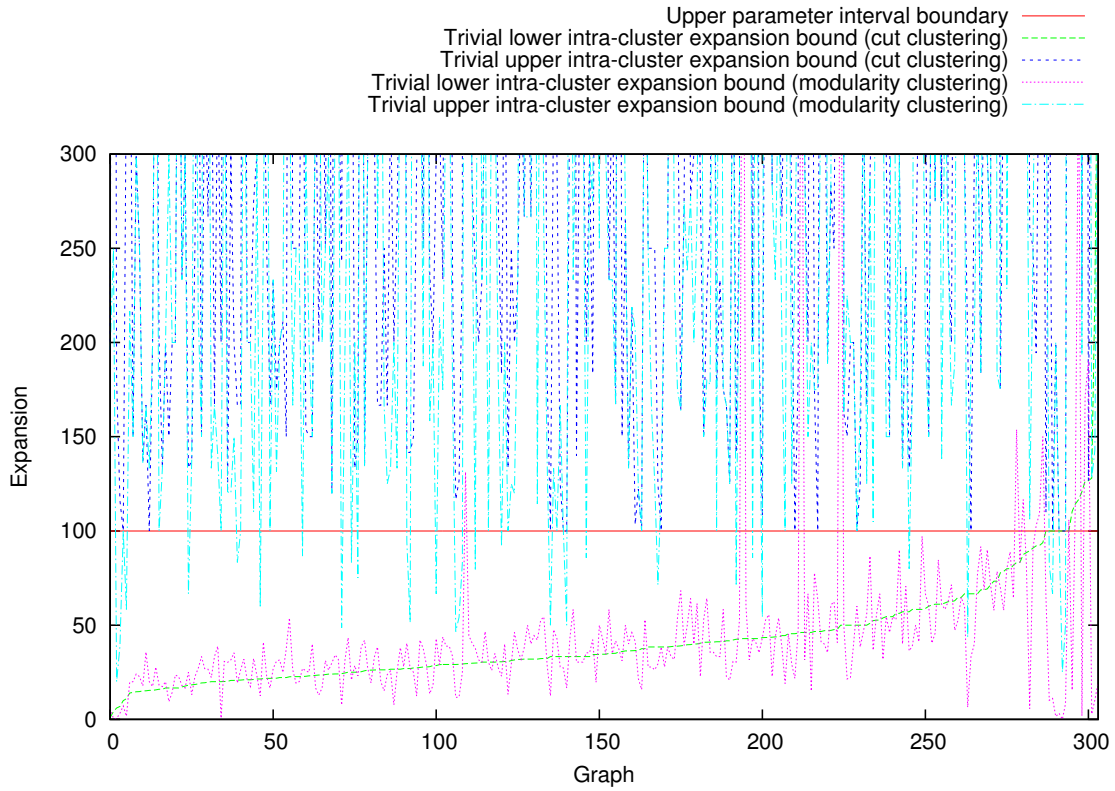


Figure 4.7: Overview of the intra-cluster expansion bounds of the cut clusterings and the modularity clusterings.

32 graphs the upper bound of the intra-cluster expansion of the modularity clustering is lower than the parameter value or the trivial lower bound of the intra-cluster expansion for the cut clustering which means that these 32 modularity clusterings are definitely worse in terms of intra-cluster expansion than the corresponding cut clustering.

For the graph PGPgiantcompo we have the situation that the lower intra-cluster expansion bounds for the cut clustering are better than the trivial lower intra-cluster expansion bound for the modularity clustering, but the trivial upper intra-cluster expansion bound for the modularity clustering is eight times higher than the lower intra-cluster expansion bound of the cut clustering, so we can not make any definitive comparison. In Figure 4.8 we have plotted the two clusterings. The left plot shows the cut clustering, it contains a lot of singletons while the right plot shows that the modularity clustering contains larger, more similar sized clusters. Looking at the inter-cluster edges one has the impression that certain clusters have relatively strong connections to other clusters. The cut clustering has 2169 clusters for the 10680 nodes while the modularity clustering calculated only 107 clusters. More extreme results can be seen for other large graphs in the test set, the cut clusterings have for some of these graphs a better intra-cluster expansion but on the other hand the clusterings contain only very small clusters.

