Karlsruhe Institute of Technology

Hierarchical Cut Clustering in Dynamic Scenarios

Student Research Project of

Christof Doll

Department of Informatics
Institute of Theoretical Informatics

Reviewer:	Prof. Dr. Dorothea Wagner
Advisor:	Dipl.-Inform./Dipl.-Math. Tanja Hartmann

Submission: 23. February 2011

Contents

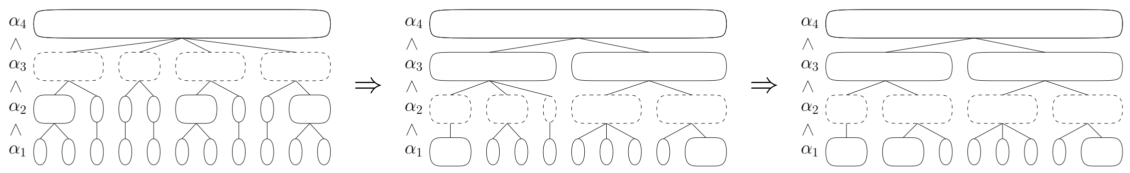
1 Introduction

Clustering data sets into groups that are in some sense homogeneous and well separated from each other is a problem that arises in many domains, e.g. the analysis of networks. These data sets can often be represented as weighted graphs, where the vertices correspond to the entities in the data sets and the edge weight between two vertices stands for the similarity of the connected entities. In the context of graphs a clustering is a partition of the vertices into disjoint vertex-induced subgraphs called clusters. The homogeneity of the entities within one cluster can be measured by the density of edges between vertices in this cluster. The more edges and the heavier they are the denser is the cluster. On the opposite only little and light weight edges between the clusters are desired. Thus, graph clustering conforms to the paradigm of *intra-cluster density* and *inter-cluster sparsity*.

In the last decades a lot of time and effort has been spent on building algorithms that cluster graphs. Most of these algorithms rely on heuristics and they cannot offer any guarantee on the shape, size and number of clusters they compute. However, Flake et al. [1] presented a graph clustering algorithm which offers good quality bounds. Their Cut Clustering Algorithm bases on minimum-cut trees developed by Gomory and Hu [3]. Beside the graph the input to their algorithm consists of a parameter α whose choice influences a lower bound for the weight of intra-cluster cuts and an upper bound for the weight of inter-cluster cuts. The value of α is as well important for the number of clusters. For a large α the Cut Clustering Algorithm returns small but many clusters and for a small value it computes few but big clusters. Flake et al. build a hierarchy of clusterings by basically running the Cut Clustering Algorithm iteratively for decreasing values of α yielding one level of the hierarchy per run. They show that in this hierarchy the clusters on lower levels are subsets of clusters on higher levels (see Figure 1a). This method is called the Hierarchical Cut Clustering Algorithm. However, it is not clear how to choose α in order to gain a clustering that fits to your application and your demands. Therefore, one motivation to build a whole hierarchy of clusterings is that afterwards you can pick one clustering out of a large number of different clusterings. Thus you can easily choose one that satisfies your demands. Furthermore it might be interesting to have clusterings of different granularity. Flake et al. clustered a citation network of scientific literature. In this example on a high level there was a cluster that mainly contained papers in the field of "Algorithms and Graphics"; on a lower level this cluster was separated into "Constraint Satisfaction", "Machine Learning" and clusters of similar granularity.

Hartmann et al. [4, 5] recently published a dynamic version of the Cut Clustering Algorithm. It can be used to compute a new clustering of a graph after an edge has been modified. Instead of recomputing the complete clustering they base on a clustering of the previous graph but for the same parameter value. Then they only recompute the parts that might change. Thus, their algorithm only covers the update for one single value of α, i.e. a single level in the hierarchy of clusterings. Although they showed that their update algorithms guarantee a certain smoothness, i.e. a similarity between the new clustering and the old one, over a large number of edge modifications the clustering might get trivial. In this case one would like to change the parameter used for the clustering algorithm. But it is not clear how to choose the parameter that produces a clustering that fits best to the demands of the user.

Inspired by the work of Hartmann et al. in this work we focus on a dynamic version of the Hierarchical Cut Clustering Algorithm. After an edge or vertex in the underlying graph has changed our algorithms can compute an updated hierarchy of the graph. We show that this updated hierarchy is what the Hierarchical Cut Clustering Algorithm would compute for the modified graph. Having a whole hierarchy of clusterings in each update step at hand makes it simple to handle the degeneration of a single clustering after a large number of modifications. After each edge modification you can pick the clustering out of the hierarchy that fits best to your application and your demands (see Figure 1).



(a) A hierarchy built using the Hierarchical Cut Clustering Algorithm

(b) After an edge modification the hierarchy is updated. Another level of the hierarchy now satisfies the demands.

(c) Again the hierarchy is updated. This time the previous level still satisfies the demands.

Figure 1: One can pick one of the clusterings in the hierarchy in each step.

In this work we firstly develop a basic algorithm that updates the hierarchy after an edge in the underlying graph has been deleted or inserted. We call this algorithm the Basic Update Algorithm. Afterwards we refine this algorithm using advanced techniques yielding two advanced update algorithms of which one deals with edge deletions and one with edge insertions. All these algorithms build upon maintaining parts of the hierarchy that stay valid. Therefore, one needs to check whether some minimum cuts in the old graph stay minimum cuts in the modified graph. Furthermore, some parts of the hierarchy that do not stay valid can only change in a specific way, e.g. some clusters may not be split up. This can be exploited as well. We also discuss vertex modifications in the underlying graph and how to change the granularity of the hierarchy.

This paper is organized as follows: In the next section we first describe the notation that we use throughout the paper. In Sections 1.2 and 1.3 we introduce the reader to the algorithms developed by Flake et al. and depict their basic ideas. In Chapter 2 we first present the Basic Update Algorithm and then the advanced ones in Chapter 3. In Chapter 4 we analyze the runtime of our algorithms and show their correctness. Chapter 5 contains a discussion of updates after the vertices of the underlying graph have been modified and how to change the hierarchy's granularity. Finally we conclude our work in Chapter 6.

1.1 Notation and preliminaries

Throughout this work we consider an undirected, weighted graph $G = (V, E, c)$ with vertices V, edges E and a non-negative edge weight function c, writing $c(u, v)$ as a shorthand for $c(\{u, v\})$ with $\{u, v\} \in E$.

In this work dynamic modifications of G will concern vertices and edges. An edge modification of G always involves edge $\{b, d\}$, with $c(b, d) = \Delta$, yielding $G^{\oplus}$ if $\{b, d\}$ is newly inserted into G, and $G^{\ominus}$ if it is deleted from G. For simplicity we will not handle changes to the weight of an edge, since this can be emulated by a deletion and an insertion. We further assume G to be connected; if that is not the case one can work on each connected component independently and the results still apply. For the modifications of vertices we postpone the introduction of notation to the corresponding section.

Furthermore we define a *cut* $(S, V \backslash S)$ to partition V into disjoint sets S and $V \backslash S$. The weight of a cut is the sum of the weights of all edges that cross the cut. We will denote it by $c(S, V \backslash S) := \sum_{\{u,v\} \in E, u \in S, v \in V \backslash S} c(u, v)$. Beyond that we use $c(S)$ as an abbreviation for $c(S, V \setminus S)$. A cut is a minimum s-t-cut for a pair of vertices $s, t \in V$ if it (1) separates s and t, i.e. $s \in S$ and $t \in V \setminus S$ or the other way round and (2) it is the cut with smallest weight among the cuts satisfying (1).

A *minimum-cut tree* $\mathrm{M}(G) = (V, E_\mathrm{M}, c_\mathrm{M})$ of G is a tree on V and represents for any vertex pair $\{u, v\} \in \binom{V}{2}$ a minimum u-v-cut in G by the cheapest edge on the unique path between u and v in $\mathrm{M}(G)$. Deleting this edge decomposes the minimum-cut tree into two connected components which represent the partitions of the cut. The weight of this cut is the weight of the edge that has been deleted. Neither must this edge be unique, nor $\mathrm{M}(G)$. For details on minimum-cut trees, see the pioneering work by Gomory and Hu [3] or the simplifications by Gusfield [6].

A *contraction* in G of $N \subseteq V$ means replacing set N by a single super-node and leaving it adjacent to all former adjacencies u of vertices of N, with edge weight equal to the sum of all former edges between N and u.

In the context of graphs, our understanding of a *cut clustering* $\mathcal{C}(G)$ of G is a partition of V into subsets C^i, which define vertex-induced subgraphs, called *clusters*, conforming to the paradigm of *intra-cluster density* and *inter-cluster sparsity.* If we want to further specify the parameter α used to compute a clustering we write $\mathcal{C}_\alpha(G)$ and C^i_α, respectively. However, if there is no need to determine the cluster exactly we omit the upper index i. We use the shorthands $\mathcal{C}$ for $\mathcal{C}(G)$, $\mathcal{C}^\oplus$ for $\mathcal{C}(G^\oplus)$ and $\mathcal{C}^\ominus$ for $\mathcal{C}(G^\ominus)$ and can optionally add a lower index α to these shorthands in order to define which value of α has been used to compute the cut clustering. Regarding a dynamic graph G we particularly designate C^b and C^d containing b and d, respectively, but not both, where b and d are the vertices incident to the modified edge. A cluster containing both b and d will be denoted by $C^{b/d}$.

A *hierarchy* of cut clusterings consists of multiple cut clusterings of the same graph G. The clusterings differ in the choice of the parameter α. In this work we assume a sequence $\alpha_1 > \alpha_2 > \ldots > \alpha_{max}$ to be given. For each α_i there is exactly one level in the hierarchy which we call *level* α_i. As clusters on lower levels are subsets of clusters on higher levels one can look at this hierarchy as a tree-like structure by interpreting the relation “is a subset of” as “is a child of”. In this tree structure we denote the subtree whose root is the cluster C_α, i.e. a cluster on level α, with $\mathrm{T}(C_\alpha)$. Note that $\mathrm{T}(C_\alpha)$ contains C_α as well.

1.2 Cut Clustering Algorithm

Flake et al. proposed two algorithms that are based on minimum-cut trees, the Cut Clustering Algorithm and the Hierarchical Cut Clustering Algorithm. In this section we will cover the Cut Clustering Algorithm that computes a single cut clustering for a specific parameter α (see Algorithm 1). In the following section we present their Hierarchical Cut Clustering Algorithm (see Algorithm 2) that computes a hierarchy of clusterings based on the Cut Clustering Algorithm.

The Cut Clustering Algorithm performs the following steps: First of all an artificial vertex t is added to the graph. Then all vertices are connected to t by weight α (see Figure 2a and 2b). Throughout this work we will denote a graph G expanded by t and the edges connecting t with all $v \in V$ by G_α. We split the algorithm into two subroutines for an easier reuse in the context of the update algorithms presented in the next chapters. The first subroutine, `ExpandGraph`, implements the steps described so far. The second subroutine, `Cluster`, computes a minimum-cut tree for the expanded graph G_α. Afterwards the recently added vertex t is removed from the tree. This decomposes the tree in several connected components. The algorithm returns these connected components as clusters of G (see Figure 2c-2e).

Algorithm 1: Cut Clustering Algorithm

```
   Input: $G(V, E, c)$, $\alpha$
1 $G_\alpha \leftarrow$ ExpandGraph $(G, \alpha)$
2 Return Cluster $(G_\alpha)$
```

Procedure `ExpandGraph`(G, α)

```
1 $V_\alpha \leftarrow V \cup t$
2 For all vertices $v \in V$ do
3     Connect $t$ to $v$ with edge of weight $\alpha$
4 $G_\alpha(V_\alpha, E_\alpha, c_\alpha) \leftarrow$ the expanded graph after connecting $t$ to $V$
5 Return $G_\alpha$
```

The following Definition 1.1 and Lemmas 1.2, 1.3 have been presented in the work of Flake et al. [1]. We recapitulate them because they are necessary to understand the steps performed by the update algorithms we present in the next chapters. For the proofs see [1].

