Dynamic Graph Clustering Using Minimum-Cut Trees

Robert Görke, Tanja Hartmann, and Dorothea Wagner

Faculty of Informatics, Universität Karlsruhe (TH), Karlsruhe Institute of Technology (KIT)*
{rgoerke,hartmn,wagner}@informatik.uni-karlsruhe.de

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 minimum-cut trees, 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.

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 in networks. Countless formalizations thereof 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. For an overview and recent results on graph clustering see, e.g., [2, 1] and references therein. Inspired by the work of Kannan et al. [8], Flake et al. [3] 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 [5], 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 [12, 11], however, we found it to be erroneous. We are not aware of any other dynamic graph-clustering algorithms in the literature.

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. [3], allowing arbitrary atomic changes in the graph, and keeping consecutive clusterings similar (a notion we call *temporal smoothness*). Our algorithms build upon partially updating an intermediate minimum-cut tree of a graph in the spirit of Gusfield's [6] simplification of the Gomory-Hu algorithm [5]. We show that, with only slight modifications, our techniques can update entire min-cut trees. 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 Sec. 1. Then, in Sec. 2, we revisit some results from [5, 6, 3], convey them to a dynamic scenario, and derive our central results. In Section 3 we give actual update algorithms, which we analyse in Sec. 4, concluding in Sec. 5.

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 solely concern edges; the reason for this is, that vertex insertions and deletions are trivial as long as the vertex is disconnected. Thus, a 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

* This work was partially supported by the DFG under grant WA 654/15-1.

simplicity we will not handle changes to the weight of an edge, since this can be done almost exactly as deletions and additions. Bridge edges in G require special treatment when deleted or inserted. However, since they are both simple to detect and to deal with, we ignore them by assuming the dynamic graph to stay connected at all times.

The *minimum-cut tree* $T(G) = (V, E_T, c_T)$ of G is a tree on V and represents for any node pair $\{u, v\} \in \binom{V}{2}$ a minimum-u-v-cut $\theta_{u,v}$ in G by the cheapest edge on the unique path between u and v in $T(G)$. Neither must this edge be unique, nor $T(G)$. For $b, d \in V$ we always call this path γ (as a set of edges). An edge $e_T = \{u, v\}$ of T induces the cut $\theta_{u,v}$ in G, sometimes denoted θ_v if the context identifies u. We sometimes identify e_T with the cut it induces in G. For details on min-cut trees, see the pioneering work by Gomory and Hu [5] or the simplifications by Gusfield [6].

A *contraction* of G by $N \subseteq V$ means replacing set N by a single super-node 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 $M \subseteq E$. A *clustering* $\mathcal{C}(G)$ of G is a partition of V into *clusters* C_i, usually conforming to the paradigm of *intra-cluster density and inter-cluster sparsity*. We start by giving some fundamental insights, which we will rely on in the following, leaving their rather basic proofs to the reader.

Lemma 1. *Let $e = \{u, v\} \in E_T$ be an edge in $T(G)$.*

Consider $G_\oplus$: If $e \notin \gamma$ then e is still a min-u-v-cut with weight $c(\theta_e)$. If $e \in \gamma$ then its cut-weight is $c(\theta_e)+\Delta$, it stays a min-u-v-cut iff $\forall u$-v-cuts θ' in G that do not separate *b, d: $c(\theta') \geq c(\theta_e)+\Delta$.*

Consider $G_\ominus$: If $e \in \gamma$ then e remains a min-u-v-cut, with weight $c(\theta_e) - \Delta$. If $e \notin \gamma$ then it retains weight $c(\theta_e)$, it stays a min-u-v-cut iff $\forall u$-v-cuts θ' in G that separate *b, d: $c(\theta') \geq c(\theta_e)+\Delta$.*

2 Theory

2.1 The Static Algorithm

Finding communities in the world wide web or in citation networks are but example applications of graph clustering techniques. In [3] 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. [8]:

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

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. [8], which bases on the related measure conductance [1], Flake et al. [3] design a graph clustering algorithm that, given parameter α, asserts the following:[2]

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

These quality guarantees—simply called quality in the following—are due to special properties of min-cut trees, which are used by the clustering algorithm, as given in Alg. 1 (comp. [3]). It performs the following steps: Add an artificial node t to G, and connect t to all other vertices by weight α. Then, compute a min-cut tree $T(G_\alpha)$ of this augmented graph. Finally, remove t and let the resulting connected components of T define the clustering. In the following, we will call the fact that a clustering

Algorithm 1: CUT-CLUSTERING

Input: Graph $G = (V, E, c)$, α

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

[1] conductance is similar to expansion but normalizes cuts by total incident edge weight instead of the number of vertices in a cut set.

[2] The disjoint union $A \cup B$ with $A \cap B = \emptyset$ is denoted by $A \dot\cup B$.

can be computed by this procedure the invariant. For the
proof that Cut-Clustering yields a clustering that obeys Eq. (2), we refer the reader to [3]. Flake at al. further show how nesting properties of min cuts [4] can be used to avoid computing the whole min-cut tree T and try to only identify those edges of T incident with t. Their recommendation for finding these edges quickly, is to start with separating high degree nodes from t. Furthermore they show that this property yields 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. For now, however, general $G_{\oplus(\ominus)}$ are considered.

A Dynamic Attempt. Saha and Mitra [12] published an algorithm that aims at the same goal as our work. Unfortunately, we discovered a methodical error in this work. Roughly speaking, it seems as if the authors implicitly 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' procedures and give counter-examples in the Appendix. These issues, alongside correct parts, are further scrutinized in-depth by Hartmann [7].

2.2 Minimum-Cut Trees and the Gomory-Hu Algorithm

We briefly describe the construction of a min-cut tree as proposed by Gomory and Hu [5] and simplified by Gusfield [6]. Although we will adopt ideas of the latter work, we first give Gomory and Hu's algorithm (Alg. 2) as the foundation.

Algorithm 2: Gomory-Hu (Minimum-Cut Tree)

Input: Graph $G = (V, E, c)$
Output: Min-cut tree of G

1 Initialize $V_* := \{V\}, E_* := \emptyset$ 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 $\quad \{u, v\} \leftarrow$ arbitrary pair from $\binom{S}{2}$
4 $\quad$ **forall** $S_j \sim S$ *in* $T_*(G)$ **do** $N_j \leftarrow$ subtree of S with $S_j \in N_j$
5 $\quad G_S = (V_S, E_S, c_S) :=$ in G contract each subtree N_j to node η_j `// subtree contraction`
6 $\quad (U, V_S \setminus U) \leftarrow$ min-u-v-cut in G_S, weight δ, $u \in U$
7 $\quad S_u \leftarrow S \cap U$, and $S_v \leftarrow S \cap (V_S \setminus U)$ `// split` $S = S_u \cup S_v$
8 $\quad V_* \leftarrow (V_* \setminus \{S\}) \cup \{S_u, S_v\}$, $E_* \leftarrow E_* \cup \{\{S_u, S_v\}\}$, $c_*(S_u, S_v) \leftarrow \delta$
9 $\quad$ **forall** *former edges* $e_j = \{S, S_j\} \in E_*$ **do**
10 $\quad\quad$ **if** $\eta_j \in U$ **then** $e_j \leftarrow \{S_u, S_j\}$ `// either reconnect` S_j `to` S_u
11 $\quad\quad$ **else** $e_j \leftarrow \{S_v, S_j\}$ `// or reconnect` S_j `to` S_v
12 **return** $T_*(G)$

The algorithm builds the min-cut tree of a graph by iteratively finding min-u-v-cuts for vertices that have not yet been separated by a previous min-cut. The *intermediate* min-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 nodes (line 1). Then, 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 5) before cutting, which yields G_S—a notation we will continue using in the following. The split of S into (S_u, S_v) is then defined by a min-u-v-cut in G_S (line 6). Afterwards, N_j is reconnected, 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 min-u-v-cut in G.

An *execution* $\mathrm{GH} = (G, F, K)$ of Gomory-Hu is characterized by graph G, sequence F of $n-1$ *step pairs* (compare to line 3) of nodes and sequence K of *split cuts* (compare to line 6). Pair $\{u, v\} \subseteq V$ is a *cut pair* of edge e of cut-tree T if θ_e is a min-u-v-cut in G.

Theorem 1. *Consider a set $M \subseteq E_T$ and let $T_\circ(G) = (V_\circ, M, c_\circ)$ be $T(G)$ with $E_T \setminus M$ contracted. Let f and f' be sequences of the elements of M and $E_T \setminus M$, respectively, and k and k' the*

corresponding sequences of edge-induced cuts of G. The Gomory-Hu *execution* $\mathrm{GH} = (G, f' \cdot f, k' \cdot k)$[3] *has* $T_\circ(G)$ *as intermediate min-cut tree (namely after* f*).*

In the following we will denote by $T_\circ$ an intermediate min-cut tree which serves as a starting point, and by T_* a working version. We prove The. 1 by induction on the edges in $f' \cdot f$, however, for the sake of brevity we move the full proof to App. A. This theorem states that if for some reason we can only be sure about a subset of the edges of a min-cut tree, we can contract all other edges to super-nodes and consider the resulting tree $T_\circ$ as the correct intermediate result of some GH, which can then be continued. One such reason could be a dynamic change in G, such as the insertion or the deletion of an edge, which by Lem. 1 maintains a subset of the old min-cuts. Thus we could already design an effort-saving algorithm for dynamically updating min-cut trees: contract all those edges of $T(G)$ which might not be valid any more, yielding $T_\circ(G_{\oplus(\ominus)})$, as depicted in Figure 1, and start a run of Gomory-Hu with this intermediate min-cut tree.

(a) $T_\circ$ by contracting all edges of γ in $T(G)$ (b) $T_\circ$ by contracting all edges of $E_T \setminus \gamma$

Fig. 1. Sketches of intermediate min-cut trees $T_\circ$; for $G_\oplus$ (a) we contract γ to a node, and for $G_\ominus$ (b) we contract each connected component induced by $E_T \setminus \gamma$, yielding a path of nodes.

2.3 Using Arbitrary Minimum Cuts in G

Gusfield [6] presented an algorithm for finding min-cut trees which avoids complicated contraction operations. In essence he provided rules for adjusting iteratively found min-u-v-cuts in G (instead of in G_S) that potentially cross, such that they are consistent with the Gomory-Hu procedure and thus non-crossing, but still minimal. We need to review and generalize some of these ideas as to fit our setting. The following lemma essentially tells us, that at any time in Gomory-Hu, for any edge e of $T_\circ$ there exists a cut pair of e in the two nodes incident to e.

