Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 16, no. 2, pp. 411–446 (2012)
DOI: 10.7155/jgaa.00269

Dynamic Graph Clustering Using Minimum-Cut Trees

Robert Görke Tanja Hartmann Dorothea Wagner

Institute of Theoretical Informatics,
Karlsruhe Institute of Technology (KIT), Germany

Abstract

Algorithms or target functions for graph clustering rarely admit quality guarantees or optimal results in general. Based on properties of minimum-cut trees, a clustering algorithm by Flake et al. does however yield such a provable guarantee, which ensures the quality of bottlenecks within the clustering. We show that the structure of minimum *s*-*t*-cuts in a graph allows for an efficient dynamic update of those clusterings, and present a dynamic graph clustering algorithm that maintains a clustering fulfilling this quality guarantee, and that effectively avoids changing the clustering. Experiments on real-world dynamic graphs complement our theoretical results.

Submitted:	Accepted:	Final:	Published:
September 2010	June 2012	July 2012	August 2012
	Article type:	Communicated by:	
	Regular paper	M. Kaufmann	

This work was partially supported by the DFG under grant WA 654/15-2.

E-mail addresses: robert.goerke@kit.edu (Robert Görke) tanja.hartmann@kit.edu (Tanja Hartmann) dorothea.wagner@kit.edu (Dorothea Wagner)

1 Introduction

Graph clustering has become a central tool for the analysis of networks in general, with applications ranging from the field of social sciences to biology and to the growing field of complex systems. The general aim of graph clustering is to identify dense subgraphs, called clusters, that are only loosely connected to each other in a given network. This is, the connections between the dense subgraphs constitute bottlenecks in the network while within the dense regions no significant bottlenecks can be found. The denser the subgraphs and the smaller the bottlenecks inbetween the clearer is the bottleneck-property, and thus, the better is the clustering. Countless formalizations of this paradigm of *intra-cluster density and inter-cluster sparsity* exist, however, the overwhelming majority of algorithms for graph clustering relies on heuristics, e.g., for some NP-hard optimization problem, and do not allow for any structural guarantee on their output.

Probably the most widespread approach nowadays is a method called *greedy modularity maximization* proposed in [5]. *Modularity* is a quality measure for clusterings and was first introduced in [23]. Finding a clustering with maximum modularity is NP-hard [2], so the greedy method improves the modularity of a clustering by merging given subgraphs, thus building new subgraphs representing a clustering of higher modularity. Although modularity is close to human intuition of clustering quality and therefore widely used, it has some specific drawbacks as for example the *resolution limit* explored in [9]. This is, modularity-based methods tend to detect subgraphs of specific size categories depending on the size of the network. Further heuristic algorithms to optimize modularity base on greedy agglomeration [21, 5], greedy movement [1], *spectral* division [22, 29], simulated annealing [16, 24] or extremal optimization [6]. Deducing a clustering from the structural information given by the *spectrum* of a graph is another technique of graph clustering which avoids optimizing a quality measure. Good introductions on *spectral* clustering are given in [28, 10] and advanced topics can be found in [4]. The top-down approach introduced by Kannan et al. [19] relies on a bottleneck-based quality measure called conductance, which is also NP-hard. It uses a polylogarithmic approximation of minimum-conductance cuts to recursively split the graph. For further clustering methods and recent results on graph clustering see, e.g., the following overviews [3, 27, 8] and references therein.

Inspired by the work of Kannan et al. [19], Flake et al. [7] recently presented a clustering algorithm which does guarantee a very reasonable bottleneck-property. Their elegant approach employs minimum-cut trees, pioneered by Gomory and Hu [13], and is capable of finding a hierarchy of clusterings by virtue of an input parameter. There has been an attempt to dynamize this algorithm, by Saha and Mitra [26, 25], however, we found it to be erroneous. We are not aware of any other dynamic graph-clustering algorithms in the literature, except for a recent advance [15], which designs and evaluates several heuristics for dynamically maintaining a clustering with high quality. There, clustering quality is measured again by the index modularity, which is NP-hard to optimize.

Our Contribution. In this work we develop the first correct algorithm that efficiently and dynamically maintains a clustering for a changing graph as found by the method of Flake et al. [7], allowing arbitrary atomic changes in the graph, and keeping consecutive clusterings similar (a notion we call *temporal smoothness*). Our algorithm builds upon partially updating a half-finished minimum-cut tree of a graph in the spirit of Gusfield's [17] simplification of the Gomory-Hu algorithm [13]. While it poses an interesting problem on its own right, updating a complete minimum-cut tree is unnecessary for clusterings and thus undesirable, as it entails additional costs. We corroborate our theoretical results on clustering by experimentally evaluating the performance of our procedures compared to the static algorithm on a real-world dynamic graph.

This paper is organized as follows. We briefly give our notational conventions and one fundamental lemma in Section 1. Then, in Section 2, we revisit some results from [13, 17, 7], convey them to a dynamic scenario in Section 3, and derive our central results. In Section 4 we give a formal description of our update algorithm, which decomposes into several subalgorithms, followed by an analysis in Section 5. We conclude in Section 6.

A preliminary version of this work has been presented at WADS'09 [14]; the main additions are an in-depth treatment of edge insertions alongside corresponding algorithms, a discussion of vertex modifications, an expanded experimental verification and a higher level of detail.

Preliminaries and Notation. 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 $u \sim v$, i.e., $\{u, v\} \in E$. We reserve the term *node* (or *super-node*) for compound vertices of abstracted graphs, which may contain several basic vertices; however, we identify singleton nodes with the contained vertex without further notice. Dynamic modifications of G will concern vertices and edges. The notation for vertex insertions and deletions is postponed to Section 4.2; the reason for this is that vertices require a special treatment, although at first glance, the insertion or deletion of a disconnected vertex in G looks trivial. 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 done exactly as deletions and insertions. 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.

A *minimum-cut tree* $T(G) = (V, E_T, c_T)$ 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 $T(G)$. An edge e_T of $T(G)$ induces a cut in G by decomposing $T(G)$ into two connected components. We sometimes identify e_T with the cut it induces in G. A vertex pair $\{u, v\}$ is called a *cut pair* of an edge e_T if e_T represents a minimum u-v-cut, i.e., if e_T is a cheapest edge on the path between u and v. Neither must the cheapest edge on a path in $T(G)$ be unique, nor $T(G)$. We denote the connectivity between u and v, which is given by the weight of the cheapest edge, by $\lambda(u, v)$. Unspecified cuts are often denoted by θ. For details on minimum-cut trees, see the pioneering work by Gomory and Hu [13] or the simplifications by Gusfield [17].

A *contraction* of G by $N \subseteq V$ means replacing set N by a single supernode η, and leaving η 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. Analogously we can contract by a set $M \subseteq E$. In the context of graphs, our understanding of a *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. Regarding a dynamic graph G and edge modifications of $\{b, d\}$ we particularly designate C_b and C_d containing b and d, respectively. We start by giving some fundamental insights about minimum cuts in modified graphs, which we will rely on in the following, leaving their rather basic proofs to the reader.

Lemma 1 *Let θ be a cut in G, and let edge $e = \{b, d\}$ be modified in G.*

Consider $G^{\oplus}$: If θ does not separate b and d then θ has still the same weight and is still a minimum u-v-cut for all its previous cut pairs $\{u, v\}$. If θ separates b and d then its weight in $G^{\oplus}$ increases by Δ; furthermore, if it was a minimum u-v-cut in G, it stays a minimum u-v-cut iff $\forall u$-v-cuts θ' in G that do not separate b, d: $c(\theta') \geq c(\theta) + \Delta$.

Consider $G^{\ominus}$: If θ separates b and d then θ has weight $c(\theta) - \Delta$ after the modification and remains a minimum u-v-cut for all its previous cut pairs $\{u, v\}$. If θ does not separate b and d then it retains weight $c(\theta)$; furthermore, if it was a minimum u-v-cut in G, it stays a minimum u-v-cut iff $\forall u$-v-cuts θ' in G that separate b, d: $c(\theta') \geq c(\theta) + \Delta$.

Remark 1 *In a cut tree $T(G)$, an edge e_T separates b and d iff e_T is on the path between b and d. Thus, according to Lemma 1, in the case of edge deletion, all edges on the path still represent valid minimum u-v-cuts after the modification. In the case of edge insertion, all edges besides those on the path remain valid.*

2 Fundamentals

Finding communities in the world wide web or in citation networks are but example applications of graph clustering techniques. In [7] Flake et al. propose and evaluate an algorithm which clusters such instances in a way that yields a certain guarantee on the quality of the clusters. The authors base their quality measure on the expansion of a cut $(S, \bar{S})$ due to Kannan et al. [19]:

$$\Psi = \frac{\sum_{u \in S, v \in \bar{S}} c(u, v)}{\min\{|S|, |\bar{S}|\}} \qquad (\textit{expansion} \text{ of cut } (S, \bar{S})) \tag{1}$$

The expansion of a graph is the minimum expansion over all cuts in the graph. For a clustering $\mathcal{C}$, expansion measures both the quality of a single cluster C, quantifying the clearest bottleneck within C, and the goodness of bottlenecks defined by cuts $(C, V \setminus C)$. Inspired by a bicriterial approach for good clusterings by Kannan et al. [19], which bases on the related measure conductance, Flake et al. [7] design a graph clustering algorithm (Algorithm 1) that, given parameter α, asserts the following (where the disjoint union $A \cup B$ with $A \cap B = \emptyset$ is denoted by $A \mathbin{\dot\cup} B$):

$$\underbrace{\frac{c(C, V \setminus C)}{|V \setminus C|}}_{\text{inter-cluster cuts}} \leq \alpha \leq \underbrace{\frac{c(P, Q)}{\min\{|P|, |Q|\}}}_{\text{intra-cluster cuts}} \quad \forall C \in \mathcal{C} \quad \forall P, Q \neq \emptyset \quad P \mathbin{\dot\cup} Q = C \tag{2}$$

2.1 The Static Algorithm

The above quality guarantees—simply called quality in the following—are due to special properties of minimum-cut trees, which are used by the clustering algorithm, as given in Algorithm 1 (compare to [7]). It performs the following steps: Add an artificial vertex t to G, and connect t to all other vertices by weight α. Then, compute a minimum-cut tree $T(G_\alpha)$ of this augmented graph. Finally, remove t and let the resulting connected components of $T(G_\alpha)$ define the clustering. In the following, we will call the fact that a clustering can be computed by this procedure the invariant. For the proof that Cut-Clustering yields a clustering that obeys Equation (2), i.e., that the invariant yields quality, we refer the reader to [7]. Flake at al. further show how nesting properties of *communities* can be used to avoid computing the whole minimum-cut tree $T(G_\alpha)$ and try to only identify those edges of $T(G_\alpha)$ incident to t. In this context the community of a vertex u is a special minimum u-v-cut, namely the unique cut that minimizes the size of the cut side containing u. Thus, in line 5 of Algorithm 1, such a partial minimum-cut tree, which is in fact a *star*, would suffice. Their recommendation for finding these edges quickly is to start with separating high degree vertices from t. Flake et al. also show that further community properties given in [11] yield a whole clustering hierarchy, if α is scaled. In the following we will use the definition of $G_\alpha = (V_\alpha, E_\alpha, c_\alpha)$, denoting by $G_\alpha^\ominus$ and $G_\alpha^\oplus$ the corresponding augmented *and* modified graphs.

Algorithm 1: Cut-Clustering

```
Input: Graph $G = (V, E, c)$, $\alpha$
1 $V_\alpha \leftarrow V \cup \{t\}$
2 $E_\alpha \leftarrow E \cup \{\{t, v\} \mid v \in V\}$
3 $c_\alpha|_E \leftarrow c$, $c_\alpha|_{E_\alpha \setminus E} \leftarrow \alpha$
4 $G_\alpha \leftarrow (V_\alpha, E_\alpha, c_\alpha)$
5 $T(G_\alpha) \leftarrow$ minimum-cut tree of $G_\alpha$
6 $T(G_\alpha) \leftarrow T(G_\alpha) - t$
7 $\mathcal{C}(G) \leftarrow$ components of $T(G_\alpha)$
```

2.2 A Dynamic Attempt

Saha and Mitra [25, 26] published an algorithm that aims at the same goal as our work. The authors describe four procedures for updating a clustering and a data structure for the deletion and the insertion of intra-cluster and inter-cluster edges. Unfortunately, we discovered a methodical error in their work.

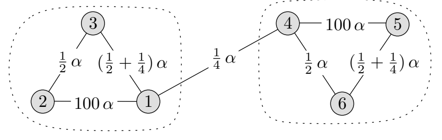
Algorithm 2: Old Inter-Edge Insertion

```
Input: $G = (V, E, c)$, $\alpha$, $\mathcal{C}$, $e_\oplus = \{b, d\}$, $b \in C_b$, $d \in C_d$
1 if inter-cluster quality of $C_d$, $C_b$ is maintained then Case 1:
2     return $\mathcal{C}$ (do nothing)
3 else if $\frac{2c(C_b,C_d)}{|V|} \geq \alpha$ then Case 2:
4     return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{\{C_b \cup C_d\}\}$ (merge $C_b$ and $C_d$)
5 Case 3 (default): enlarge $G$ to $G_\alpha$ (cp. Cut-Clustering)
6 dissolve $C_b$ and $C_d$ and contract all other vertices
7 perform line 5 and 6 of Cut-Clustering on $G_\alpha$
8 return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{$newly formed clusters from $C_b \cup C_d\}$
```

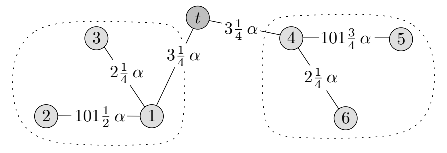
Roughly speaking, it seems as if the authors implicitly (and erroneously) assume an equivalence between quality and the invariant. A full description of issues is beyond the scope of this work, but we briefly point out errors in the authors' procedure that deals with the insertion of intra-cluster edges and give counter-examples in the following. These issues, alongside correct parts, are further scrutinized in-depth by Hartmann [18]. Algorithm 2 sketches the approach given in [26] for handling edge insertions between clusters. Summarizing, we found that Case 1 does maintain quality but not the invariant. Case 2 maintains both quality and the invariant if and only if the input fulfills the invariant, however it can be shown that this case is of purely theoretical interest and extremely improbable. Finally, Case 3 neither maintains quality nor the invariant. The following subsections illustrate these shortcomings with examples.

