Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2011,12
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Dynamic Graph Clustering Using Minimum-Cut Trees

Robert Görke, Tanja Hartmann, and Dorothea Wagner

2011

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Dynamic Graph Clustering Using Minimum-Cut Trees

Robert Görke, Tanja Hartmann, and Dorothea Wagner

Chair of Algorithmics I, Department of Informatics, Karlsruhe Institute of Technology (KIT)*
{robert.goerke, tanja.hartmann, dorothea.wagner}@kit.edu

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 quarantee, 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., the following overviews [2, 16, 4] and references therein. Inspired by the work of Kannan et al. [12], 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 [7], 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 [15, 14], 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 [9], which designs and evaluates several heuristics for dynamically maintaining a clustering with high quality. There, clustering quality is measured by the index *modularity* [13], which is NP-hard to optimize [1].

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 algorithm builds upon partially updating a half-finished minimum-cut tree of a graph in the spirit of Gusfield's [10] simplification of the Gomory-Hu algorithm [7]. 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 [7, 10, 3], 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 [8]; 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.

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

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.

The *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 $\theta_{u,v}$ 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 $\gamma_{b,d}$ (either represented as a set of edges or vertices/nodes, as convenient). An edge $e_T = \{u, v\}$ of $T(G)$ induces the cut $\theta_{u,v}$ in G, sometimes denoted θ_v if the context identifies u, by decomposing $T(G)$ into two connected components. We sometimes identify e_T with the cut it induces in G. For details on minimum-cut trees, see the pioneering work by Gomory and Hu [7] or the simplifications by Gusfield [10].

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 $M \subseteq E$. 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, which we will rely on in the following, leaving their rather basic proofs to the reader.

Lemma 1. *Let $e_T = \{u, v\} \in E_T$ be an edge in $T(G)$, and let $e = \{b, d\}$ be modified in G.*

Consider $G^{\oplus}$: If $e_T \notin \gamma_{b,d}$ then e_T is still a minimum u-v-cut with weight $c(\theta_{u,v})$. If $e_T \in \gamma_{b,d}$ then its cut-weight is $c(\theta_{u,v}) + \Delta$; furthermore, it stays a minimum u-v-cut iff $\forall u$-v-cuts θ' in G that do not separate *b, d: $c(\theta') \geq c(\theta_{u,v}) + \Delta$.*

Consider $G^{\ominus}$: If $e_T \in \gamma_{b,d}$ then e_T remains a minimum u-v-cut, now with weight $c(\theta_{u,v}) - \Delta$. If $e_T \notin \gamma_{b,d}$ then it retains weight $c(\theta_{u,v})$; furthermore, it stays a minimum u-v-cut iff $\forall u$-v-cuts θ' in G that separate *b, d: $c(\theta') \geq c(\theta_{u,v}) + \Delta$.*

2 Fundamentals

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. [12]:

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

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. [12], which bases on the related measure conductance[1], Flake

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

et al. [3] design a graph clustering algorithm (Algorithm 1) that, given parameter α, asserts the following:[2]

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

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 [3]). 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 [3]. Flake at al. further show how nesting properties of minimum cuts [5] 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. 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. 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.

Algorithm 1: Cut-Clustering

Input: Graph $G = (V, E, c)$, α
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_α
6 $T(G_\alpha) \leftarrow T(G_\alpha) - t$
7 $\mathcal{C}(G) \leftarrow$ components of $T(G_\alpha)$

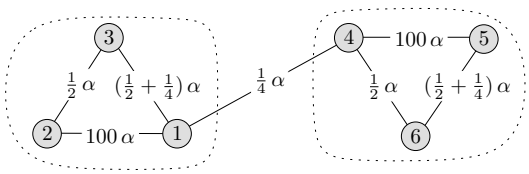
2.2 A Dynamic Attempt

Saha and Mitra [14, 15] 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. 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 [11]. Algorithm 2 sketches the approach given in [15] 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.

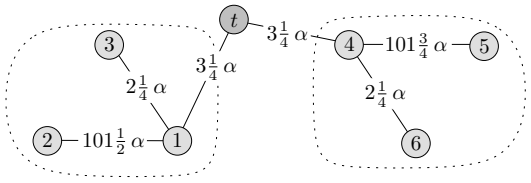
Algorithm 2: Old Inter-Edge Insertion

Input: $G = (V, E, w)$, α, $\mathcal{C}$, $e_\oplus = \{b, d\}$, $b \in C_b$, $d \in C_d$
if *inter-cluster quality of* C_d, 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): enlarge G to G_α (cp. Cut-Clustering)
dissolve C_b and C_d and contract all other vertices
perform line 5 and 6 of Cut-Clustering on G_α
return $(\mathcal{C} \setminus \{C_b, C_d\}) \cup \{$newly formed clusters from $C_b \cup C_d\}$

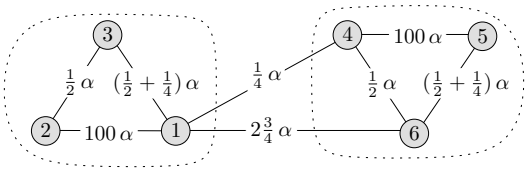
[2] The disjoint union $A \cup B$ with $A \cap B = \emptyset$ is denoted by $A \dot\cup B$.



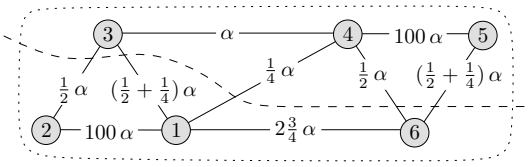
(a) Graph G^0 with clustering $\mathcal{C}(G^0)$ by the Cut-Clustering method.



(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).

Fig. 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.

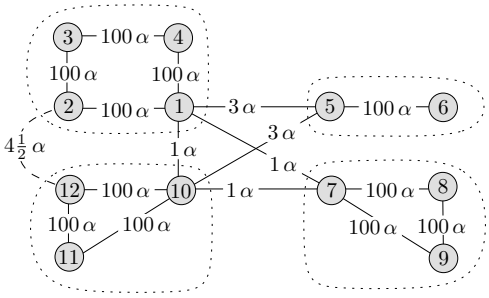
A Counter-Example for Case 1 *and* Case 2. We now give an example instance which the algorithm given in [15] fails to cluster correctly. The two upper figures (Figure 1(a), 1(b)) show the input instance, as computed by algorithm Cut-Clustering. In Figure 1(c), a first edge insertion then triggers Case 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 ($c(C_1, C_2) = 4\alpha > 3 = \alpha \cdot \min\{|C_1|, |C_2|\}$), 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 [15] 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 [15], 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 [15]: A "newly formed cluster from $C_b \cup C_d$" as returned in line 8 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 $3 \cdot \alpha = \min\{|P|, |Q|\} \geq c(P, Q) = 2 \cdot \alpha$.

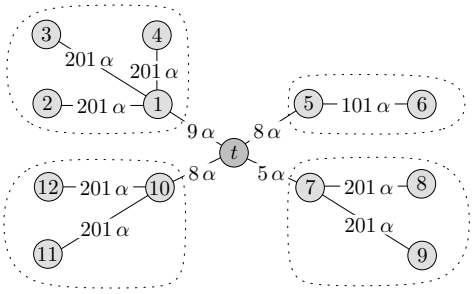
2.3 Minimum-Cut Trees and the Gomory-Hu Algorithm

We briefly describe the construction of a minimum-cut tree as proposed by Gomory and Hu [7] and simplified by Gusfield [10]. Although we will adopt ideas of the latter work, we first give Gomory and Hu's algorithm (Algorithm 3) as the foundation.

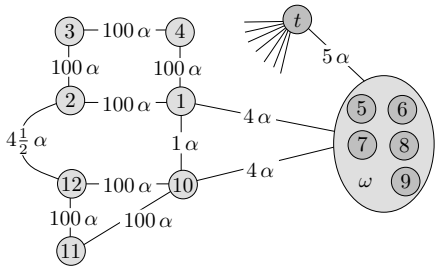
The algorithm builds the minimum-cut tree of a graph by iteratively finding minimum u-v-cuts for vertices that have not yet been separated by a previous minimum cut. Each iteration allows the construction of exactly one edge of the final minimum-cut tree. Thus, the algorithm



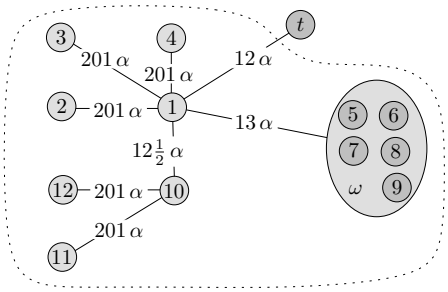
(a) Graph G with clustering $\mathcal{C}(G)$ resulting from the Cut-Clustering method.



(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)).