Altogether the cut-clustering algorithm returns for 32 graphs in our test set a cut-clustering with a better intra-cluster expansion than the modularity algorithm and for 290 of the 304 graphs in our test set we get at least a better lower bound of the intra-cluster expansion but for these clusterings we do not know which clustering is better in terms of intra-cluster expansion. However, there is only one single graph (emailgraph550K_205) in the test set for which the trivial lower intra-cluster expansion bound of the modularity clustering is higher than the trivial upper intra-cluster expansion bound of the cut clustering.

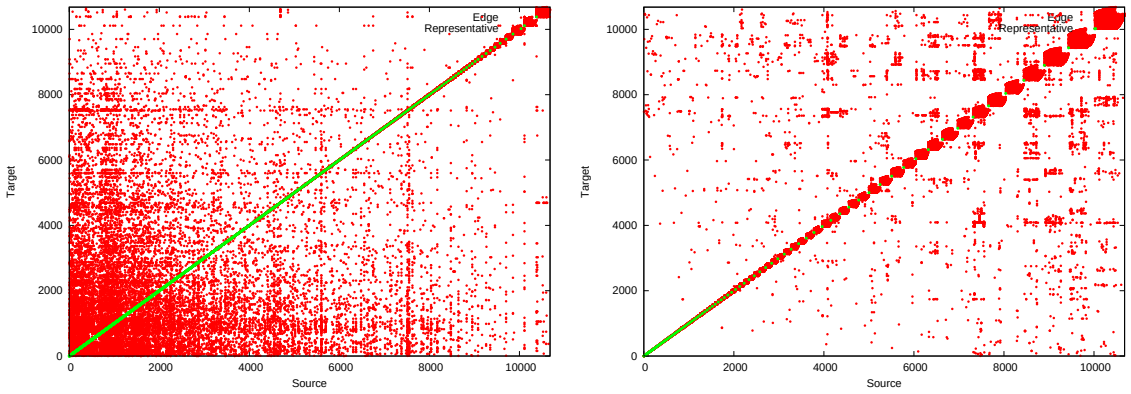


Figure 4.8: Graph "PGPgiantcompo" - the cut clustering on the left side, the modularity clustering on the right side.

5. Conclusion

In the first part of this work we proved that calculating a hierarchy of all different cut clusterings can be done with at most twice as many executions of the cut-clustering algorithm as clustering levels in the found hierarchy. Furthermore, the boundaries of the intervals of the parameter values that correspond to the levels in the clustering hierarchy are exact. This means that we get a complete overview of all different cut clusterings for a certain graph and can use the boundaries of the intervals for the guarantee the cut-clustering algorithm gives which can yield better guarantees than just using a single parameter value.

In the second part we then had a look at the clusterings and saw that in most cases the theoretical guarantee the algorithm gives for the intra-cluster expansion is better than the trivial lower bound of the intra-cluster expansion. In the individual clustering hierarchies we have seen that the modularity value is best for very low parameter values, so choosing a higher parameter value that gives a high lower bound of the intra-cluster expansion is not always good, one can even get the impression that intra-cluster expansion and modularity are conflicting measures.

For the further analysis we picked the best cut clustering in terms of modularity of each clustering hierarchy and compared it to results of an algorithm that tries to locally optimize modularity. In that comparison we noticed that merging the clusters in the cut clustering whose representatives are in the same cluster in the modularity clustering yields a new clustering that is similar to the modularity clustering. In most cases this merged clustering has a modularity value that is only slightly lower or in a few cases even higher than the modularity value of the modularity clustering.

In the individual clusterings we have seen a large diversity, from graphs for which similar clusterings are calculated by both algorithms to graphs that yielded completely different clusterings. In the examples we have examined in more detail we found that while the cut-clustering algorithm calculates clusters of very different sizes and does not cluster nodes if they do not clearly belong to a cluster the modularity algorithm creates a smaller amount of relatively equally-sized clusters.

Concluding we can say that the cut-clustering algorithm does help knowing the intra-cluster expansion of a clustering and gives a good guarantee for it but does only create clusters for structures that are really clear, i.e., the cut-clustering algorithm is very strict. This can be an advantage if the focus is on finding closely connected and possibly singleton clusters but also a disadvantage if each node should be assigned to a non-trivial cluster.