In order to compute the minimum-cut tree of G_α in Line 1 of Procedure `Cluster` one needs to compute minimum cuts for pairs of vertices in V. In general minimum cuts between two vertices are not unique. Thus, Flake et al. introduce communities. Using communities yields unique clusterings (see Theorem 1.5). Furthermore, it offers the possibility to build hierarchies of clusterings.

Definition 1.1 (Community) *Let $G(V, E)$ be an undirected graph and let $s, t \in V$ be two vertices of G. Let (S, T) be a minimum s-t-cut, where $s \in S$ and $t \in T$, that minimizes the size of S. We call S a* community *of s in G with respect to t. For a given community S of s in G we will call $s =: r(S)$ the representative of S. If t is not specified S denotes the community of s in the expanded graph G_α with respect to the artificial vertex t.*

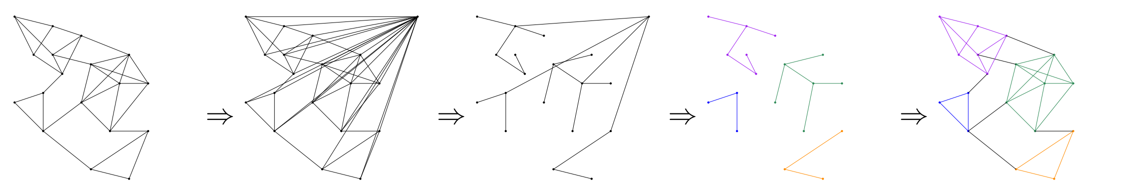
Lemma 1.2 (Communities are unique) *The community of s in G_α with respect to t is unique.*

Lemma 1.3 (Communities do not overlap) *Let $v_1, v_2 \in V$, and S_1, S_2 be their communities with respect to t in G_α. Then either S_1 and S_2 are disjoint or one is a subset of the other.*

The previous lemma shows that communities are either disjoint or they are nested. This is one type of nesting of communities that occur in the context of hierarchical cut clustering. We will introduce the second type of nesting later. Flake at al. show how the first type of nesting of communities can be used to avoid computing the whole minimum-cut tree $\mathrm{M}(G_\alpha)$ and try to only identify those edges of $\mathrm{M}(G_\alpha)$ incident to t. Thus, in line 1 of Procedure `Cluster`, such a partial minimum-cut tree, which is in fact a *star*, would suffice (see Figure 3). Procedure `Simplified Cluster` uses this idea and represents a

Procedure `Cluster`(G_α)

```
1 Compute minimum-cut tree $\mathrm{M}(G_\alpha)$
2 Remove $t$ from $\mathrm{M}(G_\alpha)$
3 Return all resulting connected components
```



(a) G (b) G_α (c) $\mathrm{M}(G_\alpha)$ (d) Deletion of t (e) Clusters of G

Figure 2: Illustration of Algorithm 1.

simplification of Procedure `Cluster`. In the remaining part of this work we will not use Procedure `Cluster` any more but replace it by Procedure `Simplified Cluster`.

The Procedure `Simplified Cluster` gradually separates communities of vertices in V from the artificial vertex t. When all vertices in V are separated from t these communities yield a partitioning of V of which each partition represents a cluster of G. In order to store the separated communities `Simplified Cluster` uses the variable $\mathcal{C}_\alpha$whose initial value is the argument $\mathcal{S}$ of the procedure. It can either be an empty set or a set of communities. This feature will be used by the update algorithms. In some cases the update algorithms know that some communities will stay valid after the modification. Using the argument $\mathcal{S}$ they can hand over this information to the Procedure `Simplified Cluster` and it does not need to recompute these communities. However, in the context of Algorithm 1 the value of $\mathcal{S}$ is always $\emptyset$.

Procedure `Simplified Cluster` iteratively picks one vertex $v \in V$ that is not yet in a community in $\mathcal{C}_\alpha$. Flake et al. suggest to start with vertices that have a high degree. In Line 3 the community C^v of v in G_α with respect to t is computed. The community C^v is then added to $\mathcal{C}_\alpha$ (see Line 4) which will finally be the set of clusters. Afterwards it is checked whether C^v covers any other community that has already been computed earlier. According to Lemma 1.3 it is sufficient to check whether a representative of a community in $\mathcal{C}_\alpha$ is in C^v. These covered communities are deleted from $\mathcal{C}_\alpha$ as they are no clusters in G. Finally after all vertices in V has been separated from t the clustering $\mathcal{C}_\alpha$ is returned.

Observation 1 *We want to call attention to the clusterings that result from the described method. The clusters in these clusterings are all communities of a specific vertex $v \in V$. On the one hand they do not overlap each other according to Lemma 1.3. On the other hand they cover all vertices in V. Thus, a cut clustering is a partition of the vertices of a graph into inclusion-maximal communities.*

In the context of the update algorithms we need to access the representatives of these inclusion-maximal communities. Therefore, we store the representative of each inclusion-maximal community.

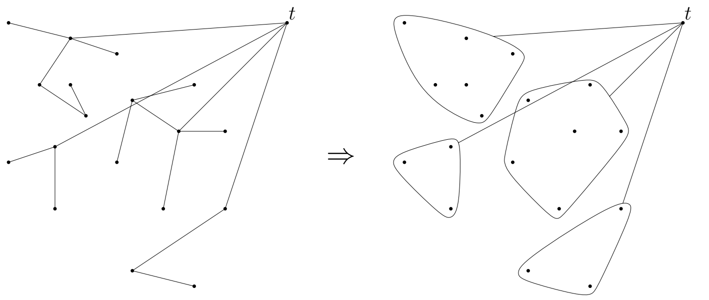
As already mentioned clusterings built with the method described above are unique. We want to state this with Theorem 1.5. But for its proof we firstly need one more lemma.

Lemma 1.4 *Consider a community C in G_α and a vertex $v \in C$. The community C' of v in G_α is a subset of C.*

Proof Assume that C' is not a subset of C. As communities do not overlap according to Lemma 1.3 it holds that C and C' are either disjoint or one contains the other as a subset. Obviously C and C' share at least v excluding the first case. Furthermore, we

Procedure `Simplified Cluster(`G_α, $\mathcal{S}$`)`

```
1 $\mathcal{C}_\alpha \leftarrow \mathcal{S}$
2 For $v \in V, v$ not yet in a community in $\mathcal{C}_\alpha$ do
3     $C^v \leftarrow$ community of $v$ in $G_\alpha$
4     $\mathcal{C}_\alpha \leftarrow \mathcal{C}_\alpha \cup \{C^v\}$
5     For $C \in \mathcal{C}_\alpha$, $r(C) \in C^v$ do
6         $\mathcal{C}_\alpha \leftarrow \mathcal{C}_\alpha \backslash \{C\}$
7 Return $\mathcal{C}_\alpha$
```



(a) A minimum-cut tree of G_α (b) A "minimum-cut star" of G_α

Figure 3: The sets in the star are the sets returned by Procedure `Simplified Cluster`.

assumed that C' is not a subset of C leading to $C \subset C'$. As C' is the community of v we can state that $c_\alpha(C') < c_\alpha(C)$. As the two cuts inducing C and C' separate $r(C)$ and t, and $c_\alpha(C') < c_\alpha(C)$ we found that C could not be the community of $r(C)$ in G_α. ■

Theorem 1.5 *The clustering returned by Procedure* `Simplified Cluster` *applied to an expanded graph G_α and a (possibly empty) set $\mathcal{S}$ of communities in G_α is unique.*

Proof Assume that Procedure `Simplified Cluster` returns two different clusterings $\mathcal{C}_\alpha$ and $\mathcal{C}'_\alpha$ for a graph G. Now consider a vertex $v \in V$ and the clusters C, C' containing v in $\mathcal{C}_\alpha$ and $\mathcal{C}'_\alpha$, respectively, such that $C \neq C'$. We will now distinguish three exhaustive cases:
Case (a): C and C' are disjoint. Since both C and C' contain v we can exclude this case.
Case (b): C and C' overlap but none is a subset of the other one. Remember that C and C' are communities in G. However according to Lemma 1.3 communities do not overlap. Thus, we can exclude this case as well.
Case (c): One is a subset of the other one, w.l.o.g. $C \subseteq C'$. Let $r' := r(C')$ be the representative of C'. If $r' \in C$ this contradicts Lemma 1.4 as C' is no subset of C. Thus C may not contain r', i.e. $r' \notin C$. Now consider the cluster $C^{r'}$ in $\mathcal{C}_\alpha$ that contains r'. According to Lemma 1.4 the community of r' must be a subset of $C^{r'}$. But this contradicts to the fact that C' is the community of r' as $C' \nsubseteq C^{r'}$. Therefore, this case cannot occur as well.
We saw that all three exhaustive cases cannot occur. Thus, there cannot be two different clusterings for one graph, i.e. the clustering computed by Procedure `Simplified Cluster` applied to an expanded graph G_α and a set $\mathcal{S}$ of communities in G_α is unique ■

1.3 Hierarchical Cut Clustering Algorithm

In the foregoing section we did not cover the choice of the parameter α but only discussed scenarios with one fixed α. In this section we will present Flake's results for a sequence of parameter values $\alpha_1 > \alpha_2 > \ldots > \alpha_{max}$. The parameter α influences number, size and shape of the clusters returned by Algorithm 1. Especially the guaranteed quality bounds for inter-cluster and intra-cluster cuts depend on α. The quality measure Flake et al. use for intra-cluster cuts is the so called *expansion*. The expansion of a cut $(S, \overline{S})$ with $S \dot\cup \overline{S} \subseteq V$ has been introduced by Kannan et al. [7] and is defined as:

$$\Psi(S, \overline{S}) = \frac{c(S, \overline{S})}{\min\{|S|, |\overline{S}|\}}$$

In this context S and $\overline{S}$ together do not need to cover V. We measure the weight of the (partial) cut $(S, \overline{S})$ by $c(S, \overline{S}) := \sum_{\{u,v\} \in E, u \in S, v \in \overline{S}} c(u, v)$. Note that for $\overline{S} = V \backslash S$ this definition is compatible with the definition of the cut weight given in Chapter 1.1.

The expansion of a (sub)graph is the minimum expansion over all cuts in the (sub)graph. The quality of a clustering $\mathcal{C}$ of G can be measured by the expansion of its clusters regarded as subgraphs of G: the expansion of a clustering $\mathcal{C}$ is the minimum expansion over all clusters $C \in \mathcal{C}$. The larger the expansion of the clustering the better it is as there are intra-cluster cuts with higher weights conforming to the intra-cluster density paradigm.

For inter-cluster cuts they use a similar measure which is the better the lower it is. Thus, it is compatible with the paradigm of inter-cluster sparsity. For a clustering $\mathcal{C}$ returned by Algorithm 1 the following equation holds. It offers an upper bound for inter-cluster cuts and a lower bound for intra-cluster cuts depending on α.

$$\underbrace{\frac{c(C, V \setminus C)}{|V \setminus C|}}_{\text{inter-cluster cuts}} \leq \alpha \leq \underbrace{\frac{c(P, Q)}{\min\{|P|, |Q|\}}}_{\text{intra-cluster cuts}} \quad \forall C \in \mathcal{C} \quad \forall P, Q \neq \emptyset \quad P \dot\cup Q = C$$

Flake et al. show how one can even build a hierarchy of different clusterings of G by varying α. If α is big enough Algorithm 1 returns a clustering that consists of singletons, i.e. each vertex forms a single cluster. If α is small enough all vertices together form a single cluster. For values in between we get a clustering between these two extremes. Flake et al. state that clusters on lower levels are subsets of the clusters on higher levels (see Figure 4).
Before we come to the nesting of clusters we firstly introduce Lemma 1.6 from Flake et al. that discusses the second type of nesting of communities. It states that for one vertex $v \in V$ the communities for different α_i are nested. We will call the property described in the following lemma the `nesting property`.