Fig. 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

Initialize $V_* \leftarrow \{V\}, E_* \leftarrow \emptyset$ and c_* empty and tree $T_*(G) := (V_*, E_*, c_*)$
while $\exists S \in V_*$ *with* $|S| > 1$ **do** `// unfold all super-nodes`
  $\{u, v\} \leftarrow$ arbitrary pair from $\binom{S}{2}$
  forall $S_j \sim S$ *in* $T_*(G)$ **do**
    $N_j \leftarrow$ subtree of S with $S_j \in N_j$
  $G_S = (V_S, E_S, c_S) \leftarrow$ in G contract each subtree N_j to node η_j `// subtree contrac.`
  $(U, V_S \setminus U) \leftarrow$ minimum u-v-cut in G_S, weight δ, $u \in U$
  $S_u \leftarrow S \cap U$ and $S_v \leftarrow S \cap (V_S \setminus U)$ `// split` $S = S_u \cup S_v$
  $V_* \leftarrow (V_* \setminus \{S\}) \cup \{S_u, S_v\}$, $E_* \leftarrow E_* \cup \{\{S_u, S_v\}\}$, $c_*(S_u, S_v) \leftarrow \delta$
  forall *former edges* $e_j = \{S, S_j\} \in E_*$ **do**
    if $\eta_j \in U$ **then** $e_j \leftarrow \{S_u, S_j\}$; `// either reconnect` S_j `to` S_u
    else $e_j \leftarrow \{S_v, S_j\}$; `// or reconnect` S_j `to` S_v
return $T_*(G)$

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 [6] 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, 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 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 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 *execution* $\mathrm{GH} = (G, F, K)$ of Gomory-Hu is characterized by graph G, sequence F of $n-1$ *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. Pair $\{u, v\} \subseteq V$ is a *cut pair* of edge $e_T = \{x, y\}$ of cut tree $T(G)$ if $\theta_{x,y}$ is a minimum u-v-cut in G.

In the situation of Cut-Clustering where G gets augmented by an artificial vertex t the minimum-cut tree $T(G_\alpha)$ attributes a cut pair $\{v_i, t\}$ to each cluster C_i with $v_i \in C_i$. We call v_i the *representative* $r(C_i)$ of C_i. The edges $e_i = \{v_i, t\}$ in $T(G_\alpha)$ induce a set of non-crossing, non-nested minimum v_i-t-cuts in G_α that together separate t from G. A set of v_i-t-cuts is *non-nested* if the cut sides containing the v_is are mutually disjoint. Remember that it is not necessary to compute the whole minimum-cut tree $T(G_\alpha)$, as shown by Flake et al. The following theorem states that given a set of non-crossing, non-nested minimum v_i-t-cuts, the existence of a GH with starting sequence F of step pairs $\{v_i, t\}$ can also be assumed.

Theorem 1. *For some vertices v_i in G let Θ be a set of non-crossing, non-nested minimum v_i-t-cuts in G_α. Let K be a sequence of all cuts in Θ and F a sequence of the associated vertex pairs $\{v_i, t\}$. There exists a sequence F' of step pairs and a sequence K' of split cuts such that* $\mathrm{GH} = (G, F \cdot F', K \cdot K')$[3] *is a feasible* GH. *The intermediate minimum-cut tree after F is denoted by $T_\circ(G_\alpha)$.*

Proof. We index the vertices v_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 v_{i+1} still shares a super-node with t. Therefore, $\{v_{i+1}, t\}$ 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 by Gomory-Hu to continue. □

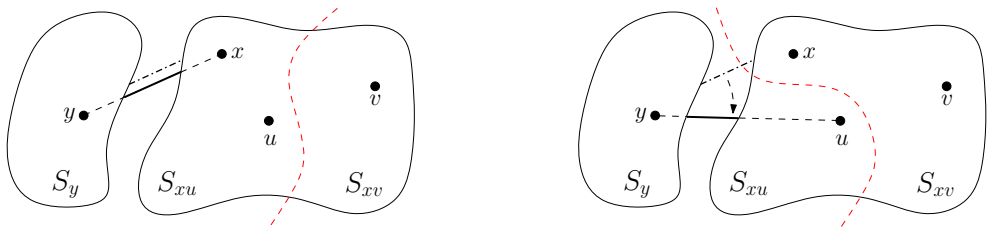
2.4 Using Arbitrary Minimum Cuts in G

Gusfield [10] 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 are non-crossing, but still minimal. We need to 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.

Lemma 2 (Gus. [10], Lemma 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} (red dashed split cut in Figure 3), wlog. with $S_y \sim S_{xu}$ in T_* (dash-dotted edge). If $x \in S_{xu}$, $\{x, y\}$ remains a cut pair of edge $\{S_{xu}, S_y\}$ (we say edge $\{S_{x(u)}, S_y\}$ gets* reconnected*; see solid edge). If $x \in S_{xv}$, i.e., the minimum 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 Lemma 2, pair $\{x, y\}$ gets *hidden*, and, in the view of vertex y, we say that its former counterpart x gets *shadowed* by u (or by S_{xu}). It is not hard to see that during

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



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

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

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

Gomory-Hu, step pairs remain cut pairs, but cut pairs need not stem from step pairs. However, 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_x, S_y, it is the nearest cut pair of edge $\{S_x, S_y\}$; if a nearest cut pair gets hidden in $T_*(G)$ by a step of Gomory-Hu, as described in Lemma 2, if $x \in S_{xv}$ the nearest cut pair of the reconnected edge $\{S_{x(u)}, S_y\}$ becomes $\{u, y\}$ (which are in the adjacent nodes S_{xu}, S_y). 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 2 (Gus. [10], Theorem 2[4]). *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 $u \in U_S$.*

Being an ingredient to the original proof of Theorem 2, 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. [10], Lemma 1[4]). *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 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 injured. The final shape of the bent cut depends on which side of the split cut contains 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.

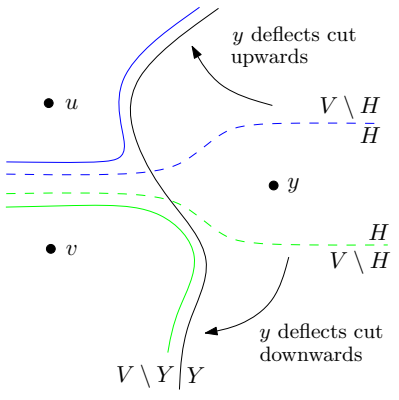


Fig. 4. Depending on y there are two different directions to which Lemma 3 bends the cut $(H, V \setminus H)$: upwards or downwards.

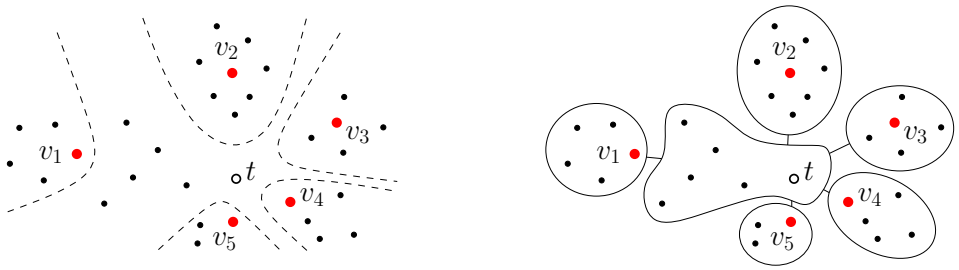
While Lemma 3 tells us how to find non-crossing cuts, it does *not* yet tell us how to proceed with the minimum-cut tree we are building up: Given a cut as by Theorem 2, Gomory and Hu state a simple mechanism which reconnects a former neighboring subtree N_j of

a node S to either of its two split parts (lines 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. 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:

Theorem 3 (Gus. [10], Theorem 3[4]). *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 2; then a neighboring subtree N_j of S, formerly connected by edge $\{S, S_j\}$, lies in U_S iff $r(S_j) \in U$.*

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 $G_\alpha^{\ominus}$ and $G_\alpha^{\oplus}$, respectively). Consider a partial minimum-cut tree $T_\circ$ (we use $T_\circ$ as a shorthand for $T_\circ(G_\alpha^{\oplus(\ominus)})$) sufficient to define a clustering, as discussed for Algorithm 1, or any intermediate state of it. By Theorem 1 tree $T_\circ$ consists of one super-node for each vertex v_i arranged starlike around a super-node containing t. An example of such an intermediate minimum-cut tree and of the corresponding cuts is depicted in Figure 5. We define valid representatives of the nodes in $T_\circ$ as follows: We choose t as representative $r(S_t)$ of the node $S_t \ni t$; for a node S_i containing v_i we choose $r(S_i) := v_i$. Note that the representatives so appointed equal those defined by Gusfield for the sequence F of step pairs $\{v_i, t\}$. Thus, $T_\circ$ is also a valid intermediate tree regarding Gusfield's algorithm, and we can continue a GH for $T_\circ$ using arbitrary split cuts instead of contracted subtrees.



