Dynamic Graph Clustering Using Minimum-Cut Trees*

Robert Görke, Tanja Hartmann, and Dorothea Wagner

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

Abstract. Algorithms or target functions for graph clustering rarely admit quality guarantees or optimal results in general. Based on properties of minimum-cut trees, a clustering algorithm by Flake et al. does however yield such a provable guarantee. We show that the structure of minimum-*s*-*t*-cuts in a graph allows for an efficient dynamic update of minimum-cut trees, and present a dynamic graph clustering algorithm that maintains a clustering fulfilling this quality 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., [2,1] and references therein. Inspired by the work of Kannan et al. [8], Flake et al. [3] recently presented a clustering algorithm which does guarantee a very reasonable bottleneck-property. Their elegant approach employs minimum-cut trees, pioneered by Gomory and Hu [4], 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 [9], however, we found it to be erroneous beyond straightforward correction. We are not aware of any other dynamic graph-clustering algorithms in the literature.

In this work we develop the first correct algorithm that efficiently and dynamically maintains a clustering for a changing graph as found by the method of Flake et al. [3], allowing arbitrary atomic changes in the graph, and keeping consecutive clusterings similar (a notion we call *temporal smoothness*). Our algorithms build upon partially updating an intermediate minimum-cut tree of a graph in the spirit of Gusfield's [6] simplification of the Gomory-Hu algorithm [4]. We show that, with only slight modifications, our techniques can update entire min-cut trees. We corroborate our theoretical

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

F. Dehne et al. (Eds.): WADS 2009, LNCS 5664, pp. 339–350, 2009.

results on clustering by experimentally evaluating the performance of our procedures compared to the static algorithm on a real-world dynamic graph.

We briefly give our notational conventions and one fundamental lemma in Sec. 1. Then, in Sec. 2, we revisit some results from [4,6,3], convey them to a dynamic scenario, and derive our central results. In Section 3 we give actual update algorithms, which we analyse in Sec. 4, concluding in Sec. 5. We do not include any proof in this extended abstract, without further notice these can all be found in the full version [5].

Preliminaries and Notation. Throughout this work we consider an undirected, weighted graph $G = (V, E, c)$ with vertices V, edges E and a non-negative edge weight function c, writing $c(u,v)$ as a shorthand for $c(\{u,v\})$ with $u \sim v$, i.e., $\{u,v\} \in E$. We reserve the term *node* (or *super-node*) for compound vertices of abstracted graphs, which may contain several basic vertices; however, we identify singleton nodes with the contained vertex without further notice. Dynamic modifications of G will solely concern edges; the reason for this is, that vertex insertions and deletions are trivial for a disconnected vertex. Thus, a modification of G always involves edge $\{b,d\}$, with $c(b,d) = \Delta$, yielding $G_\oplus$ if $\{b,d\}$ is newly inserted into G, and $G_\ominus$ if it is deleted from G. For simplicity we will not handle changes to edge weights, since this can be done almost exactly as deletions and additions. Bridges in G require special treatment when deleted or inserted. However, since they are both simple to detect and to deal with, we ignore them by assuming the dynamic graph to stay connected at all times.

The *minimum-cut tree* $T(G) = (V, E_T, c_T)$ of G is a tree on V and represents for any node pair $\{u,v\} \in \binom{V}{2}$ a minimum-u-v-cut $\theta_{u,v}$ in G by the cheapest edge on the u-v-path in $T(G)$. For $b,d \in V$ we always call this path γ (as a set of edges). An edge $e_T = \{u,v\}$ of T induces the cut $\theta_{u,v}$ in G, sometimes denoted θ_v if the context identifies u. We sometimes identify e_T with the cut it induces in G.

A *contraction* of G by $N \subseteq V$ means replacing set N by a single super-node η, and leaving η adjacent to all former adjacencies u of vertices of N, with edge weight equal to the sum of all former edges between N and u. Analogously we can contract by a set $M \subseteq E$. A *clustering* $\mathscr{C}(G)$ of G is a partition of V into *clusters* C_i, usually conforming to the paradigm of *intra-cluster density and inter-cluster sparsity*. We start by giving some fundamental insights, which we will rely on in the following.