Lemma 1.6 (nesting property) *For a vertex s in G_{α_i}, where $\alpha_i \in \{\alpha_1, \ldots, \alpha_{max}\}$, such that $\alpha_1 > \alpha_2 > \ldots > \alpha_{max}$, the communities $S_1, S_2, \ldots, S_{max}$ are such that $S_1 \subseteq S_2 \subseteq \ldots \subseteq S_{max}$ where S_i is the community of s with respect to t in G_{α_i}.*

Proof This lemma is a direct implication of Lemma 2.4 in a work by Gallo et al. [2]. In this lemma set $\lambda_l := \alpha_1, \ldots, \lambda_1 := \alpha_{max}$ and use s as the sink, t as the source. Then for each λ_i there exists a minimum cut $(X_i, \overline{X_i})$ that minimizes the sink's side $\overline{X_i}$ such that $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_l$. Thus in reverse $\overline{X_l} = S_1 \subseteq \overline{X_{l-1}} = S_2 \subseteq \ldots \subseteq \overline{X_1} = S_{max}$. ■

Flake et al. extend the statement of the foregoing lemma about communities to a similar statement about clusters. This lemma is the very basic idea of the work of Flake et al.

Lemma 1.7 (Clusters are nested) *Let $\alpha_1 > \alpha_2 > \ldots > \alpha_{max}$ be a sequence of parameter values that connect the vertices in V to t in G_{α_i}. Let $\alpha_{max+1} \leq \alpha_{max}$ be small enough to yield a single cluster in G by applying `Simplified Cluster` and $\alpha_0 \geq \alpha_1$ be large enough to yield all singletons. Then all α_{i+1} values, for $0 \leq i \leq max$, yield clusters in G that are supersets of the clusters produced by each α_i, and all clusterings together form a hierarchical tree over the clusterings of G.*

Note that in this work we do not require that the hierarchy ranges from all singleton clusters on the lowest level to one single cluster on the highest level. We use any arbitrary sequence $\alpha_1 > \alpha_2 > \ldots > \alpha_{max}$ of parameter values.

We further present the following lemma which supports the proceeding in the Hierarchical Cut Clustering Algorithm by Flake et al. (see Algorithm 2 which we will introduce shortly). Furthermore, it is very important for the update algorithms we present in the following chapters as well.

Lemma 1.8 *Let C' be a subset of a community $C \supseteq C'$ in G_α. The contraction of C' in G_α does not change the result of calling Procedure `Simplified Cluster` with the contracted graph as first argument in comparison to applying it to the uncontracted graph G_α, independently of the value of $\mathcal{S}$.*

Proof Firstly consider the Clustering $\mathcal{C}$ which we obtain by applying Procedure `Simplified Cluster` to the uncontracted graph G_α. Due to the `nesting property` there is a cluster $\overline{C}$ in this clustering with $\overline{C} \supseteq C'$. Let $r := r(\overline{C})$ be the representative of $\overline{C}$. We now show that in the contracted graph there is still a vertex whose community is $\overline{C}$. Therefore, we distinguish two cases:
Case $r \in C'$: In this case the super-node C' is the representative of $\overline{C}$ in the contracted graph. Assume that the community of the super-node C' would differ from $\overline{C}$. Neither its size nor the weight of the cut inducing it would have changed due to the contraction. Therefore, it would have already been the community of r in the uncontracted graph G_α, what yields a contradiction. Thus, there is still a vertex whose community in the contracted graph is $\overline{C}$.
Case $r \in \overline{C} \backslash C'$: Obviously no cut separating r and t might get cheaper due to the contraction of C'. Thus, $\overline{C}$ stays the community of r in the contracted graph.
We already showed that $\overline{C}$ stays a community in the contracted graph. As all the cuts not cutting through C' are not affected by the contraction no other community in the contracted graph will cover $\overline{C}$. Otherwise it would have already covered it in G_α. Thus, the clustering does not change due to the edge modification. ■

We now introduce the method of Flake et al. to compute a hierarchy of cut clusterings as described in Lemma 1.7. The so called Hierarchical Cut Clustering Algorithm (see Algorithm 2) achieves an algorithmic speedup in comparison to recomputing the clusterings from scratch for each α_i as follows: It starts to compute the clustering for the biggest parameter value α_1 by basically applying Algorithm 1. For the computation of all following levels the algorithm can use the result of the previous one. The computation of $\mathcal{C}_{\alpha_i}$ is accelerated by using $\mathcal{C}_{\alpha_{i-1}}$. Remember that the clusters of a cut clustering are communities. According to Lemma 1.6 the clusters on level α_{i-1} are subsets of communities on level α_i. Thus, the contraction of clusters in $\mathcal{C}_{\alpha_{i-1}}$ in G_{α_i} (in Line 4) is feasible due to Lemma 1.8. Contracting them still yields a valid clustering but `Simplified Cluster` is applied to a smaller instance. After the contraction of a cluster $C \in \mathcal{C}_{\alpha_i - 1}$ the edge connecting the resulting super-node with t in G_{α_i} has weight $|C| \cdot \alpha_i$.

Algorithm 2: HIERARCHICAL CUT CLUSTERING ALGORITHM

```
Input: $G(V, E, c)$, $\alpha_1 > \ldots > \alpha_{max}$
1 For $i = 1, \ldots, max$ do
2     $G_{\alpha_i} \leftarrow$ ExpandGraph $(G, \alpha_i)$
3     If $i > 1$
4         Contract all clusters of $\mathcal{C}_{\alpha_{i-1}}$ in $G_{\alpha_i}$
5     $\mathcal{C}_{\alpha_i} \leftarrow$ SimplifiedCluster$(G_{\alpha_i}, \emptyset)$
6 Return $\mathcal{C}_{\alpha_1}, \ldots, \mathcal{C}_{\alpha_{max}}$
```

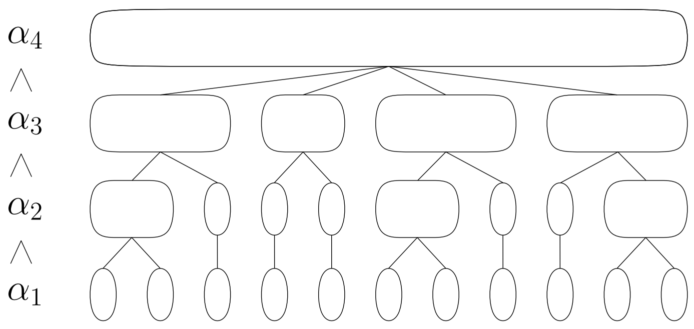


Figure 4: A hierarchy of clusterings returned by Algorithm 2.

Similar to the uniqueness of a single clustering returned by Algorithm 1 we can show that even the whole hierarchy computed by Algorithm 2 is unique.

Corollary 1.9 *The hierarchy of clusters returned by Algorithm 2 is unique.*

Proof As the clusterings on all levels of the hierarchy are unique (see Theorem 1.5) the whole hierarchy is unique as well. ■

2 Basic update techniques

In the foregoing chapter we introduced the algorithms Flake et al. developed in order to build a hierarchy of clusterings. They are restricted to static graphs. In the remainder of this work we want to discuss cut clustering methods that deal with a dynamic scenario. First of all we cover the modification of edges, i.e. edge deletions and insertions. Vertex modifications will be discussed in Chapter 5. The idea of handling edge modifications is to update the previous hierarchy to a valid hierarchy of the modified graph. In this chapter we present our basic update techniques which will be used for a simple but efficient update algorithm. Firstly in this section we cover the theory of updates without distinguishing between edge insertions and deletions. In the Sections 2.1 and 2.2 we discuss these two cases, edge deletion and edge insertion, separately. In both cases we gain similar results that are used for an algorithm which we will call the BASIC UPDATE ALGORITHM. This algorithm is presented at the end of this chapter. In the following chapter we present advanced update techniques that are more complex but more efficient as well.
Firstly we present two lemmas that lead to Theorem 2.3 which states an important idea we will need for the BASIC UPDATE ALGORITHMand the advanced update algorithms as well.

Lemma 2.1 *Consider a cluster C_{α_k} on level α_k. In the hierarchy on the levels $\alpha_1 > \ldots > \alpha_{k-1}$ all vertices contained in C_{α_k} are clustered independently of $V \backslash C_{\alpha_k}$, i.e. there is no cluster C'_{α_i} in one of the clusterings $\mathcal{C}_{\alpha_1}, \ldots, \mathcal{C}_{\alpha_{k-1}}$ that contains a vertex $x \notin C_{\alpha_k}$ and a vertex $y \in C_{\alpha_k}$ at the same time.*

Proof Assume that there is a cluster C'_{α_i} on a level $\alpha_i \in \{\alpha_1, \ldots, \alpha_{k-1}\}$ that contains at least one vertex $x \notin C_{\alpha_k}$ and one vertex $y \in C_{\alpha_k}$. Due to the `nesting property` on level α_k the community C'_{α_k} of $r(C'_{\alpha_i})$ is a superset of C'_{α_i}. Because of Lemma 1.3 cluster C_{α_k} and community C'_{α_k} must either be disjoint or one is a subset of the other one. Since C_{α_k} and C'_{α_i} share y, it follows that C_{α_k} and C'_{α_k} also share y and thus, they cannot be disjoint. Furthermore C'_{α_k} contains a vertex $x \notin C_{\alpha_k}$ what excludes that $C'_{\alpha_k} \subseteq C_{\alpha_k}$. Thus, it holds that $C_{\alpha_k} \subset C'_{\alpha_k}$. But this contradicts the premise that C_{α_k} is a cluster, i.e. it is not a subset of any other community in G_{α_k} as described in Observation 1. ■

The foregoing lemma did not yet cover any dynamic scenario where an edge has been modified. However, the next lemma and the next theorem deal with modified graphs covering the cases of the deletion and the insertion of an edge $\{b, d\}$ at the same time. Theorem 2.3 states that given that a specific part of the hierarchy stays valid one can conclude that other parts of the hierarchy stay valid as well. This is a very central idea of the Basic Update Algorithmand enables us to compute the updated hierarchy faster than applying the Hierarchical Cut Clustering Algorithm from scratch.