(a) Set Θ of non-crossing, non-nested split cuts. (b) Intermediate minimum-cut tree $T_\circ$ regarding Θ.

Fig. 5. This example shows an intermediate minimum-cut tree $T_\circ$ regarding vertices $v_1, \ldots, v_5$; in order to have $T_\circ$ define a clustering, the vertices left inside t's super-node need to be separated from t by further cuts.

3.1 Cuts That Can Stay

A clustering $\mathcal{C}$ found by Cut-Clustering results from $|\mathcal{C}|$ non-crossing, non-nested minimum v_i-t-cuts with $v_i = r(C_i)$. Thus, the clustering can be treated as an intermediate minimum-cut tree $T_\circ$with $r(S_i) = r(C_i)$. The non-crossing nature of such cuts allows for more effort-saving and temporal smoothness. Lemma 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.

[4] Lemma 2, Theorem 2, Lemma 3 and Theorem 3 have been discussed and proven in [10] and the lemmas also in [7], we thus omit the proofs.

Remark 1. Intra-cluster insertion (resulting in $G_\alpha^\oplus$; b, d are in the same cluster in G_α):

Path $\gamma_{b,d}$ does not contain any edge $\{r(C_i), t\}$ (see Figure 13). Thus by Lemma 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 v_i-t-cuts in $G_\alpha^\oplus$ that together separate t from $G^\oplus$ and therefore the previous clustering is still valid.

Remark 2. Inter-cluster insertion (resulting in $G_\alpha^\oplus$; b, d are in different clusters in G_α):

Apart from $\{r(C_b), t\}$ and $\{r(C_d), t\}$ path $\gamma_{b,d}$ does not contain any edge incident to t. Again by Lemma 1 all these edges off $\gamma_{b,d}$ still induce minimum cuts after the modification, the two other cuts need not remain minimum. The thus implied set of non-crossing, non-nested minimum v_i-t-cuts in $G_\alpha^\oplus$ yields an intermediate minimum-cut tree $T_\circ$ with $|\mathcal{C}| - 1$ super-nodes (see Figure 12) that can be used by Gomory-Hu to continue.

Remark 3. Inter-cluster deletion (resulting in $G_\alpha^\ominus$; b, d are in different clusters in G_α):

Path $\gamma_{b,d}$ contains the two edges $\{r(C_b), t\}$ and $\{r(C_d), t\}$. By Lemma 1 both cuts induced by these edges are still minimum after the modification, all other edges incident to t need not remain minimum. This implies an intermediate minimum-cut tree $T_\circ$ with three super-nodes (see Figure 10) that can be used by Gomory-Hu to continue.

Remark 4. Intra-cluster deletion (resulting in $G_\alpha^\ominus$; b, d are in the same cluster in G_α):

Path $\gamma_{b,d}$ does not contain any of the edges incident to t. The thus implied intermediate tree $T_\circ$ consists of only one super-node covering all vertices of $G_\alpha^\ominus$ (see Figure 11). However, the following lemma yields some cuts that can stay. Its proof mostly relies on properties of Gomory-Hu and on Lemma 1.

Lemma 4. *In $G_\alpha^\ominus$, let $(U, V_\alpha \setminus U)$ be a minimum u-v-cut not separating $\{b, d\}$, with $\gamma_{b,d}$ in $V_\alpha \setminus U$. Then, a cut induced by an edge $\{g, h\}$ of the old $T(G_\alpha)$, with $g, h \in U$, remains a minimum separating cut for all its previous cut pairs within U in $G_\alpha^\ominus$, and a minimum g-h-cut in particular.*

Proof. Consider the minimum u-v-cut $(U, V_\alpha \setminus U)$ in $G_\alpha^\ominus$ to be the first split cut of GH, with step pair $\{u, v\}$. As the cut has $\gamma_{b,d}$ in $V_\alpha \setminus U$, b, d also lie on side $V_\alpha \setminus U$, and thus, from the view of $\{U\}$ as next split node, b, d are pseudo-contracted. This is, for any step pair within U, $\{b, d\}$ are not separated, and by the correctness of Gomory-Hu and Lemma 1, any previous minimum g-h-cut is still valid in $G_\alpha^\ominus$. Furthermore, Lemma 2 asserts that previous cut pairs within U also stay valid. □

It is important to see that it is not necessary to maintain a full minimum-cut tree to determine the induced clustering. What we need is a set Θ of non-crossing, non-nested minimum v_i-t-cuts in $G_\alpha^{\oplus(\ominus)}$ that together separate t from $G^{\oplus(\ominus)}$. Thus, the idea based on Theorem 1 is to continue Gomory-Hu on $T_\circ$ by checking the edges incident to t in the old tree $T(G_\alpha)$.

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. During a continued GH for $T_\circ$, we might then find an edge $\{v_i, t\}$ of the old $T(G_\alpha)$ that is *not* reconfirmed by a computation in $G_\alpha^{\oplus(\ominus)}$, but a new, cheaper minimum v_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 edge $\{v_i, t\}$ of the old tree $T(G_\alpha)$ has not been reconfirmed by a continued GH execution for $T_\circ$, but instead, by a computation in $G_\alpha^\ominus$, a new, cheaper minimum v_i-t-cut $\theta' = (U, V_\alpha \setminus U)$ is found, with v_i in U.

Definition 1 (Treetop and Wood). *Consider edge $e = \{u, v\}$ off $\gamma_{b,d}$ in $T(G_\alpha)$, and cut $(U, V_\alpha \setminus U)$ in G_α induced by e with $\gamma_{b,d} \subseteq V_\alpha \setminus U$. The subtree spanning $V_\alpha \setminus U$ is called the* wood *$\sharp_e$, the one spanning U the* treetop *$\Uparrow_e$ of e (see Figure 6).*

Cutting down the generality of Lemma 5 to edges incident to t that have not been reconfirmed, this lemma tells us, that for any such minimum v_i-t-cut θ' there is a minimum v_i-t-cut $\theta = (U \cup \Uparrow_{\{v_i,t\}}, (V_\alpha \setminus U) \setminus \Uparrow_{\{v_i,t\}})$ in $G_\alpha^\ominus$ that (a) does not split $\Uparrow_{\{v_i,t\}}$, (b) but splits $V_\alpha \setminus \Uparrow_{\{v_i,t\}}$ exactly as θ' does. Figure 6 illustrates such cuts.

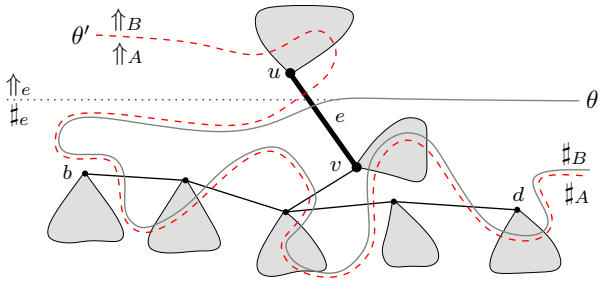


Fig. 6. Special parts of $G_\alpha^\ominus$: $\gamma_{b,d}$ (black) connects b and d; wood $\sharp_e$ and treetop $\Uparrow_e$ of edge e, both cut by θ' (dashed), adjusted to θ (solid) by Lemma 5, as to cut only $\sharp_e$.

Lemma 5. *Given $e = \{u, v\}$ off $\gamma_{b,d}$ in $T(G_\alpha)$. Let (A, B) be a cut separating u and v such that (A, B) induces a cut $(\Uparrow_A, \Uparrow_B)$ of $\Uparrow_e$ with $u \in \Uparrow_A$ and a cut $(\#_A, \#_B)$ of $\#_e$ with $v \in \#_B$. Then $c_\alpha^\ominus(\#_A \cup \Uparrow_e, \#_B) \leq c_\alpha^\ominus(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$.*

Proof. Using the fact that, in the old tree $T(G_\alpha)$, edge e represents a minimum u-v-cut, we prove Lemma 5 by contradiction. We show that cut $(\Uparrow_A, V_\alpha \setminus \Uparrow_A)$ would have been cheaper than the edge-induced minimum u-v-cut $(\Uparrow_e, V_\alpha \setminus \Uparrow_e)$ in G_α if $c^\ominus(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$ was cheaper than $c^\ominus(\#_A \cup \Uparrow_e, \#_B)$ in $G_\alpha^\ominus$. We express the costs of $(\Uparrow_A, V_\alpha \setminus \Uparrow_A)$ and $(\Uparrow_e, V_\alpha \setminus \Uparrow_e)$ with the aid of $(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$ and $(\#_A \cup \Uparrow_e, \#_B)$ considered in Lemma 5. Note that $(\Uparrow_A, V_\alpha \setminus \Uparrow_A)$ and $(\Uparrow_e, V_\alpha \setminus \Uparrow_e)$ do not separate b and d. Thus, their costs are unaffected by the deletion, by Lemma 1. We get

$$\begin{aligned}
&(i)\ c_\alpha(\Uparrow_A, V_\alpha \setminus \Uparrow_A) = c_\alpha^\ominus(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) - c_\alpha^\ominus(\#_A, \#_B \cup \Uparrow_B) + c_\alpha^\ominus(\#_A, \Uparrow_A)\\
&(ii)\ c_\alpha(\Uparrow_e, V_\alpha \setminus \Uparrow_e) = c_\alpha^\ominus(\#_A \cup \Uparrow_e, \#_B) - c_\alpha^\ominus(\#_A, \#_B) + c_\alpha^\ominus(\#_A, \Uparrow_e)
\end{aligned}$$