Lemma 2 (Gus. [6], Lem. 4[4]). *Let S be cut into S_x and S_y, with $\{x, y\}$ being a cut pair (not necessarily the step pair). Let now $\{u, v\} \subseteq S_x$ split S_x into S_{xu} and S_{xv}, wlog. with $S_y \sim S_{xu}$ in T_*. Then, $\{x, y\}$ remains a cut pair of edge $\{S_y, S_{xu}\}$ (we say edge $\{S_x, S_y\}$ gets* reconnected*). If $x \in S_{xv}$, i.e., the min-u-v-cut separates x and y, then $\{u, y\}$ is also a cut pair of $\{S_{xu}, S_y\}$.*

In the latter case of Lem. 2, we say that pair $\{x, y\}$ gets *hidden*, and, in the view of vertex y, its former counterpart x gets *shadowed* by u (or by S_u). 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, each edge in T has at least one cut pair in the incident nodes. We define the *nearest cut pair* of an edge in T_* as follows: As long as a step pair $\{x, y\}$ is in adjacent nodes S_x, S_y, it is the nearest cut pair of edge $\{S_x, S_y\}$; if a nearest cut pair gets hidden in T_* by a step of Gomory-Hu, as described in Lem. 2 if $x \in S_{xv}$, the nearest cut pair of the reconnected edge $\{S_y, S_{xu}\}$ becomes $\{u, y\}$ (which are in the adjacent nodes S_y, S_{xu}). The following theorem basically states how to iteratively find min-cuts as Gomory-Hu, without the necessity to operate on a contracted graph.

Theorem 2 (Gus. [6], The. 2[5]). *Let $\{u, v\}$ denote the current step pair in node S during some* GH*. If $(U, V \setminus U)$, $(u \in U)$ is a min-u-v-cut in G, then there exists a min-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 $S \cap (V \setminus U) = S \cap (V_S \setminus U_S)$, $(u \in U_S)$.*

Being an ingredient to the original proof of Theorem 2, the following Lem. 3 gives a constructive assertion, that tells us how to arrive at a cut described in the theorem by inductively adjusting a given min-u-v-cut in G. Thus, it is the key to avoiding contraction and using cuts in G by rendering min-u-v-cuts non-crossing with other given cuts.

[3] The term $b \cdot a$ denotes the concatenation of sequences b and a, i.e., a happens first.

[4] This lemma is also proven in [6] and [5], we thus omit a proof.

Lemma 3 (Gus. [6], Lem. 1[5]). *Let $(Y, V \setminus Y)$ be a min-x-y-cut in G ($y \in Y$). Let $(H, V \setminus H)$ be a min-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 min-u-v-cut.*

Given a cut as by Theorem 2, 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 9-11 in Alg. 2), 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_*. 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. For a suchlike run of Gomory-Hu, Gusfield shows (using Lem. 2 iteratively) that for two adjacent nodes S_u, S_v in any $T_\circ$, $r(S_u), r(S_v)$ is a cut pair of edge $\{S_u, S_v\}$, and, most importantly his Theorem 3: For $u, v \in S$ let *any* min-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 Lem. 3 and Theorem 2; then a neighboring subtree N_j of S, formerly connected by edge $\{S, S_j\}$, lies in U_S iff $r(S_j) \in U$. Since we intend to work with arbitrary intermediate min-cut trees as in Theorem 1, we do not have representatives and thus need to adapt Gusfield's Theorem 3, namely using nearest cut pairs as representatives, in order to finally enable a simplified construction of min-cut trees. The proof of the following theorem can be found in App. A.

Theorem 3 (comp. Gus. [6], The. 3[5]). *In any T_* of a* GH*, suppose $\{u, v\} \subseteq S$ is the next step pair, with subtrees N_j of S connected by $\{S, S_j\}$ and nearest cut pairs $\{x_j, y_j\}$, $y_j \in S_j$. Let $(U, V \setminus U)$ be a min-u-v-cut in G, and $(U_S, V \setminus U_S)$ its adjustion. Then $\eta_j \in U_S$ iff $y_j \in U$.*

2.4 Finding and Shaping Minimum Cuts in the Dynamic Scenario

In this section we let graph G change, i.e., we consider the addition of an edge $\{b, d\}$ or its deletion, yielding $G_\oplus$ or $G_\ominus$. First of all we define valid representatives of the nodes on $T_\circ$. By Lem. 1 and Theorem 1, given an edge addition, $T_\circ$ consists of a single super-node and many singletons, and given edge deletion, $T_\circ$ consists of a path of super-nodes; for examples see Fig. 1.

Definition 1 (Representatives in $T_\circ$).
Edge addition: *Set singletons to be representatives of themselves; for the only super-node S choose an arbitrary $r(S) \in S$.*
Edge deletion: *For each node S_i, set $r(S_i)$ to be the unique vertex in S_i which lies on γ in $T(G)$.*
New nodes during algorithm, and the choice of step pairs: *On a split of node S during the algorithm, require the step pair to be $\{r(S), v\}$ with an arbitrary $v \in S, v \neq r(S)$. Let the split be $S = S_{r(S)} \cup S_v, v \in S_v$, then define $r(S_{r(S)}) := r(S)$ and $r(S_v) := v$.*

Consider edge additions; singletons in $T_\circ$ trivially are their own representatives. Since no singleton gets split, the single super-node S gets split first, and thus only needs representatives for its parts thereafter, which are defined by the step pair, see below. With edge deletions, according to Lem. 1 each node of $T_\circ$ contains a vertex that lies on γ in the old $T(G)$, with the edges connecting these vertices being correct min-cuts in $G_\ominus$(see Fig. 1(b)), they thus are nearest cut pairs. By Lem. 2 the representatives of new nodes as defined above always define nearest cut pairs. Thus, in the case of edge additions, choosing an arbitrary step pair in S at the start is feasible.

Following Theorem 1, we define the set M of "good" edges of the old tree $T(G)$, i.e., edges that stay valid due to Lem. 1, as $M := E_T \setminus \gamma$ for the insertion of $\{b, d\}$ and to $M := \gamma$ for the deletion. Let the intermediate cut-tree $T_\circ(G_{\oplus(\ominus)})$ be $T(G)$ contracted by M. As above, let f be any sequence of the edges in M and k the corresponding cuts in G.

Lemma 4. *Given an edge addition (deletion) in G. The Gomory-Hu execution $\mathrm{GH}_{\oplus(\ominus)} = (G_{\oplus(\ominus)}, f_{\oplus(\ominus)} \cdot f, k_{\oplus(\ominus)} \cdot k)$ is feasible for $G_{\oplus(\ominus)}$ yielding $T_\circ(G)$ as the intermediate min-cut tree after sequence f, if $f_{\oplus(\ominus)}$ and $k_{\oplus(\ominus)}$ are feasible sequences of step pairs and cuts on $T_\circ(G_{\oplus(\ominus)})$.*

[5] This Lemma alongside Lemma 3, Theorem 2 and a simpler version of our Theorem 3 have been discussed in [6] and both lemmas also in [5], we thus only prove both theorems together in the Appendix.

As Lem. 4 describes a specific variant of the setting in The. 1, it also relies on induction on the split cuts in k, see App. A for its proof. It is the basis of our updating algorithms, founded on $T_\circ$'s as in Fig. 1, using arbitrary cuts in $G_{\oplus(\ominus)}$ instead of actual contractions. Still, the non-crossing nature of min-u-v-cuts allows for more effort-saving and temporal smoothness.

Definition 2 (Treetop and Wood). *Consider edge $e = \{u, v\}$ off γ, and cut $\theta = (U, V \setminus U)$ in G induced by e in $T(G)$ with γ contained in U. In the contracted graph $G_\ominus(S)$, $S \cap (V \setminus U)$ is called the* treetop $\Uparrow_e$*, and $S \cap U$ the* wood $\sharp_e$ *of e. The subtrees of S are N_b and N_d, containing b and d, respectively (see Fig. 2 for an example).*

Cuts That Can Stay. There are several circumstances which imply that a previous cut is still valid after a graph modification, making its recomputation unnecessary. The following three lemmas all give such assertions. Their proofs mostly rely on properties of Gomory-Hu-executions and on Lemma 1, they can be found in App. A.

Lemma 5. *Suppose $e_{\min}$ is the cheapest edge on γ. In $G_\oplus$, $e_{\min}$ still induces a min-b-d-cut.*

Lemma 6. *In $G_\ominus$, let $(U, V \setminus U)$ be a min-u-v-cut not separating $\{b, d\}$, with γ in $V \setminus U$. Then, a cut induced by an edge $\{g, h\}$ of the old $T(G)$, with $g, h \in U$, remains a min separating cut for all its previous cut pairs within U in $G_\ominus$, and a min g-h-cut in particular.*

Lemma 7. *Assume $g \in V$ on γ and $\{y_b, g\}, \{y_d, g\} \in \gamma$, and let wlog. $c(\{y_b, g\}) \leq c(\{y_d, g\})$. Let further $\{u, v\}$ be an edge within $\Uparrow_{\{g,h\}}$ (or $\{g, h\}$ itself) in $T(G)$. If $c_T(\{u, v\}) \leq c_T(\{y_b, g\}) - \Delta$ in the old tree, then, in $G_\ominus$, $\{u, v\}$ also induces a min-u-v-cut.*

As a corollary from Lem. 6 we get that in $T(G_\ominus)$ the entire treetops of reconfirmed edges of $T(G)$ are also reconfirmed. Cuts that can be retained save effort and encourage smoothness; however new cuts can also be urged to behave well, as follows.

The Shape of New Cuts. In contrast to the above lemmas, during a Gomory-Hu execution for $G_\ominus$, we might find an edge $\{u, v\}$ of the old $T(G)$ that is *not* reconfirmed by a computation in $G_\ominus$, but a new, cheaper min-u-v-cut $\theta' = (U, V(S) \setminus U)$ is found. For such a new cut we can still make some guarantees on its shape to resemble its "predecessor": Lemmas 8 and 9 tell us, that for any such min-u-v-cut θ', there is a min-u-v-cut $\theta = (U \setminus \Uparrow_e, (V(S) \setminus U) \cup \Uparrow_e)$ in $G_\ominus$ that (a) does not split $\Uparrow_e$, (b) but splits $V \setminus \Uparrow_e$ exactly as θ' does. Figure 2 illustrates such cuts θ (solid) and θ' (dashed).