Lemma 2.2 *Consider a community C_{α_k} in G_{α_k} such that after the modification C_{α_k} is still the community of its representative in $G^{\oplus/\ominus}_{\alpha_k}$. If C_{α_k} neither contains b nor d, it holds that for a vertex $v \in C_{\alpha_k}$ the community of v in G_{α_i} is still the community of v in $G^{\oplus/\ominus}_{\alpha_i}$ for $\alpha_i \in \{\alpha_1, \ldots, \alpha_{k-1}\}$.*

Proof For $v \in C_{\alpha_k}$ let C_{α_i} be the community of v in G_{α_i} and C'_{α_i} its community in $G^{\oplus/\ominus}_{\alpha_i}$ for an $\alpha_i \in \{\alpha_1, \ldots, \alpha_{k-1}\}$. The community of v in $G^{\oplus/\ominus}_{\alpha_k}$ is a subset of C_{α_k} according to Lemma 1.4. Thus, $C'_{\alpha_i} \subseteq C_{\alpha_k}$ holds due to the `nesting property`.
Since $b, d \notin C_{\alpha_k}$ the subsets C_{α_i} and C'_{α_i} do neither contain b nor d. Thus, the weights of the cuts inducing the two communities C_{α_i} and C'_{α_i} do not change due to the edge modification. If $c_{\alpha_i}(C'_{\alpha_i}) < c_{\alpha_i}(C_{\alpha_i})$ or $c_{\alpha_i}(C'_{\alpha_i}) = c_{\alpha_i}(C_{\alpha_i})$ and $|C'_{\alpha_i}| < |C_{\alpha_i}|$ then C_{α_i} cannot be the community of v in G_{α_i}. Thus, we conclude that $C_{\alpha_i} = C'_{\alpha_i}$. ■

Theorem 2.3 *Consider a cluster C_{α_k} on level α_k that after the edge modification is still a valid cluster. If C_{α_k} neither contains b nor d, then all clusters in the subtree $T(C_{\alpha_k})$ of C_{α_k} in the hierarchy, stay valid clusters on their respective levels of the hierarchy.*

Proof Using Lemma 2.1 we know that the vertices in C_{α_k} are clustered independently of the other vertices. Furthermore by Lemma 2.2 all communities of vertices in C_{α_k} stay communities of their corresponding representative in the updated graphs $G^{\oplus/\ominus}_{\alpha_1}, \ldots, G^{\oplus/\ominus}_{\alpha_{k-1}}$. Thus, the whole subtree $\mathrm{T}(C_{\alpha_k})$ in the hierarchy stays valid. ■

2.1 Edge deletion

In the previous section we introduced statements that are not specific to the deletion or insertion of an edge. In this section we present statements that only hold for the deletion of edges. Firstly we introduce Lemma 2.4 about how cuts in the modified graph may differ from cuts in the original graph. Then we use this lemma to deduce stronger statements in Lemma 2.5 and Theorem 2.6 which is the central idea of the Basic Update Algorithmregarding edge deletions.

Lemma 2.4 *If the community C of a vertex v in G_α is not the community of v in $G_\alpha^\ominus$ any more then the cut inducing the community C' of v in $G_\alpha^\ominus$ separates b and d.*

Proof Assume that the cut inducing C' does not separate b and d. Then the cut weight does not change due to the edge modification, i.e. $c_\alpha(C') = c_\alpha^\ominus(C')$. Furthermore it holds that $c_\alpha^\ominus(C) \leq c_\alpha(C)$. Since C' is the community of v we know that $c_\alpha^\ominus(C') < c_\alpha^\ominus(C)$ or $c_\alpha^\ominus(C') = c_\alpha^\ominus(C)$ and $|C'| < |C|$. We will now distinguish these two cases and will see that both contradict the correctness of C as v's community in G_α.
(a) $c_\alpha^\ominus(C') < c_\alpha^\ominus(C)$: Then $c_\alpha(C') < c_\alpha(C)$ and C cannot be the community of v in G_α.
(b) $c_\alpha^\ominus(C') = c_\alpha^\ominus(C)$ and $|C'| < |C|$: Then $c_\alpha(C') \leq c_\alpha(C)$ and $|C'| < |C|$ and again C cannot be the community of v in G_α. ■

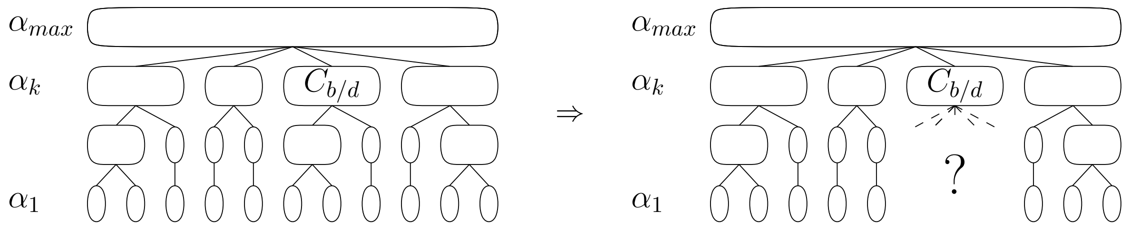
Lemma 2.5 *If in the case of an intra-cluster edge deletion the cluster $C^{b/d}$ in $\mathcal{C}$ stays the community of its representative in the modified expanded graph $G_\alpha^\ominus$ then the whole clustering of G stays valid for $G^\ominus$.*

Proof Consider a cluster $C \neq C^{b/d}$ in G_α that is not the community of its representative in $G_\alpha^\ominus$ any more. According to Lemma 2.4 the cut inducing the community in $G_\alpha^\ominus$ needs to separate b and d and thus, cut through the cluster $C^{b/d}$. However, this is forbidden by Lemma 1.3 as communities do not overlap. Thus, all clusters in G_α stay communities for their representatives in $G_\alpha^\ominus$ and the clustering of G is also valid for $G^\ominus$. ■

Lemma 2.5 is a very important statement in the context of an update after an edge deletion: Recalculating only one minimum cut can be sufficient in order to know that a whole level of the hierarchy does not change. This already allows a massive computational speedup. In the following theorem we extend the last lemma together with Theorem 2.3 to the very central idea of the Basic Update Algorithm.

Theorem 2.6 *In the case of an edge deletion let α_k denote the lowest level such that on the levels $\alpha_i \in \{\alpha_k, \dots, \alpha_{max}\}$ the vertices b and d lie in the same cluster and the cluster $C_{\alpha_i}^{b/d}$ stays the community of its representative in the modified graph. Then apart from the subtree $T(C_{\alpha_k}^{b/d})$ of $C_{\alpha_k}^{b/d}$ the hierarchy does not change (see Figure 5).*

Proof Immediate by combination of Lemma 2.5 and Theorem 2.3 applied to all other clusters on level α_k. ■



(a) The previous hierarchy

(b) The updated hierarchy

Figure 5: Illustration of Theorem 2.6: Apart from the subtree $\mathrm{T}(C_{\alpha_k}^{b/d})$ the hierarchy does not change.

2.2 Edge insertion

After covering the case of an edge deletion in detail we now introduce some lemmas that are specific to edge insertions. Again we start with a lemma which is only necessary for the following statements in Observation 2 and Corollary 2.8 that gradually lead to Theorem 2.9 which is the correspondent to Theorem 2.6.

Lemma 2.7 *Consider the community C of a vertex v in the graph G_α. If the cut θ inducing this community stays a minimum v-t-cut in $G_\alpha^\oplus$, then C is the community of v in $G_\alpha^\oplus$.*

Proof Assume that C is no longer the community of v in $G_\alpha^\oplus$. We know that $c_\alpha(\theta) = c_\alpha^\oplus(\theta)$ and $c_\alpha(\theta') \leq c_\alpha^\oplus(\theta')$ as the cut weight may only increase by the insertion of an edge. Then there exists a cut θ' separating v and t that is easier than θ or a cut θ' that is of at most the same weight but v's side is smaller than before.
First we cover the case $c_\alpha^\oplus(\theta') < c_\alpha^\oplus(\theta)$: Together with the formulas stated above we can derive $c_\alpha(\theta') \leq c_\alpha^\oplus(\theta') < c_\alpha^\oplus(\theta) = c_\alpha^\oplus(\theta)$, simplified $c_\alpha(\theta') < c_\alpha^\oplus(\theta)$. Thus θ' induces a community of v in G_α that is easier than C. This is a contradiction.
In the other case $c_\alpha^\oplus(\theta') = c_\alpha^\oplus(\theta)$ but v's side of θ' is smaller than v's side of θ. Then by transitivity $c_\alpha(\theta') \leq c_\alpha(\theta)$. Thus, θ' induces a community of v in G_α that may be easier and is definitely smaller than C. Again this is a contradiction ■

Using this lemma it is easy to deduce the following observation and corollary about communities and levels of the hierarchy that stay valid.

Observation 2 *A community C in G_α that neither contains b nor d stays a valid community for its representative in $G_\alpha^\oplus$ as $c^\oplus(C) = c(C)$. Analogously, a community C in G_α that contains both, b and d, stays a valid community for its representative in $G_\alpha^\oplus$ as $c^\oplus(C) = c(C)$.*

Corollary 2.8 *In the case of an intra-luster edge insertion, i.e. b and d lie in the same cluster, the clustering $\mathcal{C}_\alpha$ of G stays a valid clustering of $G^\oplus$ in respect to α.*

Proof Using Observation 2 we know that all clusters in $\mathcal{C}_\alpha$ stay valid communities. Thus, $\mathcal{C}_\alpha$ is as well a valid clustering of $G^\oplus$. ■

We combine all the foregoing statements in Theorem 2.9 which is the correspondent to Theorem 2.6.

Theorem 2.9 *In the case of an edge insertion let α_k be the greatest α_i (the lowest level in the hierarchy) where b and d lie in the same cluster. Then in the updated hierarchy only the clustering in the subtree of $C_{\alpha_k}^{b/d}$ may change (see Figure 5).*

Proof Immediate by combination of Corollary 2.8, Lemma 2.1 applied to $C_{\alpha_k}^{b/d}$ and Theorem 2.3 applied to all other clusters on level α_k. ■

In the next section we present the Basic Update Algorithm that bases on the ideas presented in Theorem 2.6 and Theorem 2.9.

2.3 Basic update algorithm

In the foregoing two sections we have shown very similar results for the both cases, edge deletion and edge insertion. Theorem 2.6 and 2.9 state that under specific circumstances some levels on the top of the hierarchy stay valid and that parts of the lower levels stay valid. We now present the Basic Update Algorithm.

Algorithm 3: Basic Update Algorithm

```
Input: $G^{\oplus/\ominus}(V, E^{\oplus/\ominus}, c^{\oplus/\ominus}), \alpha_1 > \ldots > \alpha_{max}, \mathcal{C}_{\alpha_1}, \ldots, \mathcal{C}_{\alpha_{max}}$
1 If an edge has been inserted                                  // Case $\oplus$
2     Let $k$ be the smallest index such that in $\mathcal{C}_{\alpha_k}$ $b, d$ are in the same cluster
3 Else                                                          // Case $\ominus$
4     Let $k$ be the smallest index such that in $\mathcal{C}_{\alpha_k}$ $b, d$ are in the same cluster and
      $C_{\alpha_k}^{b/d}$ stays the community of its representative in $G_{\alpha_k}^{\oplus/\ominus}$
5 For $i = k, \ldots, max$ do
6     $\mathcal{C}_{\alpha_i}^{\oplus/\ominus} \leftarrow \mathcal{C}_{\alpha_i}$               // Theorem 2.6 or Theorem 2.9
7 Compute $\mathcal{C}_{\alpha_i}^{\oplus/\ominus}$, $i = 1, \ldots, k-1$ using that parts of the clustering do not change
```

Algorithm 3 first of all computes the level that is denoted by level α_k in Theorem 2.6 and 2.9. This can be done by going top down and checking the condition on each level. Then all levels above level α_k (including level α_k) do not change according to the theorems. Thus, the clusterings for these levels are directly carried over in Line 6. For the lower levels the pseudo code does not specify how to compute the updated clusterings but states that one should exploit the fact that only parts of the previous clusterings may change. According to Theorem 2.6 and 2.9 these are the clusters in the subtree $\mathrm{T}(C_{\alpha_k}^{b/d})$. In this work we present two possibilities how to proceed Line 7 of Algorithm 3.

The first possibility is to use the Hierarchical Cut Clustering Algorithm presented by Flake et al. in a very basic manner. One could replace Line 7 of Algorithm 3 by the call Hierarchical Cut Clustering Algorithm($G^{\oplus/\ominus}, \alpha_1 > \ldots > \alpha_{k-1}$). In this basic version the fact that some clusters stay valid on the lower levels would not be used. But one can extend the Hierarchical Cut Clustering Algorithm such that it uses this information. Therefore, after Line 4 in Algorithm 2 all clusters that stay valid need to be contracted into one single super-node in the expanded graph. This contraction does not affect the clustering of the remaining part (the part of the graph whose clustering may change) as no cluster in the contracted part can be covered by any community in the remaining part of the graph. Therefore, the contracted super-node will form a singleton in the clustering and we can already add it to $\mathcal{S}$. The updated clustering can then be composed of the parts of the previous clustering that stay valid and the clustering computed by the modified Hierarchical Cut Clustering Algorithm. The Basic Update Algorithm is already more efficient in terms of minimum cut calculations than recomputing the whole hierarchy using the Hierarchical Cut Clustering Algorithm. For a runtime analysis see Chapter 4.