Open Problems

In the following we describe further approaches connected to this work that we did not explore in this work.

We think that results of the cut-clustering algorithm could be used as basis for other (greedy) clustering algorithms like the modularity algorithm in order to give them a solid basis of clusters greedy algorithms possibly would not find otherwise.

Regarding the clustering hierarchy it could be possible that clusterings consisting of a combination of clusters from different levels have a better modularity value than the levels themselves.

Instead of the algorithm based on intersection points the parametric maximum-flow algorithm could be used which gives in theory a much better running time. A clever use of the algorithm, i.e., not simply executing it for every node in the graph, could also lead to an improved running time in practice.

Bibliography

[boo] "boost documentation of Rational Numbers." [Online]. Available: http://www.boost.org/doc/libs/1_46_1/libs/rational/rational.html

[CM99] J. Cheriyan and K. Mehlhorn, "An analysis of the highest-level selection rule in the preflow-push max-flow algorithm," *Information Processing Letters*, vol. 69, no. 5, pp. 239 – 242, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019099000198

[DIM] "10th DIMACS Implementation Challenge – Graph Partitioning and Graph Clustering." [Online]. Available: http://www.cc.gatech.edu/dimacs10/

[FTT04] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis, "Graph clustering and minimum cut trees," *Internet Math.*, vol. 1, no. 4, pp. 385–408, 2004. [Online]. Available: http://projecteuclid.org/getRecord?id=euclid.im/1109191029

[GGT89] G. Gallo, M. Grigoriadis, and R. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM Journal on Computing*, vol. 18, p. 30, 1989. [Online]. Available: http://dx.doi.org/10.1137/0218003

[GHW09] R. Görke, T. Hartmann, and D. Wagner, "Dynamic Graph Clustering Using Minimum-Cut Trees," in *Algorithms and Data Structures, 11th International Workshop (WADS'09)*, ser. Lecture Notes in Computer Science, F. Dehne, J.-R. Sack, and R. Tamassia, Eds., vol. 5664. Springer, August 2009, pp. 339–350. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03367-4_30

[GT88] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, pp. 921–940, October 1988. [Online]. Available: http://doi.acm.org/10.1145/48014.61051

[Gö10] R. Görke, *An Algorithmic Walk from Static to Dynamic Graph Clustering.* Karlsruhe, 2010. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000018288

[JMBO01] H. Jeong, S. P. Mason, A.-L. Barabasi, and Z. Oltvai, "Centrality and lethality of protein networks," *Nature*, vol. 411, no. 11, 2001. [Online]. Available: http://dx.doi.org/10.1038/35075138

[KIT] "Dynamic network of email communication at the Department of Informatics at Karlsruhe Institute of Technology (KIT). Data collected, compiled and provided by Robert Görke and Martin Holzer of ITI Wagner and by Olaf Hopp, Johannes Theuerkorn and Klaus Scheibenberger of ATIS, all at KIT. 2011." [Online]. Available: http://i11www.iti.kit.edu/projects/spp1307/emaildata

[LEMa] "Documentation of Goldberg-Tarjan's preflow push-relabel algorithm in LEMON." [Online]. Available: http://lemon.cs.elte.hu/pub/doc/1.2.1/a00261.html

[LEMb] "Library for Efficient Modeling and Optimization in Networks (LEMON)." [Online]. Available: http://lemon.cs.elte.hu/trac/lemon

[Lis11] D. Lisowski, "Modularity-basiertes Clustern von dynamischen Graphen im Offline-Fall," 2011. [Online]. Available: http://i11www.iti.uni-karlsruhe.de/_media/teaching/theses/da-lisowski.pdf

[Max] "Maximum Flow Problem Format." [Online]. Available: http://www.avglab.com/andrew/CATS/maxflow_formats.htm

[METa] "METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering." [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[METb] "Metis User's Guide, Version 4.0." [Online]. Available: http://www.cc.gatech.edu/dimacs10/data/manual.ps

[NG04] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.69.026113

[yEd] "yEd Graph Editor." [Online]. Available: http://www.yworks.com/en/products_yed_about.html