Lemma 1. *Let $e = \{u,v\} \in E_T$ be an edge in $T(G)$.*
Consider $G_\oplus$: If $e \notin \gamma$ then e is still a min-u-v-cut with weight $c(\theta_e)$. If $e \in \gamma$ then its cut-weight is $c(\theta_e) + \Delta$, it stays a min-u-v-cut iff $\forall$u-v-cuts θ' in G that do not separate *b,d: $c(\theta') \geq c(\theta_e) + \Delta$.*

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

2 Theory

The Static Algorithm. Finding communities in the world wide web or in citation networks are but example applications of graph clustering techniques. In [3] Flake et al. propose and evaluate an algorithm which clusters such instances in a way that yields a

certain guarantee on the quality of the clusters. The authors base their quality measure on the expansion of a cut $(S,\bar{S})$ due to Kannan et al. [8]:

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

Inspired by a bicriterial approach for good clusterings by Kannan et al. [8], which bases on the related measure conductance[1], Flake et al. [3] design a graph clustering algorithm that, given parameter α, asserts:[2]

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

These quality guarantees—simply called quality in the following—are due to special properties of min-cut trees, which are used by the clustering algorithm, as given in Alg. 1 (comp. [3], we omit a textual description). 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 Eq. (2) and for a number of other interesting properties, we refer the reader to [3]. In the following we will use the definition of $G_\alpha = (V_\alpha, E_\alpha, c_\alpha)$, denoting by $G_\alpha^{\ominus}$ and $G_\alpha^{\oplus}$ the corresponding augmented *and* modified graphs. For now, however, general $G_{\oplus(\ominus)}$ are considered.

Algorithm 1. CUT-CLUSTERING

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

A Dynamic Attempt. Saha and Mitra [9] published an algorithm that aims at the same goal as our work. Unfortunately, we discovered a methodical error in this work. Roughly speaking, the authors implicitly (and erroneously) assume an equivalence between quality and the invariant, e.g., in their *CASE2* of the procedure for dynamic inter-addition: their proof of correctness requires the invariant but guarantees only quality; there is no immediate remedy for this error. We scrutinize these issues alongside counter-examples and correct parts in the full versions [5,7]. A full description is beyond the scope of this extended abstract, but we sketch out a counter-example in Fig. 1.

Minimum-Cut Trees and the Gomory-Hu Algorithm. Although we heavily build upon the construction of a min-cut tree as proposed by Gomory and Hu [4] we cannot accomodate a detailed description of their algorithm and refer the reader to their work. The algorithm builds the min-cut tree of a graph by iteratively finding min-u-v-cuts for vertices that have not yet been separated by a previous min-cut. An *intermediate* min-cut tree $T_*(G) = (V_*, E_*, c_*)$ (or simply T_* if the context is clear) is initialized as an isolated, edgeless super-node containing all original nodes. Then, until no node S of

[1] conductance is similar to expansion but normalizes cuts by total incident edge weight.
[2] The disjoint union $A\cup B$ with $A\cap B=\emptyset$ is denoted by $A\dot\cup B$.

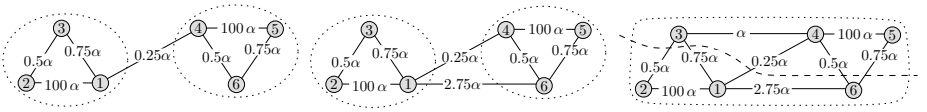
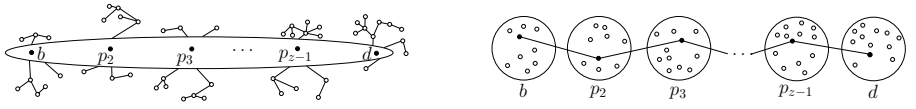


Fig. 1. Example showing error in [9]. Left: initial instance, clustered via static algorithm; middle: updated clustering after one edge-addition, preserving quality but *not* the invariant; right: update after second edge-addition, quality is violated, dashed cut weighs $11/4\alpha < \alpha \min\{|P|, |V \setminus |P\}$.