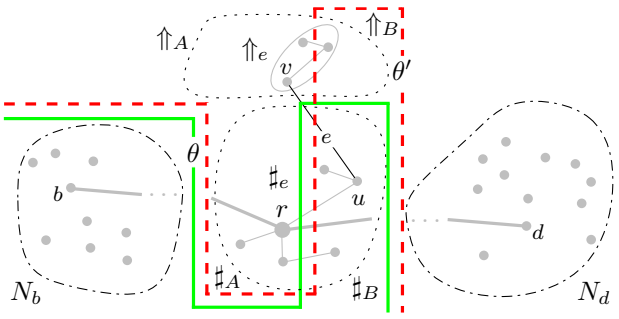


Fig. 2. Special parts of $G_\ominus$: γ (fat) connects b and d, with r on it; wood $\sharp_e$ and treetop $\Uparrow_e$ (dotted) of edge e, both cut by θ' (dashed), adjusted to θ (solid) by Lem. 9. Both $\sharp_e$ and $\Uparrow_e$ are part of some node S, with representative r, outside subtrees of r are N_b and N_d (dash-dotted). Compare to Fig. 1(b).

Lemma 8. *Given $e = \{u, v\}$ within S (off γ) in $G_\ominus(S)$. Let $(\Uparrow_A, \Uparrow_B)$ be a cut of $\Uparrow_e$ with $v \in \Uparrow_A$. Then $c_\ominus(N_b \cup \Uparrow_e, N_d \cup \sharp_e) \leq c_\ominus(N_b \cup \Uparrow_A, N_d \cup \sharp_e \cup \Uparrow_B)$. Exchanging N_b and N_d is analogous.*

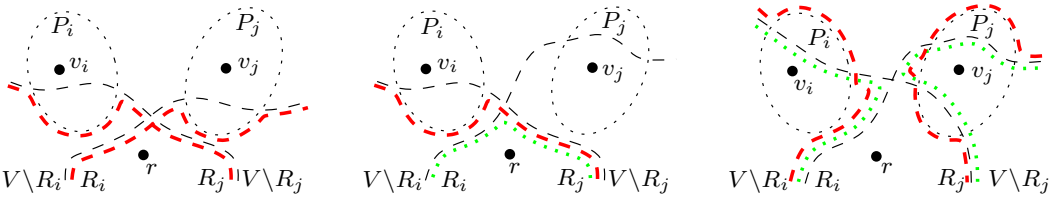
Lemma 9. *Lem. 8 can be generalized in that both considered cuts also cut the wood $\sharp_e$ in some arbitrary but fixed way.*

The proof of the above lemmas is rather technical, but conceptually it relies on the fact that if a cut which splits the treetop were cheaper, then this treetop cannot have been valid in the previous tree. While theses lemmas can be applied in order to retain treetops, even if new cuts are found,

in the following, we take a look at how new, cheap cuts can affect the treetops of *other* edges. In fact a similar treetop-conserving result can be stated.

Let G' denote an undirected, weighted graph and $\{r, v_1, \ldots, v_z\}$ a set of designated vertices in G'. Let $\Pi := \{P_1, \ldots, P_z\}$ be a partition of $V \setminus r$ such that $v_j \in P_j$. We now assume the following partition-property to hold: For each v_j it holds that for any v_j-r-cut $\theta'_j := (R_j, V \setminus R_j)$ (with $r \in R_j$), the cut $\theta_j := (R_j \setminus P_j, (V \setminus R_j) \cup P_j)$ is of at most the same weight. The crucial observation is, that Lem. 9 implies this partition-property for $r(S)$ and its neighbors in $T(G)$ that lie inside S of $T_\circ$ in $G_\ominus$. Treetops thus are the sets P_j. However, we keep things general for now.

Consider a min-v_i-r-cut $\theta'_i := (R_i, V \setminus R_i)$, with $r \in R_i$, that does not split P_i and an analog min-v_j-r-cut θ'_j (by the partition-property they exist). We distinguish three cases, given in Fig. 3, which yield the following possibilities of reshaping min-cuts:



(a) θ'_i separates v_j, r, and θ'_j separates v_i, r

(b) θ'_i does not separate v_j, r, but θ'_j separates v_i, r

(c) neither does θ'_i separate v_j, r, nor θ'_j v_i, r

Fig. 3. Three different cases concerning the positions of θ'_i and θ'_j (black, dashed), and their adjustments.

Case (a): As cut θ'_i separates v_j and r, and as v_j satisfies the partition-property, the cut $\theta_i := (R_i \setminus P_j, (V \setminus R_i) \cup P_j)$ (red dashed) has weight $c(\theta_i) \leq c(\theta'_i)$ and is thus a min-v_i-r-cut, which does not split $P_i \cup P_j$. For θ'_j an analogy holds.

Case (b): For θ'_j Case (a) applies. Furthermore, by Lem. 3 the cut $\theta_{\text{new}(j)} := (R_i \cap R_j, (V \setminus R_i) \cup (V \setminus R_j))$ (green dotted) is a min-v_j-r-cut, which does not split $P_i \cup P_j$. By Lem. 2 the previous split cut θ'_i turns out to be also a min-v_i-v_j-cut, as $\theta_{\text{new}(j)}$ separates v_i and r.

Case (c): As in case (b), by Lem. 3 the cut $\theta_{\text{new}(i)} := ((V \setminus R_j) \cup R_i, (V \setminus R_i) \cap R_j)$ (green dotted) is a min-v_i-r-cut, and the cut $\theta_{\text{new}(j)} := ((V \setminus R_i) \cup R_j, (V \setminus R_j) \cap R_i)$ (green dotted) is a min-v_j-r-cut. These cuts do not cross. So as v_i and v_j both satisfy the partition-property, cut $\theta_i := (((V \setminus R_j) \cup R_i) \setminus P_i, ((V \setminus R_i) \cap R_j) \cup P_i)$ and $\theta_j := (((V \setminus R_i) \cup R_j) \setminus P_j, ((V \setminus R_j) \cap R_i) \cup P_j)$ (both red dashed) are non-crossing min separating cuts, which neither split P_i nor P_j.

To summarize the cases discussed above, we make the following observation.

Observation 1 *During a* GH *starting from $T_\circ$ for $G_\ominus$, whenever we discover a new, cheaper min-v_i-$r(S)$-cut θ' ($v_i \sim r(S)$ in node S) we can iteratively reshape θ' into a min-v_i-$r(S)$-cut θ which neither cuts $\Uparrow_i$ nor any other treetop $\Uparrow_j$ ($v_i \sim r(S)$ in S), by means of Cases (a,b,c).*

3 Update Algorithms for Dynamic Clusterings

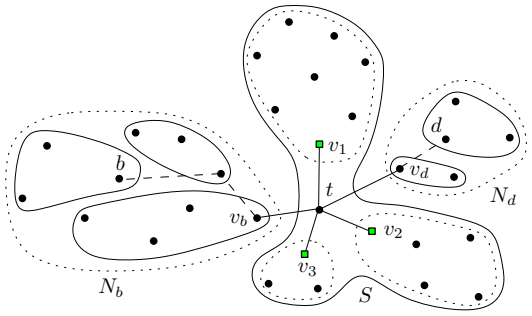


Fig. 4. $T_\circ(G^{\ominus}_{\alpha})$ for an inter-cluster deletion, t's neighbors off γ need inspection. The cuts of v_b and v_d are correct, but they might get shadowed.

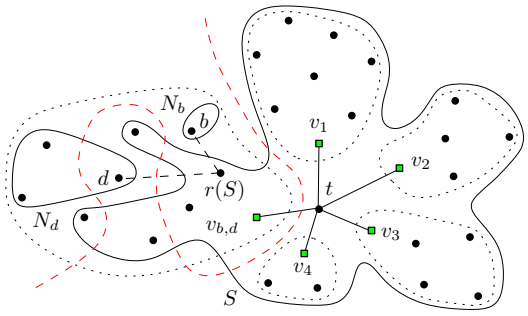


Fig. 5. $T_\circ(G^{\ominus}_{\alpha})$ for an intra-cluster deletion, edge $\{v_{b,d}, t\}$ defines a treetop (t's side). The dashed cut could be added to Θ by Alg. 5 (line 8).

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

It is important to see that it is not necessary to maintain a full min-cut tree to determine the induced clustering. By concept, we merely need to know all vertices of $T(G)$ adjacent to t; we call this set $W = \{v_1, \ldots, v_z\} \cup \{v_b, v_d\}$, with $\{v_b, v_d\}$ being the particular vertex/vertices on the path from t to b and d, respectively. We call the corresponding set of non-crossing min-v_i-t-cuts that isolate t, Θ. We will thus focus on dynamically maintaining only this information, and sketch out how to unfold the rest of the min-cut tree. From Lem. 4, 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 (Alg. 3). Just like its three counterparts, it takes as an input the old graph G and its sets $W(G)$ and $\Theta(G)$ (not the entire min-cut tree $T(G_\alpha)$), furthermore it takes the changed graph, augmented by t, $G_\alpha^\ominus$, the deleted edge $\{b, d\}$ and its weight Δ. Recall that an inter-cluster deletion yields t on γ, and thus, $T_\circ(G_\alpha)$ contains edges $\{v_b, t\}$ and $\{v_d, t\}$ cutting off the subtrees N_b and N_d of t by cuts θ_b, θ_d, as shown in Fig. 4. All clusters contained in node $S \ni t$ need to be changed or reconfirmed. To this end Algorithm 3 lists all cut vertices in S, $v_1, \ldots, v_z$, into $L(t)$, and initializes their shadows $D(v_i) = \emptyset$. The known cuts θ_b, θ_d are already added to the final list, as are v_b, v_d (line 5). Then the core algorithm, Check Cut-Vertices is called, which—roughly speaking—performs those GH-steps that are necessary to isolate t, of course, using (most of) the lemmas derived above.