The second possibility is to use the advanced techniques we present in the next chapter.

3 Advanced techniques

In this chapter we refine the Basic Update Algorithm we presented in Chapter 2. As these advanced techniques differ for the two cases, edge deletion and edge insertion, we developed two algorithms of which each is specific to one case. They extend the Basic Update Algorithm by discussing how to proceed Line 7 of Algorithm 3. However there are procedures that are used by both update algorithms. These are the Procedures `Prepare` and `Rework` which are covered in the following section about edge deletion.

3.1 Edge deletion

In this section we present lemmas that only hold for the deletion of edges. Afterwards we depict the algorithm that updates the hierarchy of clusters after an edge has been deleted. The first lemma states that in the case of an inter-cluster edge deletion the communities containing either b or d stay valid communities of their representatives. This will later be exploited in two ways. On the one hand one can contract these communities in the expanded graph. On the other hand the Procedure `Simplified Cluster` does not need to recompute these communities, i.e. we can add them to $\mathcal{S}$.

Lemma 3.1 *In the case of an inter-cluster edge deletion communities in G_α containing either b or d stay communities of its representative in $G_\alpha^\ominus$.*

Proof Consider a community C in G_α with w.l.o.g. $b \in C$ and its representative $v := r(C)$. Furthermore assume that there is an easier or smaller community C' of v in $G_\alpha^\ominus$.
First we assume $c_\alpha^\ominus(C') < c_\alpha^\ominus(C) = c_\alpha(C) - \Delta$. We distinguish the two cases that either $b \in C'$ or $d \in C'$ and that none of them or both are in C'. If either $b \in C'$ or $d \in C'$ then $c_\alpha^\ominus(C') = c_\alpha(C') - \Delta$ and thus, $c_\alpha(C') < c_\alpha(C)$. However, this contradicts the fact that C was the community of v in G_α. Otherwise if none of them or both are in C' then $c_\alpha^\ominus(C') = c_\alpha(C')$ and $c_\alpha(C') < c_\alpha(C) - \Delta < c_\alpha(C)$, and again C could not be the community of v in G_α.
Now let us assume that $c_\alpha^\ominus(C') = c_\alpha^\ominus(C) = c_\alpha(C) - \Delta$ but $|C'| < |C|$. Again we distinguish the two cases mentioned above. If either $b \in C'$ or $d \in C'$ then $c_\alpha^\ominus(C') = c_\alpha(C') - \Delta$ and thus, $c_\alpha(C') = c_\alpha(C)$. In this case $|C'| < |C|$ contradicts the fact that C is the community of v in G_α. If none of them or both are in C' then $c_\alpha^\ominus(C') = c_\alpha(C')$ and therefore, $c_\alpha(C') = c_\alpha(C) - \Delta$, i.e. $c_\alpha(C') < c_\alpha(C)$. Again this contradicts the fact that C is the community of v in G_α. ■

Furthermore we need the following lemma that is similar to Lemma 5 in Hartmann et al. [5]. This lemma states that the communities neither containing b nor d stay subsets of communities after the modification. According to Lemma 1.8 we can contract these sets of vertices as well. In contrast to the foregoing lemma the Procedure `Simplified Cluster` needs to recompute their communities, i.e. we cannot add them to $\mathcal{S}$.

Lemma 3.2 *Let C be the community of a vertex v in G_α (solid black in Figure 6) and C' its community in $G_\alpha^\ominus$. If C neither contains b nor d then C' is a superset of C (blue dotted).*

Proof Assume that C' (red dashed in Figure 6) is not a superset of C. We show that C cannot be the community of v in G_α because $C \cap C'$ is induced by a cut of at most weight $c_\alpha(C)$ in G_α but is smaller than C.

Since C' is induced by a minimum v-t-cut in $G_\alpha^\ominus$ we know that $c_\alpha^\ominus(C') \le c_\alpha^\ominus(C \cup C')$.

Furthermore it certainly holds that $c_\alpha^\ominus(C' \backslash C, V_\alpha \backslash C') \ge c_\alpha^\ominus(C' \backslash C, V_\alpha \backslash (C \cup C'))$ and $c_\alpha^\ominus(C \cap C', C' \backslash C) \le c_\alpha^\ominus(C, C' \backslash C)$ as the cut weight may only increase by cutting more edges.

Now we express $c_\alpha(C \cap C')$ and $c_\alpha(C)$ in terms of other partial cut weights. As both cuts do not separate b and d their weights are not affected by the edge deletion and thus, we can use the weight function $c_\alpha^\ominus$ on the right sides of the following two equations whereas we use c_α on the left side:

$$c_\alpha(C \cap C', V_\alpha \backslash (C \cap C')) = c_\alpha^\ominus(C', V_\alpha \backslash C') - c_\alpha^\ominus(C' \backslash C, V_\alpha \backslash C') + c_\alpha^\ominus(C \cap C', C' \backslash C)$$

$$c_\alpha(C, V_\alpha \backslash C) = c_\alpha^\ominus(C \cup C', V_\alpha \backslash (C \cup C')) - c_\alpha^\ominus(C' \backslash C, V_\alpha \backslash (C \cup C')) + c_\alpha^\ominus(C, C' \backslash C)$$

By subtracting the second equation from the first one we get:

$$\begin{aligned} c_\alpha(C \cap C', V_\alpha \backslash (C \cap C')) - c_\alpha(C, V_\alpha \backslash C) &= (c_\alpha^\ominus(C', V_\alpha \backslash C') - c_\alpha^\ominus(C \cup C', V_\alpha \backslash (C \cup C'))) \\ &- (c_\alpha^\ominus(C' \backslash C, V_\alpha \backslash C') - c_\alpha^\ominus(C' \backslash C, V_\alpha \backslash (C \cup C'))) \\ &+ (c_\alpha^\ominus(C \cap C', C' \backslash C) - c_\alpha^\ominus(C, C' \backslash C)) \\ &\le 0 \end{aligned}$$

We conclude that $c_\alpha(C \cap C') \le c_\alpha(C)$. However, this contradicts the correctness of C being the community of v in G_α as the cut inducing $C \cap C'$ is of at most the same weight and v's side is smaller than C. Thus, C' must be a superset of C. ■

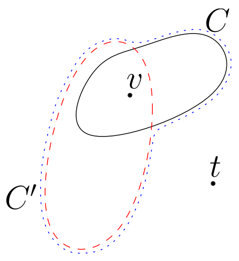


Figure 6: If C neither contains b nor d the community of $r(C)$ in $G_\alpha^\ominus$ is a superset of C.

Lemma 3.1 and 3.2 are needed in order to refine the BASIC UPDATE ALGORITHM. In the following we will describe the advanced update algorithm for edge deletions (Algorithm 4) in detail and will thereby come back to these two lemmas. First we give a rough sketch of the algorithm:
Looking at Algorithm 4 one notes that Line 1-3 correspond to Line 4-6 of Algorithm 3. In the following lines the lower levels, the levels where the clustering may change, are recomputed. Therefore, the clusterings on these levels are recalculated iteratively top down. Lemma 2.2 and Theorem 2.3 are central for this proceeding as they describe how information from an update on a higher level can be used to achieve a speedup for the recomputation of the clustering on lower levels:

Procedure `Prepare`($G^{\oplus/\ominus}$, α, $\mathcal{C}_\alpha$, V', W')

```
1  $G_\alpha^{\oplus/\ominus} \leftarrow$ ExpandGraph $(G^{\oplus/\ominus}, \alpha)$
2  For $C \in \mathcal{C}_\alpha, r(C) \in V'$
3      Add $C$ to $\mathcal{C}_\alpha^{\oplus/\ominus}$
4  Contract $V'$ in $G_\alpha^{\oplus/\ominus}$
5  $\mathcal{S} \leftarrow \{V'\}$
6  For $C \in \mathcal{C}_\alpha \backslash \mathcal{C}_\alpha^{\oplus/\ominus}, r(C) \in W'$
7      $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$
8  Return $(G_\alpha^{\oplus/\ominus}, \mathcal{C}_\alpha^{\oplus/\ominus}, \mathcal{S})$
```

Procedure `Rework`($G_\alpha^{\oplus/\ominus}$, $\mathcal{C}_\alpha^{\oplus/\ominus}$, $\mathcal{C}_\alpha$, V', W', $\mathcal{S}$)

```
1  $\mathcal{C}'_\alpha \leftarrow$ SimplifiedCluster($G_\alpha^{\oplus/\ominus}, \mathcal{S}$)
2  For $C \in \mathcal{C}'_\alpha, r(C) \in V \backslash V'$
3      Add $C$ to $\mathcal{C}_\alpha^{\oplus/\ominus}$
4  For $C \in \mathcal{C}_\alpha, b \notin C, d \notin C$
5      If $C \in \mathcal{C}_\alpha^{\oplus/\ominus}$
6          $V' \leftarrow V' \cup C$                    // Theorem 2.3 applied to $C$
7          $W' \leftarrow W' \backslash C$
8      Else If $C$ is still the community of its representative in $G_\alpha^{\oplus/\ominus}$
9          $W' \leftarrow W' \cup C$                    // Lemma 2.2 applied to $C$
10 Return $(\mathcal{C}_\alpha^{\oplus/\ominus}, V', W')$
```

Lemma 2.2 states that if a cluster C neither containing b nor d stays the community of its representative on level α_i then all the communities of vertices in C on all lower levels stay the communities for their corresponding representative. Lemma 1.8 allows us to contract each cluster that stays the community of its representative into one super-node in the expanded graph G_{α_i}. Furthermore, we do not need to recompute these communities. Therefore, we will hand over this information to the Procedure `Simplified Cluster` using its second argument $\mathcal{S}$.
Theorem 2.3 says that if a cluster C neither containing b nor d stays a cluster in the modified graph then all clusters in $\mathrm{T}(C)$, its subtree in the hierarchy, stay valid clusters on their corresponding level. Thus, no new cut will cut through one of these clusters or between two of these clusters. Even if several clusters stay valid then on a lower level no two clusters belonging to different subtrees in the hierarchy can be split by a newly found minimum cut. Therefore, all clusters that stay valid clusters in the modified graph can be contracted into one single super-node in G_{α_i}.
But how can we detect that a cluster that neither contains b nor d stays the community of its representative or even stays a cluster in the modified graph? We can do this during the computation of the updated clustering in `Rework` where we use the following heuristic in the Procedure `Simplified Cluster`: First compute the communities in $G_\alpha^{\oplus/\ominus}$ that belong to the representatives of clusters in $\mathcal{C}_\alpha$. If the community of v in $G_\alpha^{\oplus/\ominus}$ neither contains b nor d and the cut inducing it has the same weight as the cut inducing v's community in G_α then we already know that v's community does not change as both cut weights are not affected by the edge modification. If we detect a community that does not change we mark it with a cluster flag. Note that the communities in $\mathcal{S}$ need to be marked with the cluster flag as well as they are also communities that stay valid. If later in Line 6 a marked