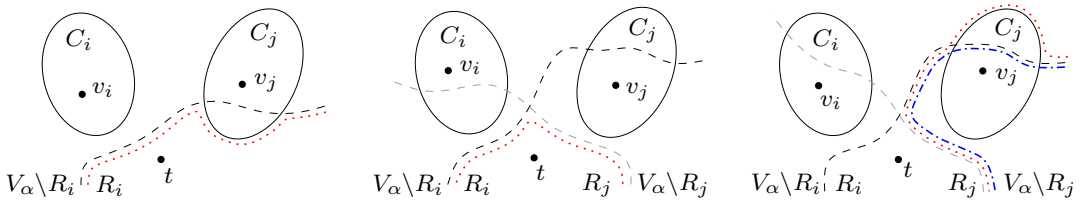
Certainly, it holds that $c_\alpha^\ominus(\#_A, \#_B) \leq c_\alpha^\ominus(\#_A, \#_B \cup \Uparrow_B)$ and that $c_\alpha^\ominus(\#_A, \Uparrow_A) \leq c_\alpha^\ominus(\#_A, \Uparrow_e)$; together with the assumption that the lemma does not hold, i.e., that $c^\ominus(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) < c^\ominus(\#_A \cup \Uparrow_e, \#_B)$ holds, we can see the following, by subtracting *(i)* and *(ii)*:

$$\begin{aligned}
c_\alpha(\Uparrow_A, V_\alpha \setminus \Uparrow_A) - c_\alpha(\Uparrow_e, V_\alpha \setminus \Uparrow_e) = {} & [c_\alpha^\ominus(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) - c_\alpha^\ominus(\#_A \cup \Uparrow_e, \#_B)]\\
& - [c_\alpha^\ominus(\#_A, \#_B \cup \Uparrow_B) - c_\alpha^\ominus(\#_A, \#_B)]\\
& + [c_\alpha^\ominus(\#_A, \Uparrow_A) - c_\alpha^\ominus(\#_A, \Uparrow_e)] < 0
\end{aligned}$$

This contradicts the fact that the edge-induced cut $(\Uparrow_e, V_\alpha \setminus \Uparrow_e)$ is a minimum u-v-cut in G_α. □

While this lemma 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. In $T(G_\alpha)$ consider the star spanning the vertices $t, v_1, \ldots, v_z$ with $v_i := r(C_i)$. For each v_i with $\gamma_{b,d} \cap C_i = \emptyset$, the clusters C_i are the treetops $\Uparrow_{\{v_i,t\}}$. Thus, by Lemma 5, for each such edge $\{v_i, t\}$ the following partition-property holds: In $G_\alpha^\ominus$, it holds that for any v_i-t-cut $\theta_i' := (R_i, V_\alpha \setminus R_i)$ (with $t \in R_i$), the cut $\theta_i := (R_i \setminus C_i, (V_\alpha \setminus R_i) \cup C_i)$ has at most the same weight as θ_i'.

Suppose we have improved a cut according to the partition-property described above; for the sake of notational simplicity we again call this cut $\theta' := (R, V_\alpha \setminus R)$, and then, processing it further, work it into a cut called θ. In $G_\alpha^\ominus$ choose two such vertices v_i, v_j with $\gamma_{b,d} \cap C_i = \emptyset$ and $\gamma_{b,d} \cap C_j = \emptyset$. Consider a minimum v_i-t-cut $\theta_i' := (R_i, V_\alpha \setminus R_i)$ (black dashed in Figure 7), with $t \in R_i$, that does not split C_i and an analog minimum v_j-t-cut θ_j' (gray dashed in Figure 7), (by the partition-property they exist). We distinguish three scenarios, given in Figure 7, which yield the following possibilities of reshaping minimum cuts:



(a) θ'_i separates v_j, t. (b) θ'_i does not separate v_j, t, but θ'_j separates v_i, t. (c) neither does θ'_i separate v_j, t, nor θ'_j v_i, t.

Fig. 7. Three different scenarios concerning the positions of θ'_i and θ'_j (black and gray dashed, respectively), and their adjustments (red bold dotted).

Scenario (a): As cut θ'_i separates v_j and t, and as $\{v_j, t\}$ satisfies the partition-property, the cut $\theta_i := (R_i \setminus C_j, (V_\alpha \setminus R_i) \cup C_j)$ (red bold dashed) has weight $c^{\ominus}_\alpha(\theta_i) \leq c^{\ominus}_\alpha(\theta'_i)$ and still separates v_i and t and is thus a minimum v_i-t-cut, which does not split $C_i \cup C_j$.

While Lemma 5 covers this case only for $\gamma_{b,d} \cap C_i = \emptyset$ and $\gamma_{b,d} \cap C_j = \emptyset$, we can even drop this limitation: Suppose $v_b \neq v_d$. Using v_b (v_d is analog) as v_j, the same assertion immediately holds, but now by pseudo-contraction. To see this, note that while $\{v_b, t\}$ is on $\gamma_{b,d}$, the cut defining C_b remains a minimum v_b-t-cut in $G^{\ominus}_\alpha$ (by Lemma 1) and thus pseudo-contracts C_b from the view of θ'_i. Remember that (again by Lemma 1) we need not do anything about the old cut if v_i equals either v_b or v_d. In the case that $v_b = v_d =: v_{b/d}$, we can not apply Lemma 1 as $\{v_{b/d}, t\}$ is not on $\gamma_{b,d}$, i.e., we cannot adjust other v_is' cuts to $C_{b/d}$ with the arguments in this scenario.

Scenario (b): The cut θ'_j basically behaves like θ'_i in Scenario (a). The difference here is that we assume that θ'_i is known and dealt with, yielding θ_i, before θ'_j is considered. This is, v_i's side of θ_i is already pseudo-contracted by θ_i. As θ_i does not separate v_j from t, indeed, but may already have been adjusted as to shadow some other cut vertices, the cut θ'_j thus is reshaped regarding the whole cut side $V_\alpha \setminus R_i$ (instead of only to cluster $C_i \subseteq V_\alpha \setminus R_i$ as by Scenario (a)). By pseudo-contraction the cut $\theta_j := (R_i \cap R_j, (V_\alpha \setminus R_i) \cup (V_\alpha \setminus R_j))$ (red bold dotted) is a minimum v_j-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 vertex from t. Regardless of whether either cut has already been adjusted as described above, we can see the following. By pseudo-contraction, the cut $((V_\alpha \setminus R_i) \cup R_j, (V_\alpha \setminus R_j) \cap R_i)$ (blue dash-dotted) is a minimum v_j-t-cut. This cut can be adjusted to $\theta_j := (((V_\alpha \setminus R_i) \cup R_j) \setminus C_j, ((V_\alpha \setminus R_j) \cap R_i) \cup C_j)$ (red bold dotted), which neither splits C_i nor C_j, by means of the partition-property of $\{v_j, t\}$. Analogously, $\theta_i := (((V_\alpha \setminus R_j) \cup R_i) \setminus C_i, ((V_\alpha \setminus R_i) \cap R_j) \cup C_i)$ (not shown) is a minimum v_i-t-cut, and θ_i and θ_j do not cross.

Being more general again, consider the case $v_b \neq v_d$ and use wlog. v_b as v_i. As $\theta'_j = (R_j, V_\alpha \setminus R_j)$ does not separate v_b and t but the cut defining C_b remains by Lemma 1 and thus pseudo-contracts C_b and does not cross C_j, we immediately obtain $\theta_j := (R_j \cup C_b, (V_\alpha \setminus R_j) \setminus C_b)$ that also does neither cross C_b nor C_j. In the case that $v_b = v_d =: v_{b/d}$ is used as v_i, any newly found minimum $v_{b/d}$-t-cut can be reshaped by Lemma 5 such that it does not cross the treetop of $\{v_{b/d}, t\}$ which consists of all other old clusters and t (Note that it can *not* be adjusted such that it definitely spares $C_{b/d}$ as $\{v_{b/d}, t\}$ has no partition-property). Thus, θ'_j is not forced to cut through C_j (in contrast to the blue dash-dotted cut) and never will. However, it still may cut through $C_{b/d}$.