A Counter-Example for Case 1 and Case 2. We now give an example instance which the algorithm given in [26] fails to cluster correctly. The two upper figures (Figure 1(a), 1(b)) show the input instance, as computed by algorithm Cut-Clustering. In Figure 1(c), a first edge insertion then triggers Case 1, and thus the clustering is kept unchanged. Note that here, quality is still maintained. Then in Figure 1(d) a second edge is added and handled by Case 2, since inter-cluster quality is violated ($c(C_1, C_2) = 4\alpha > 3\alpha = \alpha \cdot \min\{|C_1|, |C_2|\}$), and the condition for Case 2 in line 3 of the algorithm is fulfilled ($2 \cdot 4\alpha/6 > \alpha$). Thus the two clusters are merged. In this result the dashed cut in Figure 1(d) shows an intra-cluster cut with value $c(\text{dashed}) = 2.75 \cdot \alpha < 3 \cdot \alpha$, which violates intra-cluster quality, as claimed in Equation (2).

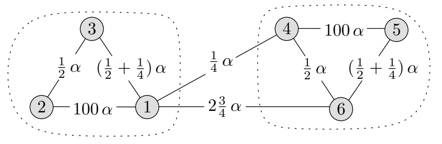
A Counter-Example for Case 3. Finally we give an example instance which the algorithm given in [26] fails to cluster correctly due to shortcomings in Case 3. Figures 2(a) and 2(b) describe the graph and the minimum-cut tree before edge $\{2, 12\}$ is inserted. Then the edge is added and Figure 2(c) describes the resulting construction given in [26], on which a procedure called "adapted Cut-Clustering" (line 7) is then applied, yielding Figure 2(d). The result does neither conform to Equation (2) (quality) nor to what is attempted to be proven in [26]: A "newly formed cluster from $C_b \cup C_d$" as returned in line 8



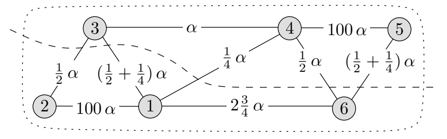
(a) Graph G^0 with clustering $\mathcal{C}(G^0)$ by the Cut-Clustering method.



(b) Minimum-cut tree $T(G^0_\alpha)$ inducing the clustering $\mathcal{C}(G^0)$ shown in Figure (a).



(c) Adding edge $\{1,6\}$, weighted by $11/4\alpha$, yields G^1 with clustering $\mathcal{C}(G^1)$ resulting from Case 1 of the inter-edge-add algorithm.



(d) Adding edge $\{3,4\}$, weighted by α, yields G^2 with the trivial clustering $\mathcal{C}(G^2)$ by Case 2, which violates Equation (2) (dashed cut).

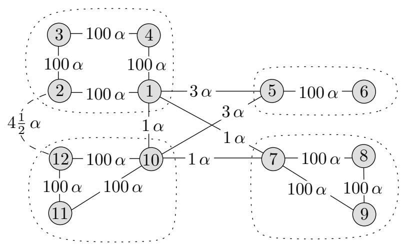
Figure 1: A dynamic instance violating the clustering quality. Weights are parameterized by the same α as used in Cut-Clustering. After two modifications to G^0 the algorithm returns one cluster which can be cut (dashed) with a cut value that violates quality.

does not exist as the clustering resulting from the line before consists of a single cluster containing all vertices. Ignoring line 8, the cut $(\{7,8,9\}, V \setminus \{7,8,9\})$ is an intra-cluster cut that with $P := \{7,8,9\}$ and $Q := V \setminus \{7,8,9\}$ violates Equation (2) as $3 \cdot \alpha = \alpha \cdot \min\{|P|,|Q|\} \geq c(P,Q) = 2 \cdot \alpha$.

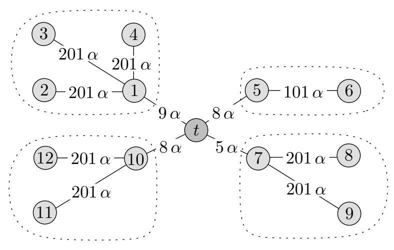
2.3 Minimum-Cut Trees and the Gomory-Hu Algorithm

We briefly describe the construction of a minimum-cut tree as proposed by Gomory and Hu [13] and simplified by Gusfield [17]. Although we will adopt ideas of the latter work, we first give Gomory and Hu's algorithm (Algorithm 3) as the foundation.

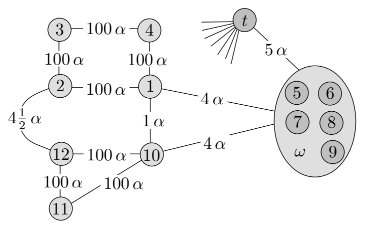
The algorithm builds the minimum-cut tree of a graph by iteratively finding minimum u-v-cuts for vertices that have not yet been separated by a previous minimum cut. Each iteration allows the construction of exactly one edge of the final minimum-cut tree. Thus, the algorithm needs exactly $n-1$ iterations, and the runtime of each iteration is the time of one maximum-flow calculation, which yields a minimum u-v-cut, plus some overhead for vertex contraction and re-organization of intermediate structures. For a rough idea of the total runtime take the well known maximum-flow algorithm by Goldberg and Tarjan [12] which uses the push-relabel method to compute a maximum flow in $O(nm\ \log(n^2/m))$ time and neglect the overhead. Then the worst case runtime for Gomory-Hu is $O(n^4)$. At the beginning, the *intermediate* minimum-cut tree $T_*(G) = (V_*, E_*, c_*)$ (or simply T_* if the context is clear) is initialized as an isolated, edgeless super-node containing all original vertices (line 1). Then,



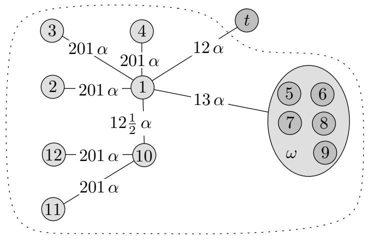
(a) Graph G with clustering $\mathcal{C}(G)$ resulting from the Cut-Clustering method.



(b) Minimum-cut tree $T(G_\alpha)$ inducing the clustering $\mathcal{C}(G)$ shown in Figure (a).



(c) Graph G'_α, resulting from $G_\oplus$ by adding the sink t and contracting the vertices in $\{5,6\} \cup \{7,8,9\}$ (as by Case 3).



(d) Minimum-cut tree $T(G'_\alpha)$ of graph G'_α, yielding a single cluster violating quality by cut $(\{7,8,9\}, V \setminus \{7,8,9\})$ (see Figure (a)).

Figure 2: Counter-example for the correctness of Case 3.

Algorithm 3: Gomory-Hu (Minimum-Cut Tree)

```
Input: Graph $G = (V, E, c)$
Output: Minimum-cut tree of $G$
1  Initialize $V_* \leftarrow \{V\}, E_* \leftarrow \emptyset$ and $c_*$ empty and tree $T_*(G) := (V_*, E_*, c_*)$
2  while $\exists S \in V_*$ with $|S| > 1$ do                 // unfold all super-nodes
3      $\{u, v\} \leftarrow$ arbitrary pair from $\binom{S}{2}$
4      forall the $S_j \sim S$ in $T_*(G)$ do
5          $N_j \leftarrow$ subtree of $S$ with $S_j \in N_j$
6      $G_S = (V_S, E_S, c_S) \leftarrow$ in $G$ contract each subtree $N_j$ to node $\eta_j$
7      $(U, V_S \setminus U) \leftarrow$ minimum $u$-$v$-cut in $G_S$, weight $\delta$, $u \in U$
8      $S_u \leftarrow S \cap U$ and $S_v \leftarrow S \cap (V_S \setminus U)$           // split $S = S_u \dot\cup S_v$
9      $V_* \leftarrow (V_* \setminus \{S\}) \cup \{S_u, S_v\}$, $E_* \leftarrow E_* \cup \{\{S_u, S_v\}\}$, $c_*(S_u, S_v) \leftarrow \delta$
10     forall the former edges $e_j = \{S, S_j\} \in E_*$ do
11         if $\eta_j \in U$ then $e_j \leftarrow \{S_u, S_j\}$ ; // either reconnect $S_j$ to $S_u$
12         else $e_j \leftarrow \{S_v, S_j\}$ ;                // or reconnect $S_j$ to $S_v$
13 return $T_*(G)$
```

until no node S of T_* contains more than one vertex, a node S is *split.* To this end, nodes $S_i \neq S$ are dealt with by contracting in G whole subtrees N_j of S in T_*, connected to S via edges $\{S, S_j\}$, to single nodes η_j (line 6) before splitting S, which yields G_S. The split of S into (S_u, S_v) is then defined by a minimum u-v-cut in G_S (line 7), which does not cross any of the previously used minimum cuts due to the contraction technique. Afterwards, each N_j is reconnected in T_*, again by S_j, to either S_u or S_v depending on which side of the cut η_j, containing S_j, ended up. It is crucial to note, that this cut in G_S can be proven to induce a minimum u-v-cut in G.

An *(partial) execution* $\text{GH} = (G, F, K)$ of Gomory-Hu is characterized by graph G, sequence F of $|F| \leq n-1$ *step pairs* of vertices (compare to line 3) and sequence K of *split cuts* (compare to line 7). A step pair $\{u, v\}$ is *hidden* if $\{u, v\}$ is no edge in the final tree $T(G)$. Hidden pairs occur if either partner moves farther away from the other by unfavorable involvement in later split cuts.

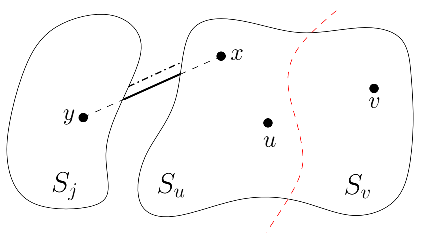
In the situation of Cut-Clustering where G gets augmented by an artificial vertex t the minimum-cut tree $T(G_\alpha)$ attributes an edge e_i to each cluster C_i. Let $\{u_i, v_i\}$ denote the step pair of e_i during Gomory-Hu with $u_i \in C_i$. We call u_i the *representative* $r(C_i)$ (or r_i for short) of C_i. The edges e_i represent a set of non-crossing, non-nested minimum u_i-v_i-cuts in G_α that together separate t from G. Here non-nested means that the cut sides containing the $u_i s$ are mutually disjoint. Remember that Flake et al. deduced from the nesting properties of communities that it is not necessary to compute the whole minimum-cut tree $T(G_\alpha)$. In the following we will apply some results of Gusfield [17] to show that even with arbitrary minimum cuts a partial minimum-cut tree suffices.

2.4 Using Arbitrary Minimum Cuts in G

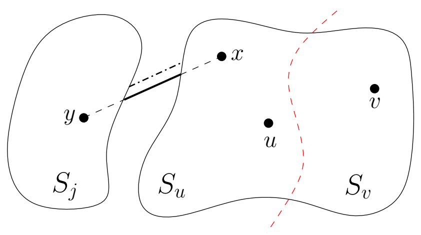
Gusfield [17] presented an algorithm for finding minimum-cut trees which avoids complicated contraction operations. In essence he provided rules for adjusting iteratively found minimum u-v-cuts in G (instead of in G_S) that potentially cross, such that they are rendered consistent with the Gomory-Hu procedure and thus made non-crossing, but still minimal. We review some of these ideas that justify our later arguments. The following lemma tells us, that at any time in Gomory-Hu, for any edge e of $T_*(G)$ there exists a cut pair of e in the two nodes incident to e. It was first formulated and proven within a longer proof in [13] and later rephrased by Gusfield [17].

Lemma 2 (Gus. [17], Lemma 4) *Let $\{S, S_j\}$ be an edge in $T_*(G)$ inducing a cut with cut pair $\{x, y\}$, wlog. $x \in S$. Let now step pair $\{u, v\} \subseteq S$ split S into S_u and S_v, wlog. S_j ending up on the same cut side with S_u, i.e. $\{S_u, S_j\}$ becomes a new edge in $T_*(G)$. If $x \in S_u$, $\{x, y\}$ remains a cut pair of edge $\{S_u, S_j\}$. If $x \in S_v$, then $\{u, y\}$ is an additional cut pair of $\{S_u, S_j\}$.*

In the latter case of Lemma 2, pair $\{x, y\}$ gets *hidden*, and, in the view of vertex y, we say that its former counterpart x gets *shadowed* by the split cut



(a) If $x \in S_u$ vertex pair $\{x, y\}$ remains a cut pair of edge $\{S_u, S_j\}$ in T_*.



(b) Although $x \notin S_u$, with $\{u, y\}$ there is still a cut pair in the adjacent nodes S_u and S_j in T_*.

Figure 3: This illustration shows the situation described in Lemma 2: There always exists a cut pair of edge $\{S_u, S_j\}$ in its incident nodes, independent of the shape of the split cut (red dashed).

associated with step pair $\{u, v\}$. It is not hard to see that during Gomory-Hu, step pairs remain cut pairs, but cut pairs need not stem from step pairs. However, by Lemma 2, each edge in $T_*(G)$ has at least one cut pair in the incident nodes. We define the *nearest cut pair* of an edge in $T_*(G)$ as follows: As long as a step pair $\{x, y\}$ is in adjacent nodes S, S_j, it is the nearest cut pair of edge $\{S, S_j\}$; if a nearest cut pair gets hidden in $T_*(G)$ by a step of Gomory-Hu, as described in Lemma 2 (if $x \in S_v$), the nearest cut pair of the reconnected edge $\{S_u, S_j\}$ becomes $\{u, y\}$ (which are in the adjacent nodes S_u, S_j). The following theorem basically states that we can iteratively find minimum cuts as Gomory-Hu does, without the necessity to operate on a contracted graph.

Theorem 1 (Gus. [17], Theorem 2) *Let $\{u, v\}$ denote the current step pair in node S during some* GH. *If $(U, V \setminus U)$, with $u \in U$, is a minimum u-v-cut in G, then there exists a minimum u-v-cut $(U_S, V_S \setminus U_S)$ of equal weight in G_S such that $S \cap U = S \cap U_S$ and $S \cap (V \setminus U) = S \cap (V_S \setminus U_S)$, with $u \in U_S$ (i.e., with equal behavior on S).*

Being an ingredient to the original proof of Theorem 1, the following Lemma 3 gives a constructive assertion, that tells us how to arrive at a cut described in the theorem by inductively adjusting a given minimum u-v-cut in G. Thus, it is the key to avoiding contraction and using cuts in G by rendering minimum u-v-cuts non-crossing with other given cuts.