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

Fig. 2. Sketches of intermediate min-cut trees $T_\circ$; for $G_\oplus$ (a) we contract γ to a node, and for $G_\ominus$ (b) we contract each connected component induced by $E_T \setminus \gamma$

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 before cutting, which yields G_S—a notation we will continue using in the following. The split of S into (S_u, S_v) is then defined by a min-u-v-cut in G_S. Afterwards, N_j is reconnected, again by S_j, to either S_u or S_v depending on which side of the cut η_j, containing S_j, ended up. Note that this cut in G_S can be proven to induce a min-u-v-cut in G. An *execution* $\mathrm{GH} = (G, F, K)$ of GOMORY-HU is characterized by graph G, sequence F of $n-1$ *step pairs* of nodes and sequence K of *split cuts*. Pair $\{u, v\} \subseteq V$ is a *cut pair* of edge e of cut-tree T if θ_e is a min-u-v-cut in G.

Theorem 1. *Consider a set $M \subseteq E_T$ and let $T_\circ(G) = (V_\circ, M, c_\circ)$ be $T(G)$ with $E_T \setminus M$ contracted. Let f and f' be sequences of the elements of M and $E_T \setminus M$, respectively, and k and k' the corresponding sequences of edge-induced cuts of G.* $\mathrm{GH} = (G, f' \cdot f, k' \cdot k)$[3] *has $T_\circ(G)$ as intermediate min-cut tree (namely after f).*

In the following we will denote by $T_\circ$ an intermediate min-cut tree which serves as a starting point, and by T_* a working version. This theorem states that if for some reason we can only be sure about a subset of the edges of a min-cut tree, we can contract all other edges to super-nodes and consider the resulting tree $T_\circ$ as the correct intermediate result of some GH, which can then be continued. One such reason could be a dynamic change in G, such as the insertion of an edge, which by Lem. 1 maintains a subset of the old min-cuts. This already hints at an idea for an effort-saving update of min-cut trees.

Using Arbitrary Minimum Cuts in G. Gusfield [6] presented an algorithm for finding min-cut trees which avoids complicated contraction operations. In essence he provided rules for adjusting iteratively found min-u-v-cuts in G (instead of in G_S) that potentially cross, such that they are consistent with the Gomory-Hu procedure and thus non-crossing, but still minimal. We need to review and generalize some of these ideas as to

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

fit our setting. The following lemma essentially tells us, that at any time in GOMORY-HU, for any edge e of $T_\circ$ there exists a cut pair of e in the two nodes incident to e.

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

In the latter case of Lem. 2, we say that pair $\{x,y\}$ gets *hidden*, and, in the view of vertex y, its former counterpart x gets *shadowed* by u (or by S_u). It is not hard to see that during GOMORY-HU, step pairs remain cut pairs, but cut pairs need not stem from step pairs. However, each edge in T has at least one cut pair in the incident nodes. We define the *nearest cut pair* of an edge in T_* as follows: As long as a step pair $\{x,y\}$ is in adjacent nodes S_x,S_y, it is the nearest cut pair of edge $\{S_x,S_y\}$; if a nearest cut pair gets hidden in T_* by a step of GOMORY-HU, as described in Lem. 2 if $x \in S_{xv}$, the nearest cut pair of the reconnected edge $\{S_y,S_{xu}\}$ becomes $\{u,y\}$ (which are in the adjacent nodes S_y,S_{xu}). The following theorem basically states how to iteratively find min-cuts as GOMORY-HU, without the necessity to operate on a contracted graph.

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

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

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

Given a cut as by Theo. 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; when avoiding contraction, this criterion is not available. For this purpose, Gusfield iteratively defines *representatives* $r(S_i) \in V$ of nodes S_i of T_*, and states his Theorem 3: For $u,v \in S$ let *any* min-u-v-cut $(U,V \setminus U)$, $u \in U$, in G split node S into $S_u \ni u$ and $S_v \ni v$ and let $(U_S,V \setminus U_S)$ be this cut adjusted via Lem. 3 and Theo. 2; then a neighboring subtree N_j of S, formerly connected by edge $\{S,S_j\}$, lies in U_S iff $r(S_j) \in U$. We do not have such representatives and thus need to adapt this, namely using nearest cut pairs as representatives:

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

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

[5] This Lemma alongside Lemma 3, Theo. 2 and a simpler version of our Theo. 3 have been discussed in [6] and the lemma also in [4].