Algorithm 3: Inter-Cluster Edge Deletion

Input: $W(G)$, $\Theta(G)$ $G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b, d\}\}, c_\alpha^\ominus)$, edge $\{b, d\}$ with weight Δ
Output: $W(G_\ominus)$, $\Theta(G_\ominus)$

1 $L(t) \leftarrow \emptyset$, $l(t) \leftarrow \emptyset$
2 **for** $i = 1, \ldots, z$ **do**
3 Add v_i to $L(t)` // old cut-vertices
4 $D(v_i) \leftarrow \emptyset$ // shadows
5 $\Theta(G_\ominus) \leftarrow \{\theta_b, \theta_d\}$, $W(G_\ominus) \leftarrow \{v_b, v_d\}$
6 **return** Check Cut-Vertices $(W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, \{b, d\}, D, L(t))$

Algorithm 4: Check Cut-Vertices

Input: $W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, \{b, d\}, D, L(t)$
Output: $W(G_\ominus), \Theta(G_\ominus)$

1 **while** $L(t)$ *has next element* v_i **do**
2 $\theta_i \leftarrow$ first min-v_i-t-cut given by FlowAlgo(v_i, t) // small side for v_i
3 **if** $c_\alpha^\ominus(\theta_i) = c_\alpha(\theta_i^{old})$ **then** // retain old cuts of the same weight
4 Add θ_i^{old} to $l(t)$ // pointed at by v_i
5 **else** // new cheaper cuts
6 Add θ_i to $l(t)$ // pointed at by v_i
7 **while** $L(t)$ *has next element* $v_j \neq v_i$ **do** // test vs. other new cuts
8 **if** θ_i *separates* v_j *and* t **then** // v_j shadowed by Lem. 3
9 Move v_j from $L(t)$ to $D(v_i)$
10 **if** $l(t) \ni \theta_j$, *pointed at by* v_j **then** Delete θ_j from $l(t)$
11 **while** $L(t)$ *has next element* v_i **do** // make new cuts cluster-preserving
12 set $(R, V_\alpha \setminus R) := \theta_i$ with $t \in R$ for $\theta_i \in l(t)$ pointed at by v_i // just nomenclature
13 $\theta_i \leftarrow (R \setminus C_i, (V_\alpha \setminus R) \cup C_i)$ // by partition-property (Lem. 9)
14 **forall** $v_j \in D(v_i)$ **do** // handle shadowed cuts ...
15 $\theta_i \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$ // ...with Cases (a) and (b)
16 **forall** $v_j \neq v_i$ *in* $L(t)$ **do** // handle other cuts ...
17 $\theta_i \leftarrow (R \cup C_j, (V_\alpha \setminus R) \setminus C_j)$ // ...with Case (c)
18 Add all vertices in $L(t)$ to $W(G_\ominus)$, and their cuts from $l(t)$ to $\Theta(G_\ominus)$
19 **return** $W(G_\ominus), \Theta(G_\ominus)$

First of all, note that if $|\mathcal{C}| = 2$ ($\mathcal{C} = \{N_b, N_d\}$ and $S = \{t\}$) then $L(t) = \emptyset$ and Alg. 3 lets Check Cut-Vertices (Alg. 4) simply return the input cuts and terminates. Otherwise, it iterates

the set of former cut-vertices $L(t)$ once, thereby possibly shortening it. We start by computing a new min-v_i-t-cut for v_i. We do this with a max-v_i-t-flow computation, which is known to yield *all* min-v_i-t-cuts [10], taking the *first* cut found by a breadth-first search from v_i (lines 2). This way we find a cut which minimally interferes with other treetops, thus encouraging temporal smoothness. If the new cut is non-cheaper, we use the old one instead, and add it to the tentative list of cuts $l(t)$ (lines 3-4). Otherwise we store the new, cheaper cut θ_i, and examine it for later adjustment. For any candidate v_j still in $L(t)$ that is separated from t by θ_i, Case (a) or (b) applies (line 8). Thus, v_j will be in the shadow of v_i, and not a cut-vertex (line 9). In case v_j has already been processed, its cut is removed from $l(t)$.

Once all cut-vertex candidates are processed, each one either induces the same cut as before, is new and shadows other former cut-vertices or is itself shadowed by another cut-vertex. Now that we have collected these relations, we actually apply Cases (a,b,c) and Lem. 9 in lines 11-17. Note that for retained, old cuts, no adjustment is actually performed here. Finally, all non-shadowed cut-vertices alongside their adjusted cuts are added to the final lists, and those returned.

Next we look at *intra*-cluster edge deletion. Looking at our starting point $T_\circ$, the safe path γ lies within some cluster $C_{b,d}$, which does not help much. In this case, t lies off γ, and thus there is an edge $\{v_{b,d}, t\}$, with $v_{b,d} \in C_{b,d}$, which defines a treetop containing all other former clusters and t, see Fig. 5. Algorithm 5 has the same in- and output as Algorithm 3, and starts by finding a new *first* min-t-$v_{b,d}$-cut. If this yields that no new, cheaper t-$v_{b,d}$-cut exists, then, by Lem. 6, we are done (line 2). Otherwise, we can at least adjust $\theta_{b,d}$ such that it does not interfere with any former cluster C_i by Lem. 9, as C_i is part of a treetop (lines 5-6); note that $C_{b,d}$ can not necessarily be preserved. Then we prepare the sets $L(t), l(t), \Theta(G_\ominus), W(G_\ominus)$ in lines 7-11. Check Cut-Vertices now performs the same tasks as for Inter-Cluster Edge Deletion: it separates all cut-vertex candidates from t in a non-intrusive manner; note that this excludes $v_{b,d}$ (line 9), as $C_{b,d}$ is no treetop, and thus defies the adjustments. After line 12 we have one

Algorithm 5: Intra-Cluster Edge Deletion

Input: $W(G), \Theta(G), G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c_\alpha^\ominus)$, edge $\{b,d\}$ with weight Δ
Output: $W(G_\ominus)$, $\Theta(G_\ominus)$ regarding $G_\ominus$

1 $\theta_{b,d} \leftarrow$ first min-t-$v_{b,d}$-cut given by FlowAlgo$(t, v_{b,d})$ // small side for 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
4 **else** // a new cut should retain treetops
5 | set $(R, V_\alpha \setminus R) := \theta_{b,d}$ with $t \in R$ // just nomenclature
6 | **forall** $C_i \neq C_{b,d}$ **do** $\theta_{b,d} := (R \cup C_i, (V_\alpha \setminus R) \setminus C_i)$ // by Lem. 9
7 | $L(t) \leftarrow \emptyset, l(t) \leftarrow \emptyset$
8 | $\Theta(G_\ominus) \leftarrow \{\theta_{b,d}\}, W(G_\ominus) \leftarrow \{v_{b,d}\}$
9 | **for** $i = 1, \ldots, z$ **do** // not including $v_{b,d}$
10 | | Add v_i to $L(t)$
11 | | $D(v_i) \leftarrow \emptyset$
12 | $W(G_\ominus), \Theta(G_\ominus) \leftarrow$ Check Cut-Vertices $(W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, \{b,d\}, D, L(t))$
13 | $W(G_\ominus) \leftarrow W(G_\ominus) \cup v_{b,d}$, $\Theta \leftarrow \Theta \cup \{\theta_{b,d}\}$
14 | Resolve all crossings in $\Theta(G_\ominus)$ by Lem. 3
15 | Isolate the sink t from all remaining unclustered vertices
16 | **return** $W(G_\ominus), \Theta(G_\ominus)$

min-$v_{b,d}$-t-cut that leaves its treetop untouched, but might cut $C_{b,d}$, and a new set $\Theta(G_\ominus)$ of non-crossing min-v_i-t-cuts (with some former $v_j \in W(G)$ possibly having become shadowed), which might, however, also cut through $C_{b,d}$. Putting all these cuts and cut-certices into $\Theta(G_\ominus)$ and $W(G_\ominus)$, we can now apply Lem. 3 (using t as "x"), to make all cuts non-crossing. Note that this can also result in shadowing $v_{b,d}$ as in Case (b) (dotted cut). Finally, some vertices from the former cluster $C_{b,d}$ might then still remain unclustered, i.e., not separated from t by any $\theta \in \Theta(G_\ominus)$. For clustering these vertices v we cannot do better than proceeding as usual: compute their set of min-v-t-cuts and render them non-crossing by Lem. 3, possibly shadowing one another or some previous cut θ. We refrain from detailing the latter steps.

Edge Addition. The good news for handling $G_\oplus$ is, that an algorithm Intra-Cluster Edge Addition need not do anything, but return the old clustering: By Lem. 1 and Theorem 1, in $T_\circ$, only path γ is contracted. But since γ lies within a cluster, the cuts in G_α, defining the old clustering, all remain valid in $G_\alpha^\oplus$, as depicted in Fig. 7 with dotted clusters and affected node S.

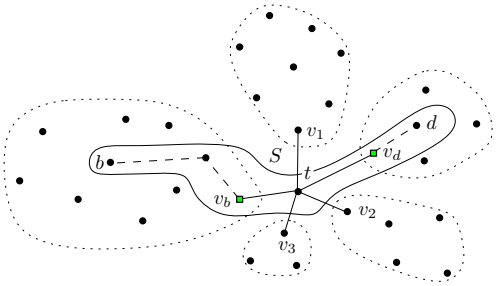


Fig. 6. $T_\circ(G_\alpha^\oplus)$ for an inter-cluster addition. At least v_b and v_d need inspection.

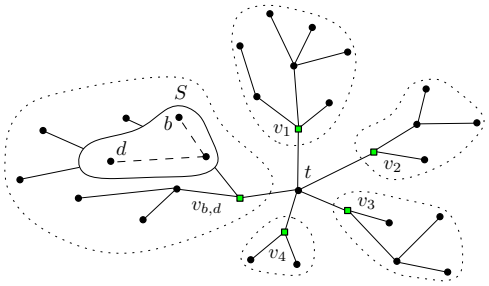


Fig. 7. $T_\circ(G_\alpha^\oplus)$ for an intra-cluster addition. All relevant min-v-t-cuts persist.

By contrast, adding an edge between clusters is more demanding. Again, γ is contracted, see region S in Fig. 6; however, t lies on γ in this case. A sketch of what needs to be done resembles the above algorithms: We compute new min-v_b-t- and min-v_d-t-cuts (or possibly only one, if it immediately shadows the other in line 12, in Alg. 6), and keep the old v_i-t-cuts. Then—proceeding as usual—we note which cuts shadow which others and reconnect nodes by Theorem 3. Similar to Alg. 5, the two new cuts may leave a "wild" set of vertices from the previous subtrees N_b, N_d, where crossings still have to be removed (via Lem. 3) in the end, and leftover vertices must be separated from t from scratch. We leave the pseudo-code to App. B.

Updating Entire Min-Cut Trees. An interesting topic on its own right and more fundamental than clustering, is the dynamic maintenance of min-cut trees. In fact the above clustering algorithms are surprisingly close to methods that update min-cut trees. Since all the results from Sec. 2 still apply, we only need to unfold whatever treetops or subtrees of t—which we gladly accept as super-nodes for the purpose of clustering—and take care to correctly reconnect subtrees. This includes, that merely examining the neighbors of t does not suffice, we must iterate through all nodes S_i of $T_\circ$. For the sake of brevity we must omit further details on such algorithms and refer the interested reader to [7].

4 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 $T_\circ$. Lemmas 6 and 9, using *first cuts* and Observation 1 nicely enforce temporal smoothness. However, in some cases we must cut back on this issue, e.g., when we examine which other cut-vertex candidates are shadowed by another one, as in line 8 of Alg. 4. Here it entails many more cut-computations and a combinatorially non-trivial problem to find an ordering of $L(t)$ to optimally preserve old clusters. Still we can state the following lemma:

Lemma 10. *Let $\mathcal{C}(G)$ fulfill the* invariant *for $G_\ominus$, i.e., let the old clustering be valid for $G_\ominus$. In the case of an inter-cluster deletion, Alg 3 returns $\mathcal{C}(G)$. For an intra-cluster deletion Alg. 5 returns a clustering $\mathcal{C}(G_\ominus) \supseteq \mathcal{C}(G) \setminus C_{b,d}$, i.e., only $C_{b,d}$ might become fragmented.*

The proof for both cases relies on the fact that any output clustering differing in cluster C_i requires at least one min-v_i-t-cut ($v_i \in C_i$) to separate b, d, invalidating $\mathcal{C}(G)$. Both proofs can be found in App. A. Considering the remaining cases, intra-cluster addition obviously retains a valid previous clustering; however, for inter-cluster addition no strong assertion can be made.