Lemma 3 (Gus. [17], Lemma 1) *Let $(Y, V \setminus Y)$ be a minimum x-y-cut in G, with $y \in Y$. Let $(H, V \setminus H)$ be a minimum u-v-cut, with $u, v \in V \setminus Y$ and $y \in H$. Then the cut $(Y \cup H, (V \setminus Y) \cap (V \setminus H))$ is also a minimum u-v-cut.*

Lemma 3 gives an instrument to protect parts of graph G from being cut although the split cut we initially intended to use wriggles through these parts: Let $(Y, V \setminus Y)$ be a given minimum x-y-cut in G ($y \in Y$) with vertices $u, v \in V \setminus Y$ and consider the minimum u-v-cut $(H, V \setminus H)$ as a preliminary version of the current split cut. Then, according to Lemma 3, we may handle side Y of the former cut as if it were contracted and thus retain it, even though the preliminary split cut tries to cut through: The lemma allows to bend the split cut such that Y is not touched. The final shape of the bent cut depends on the side of the split cut containing y—roughly speaking, the cut is "deflected" by y.

This technique will be used in many proofs in the following, so we call it *pseudo-contraction* whenever we refer to it. While Lemma 3 tells us how to find non-crossing cuts, it does *not* yet tell us how to proceed with the minimum-cut tree we are building up: Given a cut as by Theorem 1, Gomory and Hu state a simple mechanism which reconnects a former neighboring subtree N_j of a node S to either of its two split parts (lines 10-12 in Algorithm 3), by the cut side on which the contraction η_j of N_j ends up. In contrast, to establish reconnection when avoiding contraction, this criterion is not available, as N_j is not handled en-block. For this purpose, Gusfield iteratively defines *representatives* $r(S_i) \in V$ of nodes S_i of $T_*(G)$. Starting with an arbitrary vertex as $r(\{V\})$, step pairs in S_i must then always include $r(S_i)$, with the second vertex becoming the representative of the newly split off node $S_j \subset S_i$. For a suchlike run of Gomory-Hu, Gusfield shows—iteratively using Lemma 2 as the key—that for two adjacent nodes S_u, S_v in any $T_*(G)$, $\{r(S_u), r(S_v)\}$ is a cut pair of edge $\{S_u, S_v\}$, and, most importantly his Theorem 3:

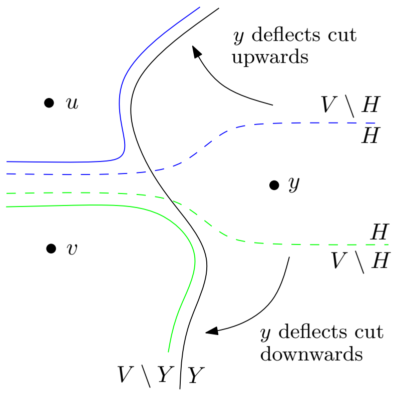
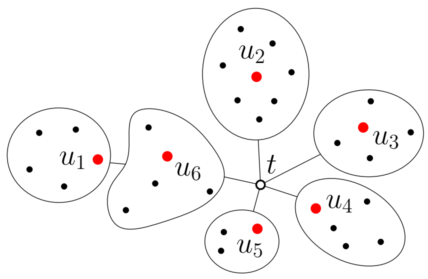


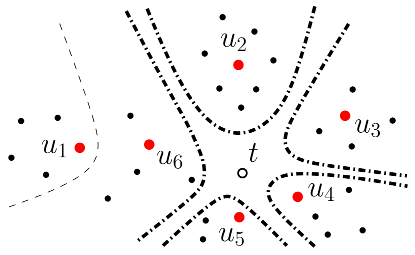
Figure 4: Depending on y there are two different directions to which Lemma 3 bends the cut $(H, V \setminus H)$: upwards or downwards.

Theorem 2 (Gus. [17], Theorem 3) *For $u, v \in S$ let* any *minimum u-v-cut $(U, V \setminus U)$, $u \in U$, in G split node S into $S_u \ni u$ and $S_v \ni v$ and let $(U_S, V \setminus U_S)$ be this cut adjusted via Lemma 3 and Theorem 1; then a neighboring subtree N_j of S, formerly connected by edge $\{S, S_j\}$, lies in U_S iff $r(S_j) \in U$.*

In Cut-Clustering we are free to use Gusfield's simplification applying arbitrary minimum cuts to compute $T(G_\alpha)$. In doing so, we preferably choose step pairs involving t. As soon as t becomes a singleton in $T_*(G_\alpha)$ we have reached an intermediate minimum-cut tree that defines a valid clustering, i.e., a clustering that fulfills the invariant. This can be easily perceived since, in the view of any further step pair, t and the clusters not containing the step pair are pseudo-contracted by Lemma 3. This is, the star with center t in $T_*(G_\alpha)$ would not change if we continued the tree construction, and the final tree would still induce the same clustering as the intermediate tree. We denote the *clustering-defining* intermediate tree by $T_c(G_\alpha)$ and identify the edges in $T_c(G_\alpha)$ with their nearest cut pairs. Figure 5 depicts an example of a clustering-defining intermediate tree, together with the set of non-crossing cuts represented by its edges.



(a) Clustering-defining intermediate tree $T_c(G_\alpha)$ representing six cuts.



(b) Set of cuts represented by the edges in $T_c(G_\alpha)$; five resulting clusters.

Figure 5: This example shows an intermediate minimum-cut tree resulting from a sequence $\{u_i, t\}$, $i = 1, \dots, 6$ of step pairs (a), where step pair $\{u_1, t\}$ is hidden. The bold cuts in (b) induce a clustering with representatives $r(C_i) = u_i$, $i = 2, \dots, 6$.

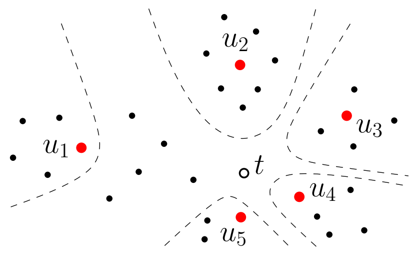
Note that the nearest cut pairs that induce the representatives $r(S_i)$ of the nodes adjacent to t equal the step pairs of the edges incident to t, i.e., those step pairs are not hidden. Thus, the representatives $r(S_i)$ correspond to the representatives $r(C_i)$ attributed to the clusters by the final tree. Conversely, according to the following theorem, any set of non-crossing, non-nested minimum u_i-t-cuts in G_α defines an intermediate minimum-cut tree that can be turned into a clustering-defining intermediate tree $T_c(G_\alpha)$. Figure 6 exemplarily extracts an intermediate tree from a cut set.

Theorem 3 *For some vertices u_i in G let Θ be a set of non-crossing, non-nested minimum u_i-t-cuts in G_α. Let K be any sequence of the cuts in Θ and F the sequence of the associated vertex pairs $\{u_i, t\}$. There exists a sequence F' of step pairs that include t and a sequence K' of split cuts such that* GH $=$ $(G, F \cdot F', K \cdot K')$ *is a feasible* GH *resulting in an intermediate tree where t is a singleton (here $\cdot$ denotes the concatenation operator). The intermediate minimum-cut tree after F is denoted by $T_\circ(G_\alpha)$.*

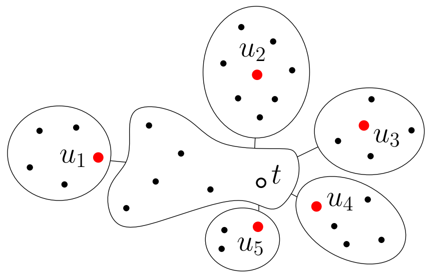
Proof. We index the vertices u_i by the order in F. As the associated split cuts in K are non-crossing and non-nested, it holds that after the i-th iteration of Gomory-Hu (or Gusfield's simlification) u_{i+1} still shares a super-node with t. Therefore, $\{u_{i+1}, t\}$ is a valid step pair in the next iteration step. After the application of F we get a valid intermediate minimum-cut tree $T_\circ(G_\alpha)$, which can be used to continue until t becomes a singleton. Note that the shape of $T_\circ(G_\alpha)$ is independent from the actual order in K. $\square$

3 Finding and Shaping Minimum Cuts in Dynamic Scenarios

In this section we let graph G change, i.e., we consider the insertion of an edge $\{b, d\}$ or its deletion, yielding $G^\oplus$ or $G^\ominus$ (for the augmented graph G_α we get



(a) Set Θ of five non-crossing, non-nested split cuts.



(b) Intermediate tree $T_\circ$ with respect to Θ.

Figure 6: This example shows an intermediate minimum-cut tree $T_\circ$ regarding vertices $u_1, \ldots, u_5$ (b); in order to have $T_\circ$ define a clustering, the vertices inside t's super-node need to be separated from t by further cuts.

$G^\ominus_\alpha$ and $G^\oplus_\alpha$, respectively). Furthermore, we implicitly use arbitrary split cuts according to Gusfield instead of contracted subtrees whenever we consider the process of constructing a clustering-defining tree T_c.

3.1 Cuts That Can Stay

A clustering $\mathcal{C}$ found by Cut-Clustering results from $|\mathcal{C}|$ non-crossing, non-nested minimum u_i-t-cuts represented in $T_c(G_\alpha)$. If any of such cuts are still valid after a graph modification, they define an intermediate tree $T_\circ(G^{\oplus(\ominus)}_\alpha)$ that can be turned into a new clustering-defining tree T_c for $G^{\oplus(\ominus)}_\alpha$ according to Theorem 3. Remark 1 implies that some previous cuts are still valid after a graph modification, making their recomputation unnecessary. The following four remarks and Lemma 4 concretize this assertion. Thus, the idea of our dynamic approach is to construct $T_\circ(G^{\oplus(\ominus)}_\alpha)$ from the cuts detected by the remarks below and continuing on $T_\circ$ by checking the edges incident to t in the old tree $T_c(G_\alpha)$ for further remaining cuts.

Remark 2 Intra-cluster insertion *(resulting in $G^\oplus_\alpha$; vertices b, d are in the same the cluster in G_α):*

None of the edges $\{r(C_i), t\}$ separates b and d (see Figure 14). Thus by Remark 1 all edges $\{r(C_i), t\}$ are still minimum $r(C_i)$-t-cuts after the modification. This implies a set of non-crossing, non-nested minimum u_i-t-cuts in $G^\oplus_\alpha$ that together separate t from $G^\oplus$ and therefore the previous clustering is still valid.

Remark 3 Inter-cluster insertion *(resulting in $G^\oplus_\alpha$; vertices b, d are in different clusters in G_α):*

Apart from $\{r(C_b), t\}$ and $\{r(C_d), t\}$ none of the edges incident to t separates b and d. Again by Remark 1 all these edges not separating b and d still induce minimum cuts after the modification, the two other cuts do not necessarily remain minimum. The thus implied set of non-crossing, non-nested minimum

u_i-t-cuts in $G_\alpha^\oplus$ yields an intermediate minimum-cut tree $T_\circ$ with $|\mathcal{C}(G)| - 1$ nodes (see Figure 13) that can be completed as to define a clustering.

Remark 4 Inter-cluster deletion *(resulting in $G_\alpha^\ominus$; vertices b, d are in different clusters in G_α):*

Only the two edges $\{r(C_b), t\}$ and $\{r(C_d), t\}$ separate b and d. By Remark 1 both cuts represented by these edges are still minimum after the modification, all other edges incident to t do not necessarily remain minimum. This implies an intermediate minimum-cut tree $T_\circ$ with three nodes (see Figure 11) that can be completed as to define a clustering.

Remark 5 Intra-cluster deletion *(resulting in $G_\alpha^\ominus$; vertices b, d are in the same cluster in G_α):*

None of the edges incident to t separate b and d. The thus implied intermediate tree $T_\circ$ consists of only one node covering all vertices of $G_\alpha^\ominus$ (see Figure 12). However, the following lemma yields a condition that allows to preserve the previous clustering. Its proof mostly relies on properties of Gomory-Hu *and on Lemma 1.*

Lemma 4 *In the case of an intra-cluster deletion, let $C_{b/d}$ denote the cluster containing b and d. In $T_c(G_\alpha)$ consider the edge $e_{b/d}$ representing the minimum $r(C_{b/d})$-t-cut. If $e_{b/d}$ still represents a minimum $r(C_{b/d})$-t-cut in $G_\alpha^\ominus$ the previous clustering is still valid.*

Proof. Consider the minimum $r(C_{b/d})$-t-cut $(C_{b/d}, V_\alpha \setminus C_{b/d})$ in $G_\alpha^\ominus$. In the view of any further step pair $\{r(C_i), t\}$ in $V_\alpha \setminus C_{b/d}$ the cut side $C_{b/d}$, and in particular the pair $\{b.d\}$, is sheltered by pseudo-contraction. This is, in $G_\alpha^\ominus$ there exists a minimum $r(C_i)$-t-cut that does not separate b and d, and thus, the connectivity $\lambda(r(C_i), t)$ is the same as in G_α and the previous minimum $r(C_i)$-t-cut is still valid. □

3.2 The Shape of New Cuts

Most cases in the above remarks of Section 3.1 leave at least parts of the clustering of the updated graph $G^{\oplus(\ominus)}$ unfinished. When continuing on $T_\circ$ by checking the edges incident to t in the old tree $T_c(G_\alpha)$ we might then find an edge $\{r(C_i), t\}$, i.e., a minimum $r(C_i)$-t-cut, that is *not* reconfirmed by a computation in $G_\alpha^{\oplus(\ominus)}$, but a new, cheaper minimum $r(C_i)$-t-cut is found. As we shall see in this section, for such a new cut we can still make some guarantees on its shape as to resemble its "predecessor", thereby both enforcing smoothness and saving runtime.

New Cuts After Edge Deletions. We first discuss the case of edge deletions. Let us assume that the minimum $r(C_i)$-t-cut represented in the old tree $T_c(G_\alpha)$ has not been reconfirmed, but instead, by a computation in $G_\alpha^\ominus$, a new, cheaper minimum $r(C_i)$-t-cut $\theta' = (U, V_\alpha \setminus U)$ is found, with $r(C_i) \in U$.

If the cluster C_i neither contains b nor d the following lemma tells us that for any such minimum $r(C_i)$-t-cut θ' there is a minimum $r(C_i)$-t-cut $\theta = (U \cup C_i, (V_\alpha \backslash U) \backslash C_i)$ in $G_\alpha^\ominus$ that does not split C_i, but splits $V_\alpha \backslash C_i$ exactly as θ' does. If C_i contains b and d there is a minimum $r(C_i)$-t-cut $\theta = (U \cap C_i, (V_\alpha \setminus U) \cup \bigcup_{C \in \mathcal{C}(G) \setminus \{C_i\}} C)$ in $G_\alpha^\ominus$ that does not split $\bigcup_{C \in \mathcal{C}(G) \setminus \{C_i\}} C$, but splits C_i exactly as θ' does.