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

Observation 1 *During a* GH *starting from $T_\circ$ for $G^{\ominus}$ and checking the remaining edges incident to t in old $T(G_\alpha)$, whenever we discover a new, cheaper minimum v_i-t-cut θ' ($\gamma_{b,d} \cap C_i = \emptyset$) we can iteratively reshape θ' into a minimum v_i-t-cut θ which neither cuts C_i nor any other cluster C_j with $\gamma_{b,d} \cap C_j = \emptyset$, by means of Cases (a,b,c). For v_b and v_d ($v_b \neq v_d$) the clusters C_b and C_d are preserved anyway. In contrast, the cluster $C_{b/d}$ ($v_b = v_d =: v_{b/d}$) can not be protected but at least any new minimum $v_{b/d}$-t-cut spares all other old clusters.*

New Cuts After Edge Insertions. The bigger picture of 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 cannot 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 a continued GH for $T_\circ$; namely $\{r(C_b), t\}$ and $\{r(C_d), t\}$, i.e., those edges that are incident to t and contained in $\gamma_{b,d}$. We call the representatives $r(C_b)$ and $r(C_d)$, v_b and v_d. The checked edges are reconfirmed by a computation in $G_\alpha^\oplus$ if there exists a minimum v_b-t-cut and a minimum v_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 $\{v_b, t\}$ or $\{v_d, t\}$ is not reconfirmed, depending on the shape of the new cut, we may still be able to retain the associated cluster.

Substituting the general edge $\{u, v\}$ in Lemma 6 (and Figure 8) by $\{v_b, t\}$, this lemma tells us that for any new minimum v_b-t-cut $\theta'_b = (U, V_\alpha(S) \setminus U)$ that has v_b, b, d *on the same side* (with S super-node in $T_\circ$ containing t and $v_b \in U$) there is a minimum v_b-t-cut $\theta_b = (U \cup C_b, (V_\alpha \setminus U) \setminus C_b)$ in $G_\alpha^\oplus$ that (a) does not split C_b, (b) but splits $V_\alpha \setminus C_b$ exactly as θ'_b does. (The attribute of a cut to have v_b, b, d *on the same side* will later be introduced as cav, as by "cut attribute regarding v".) Considering any new minimum v_d-t-cut is analogous.

Furthermore, there are also new minimum v_b-t-cuts conceivable that, conversely, do not separate t, b, d. For those cuts we apply Lemma 6 with $t =: u$ and $v_b =: v$ which yields the following insight: For any new minimum v_b-t-cut $\theta'_b := (U, V_\alpha(S) \setminus U)$ that has t, b, d *on the same side* (with S super-node in $T_\circ$ containing t and $v_b \in U$) there is a minimum v_b-t-cut $\theta_b := (U \cap C_b, V_\alpha \setminus (U \cap C_b))$ in $G_\alpha^\oplus$ that (a) does not split C_d and does not shadow any other cluster, (b) but splits C_b exactly as θ'_b does. (The attribute of a cut to have t, b, d *on the same side* will later be introduced as cat, as by "cut attribute regarding t".)

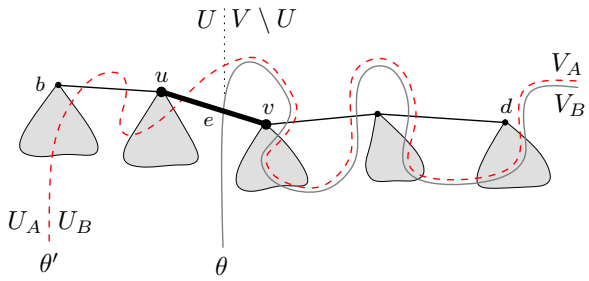


Fig. 8. Special parts of $G_\alpha^\oplus$: $\gamma_{b,d}$ (black) connects b and d; side U and side $V\alpha \setminus U$ given by edge e, both cut by θ' (dashed), adjusted to θ (solid) by Lemma 6.

Lemma 6. *Given $e = \{u, v\}$ on $\gamma_{b,d}$ in $T(G_\alpha)$ and cut $(U, V_\alpha \setminus U)$ induced by e with $u \in U$. Let (A, B) be a cut separating u and v such that (A, B) induces a cut (U_A, U_B) of U with $u \in U_A$ and a cut (V_A, V_B) of $V_\alpha \setminus U$ with $v \in V_B$. Furthermore let b, d, u share the same side of cut $(V_A \cup U_A, V_B \cup U_B)$. Then $c_\alpha^\oplus(V_A \cup U, V_B) \leq c_\alpha^\oplus(V_A \cup U_A, V_B \cup U_B)$.*

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 $(U_A, V_\alpha \setminus U_A)$ would be cheaper than the edge-induced minimum u-v-cut $(U, V_\alpha \setminus U)$ in G_α if $c_\oplus(V_A \cup U_A, V_B \cup U_B)$ was cheaper than $c_\oplus(V_A \cup U, V_B)$ in $G_\alpha^\oplus$. We express the costs of $(U_A, V_\alpha \setminus U_A)$ and $(U, V_\alpha \setminus U)$ with the aid of $(V_A \cup U_A, V_B \cup U_B)$ and $(V_A \cup U, V_B)$ considered in Lemma 6. Note that $c_\alpha(U_A, V_\alpha \setminus U_A) = c_\alpha^\oplus(U_A, V_\alpha \setminus U_A) - \Delta$ and $c_\alpha(U, V_\alpha \setminus U) = c_\alpha^\oplus(U, V_\alpha \setminus U) - \Delta$. Thus, for our contradiction, it will do to show that $c_\alpha^\oplus(U_A, V_\alpha \setminus U_A)$ would be cheaper than $c_\alpha^\oplus(U, V_\alpha \setminus U)$. We get

$$\begin{aligned}
&(i)\ c_\alpha^\oplus(U_A, V_\alpha \setminus U_A) = c_\alpha^\oplus(V_A \cup U_A, V_B \cup U_B) - c_\alpha^\oplus(V_A, V_B \cup U_B) + c_\alpha^\oplus(V_A, U_A)\\
&(ii)\ c_\alpha^\oplus(U, V_\alpha \setminus U) \quad = c_\alpha^\oplus(V_A \cup U, V_B) \qquad - c_\alpha^\oplus(V_A, V_B) \qquad + c_\alpha^\oplus(V_A, U)
\end{aligned}$$

Again we observe two inequalities: $c^{\oplus}_{\alpha}(V_A, V_B) \leq c^{\oplus}_{\alpha}(V_A, V_B \cup U_B)$ and $c^{\oplus}_{\alpha}(V_A, U_A) \leq c^{\oplus}_{\alpha}(V_A, U)$; together with the contradicting assumption that $c_{\oplus}(V_A \cup U_A, V_B \cup U_B) < c_{\oplus}(V_A \cup U, V_B)$, by subtracting (i) and (ii), we get:

$$\begin{aligned} c^{\oplus}_{\alpha}(U_A, V_\alpha \setminus U_A) - c^{\oplus}_{\alpha}(U, V_\alpha \setminus U) = &[c^{\oplus}_{\alpha}(V_A \cup U_A, V_B \cup U_B) - c^{\oplus}_{\alpha}(V_A \cup U, V_B)] \\ &- [c^{\oplus}_{\alpha}(V_A, V_B \cup U_B) - c^{\oplus}_{\alpha}(V_A, V_B)] \\ &+ [c^{\oplus}_{\alpha}(V_A, U_A) - c^{\oplus}_{\alpha}(V_A, U)] < 0 \end{aligned}$$

This contradicts the fact that the edge-induced cut $(U, V_\alpha \setminus U)$ is a minimum u-v-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 v_b-t-cut, with v_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 $T(G_\alpha)$ consider the edges $\{v_b, t\}$ and $\{v_d, t\}$ on path $\gamma_{b,d}$. The clusters C_b and C_d are associated with the vertices v_b and v_d. Returning to the notation used before in the case of edge deletion, by Lemma 6 the following partition-property for $\{v_b, t\}$ holds: For any v_b-t-cut $\theta'_b := (R_b, V_\alpha \setminus R_b)$ (with t in R_b) that has v_b, b, d *on the same side* the cut $\theta_b := (R_b \setminus C_b, (V_\alpha \setminus R_b) \cup C_b)$ is of at most the same weight in $G^{\oplus}_{\alpha}$. For v_d an analogous property holds. In order to see which cuts are eligible for adjustment by this partition-property, we introduce the following attributes for cuts: We say that a v_b-t-cut possesses the cut attribute cav(b) if it has v_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 cav(b) and cat(b) are mutually exclusive). Analogously, cav(d) and cat(d) denote the cut attributes for v_d-t-cuts.