Running Times. We universally express running times of our algorithms in terms of the number of necessary max-flow computations, leaving open how these are done. A summary of tight bounds is given in Tab. 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 (*guaran. 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.

	worst case	old clustering still valid		
		lower bound	upper bound	guaran. smooth
Inter-Del	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	Yes
Intra-Del	$\lvert\mathcal{C}(G)\rvert + \lvert C_{b,d}\rvert - 1$	1	$\lvert\mathcal{C}(G)\rvert + \lvert C_{b,d}\rvert - 1$	No (1)
Inter-Add	$\lvert C_b\rvert + \lvert C_d\rvert$	1	$\lvert C_b\rvert + \lvert C_d\rvert$	No (2)
Intra-Add	0	0	0	Yes

Table 1. Bounds on the number of max-flow calculations.

For *Inter-Del* (Alg. 3) we require at most $|\mathcal{C}(G)| - 2$ cuts, separating t from *all* (no shadowing) neighbors, except v_b and v_d (comp. Fig. 4). Since this is exactly what happens in case the old clustering remains valid, the other bounds are equal and we know we will find the old clustering. Algorithm 5 (*Intra-Del*) needs to examine all clusters within t's treetop (being treetops themselves), 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 $v_{b,d}, t$ getting hidden. Obviously, we attain the lower bound if we cut away $v_{b,d}$ from t, directly preserving $C_{b,d}$ and the entire treetop of t. For *Inter-Add* (Alg. 6), 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., v_b might become shadowed by some other $v \in C_b \cup C_d$, which ultimately yields the upper bound. In case the previous clustering is valid, however, we might get away with simply cutting off v_b and v_d at once, alongside their former clusters. This means, there is no guarantee that we return the previous clustering; still, with two cuts (v_b-t and v_d-t), we are quite likely to do so. Row *Intra-Add* is obvious. Note that a computation from scratch (static algorithm) entails a tight upper bound of $|V|$ max-flow computations for all four cases, in the worst case.

Further Speed-Up. For the sake of brevity we omit a few ideas for effort-saving in the pseudocode. Apart from the minor Lemmas 5 and 7, one heuristic is to decreasingly order vertices in the list $L(t)$, e.g., in line 10 of Alg. 5 or in line 3 of Alg. 3; for their static algorithm Flake et al. [3] found that this effectively reduces the number of cuts necessary to compute before t is isolated.

Since individual min-u-v-cuts are constantly required, another dimension of effort-saving lies in dynamically maintaining max-u-v-flows. In fact there are techniques for doing this, two of which we briefly mention here, but leave to read up in [7] and references therein, for readers interested in a detailed description, since that is beyond the scope of this work. Given an initial max-u-v-flow and a graph modification, Kohli and Torr [9] present a method for dynamically maintaining max-u-v-flows that first adjusts the residual graph in a special way, such that the flow is still valid, and then use any augmenting-path flow algorithm on this residual graph. Another approach is to build up a topologically ordered DAG on vertex subsets of G, directed from u to v. The nodes of this DAG consist of the strongly connected components in the residual graph of a max-u-v-flow, as described by Picard and Queyranne [10]. This DAG can be used to manage all min-u-v-cuts, and can efficiently be updated. Actual effort-saving by these methods depends on the dynamics, in particular hidden step pairs and shadowing prevents strong assertions.

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 Universität Karlsruhe. 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. We process a queue of 12 560 elementary modifications, 9 000 of which are actual edge modifications, on the initial graph G shown in Figure 8 ($|V| = 310, |E| = 450$).

This queue represents about one week, starting on Saturday (21.10.06); a spam-attack lets the graph slightly grow/densify over the course. We delete zero-weight edges and isolated nodes. Following the recommendations of Flake et al. [3] we choose $\alpha = 0.15$ for the initial graph, yielding 45 clusters, see Fig. 8 for an illustration. We compare their static algorithm (see Sec. 2.1) and our dynamic algorithm in terms of the number of max-flow computations necessary to maintain the clustering. For the 9 000 proper steps, static computation needed 2 080 897 max-flows, and our dynamic update needed 198 790, saving more than 90% max-flows, such that in 96% of all modifications, the dynamic algorithm was quicker. Surprisingly, inter-cluster additions have the greatest impact on effort-saving, followed by the trivial intra-cluster additions. By contrast, both deletion operations only mildly outperform the static algorithm. Out of the 9 000 total operations, 49 of the inter-cluster, and 222 of the intra-cluster deletions are the only ones, where the static algorithm happens to be quicker. See App. C for details on these results.

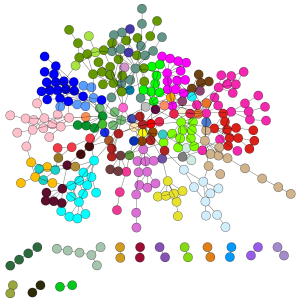


Fig. 8. Initial real world e-mail graph, color-clustered.

5 Conclusion

We have proven a number of results on the nature of min-u-v-cuts in changing graphs, which allow for feasible update algorithms of a minimum-cut tree. In particular we have presented algorithms which efficiently update specific parts of such a tree and thus fully dynamically maintain a graph clustering based on minimum-cut trees, as defined by Flake et al. [3] 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., changes to the clusterings are kept at a minimum, whenever possible. Furthermore, we disprove an earlier attempt to dynamize such clusterings [12, 11]. Our experiments on real-world dynamic graphs affirm our theoretical results and show a significant practical speedup over the static algorithm of Flake et al. [3]. Future work on dynamic minimum-cut tree clusterings will include a systematic comparison to other dynamic clustering techniques and a method to dynamically adapt the parameter α.

References

1. 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.
2. U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer, February 2005.
3. G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004.
4. 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.
5. 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.
6. D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.
7. T. Hartmann. Clustering Dynamic Graphs with Guaranteed Quality. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, October 2008.
8. 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.
9. P. Kohli and P. H. Torr. Dynamic Graph Cuts for Efficient Inference in Markov Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2079–2088, December 2007.

10. J.-C. Picard and M. Queyranne. On the Structure of All Minimum Cuts in a Network and Applications. *Mathematical Programming, Series A*, 22(1):121, December 1982.
11. 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.
12. 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.

Appendix

A Omitted Proofs

Proof (of Theorem 1). The proof uses induction on the $n-1$ edges in $f' \cdot f$. The edges are regarded as step pairs in a Gomory-Hu execution GH. The set $M \subseteq E_T$ denotes the step pairs already applied in the execution, and $T_*(G) = (V_*, E_*, c_*)$ denotes the current working version of the intermediate min-cut tree.

Induction base case: GOMORY-HU starts with a single node S containing V, such that $V_* = \{V\}$ and $E_* = \emptyset$. The contracted graph G_S thus equals G as nothing is contracted yet with $M = \emptyset$. Therefore, T_* corresponds to a $T_\circ$ that is formed by contracting $E_T \setminus M = E_T$ in $T(G)$.

Now take the first pair $\{u, v\}_1$ of $f' \cdot f$ as a step pair for the algorithm. Since the current split node is $S = \{V\}$, $\{u, v\}_1$ is a valid step pair in S. At the same time $\{u, v\}_1$ represents an edge in $T(G)$ and therefore induces a min-u-v-cut $(U, V \setminus U)$ in $G = G_S$ as a valid split cut, with $u \in U$. By splitting and replacing $S = V$ by $S_u = U$ and $S_v = V \setminus U$ connected with a new edge, we get an intermediate min-cut tree T_* with $V_* = \{S_u, S_v\} = \{U, (V \setminus U)\}$ and $E_* = \{\{S_u, S_v\}\}$. The only edge in T_* created by the step pair $\{u, v\}_1$, has weight $c_T(\{u, v\}_1) = c(U, V \setminus U)$. So after one iteration the intermediate min-cut tree T_* exactly corresponds to $T_\circ$ formed by contracting all edges of $E_T \setminus M$ in $T(G)$, with $M = \{\{u, v\}_1\}$, fulfilling our claim. Note, that the step pair $\{u, v\}_1$ is not hidden until now.

Induction hypothesis: We now assume the following: The first w pairs $\{u, v\}_1, \ldots, \{u, v\}_w$ in $f' \cdot f$ are valid step pairs regarding the various split nodes S, and the related edge-induced cuts in G are valid split cuts regarding the various contracted graphs G_S. The current intermediate min-cut tree $T_*(G)$ after these w iterations exactly corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$, with $M' = \{u, v\}_1, \ldots, \{u, v\}_w$ being the set of the first w step pairs in $f' \cdot f$. Furthermore we assume that none of the step pairs in M' is hidden yet.

Induction step: Let nodes u, v constitute the next step pair $\{u, v\}_{w+1}$ in $f' \cdot f$ with split node S. The related cut $(U, V \setminus U)$ in G induced by the edge $\{u, v\}_{w+1}$ in T_*, with $u \in U$, serves as the current split cut. We first need to show, that this cut is also a min-u-v-cut in the current contracted graph G_S. Let $N(j)$ denote the set of vertices in a subtree N_j of the current split node S. Then the current contracted graph G_S results from G by contracting the set $N(j)$ in G for all subtrees of S. The cut $(U, V \setminus U)$ induced by the edge $\{u, v\}_{w+1}$ is a min-u-v-cut in G. Moreover, it does not separate any two vertices g and h lying in the same set Nj, as otherwise the edge $\{u, v\}_{w+1}$ would lie on the unique path $\gamma_{g,h}$ from g to h in $T(G)$, contradicting the assumption that g and h belong to the same subtree of S.Thus the cut $(U, V \setminus U)$ is also a min-u-v-cut in the contracted graph G_S and hence is a valid split cut for the $(w+1)$-th iteration.