Finding and Shaping Minimum Cuts in the Dynamic Scenario. In this section we let graph G change, i.e., we consider the addition of an edge $\{b,d\}$ or its deletion, yielding $G_\oplus$ or $G_\ominus$. First of all we define valid representatives of the nodes on $T_\circ$(omitting proofs). By Lem. 1 and Theo. 1, given an edge addition, $T_\circ$ consists of a single super-node and many singletons, and given edge deletion, $T_\circ$ consists of a path of super-nodes; for examples see Fig. 2.

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

Following Theo. 1, we define the set M of "good" edges of the old tree $T(G)$, i.e., edges that stay valid due to Lem. 1, as $M := E_T \setminus \gamma$ for the insertion of $\{b,d\}$ and to $M := \gamma$ for the deletion. Let $T_\circ(G_{\oplus(\ominus)})$ be $T(G)$ contracted by M. As above, let f be any sequence of the edges in M and k the corresponding cuts in G. We now state a specific variant of the setting in Theo. 1 which is the basis of our updating algorithms, founded on $T_\circ$s as in Fig. 2, using arbitrary cuts in $G_{\oplus(\ominus)}$ instead of actual contractions.

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

Cuts That Can Stay. The non-crossing nature of min-u-v-cuts allow for more effort-saving and temporal smoothness. There are several circumstances which imply that a previous cut is still valid after a graph modification, making its recomputation unnecessary. The following lemma gives one such assertion (we omit a few others here), based on the definition of a *treetop* and of *wood* (comp. Fig. 3): Consider edge $e = \{u,v\}$ off γ, and cut $\theta = (U, V \setminus U)$ in G induced by e in $T(G)$ with γ contained in U. In $G_\ominus(S)$, $S \cap (V \setminus U)$ is called the *treetop* $\Uparrow_e$, and $S \cap U$ the *wood* $\sharp_e$ of e. The subtrees of S are N_b and N_d, containing b and d, respectively.

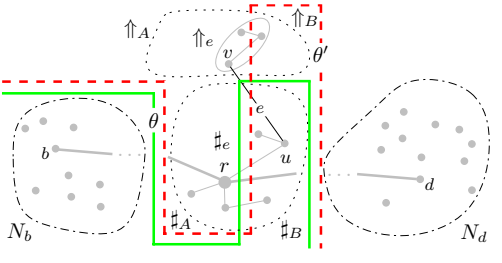
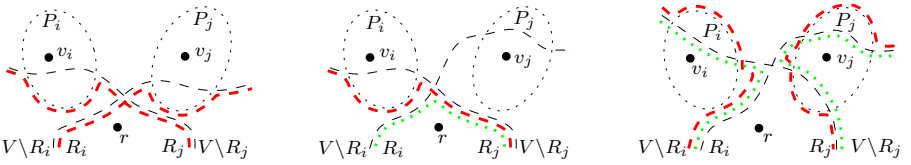


Fig. 3. Special parts of $G_\ominus$: γ (fat) connects b and d, with r on it; wood $\sharp_e$ and treetop $\Uparrow_e$ (dotted) of edge e, both cut by θ' (dashed), adjusted to θ (solid) by Lem. 6. Both $\sharp_e$ and $\Uparrow_e$ are part of some node S, with representative r, outside subtrees of r are N_b and N_d (dash-dotted). Compare to Fig. 2(b).

Lemma 5. *In $G_\ominus$, let $(U, V \setminus U)$ be a min-u-v-cut not separating $\{b,d\}$, with γ in $V \setminus U$. Then, a cut induced by edge $\{g,h\}$ of the old $T(G)$, with $g,h \in U$, remains a min separating cut for all its previous cut pairs within U in $G_\ominus$, and a min g-h-cut in particular.*



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

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

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

Fig. 4. Three cases concerning the positions of θ'_i and θ'_j, and their adjustments

As a corollary from Lem. 5 we get that in $T(G_\ominus)$ the entire treetops of reconfirmed edges of $T(G)$ are also reconfirmed. This saves effort and encourages smoothness; however new cuts can also be urged to behave well, as follows.