In $G^{\oplus}_{\alpha}$ consider a minimum v_b-t-cut $\theta'_b := (R_b, V_\alpha \setminus R_b)$ (black dashed in Figure 9), with t in R_b, which is cheaper than the previous cut plus Δ and an analog minimum v_d-t-cut $\theta'_d := (R_d, V_\alpha \setminus R_d)$ (gray dashed). Each cut either satisfies cav or cat as it does not cross the inserted edge $\{b, d\}$, by Lemma 1. We distinguish the following three scenarios (see Figures 9(a)-9(c)).

Scenario (a): The attribute cav for cuts together with the partition-property for $\{v_b, t\}$ and $\{v_d, t\}$ allows to conserve both clusters C_b and C_d as follows. As cut θ'_b separates v_d, t and satisfies cav(b) it also possesses cav(d), and as $\{v_d, t\}$ satisfies the partition-property, the cut $\theta_b := (R_b \setminus C_d, (V_\alpha \setminus R_b) \cup C_d)$ (red bold dotted) has weight $c^{\oplus}_{\alpha}(\theta_b) \leq c^{\oplus}_{\alpha}(\theta'_b)$ and is thus a minimum v_b-t-cut that does not split $C_b \cup C_d$.

Scenario (b): Suppose neither θ'_d nor θ'_b separates the other representative from t. By pseudo-contraction θ'_b can be reshaped to the blue dash-dotted cut which possesses cat(b). Applying Lemma 6 to this minimum v_b-t-cut yields $\theta_b := (R_d \cap C_b, V_\alpha \setminus (R_d \cap C_b))$ (red dotted) as a minimum v_b-t-cut, which does not shadow any of the old clusters. Furthermore, θ'_d, which still possesses cav(d), can be adjusted such that at least cluster C_d is not split and thus becomes pseudo-contracted. A similar situation also appears in Scenario (c(ii)).

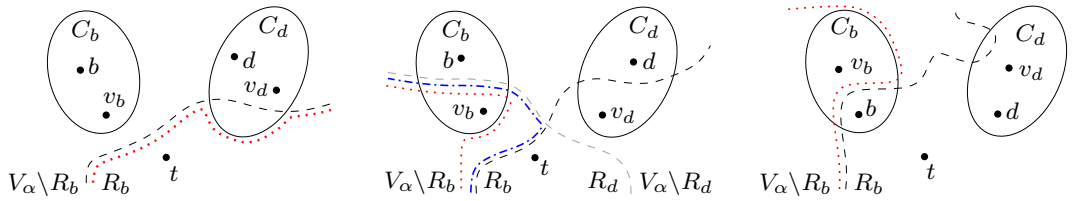
Scenario (c): Suppose cut θ'_b satisfies cat(b). Applying Lemma 6 to θ'_b which satisfies cat(b) yields a cut shaped as shown in the figure (red dotted), in particular, a cut that does not shadow any of the old clusters and does not separate v_d, t. The second cut θ'_d (not shown) now has one of three possible, mutually exclusive characteristics:

(i) θ'_d also possesses cat(d): Then it has a shape analog 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. Furthermore, C_b is not cut by θ'_d and vice versa. The same assertions hold for θ'_b being a reconfirmed, remaining cut pseudo-contracting C_b.

(ii) θ'_d possesses cav(d) but does not separate v_b, t: Due to the partition-property of $\{v_d, t\}$ it then can be adjusted such that at least cluster C_d is not split and thus becomes pseudo-contracted by the reshaped cut θ'_d. The same assertion holds 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).

(iii) θ'_d separates v_b, t with cav(d): Then, independent of the shape of θ'_b, we get Scenario (a) with roles of v_b and v_d swapped.

Observation 2 *During a* GH *starting from $T_\circ$ for $G^{\oplus}$ and checking the two remaining edges incident to t in old $T(G_\alpha)$, whenever we discover a new minimum cut that is cheaper than the previous*



(a) θ'_b separates v_d, t and possesses cav(b).

(b) θ'_b does not separate v_d, t with cav(b); θ'_d does not separate v_b, t with cav(d).

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

Fig. 9. Three different scenarios concerning the positions of θ'_b and θ'_d (black and gray dashed), and their adjustments.

cut plus Δ and possesses cav *we can reshape this cut such that is 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 former cluster. A new minimum cut that possesses* cat *can be adjusted such that it does not shadow any old, remaining cluster off $\gamma_{b,d}$ 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 both considered cuts not to cut through any of the remaining clusters off $\gamma_{b,d}$.*

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 1). On the other hand, for inter-cluster insertions we have established rules which enable cluster preservation for a number of cases. However, the 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 $T(G_\alpha)$ adjacent to t, i.e., all representatives of the clusters. 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, non-nested minimum v_i-t-cuts that isolate t, Θ. We will thus focus on dynamically maintaining only this information. From Remarks 1-4, 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.

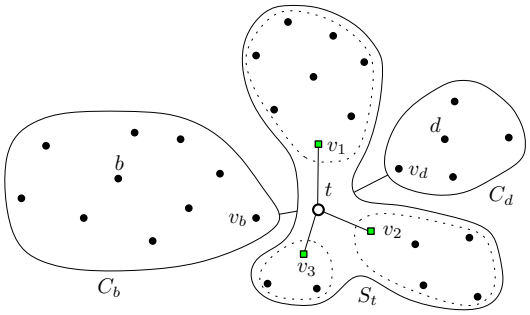


Fig. 10. $T_\circ(G_\alpha^\ominus)$ for an inter-cluster deletion, t's neighbors off $\gamma_{b,d}$ need inspection. The cuts of v_b and v_d are correct, but they might get shadowed.

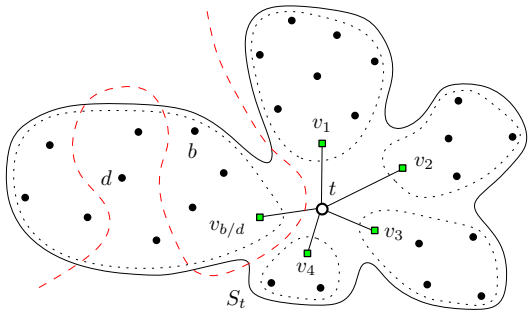


Fig. 11. $T_\circ(G_\alpha^\ominus)$ for an intra-cluster deletion, edge $\{v_{b/d}, t\}$ defines a treetop (t's side). The dashed cut could be added to Θ by Algorithm 6 (line 12).

Algorithm 4: Inter-Cluster Edge Deletion

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

1 $\Theta_{\text{ten}} \leftarrow \{\theta_b, \theta_d\}$, $W_{\text{ten}} \leftarrow \{v_b, v_d\}$
2 $D(v_b) \leftarrow \{v_b\}$, $D(v_d) \leftarrow \{v_d\}$
3 **for** $i = 1, \ldots, z$ **do** // not including v_b, v_d
4 Add v_i to W_{ten} // old cut-vertices
5 $D(v_i) \leftarrow \{v_i\}$ // shadows
6 **return** Check Cut-Vertices $(W(G), \Theta(G), G_\alpha^\ominus, \{b,d\}, D, W_{\text{ten}}, \Theta_{\text{ten}})$

Edge Deletion. Our first algorithm handles inter-cluster deletion (Algorithm 4). Just like its three counterparts, it takes as an input the sets $W(G)$ and $\Theta(G)$ of the old graph G (not the entire minimum-cut tree $T(G_\alpha)$), 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 t on $\gamma_{b,d}$, and thus, $T_\circ(G_\alpha)$ contains edges $\{v_b,t\}$ and $\{v_d,t\}$ cutting off the subtrees C_b and C_d of t by cuts θ_b, θ_d, as shown in Figure 10. All clusters contained in node $S_t \ni t$ need to be changed or reconfirmed. To this end Algorithm 4 lists all cut vertices, $v_b, v_d, v_1, \ldots, v_z$, into W_{ten}, and initializes their shadows $D(v_i) = \{v_i\}$ 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 Cut-Vertices is called, which—roughly speaking—performs those GH-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 $S_t = \{t\}$) then $W_{\text{ten}} = \{v_b, v_d\}$ and Algorithm 4 lets Check Cut-Vertices (Algorithm 5) simply return the input cuts and terminates. Otherwise, it iterates the set of former cut-vertices W_{ten}, thereby possibly shortening it, due to shadowing. We start by computing a new minimum v_i-t-cut for v_i. If the new cut is non-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 v_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, 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 (Scenario (b)), its cut is removed from Θ_{ten}.

Once all cut-vertex candidates 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 cut-vertices or it is itself shadowed by another cut-vertex. 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 cut-vertices alongside their adjusted, non-crossing, non-nested cuts are returned.