Now we can prove that after splitting and replacing the current split node S and after reconnecting the subtrees of S the resulting intermediate min-cut tree $T_*(G)$, i.e., the intermediate min-cut tree after $w+1$ iterations, again corresponds to $T_\circ(G)$ formed by contracting $T(G)$ by the edges $E_T \setminus M'$, with $M' = \{\{u, v\}_1, \ldots, \{u, v\}_{w+1}\}$ being the set of the first $w+1$ step pairs in $f' \cdot f$. To this end we show that none of the step pairs $\{\{u, v\}_1, \ldots, \{u, v\}_w\}$, which created the edges of the previous intermediate min-cut tree, gets hidden by the splitting of S. However, since these step pairs directly correspond to edges in $T(G)$ it immediately follows that they never get separated. As the new edge $\{S_u, S_v\}$ in $T_*(G)$ is created by the step pair $\{u, v\}_{w+1}$, which represents an edge in $T(G)$, and as all other step pairs in $M' = \{\{u, v\}_1, \ldots, \{u, v\}_w\}$ also represent edges in $T(G)$ as well as in $T_*(G)$ (by the induction hypothesis), none of the step pairs $\{\{u, v\}_1, \ldots, \{u, v\}_w\}$ gets separated by the split cut related to $\{u, v\}_{w+1}$. Therefore, after $w+1$ iterations, the new intermediate min-cut tree $T_*(G)$ exactly corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$ in $T(G)$, with $M' = \{\{u, v\}_1, \ldots, \{u, v\}_{w+1}\}$ being the set of the first $w+1$ step pairs in $f' \cdot f$.

Proof (of Theorem 3). This proof uses induction on the subtrees of a split node S in an intermediate min-cut tree $T_*(G)$ and shows constructively that there always exists a split cut $(U_S, V_S \setminus U_S)$ in G_S as described in Theorem 2, which is by the way also a min-u-v-cut in G and does not split any subtree of S. Furthermore, the proof shows that the two sides of this split cut pick the subtrees as described. For each subtree N_j of S the connecting edge $e_j = \{S, S_j\}$ induces the min-y_j-x_j-cut $\theta_j := (N(j), V \setminus N_j)$ in G, with $y_j \in N(j)$. As it holds that $S \subset V \setminus N(j)$, for each subtree N_j the step pair $\{u, v\}$ lies on the $V \setminus N(j)$-side of the minimum y_j-x_j-cut θ_j induced by the connection edge e_j (see Figure 9). Now let $(U, V \setminus U)$ denote an arbitrary minimum u-v-cut in G, with $u \in U$.

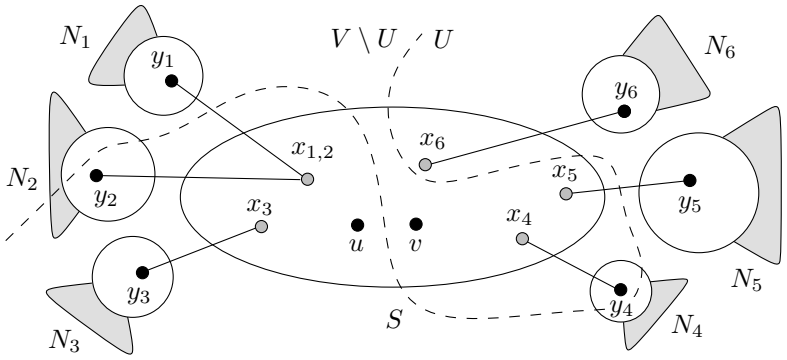


Fig. 9. Intermediate min-cut tree $T_\star(G)$ with subtrees $N_1, \ldots, N_6$ and nearest cut pairs $\{x_2, y_1\}, \ldots, \{x_6, y_6\}$.

Induction base case: We apply Lemma 3 to θ_1 and $(U, V \setminus U)$ and get a minimum u-v-cut $(U_1, V \setminus U_1)$, with $u \in U_1$, that does not separate any vertices in $N(1)$ and splits $V \setminus N(1)$ the same way as $(U, V \setminus U)$ does. So, as it holds that $S \subseteq V \setminus N(1)$, also S gets split the same way, and we get

$$S \cap U_1 = S \cap U$$
$$\text{and} \quad S \cap V \setminus U_1 = S \cap V \setminus U.$$

With $y_1 \in N(1)$, by Lemma 3, we further get

$$\begin{aligned} N(1) \cup U &= U_1 && \text{if } y_1 \in U \text{ and} \\ N(1) \cup (V \setminus U) &= V \setminus U_1 && \text{otherwise, i.e., if } y_1 \in V \setminus U, \end{aligned}$$

and therefore, it holds that $N(1) \subseteq U_1$ if and only if $y_1 \in U$. Thus this induces that the related sides of $(U_1, V \setminus U_1)$ and $(U, V \setminus U)$ only differ in $N(1)$, i.e., $U_1 \setminus N(1) = U \setminus N(1)$ and $(V \setminus U_1) \setminus N(1) = (V \setminus U) \setminus N(1)$.

Induction hypothesis: We now assume the cut $(U_z, V \setminus U_z)$ to be a minimum u-v-cut in G, with $u \in U_z$, that does not separate any vertices in any subtree N_j, $j = 1, \ldots, z$, and splits V the same way as $(U, V \setminus U)$ does. More precisely, we assume that it holds that

$$S \cap U_z = S \cap U$$
$$\text{and} \quad S \cap V \setminus U_z = S \cap V \setminus U.$$

and that $N(j) \subseteq U_z$ if and only if $y_j \in U$ for $j = 1, \ldots, z$, while the related sides of $(U_z, V \setminus U_z)$ and $(U, V \setminus U)$ only differ in the sets $N(j)$, $j = 1, \ldots, z$. More formally, this is,

$$U_z \setminus \{N(j) | j = 1, \ldots, z\} = U \setminus \{N(j) | j = 1, \ldots, z\} \text{ and}$$
$$(V \setminus U_1) \setminus \{N(j) | j = 1, \ldots, z\} = (V \setminus U) \setminus \{N(j) | j = 1, \ldots, z\}.$$

Induction step: We apply Lemma 3 to cut $\theta_{z+1} = (N(z+1), V \setminus N(z+1))$, which is induces by the connection edge $e_{z+1} = \{S, S_{z+1}\}$ of subtree N_{z+1}, and cut $(U_z, V \setminus U_z)$. So we get a minimum u-v-cut $(U_{z+1}, V \setminus U_{z+1})$, with $u \in U_{z+1}$, that does not separate any vertices in $N(z+1)$ and splits $V \setminus N(z+1)$ the same way as $(U_z, V \setminus U_z)$ does. So, as it holds that $S \subseteq V \setminus N(z+1)$, also S gets split the same way, and we get

$$\begin{aligned} S \cap U_{z+1} = S \cap U_z &\underset{\text{induction hypothesis}}{=} S \cap U \\ \text{and} \quad S \cap V \setminus U_{z+1} = S \cap V \setminus U_z &\underset{\text{induction hypothesis}}{=} S \cap V \setminus U. \end{aligned} \tag{3}$$

With $y_{z+1} \in N(z+1)$, by Lemma 3, we further get

$$\begin{aligned} N(z+1) \cup U_z = U_{z+1} \qquad & \text{if } y_{z+1} \in U_z \text{ and} \\ N(z+1) \cup (V \setminus U_z) = V \setminus U_{z+1} \qquad & \text{otherwise, i.e., if } y_{z+1} \in V \setminus U_z, \end{aligned}$$

and therefore, it holds that $N(z+1) \subseteq U_{z+1}$ if and only if $y_{z+1} \in U_z$. As, by induction hypothesis, the related sides of $(U_z, V \setminus U_z)$ and $(U, V \setminus U)$ *do not* differ in $N(z+1)$, if follows that $y_{z+1} \in U_z$ if and only if $y_{z+1} \in U$, and therefore, it holds that

$$N(z+1) \subseteq U_{z+1} \text{ if and only if } y_{z+1} \in U. \tag{4}$$

Furthermore, as a consequence of Lemma 3 it holds that the related sides of $(U_{z+1}, V \setminus U_{z+1})$ and $(U_z, V \setminus U_z)$ only differ in $N(z+1)$, i.e., $U_{z+1} \setminus N(z+1) = U \setminus N(z+1)$ and $(V \setminus U_{z+1}) \setminus N(z+1) = (V \setminus U_z) \setminus N(z+1)$. So for all sets $N(j)$, $j = 1, \ldots, z$, it follows that $N(j) \subseteq U_{z+1}$ if and only if $N(j) \subseteq U_z$. By induction hypothesis and (4) we finally get for $j = 1, \ldots, z+1$

$$N(j) \subseteq U_{z+1} \text{ if and only if } y_j \in U. \tag{5}$$

So with Assertion (3) and Assertion (5) we finally proved the existence of a minimum u-v-cut in G that splits S the same way as $(U, V \setminus U)$ does, and that does not separate any vertices of any subtree of S. It is easy o see that such a minimum u-v-cut is also a minimum u-v-cut in graph $G(S)$, which results from G by contracting all subtrees of S. So Theorem 2 and Theorem 3 are both proven true.

Proof (of Lemma 4). The Gomory-Hu execution $\mathrm{GH}_{\oplus(\ominus)}$, by definition, uses the same sequence k of split cuts as execution GH does, which considers the graph G and reaches $T_\circ(G)$ as intermediate min-cut tree after the application of k. Therefore, execution $\mathrm{GH}_{\oplus(\ominus)}$ also has $T_\circ(G)$ as intermediate min-cut tree on condition that k represents a feasible sequence of split cuts concerning the modified graph $G_{\oplus(\ominus)}$. This then implies f to be a feasible sequence of step pairs. Similar to the proof of Theorem 1, this proof uses induction on the split cuts in k.

Induction base case: The execution $\mathrm{GH}_{\oplus(\ominus)}$ starts with the first split cut induced by the first edge $\{u, v\}_1$ in f. As the first split cut is applied to the contracted graph $G_S^{\oplus(\ominus)} = G_{\oplus(\ominus)}$, and $\{u, v\}_1 \in M$ induces a minimum u-v-cut in $G_{\oplus(\ominus)}$ (by the choice of M and Corollary 1), the first split cut is feasible.

Induction hypothesis: We now assume the split cuts induced by the edges $\{u, v\}_2, \ldots, \{u, v\}_z$ in f to be feasible regarding the various contracted graphs $G_S^{\oplus(\ominus)}$ in $z-1$ further iterations.

Induction step: Consider the next split cut induced by the edge $\{u, v\}_{z+1}$ in f, which constitutes the step pair in the current split node S. For the following argumentation we need to distinguish the cases of edge addition and edge deletion.

Edge addition ($M = E_T \setminus \gamma$): If it holds for the modified vertices b and d that $\{b, d\} \not\subseteq S$, it follows that $G_S^{\oplus(\ominus)} = G_S$ in this iteration, as the modified edge $\{b, d\}$ then is contracted (Note that b and d never lie in different subtrees of S, as the edges on γ, which correspond to the cuts that separate b and d, are not included in M, and f respectively). With $G_S^{\oplus(\ominus)} = G_S$ the current split cut is feasible.