The Shape of New Cuts. In contrast to the above lemmas, during a Gomory-Hu execution for $G_\ominus$, we might find an edge $\{u,v\}$ of the old $T(G)$ that is *not* reconfirmed by a computation in $G_\ominus$, but a new, cheaper min-u-v-cut $\theta' = (U, V(S)\setminus U)$ is found. For such a new cut we can still make some guarantees on its shape to resemble its "predecessor": Lemma 6 tells us, that for any such min-u-v-cut θ', there is a min-u-v-cut $\theta = (U \setminus \Uparrow_e, (V(S)\setminus U) \cup \Uparrow_e)$ in $G_\ominus$ that (a) does not split $\Uparrow_e$, (b) but splits $V\setminus\Uparrow_e$ exactly as θ' does. Figure 3 illustrates such cuts θ (solid) and θ' (dashed).

Lemma 6. *Given $e = \{u,v\}$ within S (off γ) in $G_\ominus(S)$. Let $(\Uparrow_A, \Uparrow_B)$ be a cut of $\Uparrow_e$ with $v \in \Uparrow_A$. Then $c_\ominus(N_b \cup \Uparrow_e, N_d \cup \#_e) \leq c_\ominus(N_b \cup \Uparrow_A, N_d \cup \#_e \cup \Uparrow_B)$. Exchanging N_b and N_d is analogous. This result can be generalized in that both considered cuts are also allowed to cut the wood $\#_e$ in some arbitrary but fixed way.*

While this lemma can be applied in order to retain treetops, even if new cuts are found, we now take a look at how new, cheap cuts can affect the treetops of *other* edges. In fact a similar treetop-conserving result can be stated. Let G' denote an undirected, weighted graph and $\{r, v_1, \ldots, v_z\}$ a set of designated vertices in G'. Let $\Pi := \{P_1, \ldots, P_z\}$ be a partition of $V \setminus r$ such that $v_j \in P_j$. We now assume the following partition-property to hold: For each v_j it holds that for any v_j-r-cut $\theta'_j := (R_j, V \setminus R_j)$ (with $r \in R_j$), the cut $\theta_j := (R_j \setminus P_j, (V \setminus R_j) \cup P_j)$ is of at most the same weight. The crucial observation is, that Lem. 6 implies this partition-property for $r(S)$ and its neighbors in $T(G)$ that lie inside S of $T_\circ$ in $G_\ominus$. Treetops thus are the sets P_j. However, we keep things general for now: Consider a min-v_i-r-cut $\theta'_i := (R_i, V \setminus R_i)$, with $r \in R_i$, that does not split P_i and an analog min-v_j-r-cut θ'_j (by the partition-property they exist). We distinguish three cases, given in Fig. 4, which yield the following possibilities of reshaping min-cuts:

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

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

Case (c): As in case (b), by Lem. 3 the cut $\theta_{\text{new}(i)} := ((V \setminus R_j) \cup R_i, (V \setminus R_i) \cap R_j)$ (green dotted) is a min-v_i-r-cut, and the cut $\theta_{\text{new}(j)} := ((V \setminus R_i) \cup R_j, (V \setminus R_j) \cap R_i)$

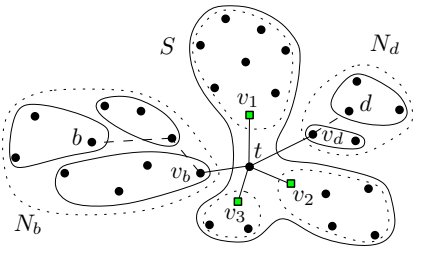


Fig. 5. $T_\circ(G_\alpha^\ominus)$ for an inter-del.; t's neighbors off γ need inspection. Cuts of v_b and v_d are correct, but might get shadowed.

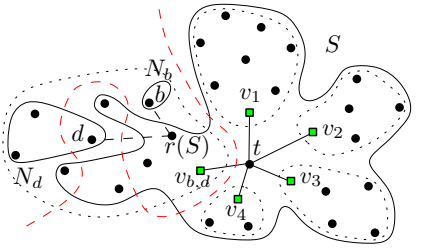


Fig. 6. $T_\circ(G_\alpha^\ominus)$ for an intra-del.; edge $\{v_{b,d},t\}$ defines a treetop (t's side). The dashed cut could be added to Θ.