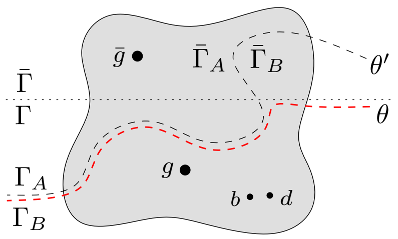


Figure 7: Situation of Lemma 5: minimum $\bar{g}$-g-cut (dotted) splits V_α (gray area) into $\bar{\Gamma}$ and Γ. Any cut θ' (black dashed) that separates $\bar{g}$ and g can be reshaped according to θ (red dashed).

Lemma 5 *For a minimum $r(C_i)$-t-cut $(C_i, V_\alpha \setminus C_i)$ in G_α that does not separate b and d denote the cut side containing b and d by Γ and let g denote the vertex in $\{r(C_i), t\} \cap \Gamma$. We further define $V_\alpha \setminus \Gamma := \bar{\Gamma}$ and $\{r(C_i), t\} \setminus \{g\} := \{\bar{g}\}$. Let (A, B) be a cut separating $r(C_i)$ and t such that (A, B) induces a cut $(\bar{\Gamma}_A, \bar{\Gamma}_B)$ of $\bar{\Gamma}$ with $\bar{g} \in \bar{\Gamma}_A$ and a cut (Γ_A, Γ_B) of Γ with $g \in \Gamma_B$. Then $c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B) \leq c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B)$.*

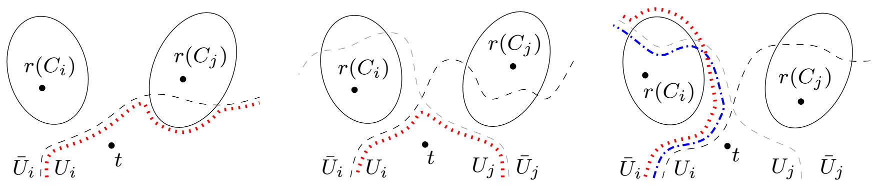
Proof. Using the fact that, in the previous graph G_α, $(C_i, V_\alpha \backslash C_i)$ is a minimum $r(C_i)$-t-cut, we prove Lemma 5 by contradiction. We show that cut $(\bar{\Gamma}_A, V_\alpha \backslash \bar{\Gamma}_A)$ would have been cheaper than the given minimum $\bar{g}$-g-cut $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ in G_α if $c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B)$ was cheaper than $c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)$ in $G_\alpha^\ominus$. We express the costs of $(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A)$ and $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ with the aid of $(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B)$ and $(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)$ considered in Lemma 5. Note that $(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A)$ and $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ do not separate b and d. Thus, their costs are unaffected by the deletion, by Lemma 1. We get

$$\begin{array}{llcll}
\textit{(i)} & c_\alpha(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A) & = & c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B) & - \; c_\alpha^\ominus(\Gamma_A, \Gamma_B \cup \bar{\Gamma}_B) \\
 & & & & + \; c_\alpha^\ominus(\Gamma_A, \bar{\Gamma}_A) \\
\textit{(ii)} & c_\alpha(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma}) & = & c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B) & - \; c_\alpha^\ominus(\Gamma_A, \Gamma_B) \\
 & & & & + \; c_\alpha^\ominus(\Gamma_A, \bar{\Gamma})
\end{array}$$

Certainly, it holds that $c_\alpha^\ominus(\Gamma_A, \Gamma_B) \leq c_\alpha^\ominus(\Gamma_A, \Gamma_B \cup \bar{\Gamma}_B)$ and that $c_\alpha^\ominus(\Gamma_A, \bar{\Gamma}_A) \leq c_\alpha^\ominus(\Gamma_A, \bar{\Gamma})$; together with the assumption that the lemma does not hold, i.e., that $c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B) < c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)$ holds, we can see the following, by subtracting *(i)* and *(ii)*:

$$\begin{array}{rcl}
c_\alpha(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A) - c_\alpha(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma}) & = & [c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B) - c_\alpha^\ominus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)] \\
 & - & [c_\alpha^\ominus(\Gamma_A, \Gamma_B \cup \bar{\Gamma}_B) - c_\alpha^\ominus(\Gamma_A, \Gamma_B)] \\
 & + & [c_\alpha^\ominus(\Gamma_A, \bar{\Gamma}_A) - c_\alpha^\ominus(\Gamma_A, \bar{\Gamma})] < 0
\end{array}$$

This contradicts the fact that the previous cut $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ is a minimum $\bar{g}$-g-cut in G_α. □



(a) θ'_i separates $r(C_j)$ and t. (b) θ'_i separates $r(C_j)$ and t, but θ'_j is known and does not separate $r(C_i)$ and t. (c) neither does θ'_i separate $r(C_j)$ and t, nor θ'_j $r(C_i)$ and t.

Figure 8: Three different scenarios concerning the positions of θ'_i and θ'_j (black and gray dashed, respectively), and their adjustments (red dotted).

While this lemma can be applied in order to retain a cluster C_i, even if new a new minimum $r(C_i)$-t-cut is found (case $b, d \notin C_i$), in the following, we take a look at how such a new, cheap cut can affect *other* clusters C_j, with $j \neq i$. In fact a similar cluster-conserving result can be stated. Consider a clustering defined by a tree $T_c(G_\alpha)$. By Lemma 5 the following conservation-property holds for each cluster C_j with $C_j \cap \{b, d\} = \emptyset$: In $G_\alpha^\ominus$ let θ'_j denote any $r(C_j)$-t-cut $(\bar{U}_j, U_j)$ (with $\bar{U}_j := (V_\alpha \setminus U_j)$, $t \in U_j$ and θ'_j not necessarily minimum with respect to $r(C_j)$ and t). The cut $\theta_j := (\bar{U}_j \cup C_j, U_j \setminus C_j)$ has at most the same weight as θ'_j.

Suppose we have improved a new minimum $r(C_i)$-t-cut in $G_\alpha^\ominus$ according to the conservation-property described above such that it does not split C_i; for the sake of notational simplicity we again call this cut $\theta'_i = (\bar{U}_i, U_i)$, and then, processing it further, work it into a cut called θ_i. In the view of C_i any other cluster C_j with $j \neq i$ and $C_j \cap \{b, d\} = \emptyset$ appears in exactly one of the following three scenarios:

(a) θ'_i separates t and $r(C_j)$ and we do not know a minimum $r(C_j)$-t-cut in $G_\alpha^\ominus$ yet.

(b) θ'_i separates t and $r(C_j)$ and we already know a minimum $r(C_j)$-t-cut θ'_j in $G_\alpha^\ominus$ that does not separate t and $r(C_i)$. In this case suppose θ'_j has also been improved according to the conservation-property, retaining C_j.

(c) θ'_i does not separate t and $r(C_j)$ and we already know a minimum $r(C_j)$-t-cut θ'_j in $G_\alpha^\ominus$ that does not separate t and $r(C_i)$. Again suppose θ'_j retains C_j.

Figure 8 shows θ'_i (black dashed), θ'_j (gray dashed) and the adjustment of θ'_i denoted by θ_i (red dotted) in the three scenarios given above. Note that if we already know a minimum $r(C_j)$-t-cut θ'_j in $G_\alpha^\ominus$ that separates t and $r(C_i)$, this cut behaves like θ'_i in Scenario (a) and using this cut as a split cut in the construction of a new clustering-defining tree $T_c(G_\alpha^\ominus)$ prevents $r(C_i)$ and t from ever becoming a step pair in a later step, since they are already separated. Thus, there is no such scenario.

Scenario (a): As cut θ'_i separates $r(C_j)$ and t, and as C_j satisfies the conservation-property, the cut $\theta_i := (\bar{U}_i \cup C_j, U_i \setminus C_j)$ (red dotted) has weight

$c^{\ominus}_{\alpha}(\theta_i) \leq c^{\ominus}_{\alpha}(\theta'_i)$ and still separates $r(C_i)$ and t and is thus a minimum $r(C_i)$-t-cut, which does not split $C_i \cup C_j$.

While Lemma 5 covers this case only for $C_j \cap \{b, d\} = \emptyset$, we can even drop this limitation: Using $r(C_b)$ ($r(C_d)$ is analogous) as $r(C_j)$ in the case of inter-cluster edge deletion, the same assertion immediately holds, but now by pseudo-contraction. To see this, note that while $\{r(C_b), t\}$ is on $\gamma_{b,d}$, the cut defining C_b remains a minimum $r(C_b)$-t-cut in $G^{\ominus}_{\alpha}$ (by Lemma 1) and thus pseudo-contracts C_b in the view of θ'_i. In the case of intra-cluster edge deletion, we can not apply Lemma 1 as $\{r(C_{b/d}), t\}$ is not on $\gamma_{b,d}$, i.e., we can not adjust other $r(C_i)$s' cuts to $C_{b/d}$ with the arguments in this scenario.

Scenario (b): Here we assume that θ'_j is already known before θ'_i is considered. This is, $r(C_j)$'s side of θ'_j is already pseudo-contracted by θ'_j. As θ'_j does not separate $r(C_i)$ from t, indeed, but may separate some other cluster representatives from t and thus may already have been adjusted accordingly, the cut θ'_i is reshaped regarding the whole cut side $\bar{U}_j$ (instead of only to cluster $C_j \subseteq \bar{U}_j$ as by Scenario (a)). By pseudo-contraction the cut $\theta_i := (U_i \cap U_j, \bar{U}_i \cup \bar{U}_j)$ (red dotted) is a minimum $r(C_i)$-t-cut, which does not split $C_i \cup C_j$. The comments from Scenario (a) carry over.

Scenario (c): In this scenario, neither newly found minimum cut separates the other representative from t. By pseudo-contraction, the cut $(\bar{U}_i \cap U_j, \bar{U}_j \cup U_i)$ (blue dash-dotted) is a minimum $r(C_i)$-t-cut. This cut can be adjusted to $\theta_i := ((\bar{U}_i \cap U_j) \cup C_i, (\bar{U}_j \cup U_i) \setminus C_i)$ (red dotted), which neither splits C_i nor C_j, by means of the conservation-property of C_i. Analogously, $\theta_j := ((\bar{U}_j \cap U_i) \cup C_j, (\bar{U}_i \cup U_j) \setminus C_j)$ (not shown) is a minimum $r(C_j)$-t-cut, and θ_i and θ_j do not cross.

If in the case of inter-cluster edge deletion we consider $r(C_b)$ ($r(C_d)$ is analogous) instead of $r(C_j)$ where $C_j \cap \{b, d\} = \emptyset$ was assumed, the cut defining C_b is a known minimum $r(C_b)$-t-cut in $G^{\ominus}_{\alpha}$ (by Lemma 1), i.e., $\theta'_b := (C_b, V_\alpha \setminus C_b)$ does not split C_i. This is, θ'_i bent along θ'_b by pseudo-contraction neither splits C_i nor C_b and both cuts do not cross. In the case of intra-cluster edge deletion, if $r(C_{b/d})$ is used as $r(C_j)$, any newly found minimum $r(C_{b/d})$-t-cut $\theta'_{b/d}$ can be reshaped by Lemma 5 such that it does not cross the union of all other old clusters and t (Note that it can *not* be adjusted such that it definitely spares $C_{b/d}$ as $C_{b/d}$ has no conservation-property). Thus, $\theta'_{b/d}$ does not cut through C_i. However, θ'_i still may cut through $C_{b/d}$.
To summarize the cases discussed above, we make the following observation.

Observation 1 *During the construction of a clustering-defining tree $T_c(G^{\ominus}_{\alpha})$ starting from $T_\circ$ and checking the remaining edges incident to t in old $T_c(G_\alpha)$, whenever we discover a new, cheaper minimum $r(C_i)$-t-cut θ' ($C_i \cap \{b, d\} = \emptyset$) we can iteratively reshape θ' into a minimum $r(C_i)$-t-cut θ which neither cuts C_i nor any other cluster C_j with $C_j \cap \{b, d\} = \emptyset$, by means of Scenarios (a,b,c). For $r(C_b)$ and $r(C_d)$ in the case of inter-cluster deletion the clusters C_b and C_d are preserved anyway. In contrast, in the case of intra-cluster deletion, the cluster $C_{b/d}$ can not be protected but at least any new minimum $r(C_{b/d})$-t-cut spares all other old clusters.*

New Cuts After Edge Insertions. The bigger picture of the last subsection's findings can be summarized as follows: After an edge deletion, it never pays off to cut through that side of a former minimum cut, which can not offer new, cheaper cuts, as it was unaffected by the update. In the following we will confer this idea to edge *insertions*.

In contrast to edge deletion, after an edge insertion between two different clusters C_b and C_d only two edges need to be checked during the construction of a new clustering-defining tree $T_c(G_\alpha^\oplus)$ from $T_\circ$; namely $\{r(C_b), t\}$ and $\{r(C_d), t\}$, i.e., those edges that are incident to t and separating b and d. The checked edges are reconfirmed by a computation in $G_\alpha^\oplus$ if there exists a minimum $r(C_b)$-t-cut and a minimum $r(C_d)$-t-cut which is as expensive as the previous minimum cut, respectively, *plus* Δ (the weight of the inserted edge), as pointed out in Lemma 1. However, if $\{r(C_b), t\}$ or $\{r(C_d), t\}$ is not reconfirmed, depending on the shape of the new cut, we may still be able to retain the associated cluster. In the following we only consider the edge $\{r(C_b), t\}$. Analog assertions hold for $\{r(C_d), t\}$.

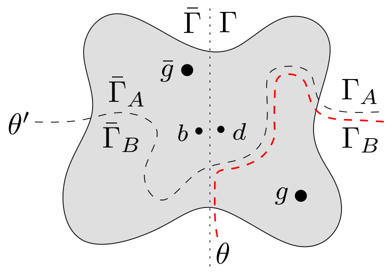


Figure 9: Situation of Lemma 6: minimum $\bar{g}$-g-cut (dotted) splits V_α (gray area) into $\bar{\Gamma}$ and Γ. Any cut θ' (black dashed) that does not separate $b, d, \bar{g}$ but separates $\bar{g}, g$ can be transformed into θ (red dashed).