Algorithm 4: Update on edge deletion

```
   Input: $G^{\ominus}(V, E\backslash\{\{b,d\}\}, c^{\ominus})$, $\alpha_1 > \ldots > \alpha_{max}$, $\mathcal{C}_{\alpha_1}, \ldots, \mathcal{C}_{\alpha_{max}}$
1  Let $k$ be the smallest index such that in $\mathcal{C}_{\alpha_k}$ $b, d$ are in the same cluster and $C^{b/d}_{\alpha_k}$
   stays the community of its representative in $G^{\ominus}_{\alpha_k}$
2  For $i = k, \ldots, max$                                  // Carry over $\mathcal{C}^{\ominus}_{\alpha_k}, \ldots, \mathcal{C}^{\ominus}_{\alpha_{max}}$
3      $\mathcal{C}^{\ominus}_{\alpha_i} \leftarrow \mathcal{C}_{\alpha_i}$                                          // Theorem 2.6
4  $V' \leftarrow V\backslash C^{b/d}_{\alpha_k}$          // Theorem 2.3 applied to clusters in $\mathcal{C}_{\alpha_k}\backslash\{C^{b/d}_{\alpha_k}\}$
5  $W' \leftarrow \emptyset$
6  For $i = k-1, \ldots, 1$
7      $(G^{\ominus}_{\alpha_i}, \mathcal{C}^{\ominus}_{\alpha_i}, \mathcal{S}) \leftarrow$ Prepare $(G^{\ominus}, \alpha_i, \mathcal{C}_{\alpha_i}, V', W')$
8      If $b, d$ are in different clusters in $\mathcal{C}_{\alpha_i}$
9          Contract $C^b_{\alpha_i}$ and $C^d_{\alpha_i}$ in $G^{\ominus}_{\alpha_i}$      // Lemma 3.1 applied to $C^b_{\alpha_i}$ and $C^d_{\alpha_i}$
10         $\mathcal{S} \leftarrow \mathcal{S} \cup \{C^b_{\alpha_i}, C^d_{\alpha_i}\}$
11     Else
12         $\alpha_j \leftarrow$ the highest level where $b$ and $d$ are in different clusters in $\mathcal{C}_{\alpha_k}$
           Contract $C^b_{\alpha_j}$ and $C^d_{\alpha_j}$ in $G^{\ominus}_{\alpha_i}$      // Lemma 3.1 applied to $C^b_{\alpha_j}$ and $C^d_{\alpha_j}$
13     For $C \in \mathcal{C}_{\alpha_i}, b \notin C, d \notin C$
14         Contract $C$ in $G^{\ominus}_{\alpha_i}$                          // Lemma 3.2 applied to $C$
15     $(\mathcal{C}^{\ominus}_{\alpha_i}, V', W') \leftarrow$ Rework$(G^{\ominus}_{\alpha_i}, \mathcal{C}^{\ominus}_{\alpha_i}, \mathcal{C}_{\alpha_i}, V', W', \mathcal{S})$
16 Return $\mathcal{C}^{\ominus}_{\alpha_1}, \ldots, \mathcal{C}^{\ominus}_{\alpha_{max}}$
```

community is removed from $\mathcal{C}_\alpha$ as it is covered by a newly found community we unmark the covered community again. At the end of Procedure `Simplified Cluster` all clusters marked with the cluster flag have already been clusters in the previous clustering. Using this approach no extra computations are needed to determine whether a cluster C of the previous clustering stays a cluster or at least the community of its representative.
Both advanced update algorithms, the algorithm for updates after edge deletions and the algorithm for update after edge insertions, use two mutually exclusive sets V' and W' in order to store which clusters and which communities do not change and thus, can be contracted. The set V' contains the vertices that can be contracted into a single super-node as the clusters covering V' on all lower levels will stay valid. Thus, if the representative of a cluster is in V' then the cluster will also be part of the updated clustering. In W' we store all vertices from which we know that their communities are still their communities in the modified graph on all lower levels and that are not in V', i.e. not in a cluster that stays valid.
According to Theorem 2.6 and 2.9 all clusters on level α_k stay valid. Theorem 2.3 can be applied to all of these clusters that neither contain b nor d. Thus, we can initialize V' with $V\backslash C^{b/d}_{\alpha_k}$ in Line 4. However, W' is initially empty as there is no information about clusters that stay communities of their representatives.

We now explain Algorithm 4 step by step and thereby explain the Procedures `Prepare` and `Rework`, which are also used by the update algorithm for edge insertions (Algorithm 5). This is the reason for choosing the notation $^{\oplus/\ominus}$ in the Procedures `Prepare` and `Rework`. As already mentioned Algorithm 4 updates the hierarchy going top down. In each iteration, i.e. for each α_i, first of all the subroutine `Prepare` is called in Line 7.

`Prepare` firstly expands the graph by adding the artificial vertex t and connecting it to all other vertices by weight α_i. Then in Line 2 and 3 all clusters from $\mathcal{C}_{\alpha_i}$ that stay valid are added to $\mathcal{C}^{\oplus/\ominus}_{\alpha_i}$. Therefore we need to check whether the representative of a cluster is in V'. Then V' is contracted into one single super-node in $G^{\oplus/\ominus}_{\alpha_i}$ in Line 4. We will justify

the correctness of the clusters' validity and the contraction when we further modify V' in `Rework`. Furthermore we initialize $\mathcal{S}$ with $\{V'\}$. The set $\mathcal{S}$ contains the communities we will later hand over to `Simplified Cluster` and thus, need not be computed by `Simplified Cluster`. As no other community in the modified graph will cover V' the super-node containing V' will be a community.
Afterwards we add all clusters that we found to stay valid communities of their representatives in G_{α_i} to $\mathcal{S}$ in Line 6. To this end we need to check whether the representative of a cluster of the previous clustering $\mathcal{C}_{\alpha_i}$ is in W'. If yes we add the cluster to $\mathcal{S}$ as well. Again we will justify this proceeding in the context of the modification of W' in `Rework`.

After the preparation in Algorithm 4 we proceed contractions that are specific to edge deletions. From Lemma 3.1 we know that if b and d are not in the same clusters the clusters containing b and d, respectively, stay valid communities of their representatives. Thus, according to Lemma 1.8 we contract them in Line 9 and add them to $\mathcal{S}$ as well.
Otherwise we look for the highest level α_j, $j < i$ where b and d lie in different clusters. Again we can contract $C^b_{\alpha_j}$ and $C^d_{\alpha_j}$ in $G^{\ominus}_{\alpha_i}$ in Line 12. Lemma 3.1 states that $C^b_{\alpha_j}$ and $C^d_{\alpha_j}$ stay communities for their representatives in $G^{\ominus}_{\alpha_j}$. According to the `nesting property` they are subsets of communities in $G^{\ominus}_{\alpha_i}$. Using Lemma 1.8 we can justify the contraction of $C^b_{\alpha_j}$ and $C^d_{\alpha_j}$ in $G^{\ominus}_{\alpha_i}$. But as they are only subsets of communities in $G^{\ominus}_{\alpha_i}$ and not necessarily communities we may not add them to $\mathcal{S}$. Furthermore we are not allowed to add these communities to W' as Lemma 3.1 gives no guarantee for the lower levels.

Furthermore we contract each cluster in $\mathcal{C}_{\alpha_i}$ that contains neither b nor d into a super-node. According to Lemma 3.2 we know that they stay subsets of communities in G_{α_i} and using Lemma 1.8 we can contract them in Line 14.

In the last step of an iteration the subroutine `Rework` is called. This procedure is as well used by the algorithm for edge insertions (Algorithm 5). At first it clusters the graph $G^{\oplus}_{\alpha_i}$ with all contractions being proceeded using the Procedure `Simplified Cluster` presented in Section 1.2 yielding $\mathcal{C}'_{\alpha_i}$. Thereby the set $\mathcal{S}$ of communities we already know is used to speed up the computation of the clustering. During the computation of the updated clustering we detect which clusters stay valid communities or even clusters in the modified graph.
Only those parts of $\mathcal{C}'_{\alpha_i}$ covering $V \backslash V'$ are of interest as we already know the clustering of V'. Thus, we complete the intermediate clustering $\mathcal{C}^{\ominus}_{\alpha_i}$ by adding the missing clusters from $\mathcal{C}'_{\alpha_i}$ in Line 3.
In the following lines V' and W' are modified. If in Procedure `Simplified Cluster` we noticed that a cluster C of the previous clustering is as well in the updated clustering, i.e. it is marked with the cluster flag, we use Theorem 2.3 in order to deduce that the whole subtree $\mathrm{T}(C)$ in the hierarchy stays valid. Thus, we can exclude these clusters from later calculations in `Simplified Cluster` by adding them to V' such that they are contracted into a single super-node in the next iterations. If C is a subset of W' we need to remove it from W' in order to keep V' and W' mutually exclusive.
However if C does not stay a valid cluster but a valid community for its representative Lemma 2.2 states that all communities of vertices in C on the lower levels stay communities for their representatives in the updated graph. Thus, in the following iterations in Line 6 of Procedure `Prepare` these communities will directly be added to $\mathcal{S}$. To this end we need to add C to W'.

After the iterative computation of all $\mathcal{C}^{\ominus}_{\alpha_i}$ Algorithm 4 returns the sequence of all updated clusterings that together form the updated hierarchy.

3.2 Edge insertion

In this section we present the algorithm that updates the hierarchy of clusters after an edge has been inserted. For this algorithm we do not need any further lemmas but rely on the lemmas discussed in Section 2.2.

Algorithm 5 works quite analog to the update algorithm for edge deletions we presented in the previous section (Algorithm 4). In particular both algorithms, Algorithm 4 and Algorithm 5, use the subroutines `Prepare` and `Rework`. We refer the reader to Section 3.1 for their description. Furthermore Line 1-4 of Algorithm 5 are equal to Line 1-4 of Algorithm 4 except the choice of k.

However, in contrast to the case of an edge deletion we know that a lot of communities do not change due to the edge insertion. Observation 2 says that all communities that neither contain b nor d will not change. Thus, we can set W' to $V \backslash (C^b_{\alpha_i} \cup C^d_{\alpha_i})$ in the very beginning of each iteration, right before the call of `Prepare`. Then the Procedure `Prepare` adds all these communities in $G^{\oplus}_{\alpha_i}$ to $\mathcal{S}$.

After the Procedure `Prepare` has been called we proceed contractions that are specific to edge insertions. According to Observation 2 we can contract the clusters that neither contain b nor d in Line 9. These are the communities Procedure `Prepare` already added to $\mathcal{S}$. For the clusters that contain either b or d on the current level α_i, i.e. for $C^b_{\alpha_i}$ and $C^b_{\alpha_i}$, we do not know whether they are still the communities of their representatives in $G^{\oplus}_{\alpha_i}$. However, according to Observation 2 we know that all clusters in $\mathrm{T}(C^b_{\alpha_i})$ and $\mathrm{T}(C^b_{\alpha_i})$, respectively, that neither contain b nor d stay valid communities on their levels and thus, subsets of communities in $G^{\oplus}_{\alpha_i}$ due to the `nesting property`. According to Lemma 1.8 we can contract these clusters in G_{α_i} as well (see Line 11).

Finally in Line 12 the clustering $\mathcal{C}^{\oplus}_{\alpha_i}$ is extended by the clusters that cover $V \backslash V'$ and V' and W' are updated by the subroutine `Rework`. Note that the update of W' does not matter in this context as it is overwritten at the beginning of the next iteration. After going through all levels from α_{k-1} to α_1 Algorithm 5 returns the updated hierarchy of clusterings.