(green dotted) is a min-v_j-r-cut. These cuts do not cross. So as v_i and v_j both satisfy the partition-property, cut $\theta_i := (((V \setminus R_j) \cup R_i) \setminus P_i, ((V \setminus R_i) \cap R_j) \cup P_i)$ and $\theta_j := (((V \setminus R_i) \cup R_j) \setminus P_j, ((V \setminus R_j) \cap R_i) \cup P_j)$ (both red dashed) are non-crossing min separating cuts, which neither split P_i nor P_j.

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

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

3 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 min-cut tree clustering, such that the invariant is maintained and thus also the quality. By concept, we merely need to know all vertices of $T(G)$ adjacent to t; we call this set $W = \{v_1, \ldots, v_z\} \cup \{v_b, v_d\}$, with $\{v_b, v_d\}$ being the particular vertex/vertices on the path from t to b and d, respectively. We call the corresponding set of non-crossing min-v_i-t-cuts that isolate t, Θ. We will thus focus on dynamically maintaining only this information, and sketch out how to unfold the rest of the min-cut tree. From Lem. 4, for a given edge insertion or deletion, we know $T_\circ$, and we know in which node of $T_\circ$ to find t, this is the node we need to examine. We now give algorithms for the deletion and the insertion of an edge running inside or between clusters.

Algorithm 2. INTER-CLUSTER EDGE DELETION

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

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

Algorithm 3. CHECK CUT-VERTICES

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

```
1  while L(t) has next element v_i do
2      θ_i ← first min-v_i-t-cut given by FLOWALGO(v_i,t)          // small side for v_i
3      if c_α^⊖(θ_i) = c_α(θ_i^old) then Add θ_i^old to l(t)       // retain old cut?
4      else
5          Add θ_i to l(t)                                        // pointed at by v_i
6          while L(t) has next element v_j ≠ v_i do               // test vs. other new cuts
7              if θ_i separates v_j and t then                    // v_j shadowed by Lem. 3
8                  Move v_j from L(t) to D(v_i)
9                  if l(t) ∋ θ_j, pointed at by v_j then Delete θ_j from l(t)
10 while L(t) has next element v_i do          // make new cuts cluster-preserving
11     set (R, V_α \ R) := θ_i with t ∈ R for θ_i ∈ l(t) pointed at by v_i
12     θ_i ← (R \ C_i, (V_α \ R) ∪ C_i)              // by partition-property (Lem. 6)
13     forall v_j ∈ D(v_i) do θ_i ← (R \ C_j, (V_α \ R) ∪ C_j)      // Cases (a) and (b)
14     forall v_j ≠ v_i in L(t) do θ_i ← (R ∪ C_j, (V_α \ R) \ C_j)          // Case (c)
15 Add all vertices in L(t) to W(G_⊖), and their cuts from l(t) to Θ(G_⊖)
```

Edge Deletion. Our first algorithm handles inter-cluster deletion (Alg. 2). Just like its three counterparts, it takes as an input the old graph G and its sets $W(G)$ and $\Theta(G)$ (not the entire min-cut tree $T(G_\alpha)$), furthermore it takes the changed graph, augmented by t, $G_\alpha^\ominus$, the deleted edge $\{b,d\}$ and its weight Δ. Recall that an inter-cluster deletion yields t on γ, and thus, $T_\circ(G_\alpha)$ contains edges $\{v_b,t\}$ and $\{v_d,t\}$ cutting off the subtrees N_b and N_d of t by cuts θ_b, θ_d, as shown in Fig. 5. All clusters contained in node $S \ni t$ need to be changed or reconfirmed. To this end Algorithm 2 lists all cut vertices in S, $v_1, \ldots, v_z$, into $L(t)$, and initializes their shadows $D(v_i) = \emptyset$. The known cuts θ_b, θ_d are already added to the final list, as are v_b, v_d (line 3). Then the core algorithm, CHECK CUT-VERTICES is called, which—roughly speaking—performs those GH-steps that are necessary to isolate t, using (most of) the above lemmas derived.