For any new minimum $r(C_b)$-t-cut $\theta'_b = (U, V_\alpha \setminus U)$ with $\{r(C_b), b\} \subseteq U$ the following lemma tells us that there is a minimum $r(C_b)$-t-cut $\theta_b = (U \cup C_b, (V_\alpha \setminus U) \setminus C_b)$ in $G_\alpha^\oplus$ that does not split C_b, but splits $V_\alpha \setminus C_b$ exactly as θ'_b does. Note that a new minimum $r(C_b)$-t-cut never separates b and d as otherwise it would not be cheaper than the previous one plus Δ. For a new minimum $r(C_b)$-t-cut $\theta'_b = (U, V_\alpha \setminus U)$ with $\{t, b\} \subseteq (V_\alpha \setminus U)$, by Lemma 6, there is a minimum $r(C_b)$-t-cut $\theta = (U \cap C_b, (V_\alpha \setminus U) \cup \bigcup_{C \in \mathcal{C}(G) \setminus \{C_b\}} C)$ in $G_\alpha^\oplus$ that does not split $\bigcup_{C \in \mathcal{C}(G) \setminus \{C_b\}} C$, but splits C_b exactly as θ'_b does. (The attribute of a cut to have $r(C_b), b, d$ *on the same side*, will later be introduced as car, as by "cut attribute regarding representative".) Analogously, cat will stand for the attribute of a cut to have t, b, d *on the same side*, as by "cut attribute regarding t".)

Lemma 6 *Consider a minimum $r(C_i)$-t-cut $(C_i, V_\alpha \setminus C_i)$ in G_α that separates b and d, i.e., $C_i \in \{C_b, C_d\}$. Let (A, B) be a cut separating $r(C_i)$ and t but not b and d, with $\{b, d\} \subseteq A$. Let g denote the vertex in $\{r(C_i), t\} \cap B$ and define $\{r(C_i), t\} \setminus \{g\} := \{\bar{g}\}$. Denote the cut side of $(C_i, V_\alpha \setminus C_i)$ containing g by Γ and $V_\alpha \setminus \Gamma := \bar{\Gamma}$. Then (A, B) induces a cut $(\bar{\Gamma}_A, \bar{\Gamma}_B)$ of $\bar{\Gamma}$ with $\bar{g} \in \bar{\Gamma}_A$ and a cut (Γ_A, Γ_B) of Γ with $g \in \Gamma_B$ such that $c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B) \leq c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B)$.*

Proof. The Proof of Lemma 6 bases on the same idea as the proof of Lemma 5. We prove it by contradiction. We show that cut $(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A)$ would be cheaper than the edge-induced minimum $\bar{g}$-g-cut $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ in G_α if $c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B)$ was cheaper than $c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)$ in $G_\alpha^\oplus$. We express the costs of $(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A)$ and $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ with the aid of $(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B)$ and $(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)$ considered in Lemma 6. Note that $c_\alpha(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A) = c_\alpha^\oplus(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A) - \Delta$ and $c_\alpha(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma}) = c_\alpha^\oplus(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma}) - \Delta$. Thus, for our contradiction, it will do to show that $c_\alpha^\oplus(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A)$ would be cheaper than $c_\alpha^\oplus(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$. We get

$$\begin{array}{rlcll}
\textit{(i)} & c_\alpha^\oplus(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A) & = & c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B) & - \; c_\alpha^\oplus(\Gamma_A, \Gamma_B \cup \bar{\Gamma}_B) \\
 & & & & + \; c_\alpha^\oplus(\Gamma_A, \bar{\Gamma}_A) \\
\textit{(ii)} & c_\alpha^\oplus(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma}) & = & c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B) & - \; c_\alpha^\oplus(\Gamma_A, \Gamma_B) \\
 & & & & + \; c_\alpha^\oplus(\Gamma_A, \bar{\Gamma})
\end{array}$$

Again we observe two inequalities:
$c_\alpha^\oplus(\Gamma_A, \Gamma_B) \leq c_\alpha^\oplus(\Gamma_A, \Gamma_B \cup \bar{\Gamma}_B)$ and $c_\alpha^\oplus(\Gamma_A, \bar{\Gamma}_A) \leq c_\alpha^\oplus(\Gamma_A, \bar{\Gamma})$; together with the contradicting assumption that $c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B) < c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)$, by subtracting (i) and (ii), we get:

$$\begin{array}{rcl}
c_\alpha^\oplus(\bar{\Gamma}_A, V_\alpha \setminus \bar{\Gamma}_A) - c_\alpha^\oplus(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma}) & = & [c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}_A, \Gamma_B \cup \bar{\Gamma}_B) - c_\alpha^\oplus(\Gamma_A \cup \bar{\Gamma}, \Gamma_B)] \\
 & - & [c_\alpha^\oplus(\Gamma_A, \Gamma_B \cup \bar{\Gamma}_B) - c_\alpha^\oplus(\Gamma_A, \Gamma_B)] \\
 & + & [c_\alpha^\oplus(\Gamma_A, \bar{\Gamma}_A) - c_\alpha^\oplus(\Gamma_A, \bar{\Gamma})] < 0
\end{array}$$

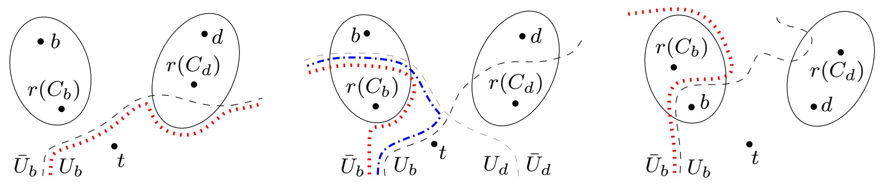
This contradicts the fact that the edge-induced cut $(\bar{\Gamma}, V_\alpha \setminus \bar{\Gamma})$ is a minimum $\bar{g}$-g-cut in G_α. □

While this lemma can be applied in order to retain the current cluster (wlog. C_b), given that a new minimum $r(C_b)$-t-cut, with $r(C_b), b, d$ on the same side, is found, in the following, we take a look at how this new cut can affect the other cluster C_d.

In a clustering-defining tree $T_c(G_\alpha)$ consider the two edges $\{r(C_b), t\}$ and $\{r(C_d), t\}$. Returning to the notation used before in the case of edge deletion, by Lemma 6, the following conservation-property holds for C_b: In $G_\alpha^\oplus$ let θ_b' denote any $r(C_b)$-t-cut $(\bar{U}_b, U_b)$ that has $r(C_b), b, d$ *on the same side*(with $\bar{U}_b := (V_\alpha \setminus U_b)$, $t \in U_b$ and θ_b' not necessarily minimum with respect to $r(C_b)$ and t). The cut $\theta_b := (\bar{U}_b \cup C_b, U_b \setminus C_b)$ is of at most the same weight as θ_b'. For C_d an analogous property holds.

In order to see which cuts are eligible for adjustment by this conservation-property, as already announced, we introduce the following attributes for cuts: We say that a $r(C_b)$-t-cut possesses the cut attribute car(b) if it has $r(C_b), b, d$ *on the same side*; and it has the cut attribute cat(b) if it has t, b, d *on the same side* (note that car(b) and cat(b) are mutually exclusive). Analogously, car(d) and cat(d) denote the cut attributes for $r(C_d)$-t-cuts.

In $G_\alpha^\oplus$ consider a new minimum $r(C_b)$-t-cut θ_b' which is cheaper than the previous cut plus Δ. Such a cut either satisfies car(b) or cat(b) as it does not cross the inserted edge $\{b, d\}$, by Lemma 1. If it possesses car(b), suppose we have already improved this cut in $G_\alpha^\oplus$ according to the conservation-property described above such that it does not split C_b; for the sake of notational simplicity



(a) θ'_b possesses car(b) and separates $r(C_d)$ and t.

(b) θ'_b with car(b) does not separate $r(C_d), t$; neither does θ'_d with car(d) separate $r(C_b), t$.

(c) θ'_b possesses cat(b). θ'_d has three possibilities (not shown).

Figure 10: Three different scenarios concerning the positions of θ'_b and θ'_d (black and gray dashed, respectively), and their adjustments (red dotted).

we again call it $\theta'_b = (\bar{U}_b, U_b)$, and then, processing it further, work it into a cut called θ_b. In the view of C_b the cluster C_d appears in exactly one of the following three scenarios:

(a) θ'_b possesses car(b) and separates t and $r(C_d)$. A new minimum $r(C_d)$-t-cut θ'_d in $G^{\oplus}_{\alpha}$ has different possibilities discussed below.

(b) θ'_b possesses car(b) and does not separate t and $r(C_d)$ and we know a new minimum $r(C_d)$-t-cut θ'_d in $G^{\ominus}_{\alpha}$ that behaves analogously. Suppose θ'_d has also been improved according to the conservation-property, retaining C_d.

(c) θ'_b possesses cat(b) and thus can be reshaped by Lemma 6 such that it does not cut through $\bigcup_{C \in \mathcal{C}(G) \setminus \{C_b\}} C$. Then a new minimum $r(C_d)$-t-cut θ'_d in $G^{\oplus}_{\alpha}$ has three possibilities, as described below.

Figure 10 shows θ'_b (black dashed), θ'_d (gray dashed) and the adjustment of θ'_b denoted by θ_b (red dotted) in the three scenarios given above.

Scenario (a): The attribute car(b) together with the conservation-property for C_d allows to conserve both clusters C_b and C_d as follows. As cut θ'_b separates $r(C_d), t$ and satisfies car(b) it also possesses car(d), and as C_d satisfies the conservation-property, the cut $\theta_b := (\bar{U}_b \cup C_d, U_b \setminus C_d)$ (red dotted) has weight $c^{\oplus}_{\alpha}(\theta_b) \leq c^{\oplus}_{\alpha}(\theta'_b)$ and is thus a minimum $r(C_b)$-t-cut that does not split $C_b \cup C_d$. If there is a new minimum $r(C_d)$-t-cut θ'_d that behaves analogously, we can choose between the two improved cuts θ_b and θ_d. In this case, in order to keep consecutive clusterings similar, we prefer that cut that shadows less of the remaining clusters in $\mathcal{C}(G_\alpha) \setminus \{C_b, C_d\}$. Minimum $r(C_d)$-t-cuts of any other shape are ignored.

Scenario (b): Here, by pseudo-contraction, θ'_b can be reshaped to the blue dash-dotted cut which possesses cat(b). Applying Lemma 6 to this minimum $r(C_b)$-t-cut yields $\theta_b := (U_d \cap C_b, V_\alpha \setminus (U_d \cap C_b))$ (red dotted) as a minimum $r(C_b)$-t-cut, which does not shadow any of the old clusters. Furthermore, θ'_d at least pseudo-contracts cluster C_d. Since in this scenario θ'_d and θ'_b are swappable, we choose that cut for reshaping that shadows more of the remaining clusters in $\mathcal{C}(G_\alpha) \setminus \{C_b, C_d\}$. A similar situation also appears in Scenario (c(ii)).

Scenario (c): Applying Lemma 6 to θ'_b which satisfies cat(b) yields a cut $\theta_b := (\bar{U}_b \cap C_b, V_\alpha \setminus (\bar{U}_b \cap C_b))$ shaped as shown in the figure (red dotted), in particular, a cut that does not shadow any of the old clusters and does not separate $r(C_d), t$. Note that similar assertions also hold for θ'_b being a reconfirmed, remaining cut pseudo-contracting C_b. However, if θ'_b is not reconfirmed, the further new cuts potentially needed to separate t from remaining vertices may cut through C_b (while θ'_b definitely does). The second cut θ'_d (not shown) now has one of three possible, mutually exclusive characteristics:

(i) θ'_d also possesses cat(d): Then it has a shape analogous to θ'_b (after the above adjustment, respectively). Thus, neither C_b nor C_d is conserved, but none of the old, remaining clusters get shadowed so far. Furthermore, C_b is not cut by θ'_d and vice versa.

(ii) θ'_d possesses car(d) but does not separate $r(C_b), t$: In this case we suppose θ'_d to be at least improved according to the conservation-property retaining and thus pseudo-contracting C_d.

(iii) θ'_d separates $r(C_b), t$ with car(d): Then, independent of the shape of θ'_b, we get Scenario (a) with roles of $r(C_b)$ and $r(C_d)$ swapped.

Observation 2 *During the construction of a clustering-defining tree $T_c(G_\alpha^\oplus)$ starting from $T_\circ$ and checking the two remaining edges incident to t in old $T_c(G_\alpha)$, whenever we discover a new minimum cut that is cheaper than the previous cut plus Δ and possesses* car*, we can reshape this cut such that it does not cut through its associated cluster. If it further separates the other vertex from t, even $C_b \cup C_d$ can be preserved, by means of Scenario (a). In Scenario (b) we can preserve at least one of the two clusters C_b and C_d, while the other cluster gets split. However, all vertices split off from the latter cluster are shadowed by the new cut associated to the preserved cluster. A new minimum cut that possesses* cat *can be adjusted such that it does not shadow any old, remaining cluster in $\mathcal{C}(G_\alpha) \setminus \{C_b, C_d\}$ and does not cut the opposite cluster (compare to Scenario (c)). In case of Scenario (c(ii)) at least one of the two clusters C_b and C_d remains. However, in contrast to Scenario (b), it might be necessary to resolve crossings between both newly computed cuts. By pseudo-contraction we can further assume all new cuts not to cut through any of the remaining clusters in $\mathcal{C}(G_\alpha) \setminus \{C_b, C_d\}$.*

Coverage of Cases and Scenarios. At this point it is reasonable to briefly review the cases treated above. It is important to note that the above elaborations are exhaustive in that all possible cases for edge-based updates have been discussed. Before we turn to formal descriptions of update algorithms based on our observations, let us summarize.

For edge deletions we have found comprehensive rules allowing for the preservation or at least for the en-bloc treatment of former clusters. Deleting an edge between clusters in fact never forces us to split any former cluster, while inside a single cluster, an edge deletion can at most require us to demolish that particular cluster. For edge insertions our rules are slightly less comprehensive. On the one hand, nothing needs to be done for intra-cluster insertions (compare to Remark 2). On the other hand, for inter-cluster insertions we have established rules which enable cluster preservation for a number of cases. However, the

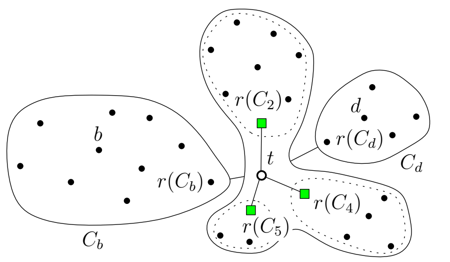


Figure 11: $T_\circ(G_\alpha^\ominus)$ for an inter-cluster deletion, edges not separating b, d need inspection. The cuts of $r(C_b)$ and $r(C_d)$ are correct, but they might get shadowed.