If it holds that $\{b,d\} \subseteq S$, the contracted graph $G_S^{\oplus(\ominus)}$ results from G_S by the addition of the edge $e_\oplus = \{b,d\}$ and, as the edge $\{u,v\}_{z+1}$ cannot lie on the path γ, the current split cut does not separate b and d. So, as the current split cut is a minimum u-v-cut in G_S, by Lemma 1 the current split cut also represents a minimum u-v-cut in $G_S^\oplus$ and hence is feasible.
Edge deletion ($M = \gamma$): As all split cuts considered so far separate the modified vertices b and d, the current intermediate min-cut tree is a path of nodes with b included in the first and d included in the last node. So if the current split node S includes b (the case when it includes d is symmetric), then S has only one subtree, which includes d. If S includes neither b nor d, then S has exactly two subtrees, with b and d in different subtrees. In both cases the graph $G_S^\ominus$ results from G_S by the deletion of the edge $e_\ominus = \{b,d\}$. Furthermore, the current split cut must separate b and d, as the edge $\{u,v\}_{z+1}$ lies on path γ. So, as the current split cut is a minimum u-v-cut in G_S, by Lemma 1 the current split cut also represents a minimum u-v-cut in $G_S^\ominus$ and hence is feasible.

As the remaining step pairs and split cuts in $f_{\oplus(\ominus)}$ and $k_{\oplus(\ominus)}$ are defined as arbitrary valid sequences, and as such sequences always exist, the assertion of the lemma is proven.

Proof (of Lemma 5). Let θ be the cut induced by $e_{\min}$; then in $G_\oplus$ it has weight $c_\oplus(\theta) = c(\theta) + \Delta$. Suppose now θ' is b-d-cut with $c_\oplus(\theta') < c_\oplus(\theta)$. Since θ' must cut edge $\{b,d\}$ in $G_\oplus$, its weight in G is $c(\theta') \leq c_\oplus(\theta') - \Delta$. This yields $c(\theta') < c(\theta)$, a contradiction to $e_{\min}$'s minimality for G.

Proof (of Lemma 6). Consider the min-u-v-cut $(U, V \setminus U)$ in $G_\ominus$ to be the first split cut of GH, with step pair $\{u,v\}$. As the cut does not separate $\{b,d\}$, wlog. let $b,d \in V \setminus U$. Let $\{U\}$ be the next split node of GH, such that b and d are contracted into $\eta_{V\setminus U}$ in $G_U^\ominus$. Since for any step pair within U, $\{b,d\}$ are not separated, by the correctness of Gomory-Hu and Lemma 1, any previous min-g-h-cut is still valid in $G_\ominus$. Furthermore, Lemma 2 asserts that previous cut pairs within U also stay valid.

Proof (of Lemma 7). By Lemma 1 $\{y_b, g\}, \{y_d, g\}$ stay valid min-cuts in $G_\ominus$. A GH starting with step pairs $\{y_b, g\}, \{y_d, g\}$ yields a path of nodes N_b, S_g, N_d as an intermediate cut tree, with $u, v \in S_g$. Suppose there is a cheaper u-v-cut θ' than that of $\{u,v\}$, then by Lemma 1 θ' must separate b and d and thus cut $\{y_b, g\}$ or $\{y_d, g\}$. But then θ' is cheaper than $c(\{y_b, g\}) - \Delta$ (and than $c(\{y_d, g\}) - \Delta$) and either violates that $(N_b, V \setminus N_b)$ remains a min-y_b-g-cut or that $(N_d, V \setminus N_d)$ remains a min-y_d-g-cut; a contradiction.

Proof (of Lemma 8). We prove this lemma regarding the subtree N_b by contradiction. The proof regarding the subtree N_d is symmetric. We show that the cut $\theta := (\Uparrow_A, N(b) \cup N(d) \cup \sharp \cup \Uparrow_B)$, which differs from θ'_b in the set $N(b)$, would be cheaper in G than the edge-induced minimum u-v-cut $\theta_{\min} := (\Uparrow, N(b) \cup N(d) \cup \sharp)$ in G, which differs from θ_b in the set $N(b)$, if θ'_b was cheaper than θ_b.

So we assume that $c_\ominus(\theta_b) > c_\ominus(\theta'_b)$. As the cuts θ and $\theta_{\min}$ both do not separate the modified vertices b and d, each of them is of the same weight in $G_\ominus(S)$, $G_\ominus$ and G, by Lemma 1. Here we consider the weights in $G_\ominus$ and get

$$\begin{aligned} c_\ominus(\theta_{\min}) &= c_\ominus(\theta_b) - c_\ominus(N(b), N(d) \cup \sharp) + c_\ominus(N(b), \Uparrow) \quad \text{and} \\ c_\ominus(\theta) &= c_\ominus(\theta'_b) - c_\ominus(N(b), N(d) \cup \sharp \cup \Uparrow_B) + c_\ominus(N(b), \Uparrow_A) \end{aligned}$$

With $(N(d) \cup \sharp) \subseteq (N(d) \cup \sharp \cup \Uparrow_B)$ and $\Uparrow_A \subseteq \Uparrow$ it holds that

$$\begin{aligned} c_\ominus(N(b), N(d) \cup \sharp) &\leq c_\ominus(N(b), N(d) \cup \sharp \cup \Uparrow_B) \quad \text{and} \\ c_\ominus(N(b), \Uparrow) &\geq c_\ominus(N(b), \Uparrow_A) \end{aligned}$$

So with the assumption that $c_\ominus(\theta_b) > c_\ominus(\theta'_b)$ we finally get

$$\begin{aligned} c_\ominus(\theta_{\min}) - c_\ominus(\theta) = {} & [c_\ominus(\theta_b) - c_\ominus(\theta'_b)] \\ & - [c_\ominus(N(b), N(d) \cup \sharp) - c_\ominus(N(b), N(d) \cup \sharp \cup \Uparrow_B)] \\ & + [c_\ominus(N(b), \Uparrow) - c_\ominus(N(b), \Uparrow_A)] > 0 \end{aligned}$$

This contradicts the fact that the edge-induced u-v-cut $\theta_{\min}$ is a minimum u-v-cut in graph G.

Proof (of Lemma 9). Again we prove this lemma regarding the subtree N_b. The proof regarding the subtree N_d is symmetric. The assertion of this lemma follows by Lemma 8. We express the cuts θ_{bb} and θ'_{bb} with the aid of the cuts θ_b and θ'_b considered in Lemma 8, which just differ in the set $\sharp_A$. So we get

$$c_\ominus(\theta_{bb}) = c_\ominus(\theta_b) - c_\ominus(\sharp_A, N(b) \cup \Uparrow) + c_\ominus(\sharp_A, N(d) \cup \sharp_B) \quad \text{and}$$
$$c_\ominus(\theta'_{bb}) = c_\ominus(\theta'_b) - c_\ominus(\sharp_A, N(b) \cup \Uparrow_A) + c_\ominus(\sharp_A, N(d) \cup \sharp_B \cup \Uparrow_B)$$

So with $c_\ominus(\theta_b) \leq c_\ominus(\theta'_b)$, by Lemma 8, we finally get

$$\begin{aligned} c_\ominus(\theta'_{bb}) - c_\ominus(\theta_{bb}) = {} & [c_\ominus(\theta'_b) - c_\ominus(\theta_b)] \\ & - [c_\ominus(\sharp_A, N(b) \cup \Uparrow_A) - c_\ominus(\sharp_A, N(b) \cup \Uparrow)] \\ & + [c_\ominus(\sharp_A, N(d) \cup \sharp_B \cup \Uparrow_B) - c_\ominus(\sharp_A, N(d) \cup \sharp_B)] \geq 0 \end{aligned}$$

Proof (of Lemma 10). Consider inter-cluster deletion (Alg. 3) first. To return a new clustering $\mathcal{C}(G_\ominus)$ different from $\mathcal{C}(G)$ the algorithm needs to find a new cheaper min-v_i-t-cut for at least one cut-vertex $v_i \in \{v_1, \ldots, v_z\}$. As the previous clustering is supposed to be 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 min-u-t-cut in the modified graph $G^\ominus_\alpha$. Then there must exists a min-cut tree $T(G^\ominus_\alpha)$ such that the edge-induced minimum v_i-t-cut represented in this new min-cut tree gets shadowed and must not separate the modified vertices b, d. This contradicts Lem. 1, which says that each new minimum v_i-t-cut in $G^\ominus_\alpha$ which is cheaper than the previous one in graph G_α needs to separate the modified vertices b, d.

Considering intra-cluster deletion (Alg. 5), all the above arguments apply to the clusters $\mathcal{C}(G) \setminus \{C_{b,d}\}$. Thus these clusters are again found; however $C_{b,d}$ might be fragmented in an almost arbitrary manner.

B Omitted Algorithms

Algorithm 6 gives the pseudo-code for the handling edge additions between clusters. Since its description is almost analogous to the above algorithms, the only detail we point out is the following. In line 5 the so called *best* min-v_b-t-cut (or min-v_d-t-cut) is used. Consider the situation sketched out in Fig. 6, and let us choose among all possible min-v_b-t-cuts $U, V \setminus U$, $v_b \in U$ (given by some max-flow). To maximize both the progress in terms of clustering and temporal smoothness, we chooses a cut that puts as many vertices of the former cluster C_b as possible into U while cutting away from t as few other cut-vertices as possible.