Algorithm 5: CHECK CUT-VERTICES

Input: $W(G), \Theta(G), G_\alpha^\ominus, \{b,d\}, D, W_{\text{ten}}, \Theta_{\text{ten}}$
Output: $W(G^\ominus), \Theta(G^\ominus)$

1 **while** W_{ten} *has next element* $v_i \notin \{v_b, v_d\}$ **do** // W_{ten} may change by loop iterations
2 $\theta_i \leftarrow$ minimum v_i-t-cut given by FLOWALGO(v_i, t)
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 Θ_{ten} // pointed at by v_i
5 **else** // new cheaper cuts
6 Add θ_i to Θ_{ten} // pointed at by v_i
7 **while** W_{ten} *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 Scenarios (a), (b)
9 Remove v_j from W_{ten}, $D(v_i) \leftarrow D(v_i) \cup D(v_j)$
10 **if** $\Theta_{ten} \ni \theta_j$, *pointed at by* v_j **then** Delete θ_j from Θ_{ten}
11 **forall** $v_i \in W_{ten}$, $v_i \notin \{v_b, v_d\}$ **do** // make new cuts cluster-preserving
12 set $(R, V_\alpha \setminus R) := \theta_i$ with $t \in R$ for $\theta_i \in \Theta_{\text{ten}}$ pointed at by v_i // just nomenclature
13 **forall** $v_j \in D(v_i)$ **do** // handle treetop and shadowed cuts ...
14 $\theta_i \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$ // ...with Scenarios (a), (b) and Lemma 5
15 **forall** $v_j \neq v_i$ *in* W_{ten} **do** // handle other cuts ...
16 **forall** $v_x \in D(v_j)$ **do** // ...with Scenario (c)
17 $\theta_i \leftarrow (R \cup C_x, (V_\alpha \setminus R) \setminus C_x)$
18 **return** $W_{\text{ten}}, \Theta_{\text{ten}}$

Next we look at *intra*-cluster edge deletion. Looking at our starting point $T_\circ$, the safe path $\gamma_{b,d}$ lies within some cluster $C_{b/d}$, which does not help much. In this case, t lies off $\gamma_{b,d}$, 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 Figure 11. Algorithm 6 has the same in- and output as Algorithm 4, and starts by finding a new minimum $v_{b/d}$-t-cut. If this yields that no new, cheaper $v_{b/d}$-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 (see the dashed cut in Figure 11), 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 $W_{\text{ten}}, \Theta_{\text{ten}}$ in lines 7-10. 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 8), as $C_{b/d}$ is neither treetop nor retained cluster, and thus defies the adjustments. After line 11 we have one minimum $v_{b/d}$-t-cut that leaves its treetop untouched, but might cut $C_{b/d}$, and we have a new set Θ_{ten} of minimum v_i-t-cuts (with some former $v_j \in W(G)$ possibly having become shadowed) which do not cut through former clusters C_i and do not cross in the treetop area 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 cut-vertices 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 $v_{b/d}$. 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 cannot 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.

Edge Insertion. The good news for handling $G^\oplus$ is, that an algorithm INTRA-CLUSTER EDGE ADDITION need not do anything, but return the old clustering (compare to Remark 1 and Figure 13). 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 t on path $\gamma_{b,d}$, thus in $T_\circ$ the only unknown cuts are those for v_b and v_d in S_t, see Figure 12. A sketch of what

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 $v_{b/d}$-t-cut given by FlowAlgo$(v_{b/d}, t)$
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 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** $\theta_{b/d} \leftarrow (R \cup C_i, (V_\alpha \setminus R) \setminus C_i)$; // by Lemma 5
7 $W_{\text{ten}} \leftarrow \emptyset, \Theta_{\text{ten}} \leftarrow \emptyset$
8 **for** $i = 1, \ldots, z$ **do** // not including $v_{b/d}$
9 Add v_i to W_{ten}
10 $D(v_i) \leftarrow \{v_i\}$
11 $W_{\text{ten}}, \Theta_{\text{ten}} \leftarrow$ Check Cut-Vertices $(W(G), \Theta(G), G_\alpha^\ominus, \{b,d\}, D, W_{\text{ten}}, \Theta_{\text{ten}})$
12 $W(G^\ominus) \leftarrow W_{\text{ten}} \cup \{v_{b/d}\}$, $\Theta(G^\ominus) \leftarrow \Theta_{\text{ten}} \cup \{\theta_{b/d}\}$
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** $W(G^\ominus), \Theta(G^\ominus)$

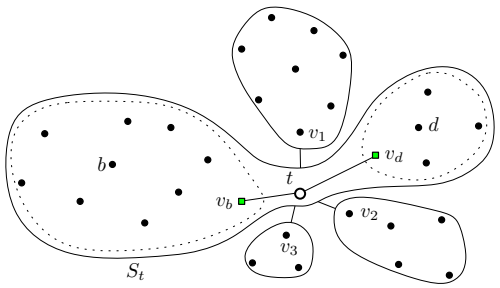


Fig. 12. $T_\circ(G_\alpha^\oplus)$ for an inter-cluster insertion. At least v_b and v_d need inspection.

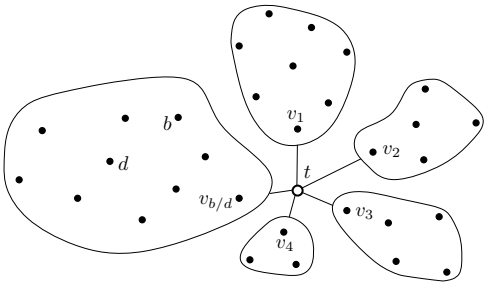


Fig. 13. $T_\circ(G_\alpha^\oplus)$ for an intra-cluster insertion. All relevant minimum v-t-cuts persist.

needs to be done, as given in Algorithm 7, is as follows: We compute new minimum v_b-t- and minimum v_d-t-cuts (line 5) and keep the former v_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 `cav` is 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 `cav` (line 9) we reshape it such that its associated cluster is conserved by Lemma 6, followed by potential shadowings of former v_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 can skip `checkShadows` and `adjustShadows`.

If possible we apply Scenario (a) in line 17–22. In case that both checked cuts satisfy `cav` and separate t from the other cut-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 cut-vertices.

Algorithm 7: 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: $W(G^\oplus), \Theta(G^\oplus)$

Θ_{ten}, $D(v_b)$, $D(v_d) \leftarrow \emptyset$, $W_{\text{ten}} \leftarrow \{v_b, v_d\}$
$cav(b)$, $cav(d) \leftarrow$ FALSE
$W(G^\oplus) \leftarrow \{v_1, \ldots, v_z\}$, $\Theta(G^\oplus) \leftarrow \{\theta_1, \ldots, \theta_z\}$ `// not including` v_b, v_d
forall $v_i \in \{v_b, v_d\}$ **do**
  $\theta_i \leftarrow$ minimum v_i-t-cut given by FLOWALGO(v_i, t)
  if $c_\alpha^\oplus(\theta_i) = c_\alpha(\theta_i^{old}) + \Delta$ **then** $\theta_i \leftarrow \theta_i^{\text{old}}$ `// retain old cuts if possible`
  else
    set $(R, V_\alpha \setminus R) := \theta_i$ with $t \in R$ `// just nomenclature`
    if θ_i *has* $\{v_i, b, d\}$ *on same side* **then** `// check attribute cav`
      $cav(i) \leftarrow$ TRUE `// cav for` θ_i
      $\theta_i \leftarrow (R \setminus C_i, (V_\alpha \setminus R) \cup C_i)$ `// by Lemma 6 for cav`
      $D(v_i) \leftarrow$ `checkShadows`(θ_i)
      $\theta_i \leftarrow$ `adjustShadows`$(\theta_i, D(v_i))$
    else `// then cat holds`
      $\theta_i \leftarrow ((V \setminus R) \cap C_i, V \setminus ((V \setminus R) \cap C_i))$ `// by Lemma 6 for cat`
  Add θ_i to Θ_{ten} `// pointed at by` v_i
if $\exists\, v_i \in \{v_b, v_d\}$: $cav(i) \wedge [\theta_i$ *separates* v_j, t; $v_i \neq v_j \in \{v_b, v_d\}]$ **then** `// Scenario (a)`
  set $(R, V_\alpha \setminus R) := \theta_i$ `// just nomenclature`
  if $cav(j) \wedge [\theta_j$ *separates* $v_i, t]$ **then** `// the other cut is feasible, too`
    wlog. $|D(v_i)| \leq |D(v_j)|$, so choose v_i for reshaping `// choose the better cut`
  $\theta_i \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$ `// by Scenario (a)`
  $W_{\text{ten}} \leftarrow \{v_i\}$, $\Theta_{\text{ten}} \leftarrow \{\theta_i\}$
else if $cav(b) \wedge cav(d)$ **then** `// Scenario (b)`
  set $(R_b, V_\alpha \setminus R_b) := \theta_b$, set $(R_d, V_\alpha \setminus R_d) := \theta_d$ `// just nomenclature`
  wlog. $|D(v_d)| \leq |D(v_b)|$, so choose v_b for reshaping `// choose the better cut`
  $\theta_b \leftarrow (R_d \cap C_b, V_\alpha \setminus (R_d \cap C_b))$ `// by Scenario (b)`
  $W_{\text{ten}} \leftarrow \{v_b, v_d\}$, $\Theta_{\text{ten}} \leftarrow \{\theta_b, \theta_d\}$
Resolve all crossings in Θ_{ten} by pseudo-contraction `// necessary for Scenario (c(ii))`
Add all vertices in W_{ten} to $W(G^\oplus)$, all cuts in Θ_{ten} to $\Theta(G^\oplus)$
Isolate the sink t from all remaining unclustered vertices `// necessary for (c(i/ii))`
return $W(G^\oplus), \Theta(G^\oplus)$