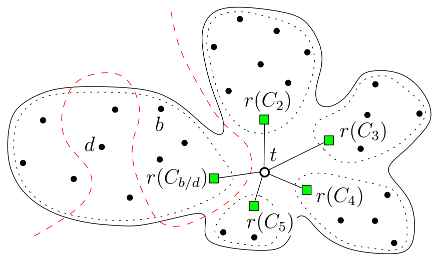


Figure 12: $T_\circ(G_\alpha^\ominus)$ for an intra-cluster deletion, any new $r(C_{b/d})$-t-cut (red dashed) spares $\bigcup_{C\in\mathcal{C}(G_\alpha)\setminus\{C_{b/d}\}} C$ and could be added to Θ by Algorithm 6 (line 12).

remaining cases can again only require us to demolish both affected clusters, not others.

4 Update Algorithms for Dynamic Clusterings

In this section we put the results of the previous sections to good use and give algorithms for updating a minimum-cut tree clustering, such that the invariant is maintained and thus also the quality.

4.1 Edge Modifications

By concept, for updating a clustering after an edge modification, we merely need to know all vertices of $T_c(G_\alpha)$ adjacent to t, i.e., all representatives of the clusters. We call this set $W(G)$, with $r_b := r(C_b), r_d := r(C_d), r_{b/d} := r(C_{b/d})$ being the particular representatives. We call the corresponding set of non-crossing, non-nested minimum $r(C_i)$-t-cuts that isolate t, $\Theta(G)$. We will thus focus on dynamically maintaining only this information. From Remarks 2-5, for a given edge insertion or deletion, we know $T_\circ$, and we know in which node of $T_\circ$ to find t, this is the node we need to examine. We now give algorithms for the deletion and the insertion of an edge running inside or between clusters.

Edge Deletion. Our first algorithm handles inter-cluster deletion (see Algorithm 4). Just like its three counterparts, it takes as an input the sets $W(G)$ and $\Theta(G)$ of the old graph G, furthermore it takes the changed graph, augmented by t, $G_\alpha^\ominus$ and the deleted edge $\{b, d\}$. Recall that an inter-cluster deletion yields an intermediate tree $T_\circ(G_\alpha^\ominus)$ that consists of three nodes and contains edges $\{r(C_b), t\}$ and $\{r(C_d), t\}$ representing cuts θ_b, θ_d defining the clusters C_b and C_d, as shown in Figure 11. All other edges incident to t in $T_c(G_\alpha)$ need to be changed or reconfirmed. To this end Algorithm 4 lists all previous representatives $r_i \in W(G)$, into the tentative list W_{ten}, and initializes their shadows $D(r_i) = \{r_i\}$ by means of Lemma 5. The known cuts θ_b, θ_d are already added to Θ_{ten} (line 1). Then the core algorithm, Check Representatives is called, which—roughly speaking—performs those Gomory-Hu steps that are necessary to isolate t, of course, using the lemmas derived above.

First of all, note that if $|\mathcal{C}| = 2$ ($\mathcal{C} = \{C_b, C_d\}$ and $\{t\}$ singleton in $T_\circ(G^\ominus_\alpha)$) then $W_{\text{ten}} = \{r_b, r_d\}$ and Algorithm 4 lets Check Representatives (Algorithm 5) simply return the input cuts and terminates. Otherwise, it iterates the set of former representatives W_{ten}, thereby possibly shortening it, due to shadowing. We start by computing a new minimum r_i-t-cut for r_i. If the new

Algorithm 4: Inter-Cluster Edge Deletion

```
   Input: $W(G)$, $\Theta(G)$ $G^\ominus_\alpha = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c^\ominus_\alpha)$, edge $\{b,d\}$
   Output: $W(G^\ominus)$, $\Theta(G^\ominus)$
1  $\Theta_{\text{ten}} \leftarrow \{\theta_b, \theta_d\}$, $W_{\text{ten}} \leftarrow \{r_b, r_d\}$
2  $D(r_b) \leftarrow \{r_b\}$, $D(r_d) \leftarrow \{r_d\}$
3  for $r_i \in W(G) \setminus \{r_b, r_d\}$ do                 // not including $r_b, r_d$
4      Add $r_i$ to $W_{\text{ten}}$                             // old representatives
5      $D(r_i) \leftarrow \{r_i\}$                                // shadows
6  return Check Representatives
   $(W(G), \Theta(G), G^\ominus_\alpha, \{b,d\}, D, W_{\text{ten}}, \Theta_{\text{ten}})$
```

Algorithm 5: Check Representatives

```
   Input: $W(G), \Theta(G), G^\ominus_\alpha, \{b,d\}, D, W_{\text{ten}}, \Theta_{\text{ten}}$
   Output: $W(G^\ominus), \Theta(G^\ominus)$
1  while $W_{ten}$ has next element $r_i \notin \{r_b, r_d\}$ do      // $W_{\text{ten}}$ may change
2      $\theta_i \leftarrow$ minimum $r_i$-$t$-cut given by FlowAlgo$(r_i, t)$
3      if $c^\ominus_\alpha(\theta_i) = c_\alpha(\theta_i^{old})$ then     // retain old cuts of same weight
4          Add $\theta_i^{\text{old}}$ to $\Theta_{\text{ten}}$                       // pointed at by $r_i$
5      else                                                  // new cheaper cuts
6          Add $\theta_i$ to $\Theta_{\text{ten}}$                               // pointed at by $r_i$
7          while $W_{ten}$ has next element $r_j \neq r_i$ do   // cp. to other cuts
8              if $\theta_i$ separates $r_j$ and $t$ then  // Scen. (a), (b) shadow $r_j$
9                  Remove $r_j$ from $W_{\text{ten}}$, $D(r_i) \leftarrow D(r_i) \cup D(r_j)$
10                 if $\Theta_{ten} \ni \theta_j$, pointed at by $r_j$ then Delete $\theta_j$ from $\Theta_{\text{ten}}$

11 forall the $r_i \in W_{ten}$, $r_i \notin \{r_b, r_d\}$ do // let cuts preserve clusters
12     set $(\bar{U}, U) := \theta_i$ with $t \in U$ for $\theta_i \in \Theta_{\text{ten}}$ pointed at by $r_i$
13     forall the $r_j \in D(r_i)$ do     // handle $C_i$ and shadowed cuts ...
14         $\theta_i \leftarrow (\bar{U} \cup C_j, U \setminus C_j)$  // ...with Scenario (a), (b), Lem. 5
15     forall the $r_j \neq r_i$ in $W_{ten}$ do           // handle other cuts ...
16         forall the $r_x \in D(r_j)$ do               // ...with Scenario (c)
17             $\theta_i \leftarrow (\bar{U} \setminus C_x, U \cup C_x)$
18 return $W_{\text{ten}}, \Theta_{\text{ten}}$
```

cut is not cheaper, we use the old one instead, and add it to the tentative list of cuts Θ_{ten} (lines 3-4). Otherwise we store the new, cheaper cut θ_i, and examine it for later adjustment: For any candidate r_j still in W_{ten} that is separated from t by θ_i, Scenario (a) or (b) applies (line 8). Note that the while loop in line 7 iterates W_{ten} from scratch. Thus, r_j will be in the shadow of r_i, and not a representative in the new clustering (line 9). In case r_j has already been processed (Scenario (b)), its cut is removed from Θ_{ten}.

Once all candidates for new representatives are processed, for each of them exactly one of the following options holds: It induces the same cut as before, it is new and shadows other former representatives or it is itself shadowed by another candidate. Now that we have collected these relations, we actually apply Lemma 5 and Scenarios (a,b,c) in lines 11-17. Note that for retained, old cuts, no adjustment is actually done here, however, for brevity, the pseudocode superficially iterates throughout W_{ten}. In fact, it is not hard to see that at most two vertices in W_{ten} are assigned to new cuts which require treatment. Finally, all non-shadowed candidates alongside their adjusted, non-crossing, non-nested cuts are returned.

Next we look at *intra*-cluster edge deletion. Looking at our starting point $T_\circ$, the modified edge $\{b, d\}$ lies within some cluster $C_{b/d}$, which does not help much, since none of the edges incident to t in $T_c(G_\alpha)$ separates b and d. However, in this case, Lemma 5 tells us that any new minimum $r_{b/d}$-t-cut spares the part of $G_\alpha^\ominus$ containing all other former clusters and t, see the dashed cut in Figure 12. Algorithm 6 has the same in- and output as Algorithm 4, and starts by finding a new minimum $r_{b/d}$-t-cut. If this yields that no new, cheaper $r_{b/d}$-t-cut exists, then, by Lemma 4, we are done (line 2). Otherwise, we first adjust $\theta_{b/d}$ such that it at least does not interfere with any former cluster C_i by Lemma 5 (lines 5-6); note that $C_{b/d}$ can not necessarily be preserved. Then we prepare the sets $W_{\text{ten}}, \Theta_{\text{ten}}$ in lines 7-10. Check Representatives now performs the same tasks as for Inter-Cluster Edge Deletion: It separates all candidates for new representatives from t in a non-intrusive manner; note that this excludes $r_{b/d}$ (line 8), as $C_{b/d}$ is not retained, and thus defies the adjustments.

After line 11 we have one minimum $r_{b/d}$-t-cut that leaves the former clusters untouched, but might split $C_{b/d}$, and we have a new set Θ_{ten} of minimum r_i-t-cuts (with some former $r_j \in W(G)$ possibly having become shadowed) which do not cut through former clusters C_i but might, however, also cut through $C_{b/d}$. These cuts may further cross each other and $\theta_{b/d}$ in the area of $C_{b/d}$. So we put all these cuts and representatives into $\Theta(G^\ominus)$ and $W(G^\ominus)$ and apply the technique of pseudo-contraction to make all cuts non-crossing. Note that this may result in shadowing $r_{b/d}$. In this case we delete the nested cut. Finally, some vertices from the former cluster $C_{b/d}$ might still remain unclustered, i.e., not separated from t by any $\theta \in \Theta(G^\ominus)$. For clustering these vertices v we can not do better than proceeding as conservatively: Compute their set of minimum v-t-cuts and render them non-crossing by pseudo-contraction, possibly shadowing one another or some previous cut θ. We refrain from detailing the latter steps.

Algorithm 6: Intra-Cluster Edge Deletion

```
    Input: $W(G), \Theta(G), G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c_\alpha^\ominus)$, edge $\{b,d\}$
    Output: $W(G^\ominus)$, $\Theta(G^\ominus)$
 1  $\theta_{b/d} \leftarrow$ minimum $r_{b/d}$-$t$-cut given by FlowAlgo$(r_{b/d}, t)$
 2  if $c_\alpha^\ominus(\theta_{b/d}) = c_\alpha(\theta_{b/d}^{old})$ then            // no cheaper cut found
 3      return $W(G), \Theta(G)$                // retain clustering by Lemma 4
 4  else                         // a new cut should spare former clusters
 5      set $(\bar{U}, U) := \theta_{b/d}$ with $t \in U$                // just nomenclature
 6      forall the $C_i \neq C_{b/d}$ do $\theta_{b/d} \leftarrow (\bar{U} \setminus C_i, U \cup C_i,)$ ;    // by Lemma 5
 7      $W_{\mathrm{ten}} \leftarrow \emptyset, \Theta_{\mathrm{ten}} \leftarrow \emptyset$
 8      for $r_i \in W(G) \setminus \{r_{b/d}\}$ do                // not including $r_{b/d}$
 9          Add $r_i$ to $W_{\mathrm{ten}}$
10          $D(r_i) \leftarrow \{r_i\}$
11      $W_{\mathrm{ten}}, \Theta_{\mathrm{ten}} \leftarrow$ Check Representatives
        ( $W(G), \Theta(G), G_\alpha^\ominus, \{b,d\}, D, W_{\mathrm{ten}}, \Theta_{\mathrm{ten}}$ )
12      $W(G^\ominus) \leftarrow W_{\mathrm{ten}} \cup \{r_{b/d}\}$, $\Theta(G^\ominus) \leftarrow \Theta_{\mathrm{ten}} \cup \{\theta_{b/d}\}$
13      Resolve all crossings in $\Theta(G^\ominus)$ by pseudo-contraction, delete nestings
14      Isolate the sink $t$ from all remaining unclustered vertices
15      return $W(G^\ominus), \Theta(G^\ominus)$
```

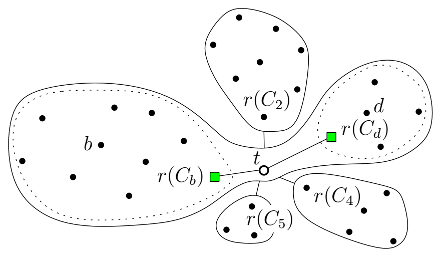


Figure 13: $T_\circ(G_\alpha^\oplus)$ for an inter-cluster insertion. At least $r(C_b)$ and $r(C_d)$ need inspection.

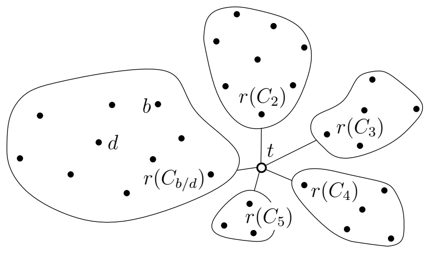


Figure 14: $T_\circ(G_\alpha^\oplus)$ for an intra-cluster insertion. All relevant minimum $r(C_i)$-t-cuts persist.