Algorithm 6: Inter-Cluster Edge Addition

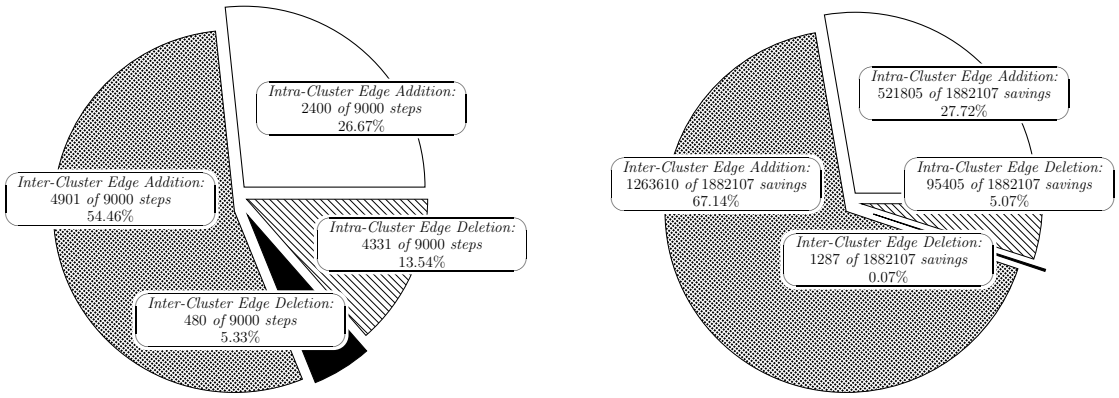
Input: $W(G), \Theta(G), G_\alpha^\oplus = (V, E \cup \{\{b,d\}\}), c_\alpha^\oplus$, edge $\{b,d\}$ with weight Δ
Output: $W(G_\ominus), \Theta(G_\ominus)$

1 $L(t) \leftarrow \{v_b, v_d\}$, $l(t) \leftarrow \emptyset$
2 $D(v_b) \leftarrow \emptyset$, $D(v_d) \leftarrow \emptyset$
3 $W(G_\oplus) \leftarrow \{v_1, \ldots, v_z\}$, $\Theta(G_\oplus) \leftarrow \{\theta_1, \ldots, \theta_z\}$
4 **while** $L(t)$ *has next element* u_i **do**
5 $\theta \leftarrow$ "best cut" given by FlowAlgo(u_i, t) // see text
6 **if** $c_\alpha^\oplus(\theta_i) = c_\alpha(\theta_i^{old}) + \Delta$ **then**
7 Move u_i from $L(t)$ to $W(G_\oplus)$
8 Add θ_i^{old} to $\Theta(G_\oplus)$
9 **else**
10 Add θ_i to $l(t)$ // pointed at by u_i
11 **while** $L(t)$ *has next element* $u_j \neq u_i$ **do**
12 **if** θ_i *separates* u_j *and* t **then**
13 Delete u_j from $L(t)$
14 **if** $l(t)$ *already contains a cut* θ_j *pointed at by* u_j **then**
15 Delete θ_j from $l(t)$
16 **while** $W(G_\oplus)$ *has next element* v_i **do**
17 **if** θ_i *separates* v_i *and* t **then**
18 Delete cut which v_i points to from $\Theta(G_\oplus)$
19 Move v_i from $W(G_\oplus)$ to $D(u_i)$
20 **while** $L(t)$ *has next element* u_i **do**
21 $(R, V_\alpha \setminus R) := \theta_i$, $t \in R$, (cut in $l(t)$ which u_i points at)
22 **forall** *vertices* v_j *in* $D(u_i)$ **do**
23 $\theta_i \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$ // by Theorem 3
24 **forall** *vertices* v_j *in* $W(G_\oplus)$ **do**
25 $\theta \leftarrow (R \cup C_j, (V_\alpha \setminus R) \setminus C_j)$ // by Theorem 3
26 Resolve all crossings in $l(t)$ // by Lem. 3
27 Add all vertices in $L(t)$ to $W(G_\oplus)$
28 Add all (non-crossing) cuts in $l(t)$ to $\Theta(G_\oplus)$
29 Isolate t
30 **return** $W(G_\oplus), \Theta(G_\oplus)$

C Experimental Evaluation

Note that updating the clustering after increasing the weight of an edge is done by one of the new algorithms regarding edge additions. The addition of an edge is considered a special case of increasing the weight of an edge. Weight decreases are handled analogously. Thus, in the following we simply talk about intra-cluster and inter-cluster edge additions and edge deletions as the four elementary modifications.

Figure 10(a) shows the proportions of the elementary modifications regarding the total number of 9 000 modifying steps. The case occurring most often is, with 54.46%, the addition of an edge between two different clusters. The inter-cluster edge deletion, by contrast, only occurs 480 times which corresponds to 5.33%. During the whole experiment the cut-clustering heuristic of Flake et al. [3] calculates 2 080 897 maximum flows. Our updating algorithms, however, only need 198 790 max-flow calculations. This yields a saving of 1 882 107 max-flow calculations which constitutes 90.45% of effort saving. Figure 10(b) shows the proportions of the elementary modifications regarding the total number of 1 882 107 savings. We see that the ratio of the percentaged savings



(a) Total number of steps decomposed into elementary modifications

(b) Total savings of max-flow calculations decomposed into elementary modifications

Fig. 10. Total number of steps and savings of max-flow calculations.

provided by edge additions to the proportion of the edge additions regarding the number of total steps is greater than one, while the proportion of edge deletions in Figure10(a) provides a smaller proportion of the total savings in Figure 10(b). More precisely, the inter-cluster edge additions are the most efficient modifications, as 54.46% of the total number of steps provide 67.14% of the saved max-flow computations. So each unit of the inter-cluster edge addition proportion on average causes 1.23% of all savings. The least efficient modifications are the inter-cluster edge deletions with 5.33% of all steps gaining only 0.07% of all savings. This corresponds to 0.01% of all savings on average per unit of the inter-cluster deletion proportion.

D Problems in [12]

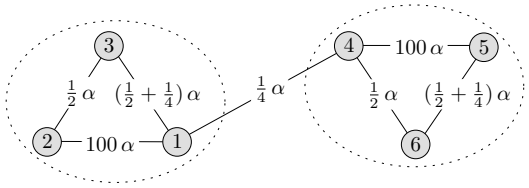
This section gives a brief overview of the errors we found in the work of B. Saha and P. Mitra [12]. A preliminary version of this work is [11]. The authors describe four procedures for updating a clustering and a data structure for the deletion and the addition of intracluster and intercluster edges. We briefly point out the errors in the authors' procedure that deals with the addition of intracluster edges. For a thorough discussion we refer the reader to Hartmann [7]. Algorithm 7

Algorithm 7: Old Inter-Edge Addition

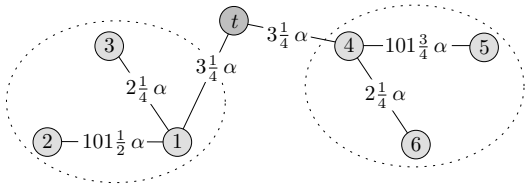
Input: $G = (V, E, w)$, α, $\mathcal{C}$, new edge $e_\oplus = \{b, d\}$, $b \in C_b$, $d \in C_d$
if *intercluster quality of* C_d *and* C_b *is maintained* **then** Case 1:
  return $\mathcal{C}$ (do nothing)
else if $\frac{2c(C_b, C_d)}{|V|} \geq \alpha$ **then** Case 2:
  return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{\{C_b \cup C_d\}\}$ (merge C_b and C_d)
Case 3 (default): dissolve C_b and C_d and contract all other nodes
perform adapted Cut-Clustering on this instance
return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{$newly formed clusters of nodes from C_b and $C_d\}$

sketches the approach given in [12] for handling edge additions 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.

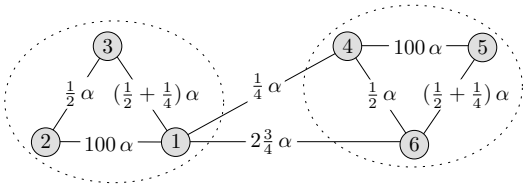
A Counter-Example for Case 1 and Case 2 We now give an example instance which the algorithm given in [12] fails to cluster correctly. The two upper figures (Fig. 11(a),11(b)) show the input instance, as computed by algorithm Cut-Clustering. In Fig. 11(c), a first edge addition



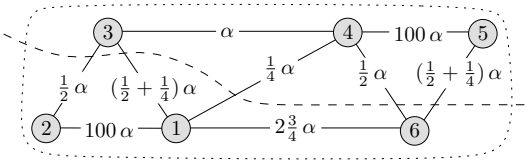
(a) Graph G^0 with clustering $\mathcal{C}(G^0)$ by the Cut-Clustering method



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



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

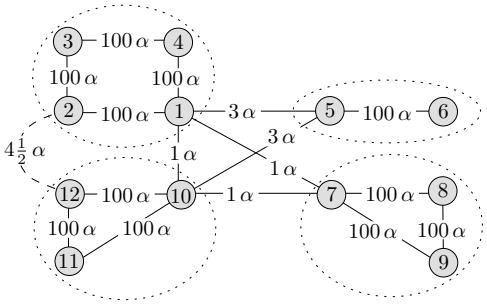


(d) Adding edge $\{3, 4\}, \alpha$ yields G^2 with the trivial clustering $\mathcal{C}(G^2)$ by Case 2, which violates Eq. (2) (dashed cut)

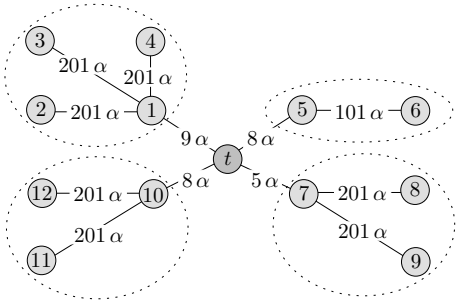
Fig. 11. A dynamic instance violating the clustering quality. Weights are parameterized by α. After two modifications to G^0 the algorithm returns one cluster which can be cut (dashed) with a cut value that violates quality.

then triggers Case 1, and thus the clustering is kept unchanged. Note that here, quality is still maintained. Then in Fig. 11(d) a second edge is added and handled by Case 2, since intercluster quality is violated ($c(C_1, C_2) = 4\alpha > 3 = \alpha \cdot \min\{|C_1|, |C_2|\}$), 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 Fig 11(d) shows an intracluster cut with value $c(\text{dashed}) = 2.75 \cdot \alpha < 3 \cdot \alpha$, which violates intracluster quality, as claimed in Eq. (2).

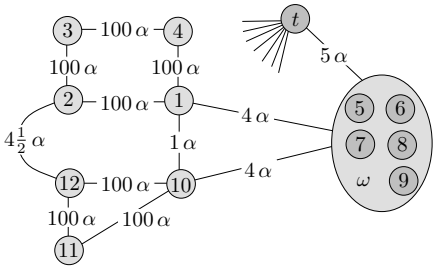
A Counter-Example for Case 3 Finally we give an example instance which the algorithm given in [12] fails to cluster correctly due to shortcomings in Case 3.



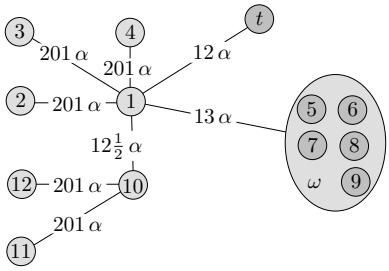
(a) Graph G with clustering $\mathcal{C}(G)$ resulting from the cut-clustering method



(b) Min-cut tree $T(G_\alpha)$ inducing the clustering $\mathcal{C}(G)$ shown above



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



(d) Min-cut tree $T(G'_\alpha)$ of graph G'_α

Fig. 12. Counter-example for the correctness of Case 3. Figures (a) and (b) describe the graph and the min-cut tree before edge $\{2, 12\}$ is inserted. The the edge is added and Figure (c) describes the resulting construction given in [12], on which Cut-Clustering is then applied, yielding Fig. (d). The result does neither conform to Eq. (2) nor to what is attempted to be proven in [12].