Algorithm 5: Update on edge insertion

```
   Input: $G^{\oplus}(V, E \cup \{\{b,d\}\}, c^{\oplus})$, $\alpha_1 > \ldots > \alpha_{max}$, $\mathcal{C}_{\alpha_1}, \ldots, \mathcal{C}_{\alpha_{max}}$
 1 Let $k$ be the smallest index such that in $\mathcal{C}_{\alpha_k}$ $b, d$ are in the same cluster
 2 For $i = k, \ldots, max$                                   // Carry over $\mathcal{C}^{\oplus}_{\alpha_k}, \ldots, \mathcal{C}^{\oplus}_{\alpha_{max}}$
 3     $\mathcal{C}^{\oplus}_{\alpha_i} \leftarrow \mathcal{C}_{\alpha_i}$                                        // Theorem 2.6
 4 $V' \leftarrow V \backslash C^{b/d}_{\alpha_k}$            // Theorem 2.3 applied to clusters in $\mathcal{C}_{\alpha_k} \backslash \{C^{b/d}_{\alpha_k}\}$
 5 For $i = k-1, \ldots, 1$                                    // Compute $\mathcal{C}^{\oplus}_{\alpha_1}, \ldots, \mathcal{C}^{\oplus}_{\alpha_{k-1}}$
 6     $W' \leftarrow V \backslash (C^b_{\alpha_i} \cup C^d_{\alpha_i})$
 7     $(G^{\oplus}_{\alpha_i}, \mathcal{C}^{\oplus}_{\alpha_i}, \mathcal{S}) \leftarrow$ Prepare $(G^{\oplus}, \alpha_i, \mathcal{C}_{\alpha_i}, V', W')$
 8     For $C \in \mathcal{C}_{\alpha_i}, b \notin C, d \notin C$
 9         Contract $C$ in $G^{\oplus}_{\alpha_i}$                   // Observation 2 applied to $C$
10     For $C \in T(C^b_{\alpha_i}) \cup T(C^d_{\alpha_i}), b \notin C, d \notin C$
11         Contract $C$ in $G^{\oplus}_{\alpha_i}$                   // Observation 2 applied to $C$
12     $(\mathcal{C}^{\oplus}_{\alpha_i}, V', W') \leftarrow$ Rework($G^{\oplus}_{\alpha_i}, \mathcal{C}^{\oplus}_{\alpha_i}, \mathcal{C}_{\alpha_i}, V', W', \mathcal{S}$)
13 Return $\mathcal{C}^{\oplus}_{\alpha_1}, \ldots, \mathcal{C}^{\oplus}_{\alpha_{max}}$
```

4 Analysis

4.1 Proof of correctness

Firstly consider the following scenario: For a graph G we compute the hierarchy using the Hierarchical Cut Clustering Algorithm (Algorithm 2). Then the graph is modified and we update the hierarchy of clusterings. Then again we modify the already modified graph and want to update the hierarchy a second time.

Note that for updating a hierarchy using the Basic Update Algorithm or the advanced update algorithms (Algorithms 3-5) we presuppose that the hierarchy that is to be updated consists of communities. Therefore, in order to enable consecutive updates of the hierarchy as the graph stepwise changes we need to guarantee that the clusterings provided by Algorithms 3-5 also consist only of communities. We will call this property of the hierarchy the `invariant`.

It is easy to show that Algorithms 3-5 maintain the `invariant`. Clusters in the new hierarchy can be computed by either carrying them over directly or by using the algorithms presented by Flake et al. In the first case we showed in the previous chapter that clusters are only carried over if they are still communities. In the second case the Procedure `Simplified Cluster` only returns clusters that are communities as stated in Section 1.2 even if contractions of subsets of communities are proceeded as described in Lemma 1.8. Thus, the updated hierarchies consist of communities.

Furthermore the `invariant` implies as well that we obtain a valid hierarchy, i.e. a hierarchy that the Hierarchical Cut Clustering Algorithm would return. According to Corollary 1.9 there exists exactly one hierarchy of clusterings consisting of clusters that are communities. Thus, the hierarchies returned by Algorithms 3-5 are the hierarchies that the Hierarchical Cut Clustering Algorithm would have returned.

4.2 Runtime

We measure the runtime in terms of the number of minimum-cut calculations that are necessary to update a hierarchy. Using this approach we do not decide which algorithm is used to compute the minimum cuts.
We compare our update algorithms to a recomputation of the hierarchy from scratch using the static Hierarchical Cut Clustering Algorithm. We denote the number of minimum-cut calculations the Hierarchical Cut Clustering Algorithm needs to build a hierarchy of clusterings of the graph G using the parameter sequence $\alpha_1 > \ldots > \alpha_{max}$ and a set of communities $\mathcal{S}$ by $\#\mathrm{HCCA}(G, \alpha_1 > \ldots > \alpha_{max}, \mathcal{S})$. We firstly give a runtime analysis of the Hierarchical Cut Clustering Algorithm, i.e. explain on what the value of $\#\mathrm{HCCA}(\cdot,\cdot,\cdot)$ depends: The Hierarchical Cut Clustering Algorithm needs one minimum-cut calculation per community it computes. As each cluster is a community the number of minimum-cut calculations is greater than or equal to the number of clusters in the whole hierarchy. If `Simplified Cluster`, which is a subroutine of the Hierarchical Cut Clustering Algorithm, finds a community that will later be covered by another community then it spends more minimum-cut calculations than there are clusters in the hierarchy, i.e. the number of clusters in the hierarchy is only a lower bound for $\#\mathrm{HCCA}(G, \alpha_1 > \ldots > \alpha_{max}, \emptyset)$. Flake et al. choose the next vertex whose community shall be computed according to the sum of the weights of its adjacent edges. They always compute the community of the vertex with the highest weighted degree that is not yet contained in a community. Using this heuristic $\#\mathrm{HCCA}(G, \alpha_1 > \ldots > \alpha_{max}, \emptyset)$ is commensurate with the number of clusters [1].

The analysis we present in the following presupposes that the Hierarchical Cut Clustering Algorithm uses a fixed ordering of the vertices and starts to compute the community of the first vertex of the ordering and then proceeds with the next vertex that is not yet contained in a community and so on. Note that the analysis holds as well for other orderings than the one Flake et al. suggested.

In Section 4.2.1 we analyze the Basic Update Algorithm. The Basic Update Algorithm carries over the upper levels that do not change at all. Then all clusters that stay valid are contracted into one single super-node and afterwards the Hierarchical Cut Clustering Algorithm is called. In Section 4.2.2 we discuss the runtime of the advanced update algorithms.

4.2.1 The Basic Update Algorithm

General case	
Edge deletion	$max - k + 1 + \#\text{HCCA}(\widehat{G}, \alpha_1 > \ldots > \alpha_{k-1}, \{V \backslash C_{\alpha_k}^{b/d}\})$
Edge insertion	$\#\text{HCCA}(\widehat{G}, \alpha_1 > \ldots > \alpha_{k-1}, \{V \backslash C_{\alpha_k}^{b/d}\})$
Best case	
Edge deletion	max
Edge insertion	0
Worst case	
Edge deletion	$\#\text{HCCA}(G, \alpha_1 > \ldots > \alpha_{max}, \emptyset)$
Edge insertion	$\#\text{HCCA}(G, \alpha_1 > \ldots > \alpha_{max}, \emptyset)$

Table 1: Number of minimum-cut calculations the Basic Update Algorithm needs in order to update the hierarchy.

In this section we will firstly express the runtime of the Basic Update Algorithm in a general formula that depends on the parameter k, which is defined by Theorem 2.6 and 2.9, respectively, and will then adjust this formula to the best and the worst case.
For edge deletions we need to check on the upper levels whether the community containing b and d stays valid. Thus, we need one minimum-cut calculation per level we check upon this condition. We check this condition going from the highest level to level α_k. If level α_{k-1} is an intra-cluster level we check it as well but find $C^{b/d}$ not to stay valid. However, we do not count this additional minimum-cut calculation as we assume that we can use this cut later when we compute the clustering of level α_{k-1}. Thus, in total we spend $max - k + 1$ minimum-cut calculations for the upper levels, where $1 \leq k \leq max + 1$.
In the case of an edge insertion we do not need any minimum-cut calculations in order to determine which levels of the hierarchy stay valid.
In both cases we contract all clusters that stay valid on the lower levels into one single super-node yielding $\widehat{G}$. Note that $\widehat{G}$ contains $|C_{\alpha_k}^{b/d}| + 1$ vertices what is an assumably small number in comparison to $|V|$. The contracted super-node forms a singleton in the clustering. Thus, we can initialize $\mathcal{S}$ with $\{V \backslash C_{\alpha_k}^{b/d}\}$. Then we apply the Hierarchical Cut Clustering Algorithm for the lower levels $\alpha_1 > \ldots > \alpha_{k-1}$. Thus, the number of minimum-cut calculations the Basic Update Algorithm needs for the lower levels is $\#\text{HCCA}(\widehat{G}, \alpha_1 > \ldots > \alpha_{k-1}, \{V \backslash C_{\alpha_k}^{b/d}\})$.
For the total number of minimum-cut calculations the Basic Update Algorithm needs we sum up the two values for the upper levels and the lower levels yielding the formulas listed in Line 1 and 2 of Table 1.

We will now adjust these formulas to the best case, i.e. the case in which the Basic Update Algorithm can carry over as much as possible. This is, the complete hierarchy does not change and all levels are intra-cluster levels where $C^{b/d}$ stays the community of its representative – as well if an edge has been deleted and if it has been inserted. Then it holds that $k = 1$, i.e. we do not need to call the Hierarchical Cut Clustering Algorithm at all. In the case of an edge insertion this means that we do not afford any minimum-cut calculations. In the edge-deletion case we spend one minimum-cut calculation per level, i.e. max minimum-cut calculations (see Line 3 and 4 of Table 1).

We now discuss the worst case. In the worst case no level can be carried over. Thus, we cannot contract any clusters that stay valid and directly start using the Hierarchical Cut Clustering Algorithm. This leads to $\#\mathrm{HCCA}(G, \alpha_1 > \ldots > \alpha_{max}, \emptyset)$ minimum-cut calculations (see Line 5 and 6 of Table 1), i.e. the Basic Update Algorithm is at least as good as the Hierarchical Cut Clustering Algorithm.

However, these upper boundaries for the number of minimum-cut calculations seem to be far from sharp boundaries in an average case analysis. As already mentioned the Basic Update Algorithm needs at most one minimum-cut calculation to verify that a level can be carried over in contrast to at least $|\mathcal{C}_{\alpha_i}|$ minimum-cut calculations for recomputing it using the Hierarchical Cut Clustering Algorithm. Even though we cannot guarantee that levels can be carried over it seems to be probable that one or even several levels can be carried over. Consequently it seems to be probable that the Basic Update Algorithm needs less minimum-cut calculations than the worst case boundary declares. Besides the number of minimum-cut calculations the Basic Update Algorithm further saves some effort due to the contraction of $V \backslash C^{b/d}_{\alpha_k}$. As this set is potentially large the contraction leads to a graph $\widehat{G}$ with a potentially small number of vertices. This cannot be measured by the number of minimum-cut calculations but might reduce the runtime of each single minimum cut-calculation drastically.

4.2.2 The advanced update algorithms

Lower bound if the hierarchy does not change	
Edge deletion	$max - k + 1 + \$\mathrm{T}(C^{b/d}_{\alpha_k})$
Edge insertion	$\$\mathrm{T}(C^{b/d}_{\alpha_k})$
Upper bound on the upper levels	
Edge deletion	$max - k + 2$
Edge insertion	0
Upper bound on the lower levels (per level)	
Intra-cluster deletion	$\lvert\mathcal{C}_\alpha\rvert - 1 + \lvert C^{b/d}\rvert$
Inter-cluster deletion	$\lvert\mathcal{C}_\alpha\rvert - 2$
Inter-cluster insertion	$\lvert C^b\rvert + \lvert C^d\rvert$

Table 2: Number of minimum-cut calculations of the advanced update algorithms.