Procedure `checkShadows`(θ)

forall $v_j \in \{v_1, \ldots, v_z\}$ **do**
  if θ *separates* v_j *and* t **then** delete θ_j from $\Theta(G^\oplus)$, move v_j from $W(G^\oplus)$ to D
return D

Procedure `adjustShadows`(θ, D)

set $(R, V_\alpha \setminus R) := \theta$, $t \in R$ `// just nomenclature`
forall $v_j \in D$ **do** $\theta \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$ `// by pseudo-contraction`
forall $v_j \in \{v_1, \ldots, v_z\} \setminus D$ **do** $\theta \leftarrow (R \cup C_j, (V_\alpha \setminus R) \setminus C_j)$ `// by pseudo-contraction`
return θ

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 $\mathcal{C}$ 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 v_i-t-cut for a vertex $v_i \in C_i \in \mathcal{C}$ which has at least weight 2α. Thus, $v_i \neq d$. But the v_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.

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 cut-vertex candidates 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 still can state the following lemma:

Lemma 8. *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, Algorithm 4 returns $\mathcal{C}(G)$. Furthermore, in line 14, for each considered vertex v let Algorithm 6 use that cut among all minimum v-t-cuts which has the fewest number of vertices on the side containing v, i.e., has the smallest side for v. For an intra-cluster deletion Algorithm 6 then returns a clustering $\mathcal{C}(G^{\ominus}) \supseteq \mathcal{C}(G) \setminus C_{b/d}$, i.e., only $C_{b/d}$ might become fragmented.*

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 case of intra-cluster deletion. However, since most minimum u-v-cuts were unique in our instance, these are more often than not satisfied implicitly. The proof for both cases relies on the fact that any output clustering differing in cluster C_i requires at least one minimum v_i-t-cut ($v_i \in C_i$) to separate b, d, invalidating $\mathcal{C}(G)$.

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 v_i-t-cut for at least one

cut-vertex $v_i \in \{v_1, \ldots, v_z\}$. 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 $G_\alpha^{\ominus}$. However, each cheaper minimum v_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 $G_\alpha^{\ominus}$.

Considering intra-cluster deletion (Algorithm 6), all the above arguments apply to the clusters in $\mathcal{C}(G) \setminus \{C_{b/d}\}$. Thus, these clusters are again found. Furthermore, a new minimum $v_{b,d}$-t-cut calculated by Algorithm 6 always saves the treetop of $\{v_{b,d}, t\}$ and thus does not shadow any cluster in $\mathcal{C}(G) \setminus \{C_{b/d}\}$. We show now that these clusters also do not get shadowed by any new minimum v-t-cut while isolating t from a remaining "wild" set of vertices formerly contained in $C_{b/d}$ (see line 14 in Algorithm 6): As $C_{b/d}$ is also valid for $G^{\ominus}$, there must exist a vertex $u \in C_{b,d}$ that serves as a witness in $G_\alpha^{\ominus}$ for the cut $\theta_{b,d}$ defining $C_{b,d}$. This is, there exists a GH for $G_\alpha^{\ominus}$ such that $V_\alpha \setminus C_{b,d}$ is pseudo-contracted by $\theta_{b,d}$ from the view of $C_{b,d}$, and thus, this GH yields a minimum-cut tree $T(G_\alpha^{\ominus})$ in which the represented minimum v-t-cut is nested in $C_{b/d}$ and does not shadow any previous cluster. The v-side of this minimum v-t-cut contains the v-side of the "smallest" minimum v-t-cut as calculated by Algorithm 6 by assumption. This is, the minimum v-t-cut in $G_\alpha^{\ominus}$ found by Algorithm 6 does not shadow any cluster in $\mathcal{C}(G) \setminus \{C_{b/d}\}$. □

Considering the remaining cases, intra-cluster insertion obviously retains a valid previous clustering; for inter-cluster insertion an assertion akin to that used for intra-cluster deletion can be made (regarding our experiments, the comments from Lemma 8 carry over):

Lemma 9. *Let $\mathcal{C}(G)$ fulfill the* invariant *for $G^{\oplus}$, i.e., let the old clustering be valid for $G^{\oplus}$. Furthermore, in line 5 as well as in line 30 let Algorithm 7 use the cut with the smallest side for v under all minimum v-t-cuts for each considered vertex v. For an inter-cluster insertion Algorithm 7 then returns a clustering $\mathcal{C}(G^{\oplus}) \supseteq \mathcal{C}(G) \setminus \{C_b, C_d\}$, where C_b, C_d are either retained or fragmented individually.*

Proof. Considering inter-cluster insertion (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, these clusters are neither shadowed by any of the two checked cuts θ_b, θ_d, as we assume "smallest" cuts in line 5, nor by new minimum v-t-cuts possibly calculated by Algorithm 7 while isolating t from a remaining "wild" set of vertices formerly contained in $C_b \cup C_d$ (line 30). □

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 (*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.

	best case	worst case	old clustering still valid		
			lower bound	upper bound	guaran. smooth
Inter-Del	$\lvert\mathcal{C}(G^{\ominus})\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	$\lvert\mathcal{C}(G)\rvert - 2$	Yes
Intra-Del	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)
Inter-Ins	2	$\lvert C_b\rvert + \lvert C_d\rvert$	2	$\lvert C_b\rvert + \lvert C_d\rvert$	No (2)
Intra-Ins	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 v_b and v_d (compare

to Figure 10). 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 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-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., v_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 v_b and v_d, 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. The row for Intra-Cluster Edge Insertion is obvious.

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). 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

	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.

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. [3], 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

	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 decomposed into different scenarios.

the observed savings are not induced by trivial clusterings. Thus, comparing dynamic and static flow computations is warrantable: For the 61 870 proper steps, static computation needed 3 300 413 maximum flows, and our dynamic update needed 736 826, saving more than 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.

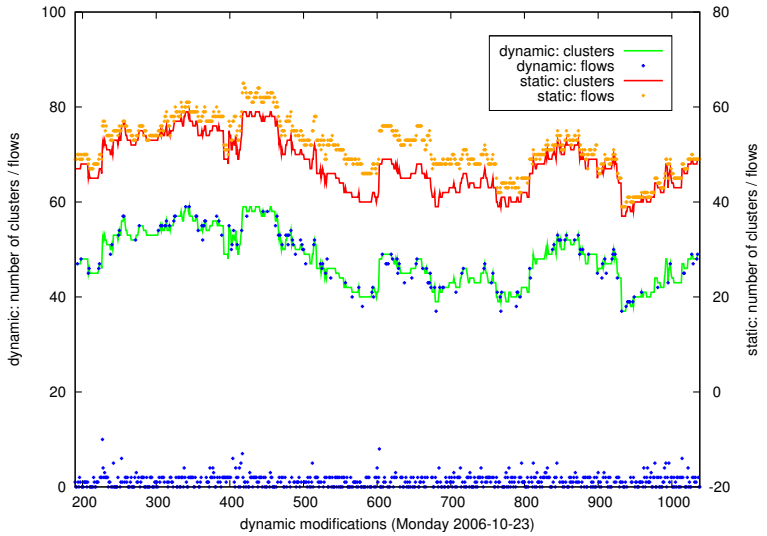


Fig. 14. Numbers of clusters and flows regarding consecutive clusterings (the two y-axes have a different offset for better readability).

This running time is also visible in Figure 14, 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. [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 [15, 14]. 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. [3].

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.

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. [3] 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.

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. S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, 2009.
5. 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.
6. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.
7. 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.
8. R. Görke, T. Hartmann, and D. Wagner. Dynamic Graph Clustering Using Minimum-Cut Trees. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures, 11th International Workshop*, volume 5664 of *Lecture Notes in Computer Science*. Springer, August 2009.
9. 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.
10. D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.
11. T. Hartmann. Clustering Dynamic Graphs with Guaranteed Quality. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, October 2008. Diplomarbeit Informatik.
12. 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.
13. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004.
14. 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.
15. 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.
16. S. E. Schaeffer. Graph Clustering. *Computer Science Review*, 1(1):27–64, August 2007.