Edge Insertion. The good news for handling $G^\oplus$ is that an algorithm Intra-Cluster Edge Addition does not need to do anything, but return the old clustering (compare to Remark 2 and Figure 14). By contrast, inserting an edge between clusters is more demanding (Algorithm 7). Again, the algorithm takes as an input the sets $W(G)$ and $\Theta(G)$ of the old graph G, the changed graph $G_\alpha^\oplus$, the inserted edge $\{b,d\}$ and its weight Δ. An *inter*-cluster insertion yields an intermediate tree $T_\circ(G_\alpha^\oplus)$ with $|\mathcal{C}(G)| - 1$ nodes, thus in $T_\circ$ the only unknown cuts are those for $r(C_b)$ and $r(C_d)$ in $T_c(G_\alpha)$, see Figure 13. A sketch of what needs to be done, as given in Algorithm 7, is as follows: We compute new minimum r_b-t- and minimum r_d-t-cuts (line 5) and keep the former r_i-t-cuts for the remaining clusters in $\Theta(G^\oplus)$ and $W(G^\oplus)$ (line 3). To also conserve the clusters C_b and C_d we try to apply Scenario (a). To this end the attribute car is

Algorithm 7: Inter-Cluster Edge Addition

```
Input: $W(G), \Theta(G), G^{\oplus}_{\alpha} = (V, E \cup \{\{b,d\}\}, c^{\oplus}_{\alpha})$, $\{b,d\}$ with weight $\Delta$
Output: $W(G^{\oplus}), \Theta(G^{\oplus})$
1  $\Theta_{\text{ten}}$, $D(r_b)$, $D(r_d) \leftarrow \emptyset$, $W_{\text{ten}} \leftarrow \{r_b, r_d\}$
2  $car(b)$, $car(d) \leftarrow$ False
3  $W(G^{\oplus}) \leftarrow W(G) \setminus \{r_b, r_d\}$, $\Theta(G^{\oplus}) \leftarrow \Theta(G) \setminus \{\theta_b, \theta_d\}$
4  forall the $r_i \in \{r_b, r_d\}$ do
5      $\theta_i \leftarrow$ minimum $r_i$-$t$-cut given by FlowAlgo$(v_i, t)$
6      if $c^{\oplus}_{\alpha}(\theta_i) = c_{\alpha}(\theta_i^{old}) + \Delta$ then $\theta_i \leftarrow \theta_i^{\text{old}}$      // retain old cuts
7      else
8          set $(\bar{U}, U) := \theta_i$ with $t \in U$      // just nomenclature
9          if $\theta_i$ has $\{r_i, b, d\}$ on same side then      // check attribute car
10             $car(i) \leftarrow$ True      // car for $\theta_i$
11             $\theta_i \leftarrow (\bar{U} \cup C_i, U \setminus C_i)$      // by Lemma 6 for car
12             $D(r_i) \leftarrow$ checkShadows$(\theta_i)$
13             $\theta_i \leftarrow$ adjustShadows$(\theta_i, D(r_i))$
14         else      // then cat holds
15             $\theta_i \leftarrow (\bar{U} \cap C_i, V_{\alpha} \setminus (\bar{U} \cap C_i))$      // by Lemma 6 for cat
16     Add $\theta_i$ to $\Theta_{\text{ten}}$      // pointed at by $r_i$
17 if $\exists\, r_i \in \{r_b, r_d\}$: $car(i) \wedge [\theta_i$ separates $r_j, t$; $r_i \neq r_j \in \{r_b, r_d\}]$ then
   // Scenario (a)
18     set $(\bar{U}, U) := \theta_i$, $t \in U$      // just nomenclature
19     if $car(j) \wedge [\theta_j$ separates $r_i, t]$ then // other cut is feasible, too
20         wlog. $|D(r_i)| \leq |D(r_j)|$, choose $r_i$ for reshaping      // better cut
21     $\theta_i \leftarrow (\bar{U} \cup C_j, U \setminus C_j)$      // by Scenario (a)
22     $W_{\text{ten}} \leftarrow \{r_i\}$, $\Theta_{\text{ten}} \leftarrow \{\theta_i\}$
23 else if $car(b) \wedge car(d)$ then      // Scenario (b)
24     set $(\bar{U}_b, U_b) := \theta_b$, set $(\bar{U}_d, U_d) := \theta_d$      // just nomenclature
25     wlog. $|D(r_d)| \leq |D(r_b)|$, choose $r_b$ for reshaping      // better cut
26     $\theta_b \leftarrow (U_d \cap C_b, V_{\alpha} \setminus (U_d \cap C_b))$      // by Scenario (b)
27     $W_{\text{ten}} \leftarrow \{r_b, r_d\}$, $\Theta_{\text{ten}} \leftarrow \{\theta_b, \theta_d\}$
28 Resolve all crossings in $\Theta_{\text{ten}}$ by pseudo-contraction      // necessary for
   Scenario (c(ii))
29 Add all vertices in $W_{\text{ten}}$ to $W(G^{\oplus})$, all cuts in $\Theta_{\text{ten}}$ to $\Theta(G^{\oplus})$
30 Isolate the sink $t$ from all remaining unclustered vertices      // necessary
   for Scenario (c(i/ii))
31 return $W(G^{\oplus}), \Theta(G^{\oplus})$
```

checked for the new cuts θ_b, θ_d (line 10). If θ_b or θ_d is cheaper than the old cut weight plus Δ, i.e., is not reconfirmed, and satisfies car (line 9) we reshape it such that its associated cluster is conserved by Lemma 6, followed by potential shadowings of former r_i (line 12). Otherwise (if it satisfies cat, line 14) it does not shadow or cut any other cluster by pseudo-contraction and Lemma 6, so we

```
Procedure checkShadows($\theta$)
1 forall the $r_j \in W(G) \setminus \{r_b, r_d\}$ do
2     if $\theta$ separates $r_j$ and $t$ then delete $\theta_j$ from $\Theta(G^\oplus)$, move $r_j$ from
      $W(G^\oplus)$ to $D$
3 return $D$
```

```
Procedure adjustShadows($\theta$, $D$)
1 set $(\bar{U}, U) := \theta$, $t \in U$                         // just nomenclature
2 forall the $r_j \in D$ do $\theta \leftarrow (\bar{U} \cup C_j, U \setminus C_j)$ // by pseudo-contraction
3 forall the $r_j \in W(G) \setminus (\{v_b, v_d\} \cup D)$ do $\theta \leftarrow (\bar{U} \setminus C_j, U \cup C_j)$
4 return $\theta$
```

can skip checkShadows and adjustShadows.

If possible we apply Scenario (a) in line 17–22. In case that both checked cuts satisfy car and separate t from the other vertex, respectively, as an optional step for the sake of temporal smoothness, the algorithm chooses that cut for reshaping that shadows less of the remaining clusters (line 20). If Scenario (a) is skipped, then both checked cuts have been reconfirmed before or both cuts meet Scenario (b) (line 23–27) or at least one meets Scenario (c(i)) or (c(ii)). Scenario (b) finally ends up with two cuts akin to those of Scenario (c(ii)) but non-crossing. As an optional step for the sake of temporal smoothness, the algorithm here chooses that cut for reshaping that shadows more of the remaining clusters (line 25). In Scenario (c(ii)) it might become necessary to make the two cuts in Θ_{ten} non-crossing (line 28). Furthermore, some vertices of the former clusters C_b and C_d might still remain unclustered. For clustering these vertices we compute their set of minimum v-t-cuts and render them non-crossing by pseudo-contraction, possibly shadowing one another or some previous representatives.

4.2 Vertex Modifications

In this section we consider the insertion and deletion of single vertices in a dynamic graph $G = (V, E)$. Augmenting V by a new vertex d realizes a vertex insertion; to delete a vertex d from G the vertex d needs to be isolated, i.e., d is only removable from G if all incident edges were removed by edge modifications first. Thus, a vertex modification of G solely involves vertex d yielding $G^\oplus$ if it is newly inserted, and $G^\ominus$ if it is deleted from G.

Note that a disconnected vertex d in G is no longer disconnected in the augmented graph G_α, as d is adjacent to t via an edge with weight α. The following lemma describes how a clustering $\mathcal{C}(G)$ resulting from Cut-Clustering behaves with respect to a disconnected vertex d in G.

Lemma 7 *Given a disconnected vertex d in graph $G = (V, E)$ and a clustering $\mathcal{C}$ for G resulting from* Cut-Clustering*. Furthermore let Θ denote the set of* *non-crossing, non-nested split cuts induced by $\mathcal{C}$. Then Θ contains cut $\theta_d := (\{d\}, V \setminus \{d\})$ with $\{d, t\}$ as cut pair. This is, $C_d := \{d\} \in \mathcal{C}$.*

Proof. It is easy to see that θ_d has weight α with respect to G_α, and thus is a minimum d-t-cut. Assume vertex $d \in C_i \neq C_d$, i.e., $|C_i| \geq 2$. Cut $\theta_i := (C_i, V_\alpha \setminus C_i)$ is a minimum $r(C_i)$-t-cut for a vertex $r(C_i) \in C_i \in \mathcal{C}$ which has at least weight 2α. Thus, $r(C_i) \neq d$. But the $r(C_i)$-t-cut $(C_i \setminus \{d\}, V_\alpha \cup \{d\})$ has weight $c(\theta_i) - \alpha$ which makes it cheaper than θ_i. This contradicts the assumption that C_i containing d is a valid cluster in $\mathcal{C}$. ⊒

With Lemma 7 a disconnected vertex in G always forms a singleton cluster. This fact makes updating a clustering $\mathcal{C}$ after a vertex modification very simple: A vertex deletion removes the corresponding cluster in $\mathcal{C}$, a vertex insertion creates a new singleton in $\mathcal{C}$ and stores the new vertex together with t as associated cut pair, i.e., the vertex becomes the representative of the new cluster.

5 Performance of the Algorithm

Temporal Smoothness. Our secondary criterion, which we left unformalized, to preserve as much of the previous clustering as possible, in parts synergizes with effort-saving, an observation foremost reflected in the usage of $T_\circ$. Lemma 4 and Observations 1 and 2 nicely enforce temporal smoothness. However, in some cases we must cut back on this issue, e.g., when we examine which other candidates for new representatives are shadowed by another one, as for example in line 8 of Algorithm 5. Here it entails many more cut-computations and a combinatorially non-trivial problem to find best-shaped cuts and an ordering of W_{ten} to optimally preserve old clusters. For our experiments we ordered the vertices in W_{ten} by decreasing degrees according to the recommendation of Flake et al. for the static Cut-Clustering. Independent of the ordering of W_{ten} we can give the following stability guarantee:

Lemma 8 *Let Algorithm 6 use for each considered vertex v in line 14 that cut among all minimum v-t-cuts which has the fewest number of vertices on the side containing v, i.e., the* community *of v with respect to t. Assume further Algorithm 6 applies cut $(C_{b/d}, V_\alpha \setminus C_{b/d})$ if a minimum v-t-cut of the same weight is found.*

Let Algorithm 7 use the communities of r_b and r_d in line 5, and in line 30 the communities for each considered vertex v. Assume further Algorithm 6 applies the cuts $(C_b, V_\alpha \setminus C_b)$ and $(C_d, V_\alpha \setminus C_d)$ if a respective minimum v-t-cut of the same weight is found.

With these slight changes each of our update algorithms guarantees stability, i.e., returns the previous clustering $\mathcal{C}(G)$ if $\mathcal{C}(G)$ still fulfills the invariant *for the modified graph $G^{\oplus(\ominus)}$.*

As our experiments serve as a first proof of concept, focusing on the number of necessary maximum-flow calculations, we did not explicitly implement the prerequisites of this lemma for the cases of intra-cluster deletion and inter-cluster insertion. However, since most minimum u-v-cuts were unique in our instance, these are more often than not satisfied implicitly. The proof for the deletion algorithms relies on the fact that any output clustering differing in cluster C_i requires at least one minimum $r(C_i)$-t-cut ($r(C_i) \in C_i$) to separate b, d, invalidating $\mathcal{C}(G)$. Intra-cluster insertion obviously retains a valid previous clustering; for inter-cluster insertion an argument akin to that used for the deletion algorithms can be used.

Proof. Consider inter-cluster deletion (Algorithm 4) first. To return a new clustering $\mathcal{C}(G^{\ominus})$ different from $\mathcal{C}(G)$ the algorithm needs to find a new cheaper minimum $r(C_i)$-t-cut for at least one representative $r_i \in W(G) \setminus \{r_b, r_d\}$. As the previous clustering is also valid for $G^{\ominus}$, there must exist another vertex $u \in C_i$ that serves as a witness that the cut θ_i (defining C_i) still constitutes a minimum u-t-cut in the modified graph $G^{\ominus}_{\alpha}$. However, each cheaper minimum r_i-t-cut found by the algorithm also separates u and t. This contradicts the existence of a vertex $u \in C_i$ such that $\{u, t\}$ is a cut pair for θ_i in $G^{\ominus}_{\alpha}$.

Considering intra-cluster deletion (Algorithm 6), a new minimum $r_{b,d}$-t-cut calculated by Algorithm 6 always spares the part of $G^{\ominus}_{\alpha}$ containing all other clusters and t according to Lemma 5 and thus does not shadow any cluster in $\mathcal{C}(G) \setminus \{C_{b/d}\}$. If the previous representative $r_{b/d}$ still serves as a witness that the cut defining $C_{b/d}$ still constitutes a minimum $r_{b/d}$-t-cut in the modified graph $G^{\ominus}_{\alpha}$ the lemma holds. Otherwise, the new cut $\theta_{b,d}$ is the community of $r_{b/d}$ with respect to t. Furthermore, all the above arguments for the inter-cluster deletion case also apply here to the clusters in $\mathcal{C}(G) \setminus \{C_{b/d}\}$. Thus, these clusters are again found. We show now that (a) while isolating t from the remaining unclustered vertices, any new minimum v-t-cut $(C_v, V_\alpha \setminus C_v)$ with $v \in C_v$ is nested in $C_{b/d}$. In this context nested means that $C_v \subseteq C_{b/d}$. And we show that (b) Algorithm 6 finds a witness $u_{b/d}$ that the cut defining $C_{b/d}$ still constitutes a minimum $u_{b/d}$-t-cut in the modified graph $G^{\ominus}_{\alpha}$.

To see (a) recall that $(C_v, V_\alpha \setminus C_v)$ is supposed to be the community of v with respect to t. For the sake of notational simplicity we also associate the cut side C_v with the community of v. Since $(C_v, V_\alpha \setminus C_v)$ pseudo-contracts $V_\alpha \setminus C_{b/d}$ in the view of $C_{b/d}$, there exists a Gomory-Hu execution for $G^{\ominus}_{\alpha}$ that yields a minimum-cut tree $T(G^{\ominus}_{\alpha})$ in which the represented minimum v-t-cut is nested in $C_{b/d}$, and thus, $C_v \subseteq C_{b/d}$ particularly.