In this section we discuss the runtime of the advanced update algorithms. We will analyze (i) the number of minimum-cut calculations the advanced update algorithms need at least if the hierarchy does not change at all and (ii) the number of minimum-cut calculations the advanced update algorithms need at most regardless whether the hierarchy changes or not.

We firstly recall the heuristic that the algorithms use during the computation of the clusterings on the lower levels in order to recognize whether clusters in $\mathcal{C}_\alpha$ stay valid communities in $G_\alpha^{\oplus/\ominus}$ or even clusters in $\mathcal{C}_\alpha^{\oplus/\ominus}$: First compute the communities in $G_\alpha^{\oplus/\ominus}$ that belong to the representatives of clusters in $\mathcal{C}_\alpha$. If afterwards there are still vertices that do not belong

to communities we proceed with the same ordering of vertices as the Hierarchical Cut Clustering Algorithm would do.
The representatives of clusters in $\mathcal{C}_\alpha$ would have been a good choice while computing the clustering $\mathcal{C}_\alpha$. As we expect the changes of the clustering due to the single edge modification, i.e. the differences between $\mathcal{C}_\alpha$ and $\mathcal{C}_\alpha^{\oplus/\ominus}$, to be small these representatives are as well a good choice for computing $\mathcal{C}_\alpha^{\oplus/\ominus}$.

At first we discuss the case that the whole hierarchy stays valid. What is the number of minimum-cut calculations the advanced update algorithms need at least to recognize that the whole hierarchy stays valid?
For the upper levels we need one minimum-cut calculation per level in the case of an edge deletion and zero minimum-cut calculations in the case of an edge insertion.
Now we turn to the update of the lower levels. As we want to deduce a lower bound for the number of minimum-cut calculations we assume that according to the ordering we described above the first $|\mathcal{C}_\alpha^{\oplus/\ominus}|$ vertices the algorithms pick are the representatives of the clusters in $\mathcal{C}_\alpha^{\oplus/\ominus}$.
Begin with considering level α_{k-1}. The algorithm needs to recalculate the minimum-cuts inducing the clusters that are children of $C_{\alpha_k}^{b/d}$. As the hierarchy does not change these clusters are reconfirmed and we can apply Theorem 2.3 to all clusters that neither contain b or d. On level α_{k-2} then the algorithms need to check whether the clusters that are children of $C_{\alpha_{k-1}}^{b}$, $C_{\alpha_{k-1}}^{d}$ or $C_{\alpha_{k-1}}^{b/d}$ change and thus, recompute these clusters. Afterwards we can again apply Theorem 2.3 as we found that these clusters stayed valid. On the levels $\alpha_{k-3}, \ldots, \alpha_1$ we proceed using the same method.
Thus, we always need to reconfirm the clusters that are children of $C_{\alpha_k}^{b/d}$ or of C^b, C^d or $C^{b/d}$ on lower levels. We denote the number of these clusters by \$T$(C_{\alpha_k}^{b/d})$ (see Figure 7 for illustration). As we assumed that we directly pick the representatives of the clusters in $\mathcal{C}_\alpha^{\oplus/\ominus}$ we need one minimum-cut calculation per cluster we compute, i.e. $|$\$T$(C_{\alpha_k}^{b/d})|$ minimum-cut calculations in total for the lower levels. Again we need to add the bound for the upper levels to the one for the lower levels in order to gain a lower bound for the advanced update algorithms (see Line 1 and 2 of Table 2).
Note, that for $k = 1$ this case is what we described as best case in the last section. For $k = 1$ the bounds of the advanced update algorithms evaluate to the same values as the best case bounds of the Basic Update Algorithm.

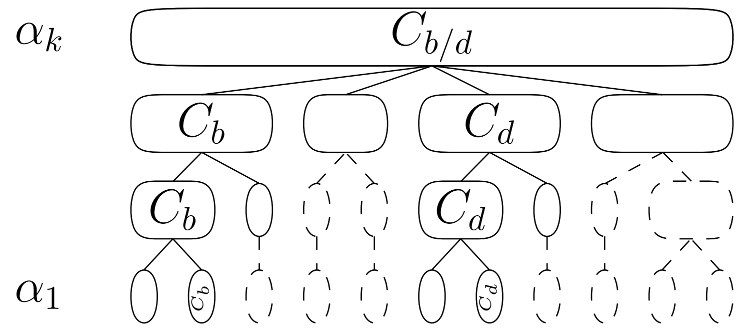


Figure 7: The solid clusters belong to \$T$(C_{\alpha_k}^{b/d})$.

It is hard to give a good boundary for the number of minimum-cut calculations of the whole algorithm in the worst case. For the upper levels the analysis is analogue to the analysis of the general case of the Basic Update Algorithm. However, we do not assume that we can reuse the cut we calculated in order to check whether $C_{\alpha_{k-1}}^{b/d}$ changes. For the lower levels we give boundaries per level distinguishing the following three cases:

intra-cluster deletion, inter-cluster deletion, and inter-cluster insertion. Note that an intra-cluster insertion does not occur on a level below level α_k. Thus, we exclude this case from the following analysis. The maximum number of minimum-cut calculations on a level depends on the number of vertices in the expanded graph after all contractions have been proceeded. In the worst case we need to compute the community of each vertex in the graph. Thus, the maximum number of minimum-cut calculations is the number of vertices after all contractions minus the number of communities we already know, i.e. $|\mathcal{S}|$.

We start computing this number for an intra-cluster edge deletion. As we are on a level below level α_k the community of $r(C^{b/d})$ changes and we need to recompute the clustering on this level. Now consider Algorithm 4. In Line 14 it contracts all clusters that neither contain b nor d. Thus, after these contractions the graph has at most $|\mathcal{C}_\alpha| - 1 + |C^{b/d}|$ vertices. As $\mathcal{S}$ might be empty this is an upper boundary for the number of minimum-cut calculations (see Table 2).

In the case of an inter-cluster edge deletion all cluster in $\mathcal{C}_\alpha$ are contracted. Furthermore C^b and C^d stay valid communities. Thus, Algorithm 4 needs at most $|\mathcal{C}_\alpha| - 2$ minimum-cut calculations (see Table 2).

For an inter-cluster edge insertion all clusters of $\mathcal{C}_\alpha$ except C^b and C^d are contracted leading to a graph with $|\mathcal{C}_\alpha| - 2 + |C^b| + |C^d|$ vertices. Furthermore all clusters except C^b and C^d stay valid communities in the modified graph. Thus, $\mathcal{S}$ contains $|\mathcal{C}_\alpha| - 2$ communities. Therefore, the maximum number of minimum-cut calculations is $|\mathcal{C}_\alpha| - 2 + |C^b| + |C^d| - (|\mathcal{C}_\alpha| - 2) = |C^b| + |C^d|$ (see Table 2).

In the analysis above the difficulty of giving a good bound for the number of minimum-cut calculations are the terms $|C^b|$, $|C^d|$ and $|C^{b/d}|$, i.e. the uncontracted clusters. One could use Flake et al's heuristic in order to cluster $C^b \cup C^d$ and $C^{b/d}$, respectively, yielding a number of minimum-cut calculations that is proportional to the number of clusters covering these sets of vertices. Furthermore, one could argue that the size of $\mathcal{C}_\alpha$ is similar to the size of $\mathcal{C}_\alpha^{\oplus/\ominus}$. It follows that the number of minimum-cut calculations per level is proportional to the number of clusters in $\mathcal{C}_\alpha^{\oplus/\ominus}$.

For a detailed and well-founded runtime analysis of the advanced update algorithms one should implement and evaluate them in comparison to the Hierarchical Cut Clustering Algorithm using real world data. Thereby one should test different heuristics how to handle the problem of uncontracted clusters, e.g. (i) Flake et al's heuristic and (ii) first use the representatives of the children of C^b, C^dor $C^{b/d}$ in the previous hierarchy.

5 Further dynamisation

In the foregoing parts of this work we only covered the modification of edges in a graph. In the following two sections we discuss further possibilities to dynamise the Hierarchical Cut Clustering Algorithm. Firstly we show that the granularity of the hierarchy can easily be changed and secondly we cover vertex modifications, i.e. the insertion and deletion of vertices.

5.1 Changing $\alpha_1 > \ldots > \alpha_{max}$

First of all consider the following scenario: We start with an initial hierarchy and stepwise update the hierarchy. After a vast number of updates the hierarchy might be somehow degenerated. For example two neighboring levels might be the same. In the worst case all levels could be the same. One can also imagine that differences between two neighboring clusterings are too strong.

In these cases it might be reasonable to change one or several parameters in the sequence $\alpha_1 > \ldots > \alpha_{max}$ in order to change the granularity of the hierarchy. Obviously deleting a single level is no problem at all as afterwards clusters on lower levels are still subsets of the clusters on the higher levels. Now consider that we want to insert a level α_i. Firstly, we need to compute the clustering on this level. Therefore, we can step into the Hierarchical Cut Clustering Algorithm and contract the clusters on level α_{i-1} and then compute the clustering on level α_i. According to Lemma 1.7 the freshly inserted level is compatible with upper part of the old hierarchy and as well with the lower part of it. Note that we can simulate the change of a single α_i by inserting and deleting a level.

5.2 Vertex modifications

Now we will discuss the modification of vertices, i.e. the insertion and deletion of a vertex. We restrict this discussion to vertices that are disconnected in the graph G. Therefore, before deleting a vertex one need to remove all incident edges by edge modifications and if a vertex shall be added the incident edges are added after the vertex modification has been proceeded. Note that even though we allow G to be disconnected in this section the expanded graph G_α is always connected.

Hartmann et al. showed that disconnected vertices form singletons in the clustering [4, 5] independently of the value of α and thus, the clustering can easily be updated after a vertex modification by simply adding or deleting a singleton. This can easily be extended to the context of hierarchical cut clustering: we need to add or delete a singleton on every level of the hierarchy. Obviously in the updated hierarchy the clusters still obey the nesting property described in Lemma 1.7, i.e. the hierarchy is a valid clustering hierarchy for the modified graph.

As a singleton, that contains a disconnected vertex v, is the community of v (see Hartmann et al.) the updated hierarchy again consists only of communities. Thus, we can further apply the update techniques we presented after more edge or vertex modifications has been proceeded.

6 Conclusion

In this work we presented different methods to dynamise the Hierarchical Cut Clustering Algorithm presented by Flake et al. We covered changes of the underlying graph by edge and vertex modifications as well as changing the parameter sequence $\alpha_1 > \ldots > \alpha_{max}$, i.e. the granularity of the hierarchy. Of these three methods the modifications of edges are most difficult to handle and cover the main part of this work. To deal with edge modifications, we firstly developed the Basic Update Algorithm that is simple but efficient. Afterwards we presented advanced update techniques that are more complex but even more efficient. The runtime of these update algorithms is potentially low in comparison to recomputing the hierarchy from scratch using the Hierarchical Cut Clustering Algorithm.

Future work will include the implementation of the algorithms presented in this work and their empirical evaluation. Furthermore it is an open question how to choose a good parameter sequence $\alpha_1 > \ldots > \alpha_{max}$. Moreover, it is an interesting question how to deal with offline changes and multiple graph modifications at once.

References

[1] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004.

[2] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[3] R. E. Gomory and T. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, December 1961.

[4] R. Görke, T. Hartmann, and D. Wagner. Dynamic Graph Clustering Using Minimum-Cut Trees. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH), 2009. Informatik, Uni Karlsruhe, TR 2009-10.

[5] R. Görke, T. Hartmann, and D. Wagner. Dynamic Graph Clustering Using Minimum-Cut Trees. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures, 11th International Workshop (WADS'09)*, volume 5664 of *Lecture Notes in Computer Science*, pages 339–350. Springer, August 2009.

[6] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.

[7] R. Kannan, S. Vempala, and A. Vetta. On Clusterings - Good, Bad and Spectral. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 367–378, 2000.