Aspect (b) holds due to Lemma 3.9 given in [7] that states that communities are either disjoint or nested: Suppose $u_{b/d}$ to be a witness for $C_{b/d}$ but shadowed by a community C_v in $G^{\ominus}_{\alpha}$, i.e., $u_{b,d} \in C_v$. Then the community C_u of $u_{b,d}$ is nested in C_v. Furthermore, it is $C_v \subseteq C_{b/d}$ according to (b) which yields that $C_u \subseteq C_v \subseteq C_{b/d}$ all induce cuts with the same costs in $G^{\ominus}_{\alpha}$. Thus, v is also a witness for $C_{b/d}$ and the algorithm applies $(C_{b/d}, V_\alpha \setminus C_{b/d})$ as split cut. Furthermore, according to (b) this split cut is not reshaped. Obviously, if $u_{b/d}$ is not shadowed, the algorithm considers this vertex anyway.

Considering inter-cluster insertion (see Algorithm 7), all clusters in $\mathcal{C}(G) \setminus \{C_b, C_d\}$ are known to be still valid for $G^{\oplus}$. With the same arguments already used above for the case of intra-cluster deletion, for any considered vertex $v \in C_b$ and $v \in C_d$ (particularly for r_b and r_d) that is not reconfirmed, the community $(C_v, V_\alpha \setminus C_v)$ is nested in C_b and C_d, respectively, and the algorithm finds a witness for each of the cuts induced by C_b and C_d. $\square$

Running Times. We universally express running times of our algorithms in terms of the number of necessary maximum-flow computations, leaving open how these are done. A summary of tight bounds is given in Table 1. The columns *lower bound*/*upper bound* denote bounds for the—possibly rather common—case that the old clustering is still valid after some graph update. As discussed in the last subsection, the last column (*smooth*) states whether our algorithms *always* return the previous clustering, in case its valid; the numbers in brackets denotes a tight lower bound on the running time, in case our algorithms do find that previous clustering.

	best case	worst case	old clustering still valid		
			lower b.	upper bound	smooth
InterD	$\lvert\mathcal{C}(G^{\ominus})\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	Yes
IntraD	1	$\lvert\mathcal{C}(G)\rvert + \lvert C_{b/d}\rvert - 1$	1	$\lvert\mathcal{C}(G)\rvert + \lvert C_{b/d}\rvert - 1$	No (1)
InterI	2	$\lvert C_b\rvert + \lvert C_d\rvert$	2	$\lvert C_b\rvert + \lvert C_d\rvert$	No (2)
IntraI	0	0	0	0	Yes

Table 1: Bounds on the number of maximum-flow calculations.

For Inter-Cluster Edge Deletion (Algorithm 4) in the best case we only calculate as many cuts as new clusters arise while separating t from *all* neighbors, except $r(C_b)$ and $r(C_d)$ (compare to Figure 11). We require at most $|\mathcal{C}(G)| - 2$ cuts, as neighbors of t might get shadowed after their processing. In this case we calculate more cuts than finally needed to define the new clustering. Since $|\mathcal{C}(G^{\ominus})| = |\mathcal{C}(G)|$ in case the old clustering remains valid, the other bounds are correct and we know we will find the old clustering. Algorithm 6 (Intra-Cluster Edge Deletion) needs to examine all clusters in $\mathcal{C}(G) \setminus \{C_{b/d}\}$, and potentially all vertices in $C_{b/d}$—even if the previous clustering is retained, e.g., with every vertex shadowing the one cut off right before, and pair $\{r(C_{b/d}), t\}$ getting hidden. Obviously, we attain the lower bound if we cut away $r(C_{b/d})$ from t, directly preserving $C_{b/d}$ and all the other clusters. For Inter-Cluster Edge Insertion (Algorithm 7), we potentially end up separating every single vertex in $C_b \cup C_d$ from t, one by one, even if the previous clustering is valid, as, e.g., $r(C_b)$ might become shadowed by some other $v \in C_b$, which ultimately yields the upper bound. In case the previous clustering is valid, however, we might get away with simply cutting off $r(C_b)$ and $r(C_d)$, alongside their former clusters. This means, there is no guarantee that we return the previous clustering; still, with two cuts ($r(C_b)$-t and $r(C_d)$-t), we are quite likely to do so. The row for Intra-Cluster Edge Insertion is obvious.

	total	Inter-Del	Intra-Del	Inter-Ins	Intra-Ins
modifications	61870	3742	26179	10010	21939
%	100	6.0482	42.3129	16.1791	35.4598
advantage static	40	0	40	0	0
of total modifications %	0.0647	0.0000	0.0647	0.0000	0.0000
advantage dynamic	61830	3742	26139	10010	21939
of total modifications %	99.9353	6.0482	42.2483	16.1791	35.4598

Table 2: Total number of modifications decomposed into different scenarios.

Note that a computation from scratch (static algorithm) entails a tight upper bound of $|V|-1$ maximum-flow computations for all four cases, in the worst case; although, in practice, the heuristic recommended by Flake et al. usually finds a new clustering in time proportional to the total number of new clusters. In the best case it needs as many cut computations as new clusters arise. Comparing this to the bound for updating an inter-cluster deletion in the best case, lets us expect only little effort saving for this case; while the case of intra-cluster insertion promises the biggest effect of effort saving.

Experiments. In this brief section, we very roughly describe some experiments we made with an implementation of the update algorithms described above, just for a first proof of concept. The instance we use is a network of e-mail communications within the Fakultät für Informatik at KIT (formerly Universität Karlsruhe) [20]. Vertices represent members and edges correspond to e-mail contacts, weighted by the number of e-mails sent between two individuals during the last 72 hours. This means, each e-mail has a fixed time to live. After that time the contribution of the e-mail to the weight of the edge expires and the weight of the edge decreases. We process a queue of 69 739 elementary modifications, 61 870 of which are actual edge modifications, on an initial graph with $|V| = 247$ and $|E| = 307$. This queue represents about three months, starting on Sunday (2006-10-22). The number of vertices varies between 172 and 557, the number of edges varies between 165 and 1190. We delete zero-weight edges and isolated nodes. Following the recommendations of Flake et al. [7], we choose $\alpha = 0.15$ for the initial graph, yielding 73 clusters. We compare their static algorithm (see Section 2.1) and our dynamic algorithm in terms of the number of maximum-flow computations necessary to maintain a clustering. Forty times out of the 61 870 total operations, the static computation needed less maximum flows than the dynamic update. In all remaining cases (99.93%) the update algorithm was at an advantage (see Table 2).

The first two rows of Table 3 show the numbers of clusters found by the static and dynamic approach over the whole experiment. As both algorithms range at similar levels we can be sure the observed savings are not induced by trivial clusterings. Thus, comparing dynamic and static flow computations is justified: For the 61 870 proper steps, static computation needed 3 300 413 maximum flows, and our dynamic update needed 736 826, saving more than

	total	Inter-Del	Intra-Del	Inter-Ins	Intra-Ins
static clusters	3186155	314979	1090890	748442	1031844
%	100	9.8859	34.2384	23.4904	32.3852
dynamic clusters	3185398	314923	1090414	748287	1031774
%	100	9.8865	34.2316	23.4912	32.3907
static flows	3300413	324098	1131538	773730	1071047
%	100	9.8199	34.2847	23.4434	32.4519
dynamic flows	736826	308904	403499	24423	0
of total static flows %	22.3253	9.3596	12.2257	0.7400	0.0000
amortized static costs	1.0359	1.0290	1.0373	1.0338	1.0380
amortized dynamic costs	0.2313	0.9809	0.3700	0.0326	0.0000
flow savings	2563587	15194	728039	749307	1071047
of total static flows %	77.6747	0.4604	22.0590	22.7034	32.4519
average flow savings	41.4351	4.0604	27.8100	74.8558	48.8193

Table 3: Total number of clusters, flows and savings for different scenarios.

77% maximum flows, such that one dynamic cluster on average costs 0.23 flow computations. The amortized costs of 1.03 flows for a static cluster affirm the running time to be proportional to the total number of new clusters, as stated by Flake et al. This running time is also visible in Figure 15, which shows the consecutive development of the graph structure over one day (Monday, 2006-10-23). Obviously, the static and dynamic clusterings (upper red and lower green line) behave similarly. Note that the scale for static clusterings and flows is offset by about 20 clusters/flows for readability. However, the dynamic flows (blue dots) cavort around the clusters or, even better, near the ground, which means there are only few flow computations needed. In contrast, most of the static flow amounts (orange dots) are still proportional but clearly higher than the number of clusters in the associated static clustering.

Regarding the total number of edge modifications the savings finally average out at 41.4 flows (Table 3), while inter-cluster insertions save the most effort per modification. This is, the case of inter-cluster insertion surprisingly outperforms the trivial intra-cluster insertions.

6 Conclusion

We have proven a number of results on the nature of minimum u-v-cuts in changing graphs, allowing for feasible algorithms which efficiently update specific parts of a minimum-cut tree and thus fully dynamically maintain a graph clustering based on such trees, as defined by Flake et al. [7] for the static case, under arbitrary atomic changes. The striking feature of graph clusterings computed by this method is that they are guaranteed to yield a certain expansion—a bottleneck measure—within and between clusters, tunable by an input parameter α. As a secondary criterion for our updates we encourage temporal smoothness, i.e.,

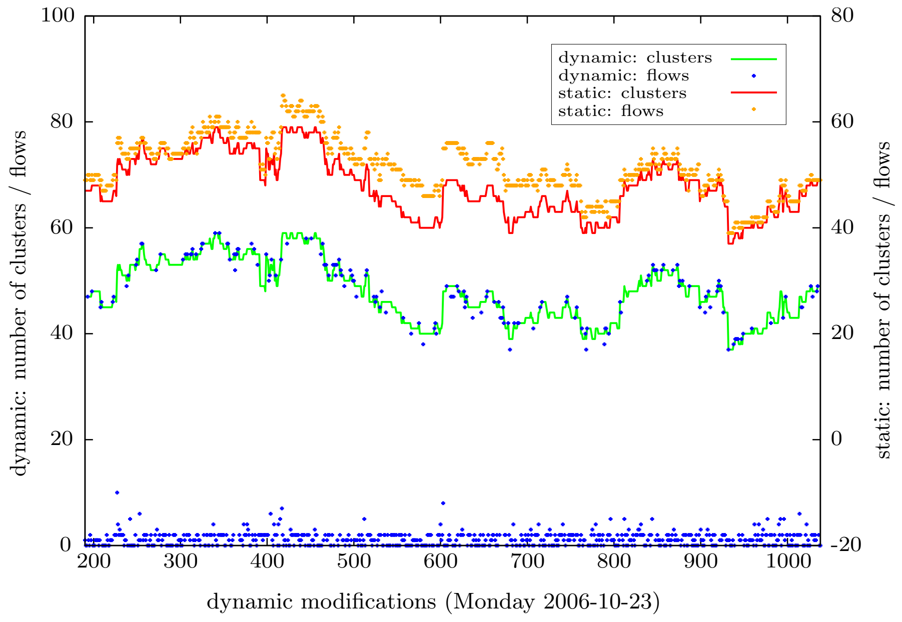


Figure 15: Numbers of clusters and flows regarding consecutive clusterings (the two y-axes have a different offset for better readability).

changes to the clusterings are kept at a minimum, whenever possible. Furthermore, we disprove an earlier attempt to dynamize such clusterings [26, 25]. Our experiments on real-world dynamic graphs confirm our theoretical results and show a significant practical speedup over the static algorithm of Flake et al. [7].

Future work on dynamic minimum-cut tree clusterings will include analyzing the potential of choosing "better" cuts (if a minimum cut is not unique) and specific orderings of W_{ten} in the algorithms in order to further improve temporal smoothness. The dynamic update of a single maximum s-t-flow is a means to gain further speedup beyond the number of cut calculations. Since our algorithm is able to deal with arbitrary cuts, on the one hand any existing max-flow-min-cut-implementation and any existing update technique can be used, and on the other hand, there is no need for prospectively designed update algorithms to respect any restrictions.

Moreover, we intend to systematically compare our work to other dynamic clustering techniques and to investigate a method for dynamically adapting the parameter α. Related to the latter, we will try to expand our update algorithms to the Hierarchical Cut-Clustering method also given by Flake et al. [7] which considers a sequence of values for α. How to deal with offline changes is another interesting question for both methods, Cut-Clustering and Hierarchical Cut-Clustering.

We thank the anonymous reviewers for their thoughtful comments and their helpful suggestions on how to achieve an even better readability and understanding.

References

[1] V. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), 2008.

[2] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Höfer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, February 2008.

[3] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer, February 2005.

[4] F. R. K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[5] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004.

[6] J. Duch and A. Arenas. Community Detection in Complex Networks using Extremal Optimization. *Physical Review E*, 72(027104):1–4, 2005.

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

[8] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, 2009.

[9] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Science of the United States of America*, 104(1):36–41, 2007.

[10] M. Gaertler. Clustering with Spectral Methods. Diplomarbeit, Fachbereich Informatik und Informationswissenschaft, Universität Konstanz, March 2002.

[11] 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.

[12] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[13] 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.

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

[15] R. Görke, P. Maillard, C. Staudt, and D. Wagner. Modularity-Driven Clustering of Dynamic Graphs. In P. Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 436–448. Springer, May 2010.

[16] R. Guimerà and L. A. N. Amaral. Functional Cartography of Complex Metabolic Networks. *Nature*, 433:895–900, February 2005.

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

[18] T. Hartmann. Clustering Dynamic Graphs with Guaranteed Quality. Master's thesis, Department of Informatics, October 2008. Diplomarbeit.

[19] 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.

[20] KIT. Email Network of KIT Informatics, 2011. 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.

[21] M. E. J. Newman. Fast Algorithm for Detecting Community Structure in Networks. *Physical Review E*, 69:066133, 2004.

[22] M. E. J. Newman. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Science of the United States of America*, 103(23):8577–8582, June 2006.

[23] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113):1–16, 2004.

[24] J. Reichardt and S. Bornholdt. Statistical Mechanics of Community Detection. *Physical Review E*, 74(016110):1–16, 2006.

[25] B. Saha and P. Mitra. Dynamic Algorithm for Graph Clustering Using Minimum Cut Tree. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 667–671. IEEE Computer Society, December 2006.

[26] B. Saha and P. Mitra. Dynamic Algorithm for Graph Clustering Using Minimum Cut Tree. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 581–586. SIAM, 2007.

[27] S. E. Schaeffer. Graph Clustering. *Computer Science Review*, 1(1):27–64, August 2007.

[28] U. von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[29] S. White and P. Smyth. A Spectral Clustering Approach to Finding Communities in Graphs. In *Proceedings of the fifth SIAM International Conference on Data Mining*, pages 274–285. SIAM, 2005.