First of all, note that if $|\mathscr{C}| = 2$ ($\mathscr{C} = \{N_b, N_d\}$ and $S = \{t\}$) then $L(t) = \emptyset$ and Alg. 2 lets CHECK CUT-VERTICES (Alg. 3) simply return the input cuts and terminates. Otherwise, it iterates the set of former cut-vertices $L(t)$ once, thereby possibly shortening it. We start by computing a new min-v_i-t-cut for v_i. We do this with a max-v_i-t-flow computation, which is known to yield *all* min-v_i-t-cuts, taking the *first* cut found by a breadth-first search from v_i (lines 2). This way we find a cut which minimally interferes with other treetops, thus encouraging temporal smoothness. If the new cut is non-cheaper, we use the old one instead, and add it to the tentative list of cuts $l(t)$ (lines 3-3). Otherwise we store the new, cheaper cut θ_i, and examine it for later adjustment. For any candidate v_j still in $L(t)$ that is separated from t by θ_i, Case (a) or (b) applies (line 7). Thus, v_j will be in the shadow of v_i, and not a cut-vertex (line 8). In case v_j has already been processed, its cut is removed from $l(t)$. Once all cut-vertex candidates are processed, each one either induces the same cut as before, is new and shadows other former cut-vertices or is itself shadowed by another cut-vertex. Now that we have

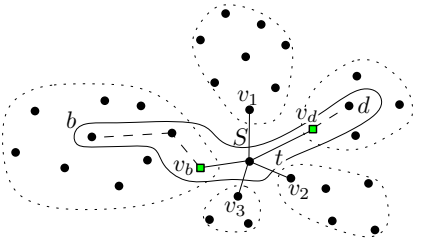


Fig. 7. $T_\circ(G_\alpha^\oplus)$ for an inter-cluster addition. At least v_b and v_d need inspection.

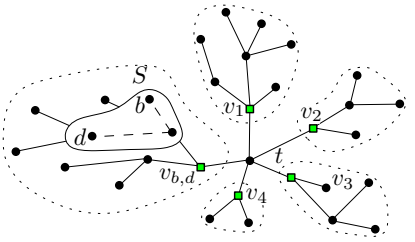


Fig. 8. $T_\circ(G_\alpha^\oplus)$ for an intra-cluster addition. All relevant min-v-t-cuts persist.

collected these relations, we actually apply Cases (a,b,c) and Lem. 6 in lines 10-14. Note that for retained, old cuts, no adjustment is actually performed here. Finally, all non-shadowed cut-vertices alongside their adjusted cuts are added to the final lists.

Unfortunately we must completely omit *intra*-cluster edge deletion here, please find it detailed in the full versions [5,7]. In a way roughly analogue to the former case we again call CHECK CUT-VERTICES and after that merely have to clean up a potentially "wild" area of leftover vertices from cluster $C_{b,d}$.

Edge Addition. The good news for handling $G_\oplus$ is, that an algorithm INTRA-CLUSTER EDGE ADDITION only needs to return the old clustering: By Lem. 1 and Theo. 1, in $T_\circ$, only path γ is contracted. But since γ lies within a cluster, the cuts in G_α, defining the old clustering, all remain valid in $G_\alpha^\oplus$, as depicted in Fig. 8 with dotted clusters and affected node S. By contrast, adding an edge between clusters is more demanding. Again, γ is contracted, see region S in Fig. 7; however, t lies on γ in this case. A sketch of what needs to be done resembles the above algorithms: We compute new min-v_b-t- and min-v_d-t-cuts (or possibly only one, if it immediately shadows the other), and keep the old v_i-t-cuts. Then—proceeding as usual—we note which cuts shadow which others and reconnect nodes by Theo. 3.

Updating Entire Min-Cut Trees. An interesting topic on its own right and more fundamental than clustering, is the dynamic maintenance of min-cut trees. In fact the above clustering algorithms are surprisingly close to methods that update min-cut trees. Since all the results from Sec. 2 still apply, we only need to unfold treetops or subtrees of t—which we gladly accept as super-nodes for the purpose of clustering—and take care to correctly reconnect subtrees. This includes, that merely examining the neighbors of t does not suffice, we must iterate through all nodes S_i of $T_\circ$. For the sake of brevity we must omit further details on such algorithms and refer the reader to the full version [5].

4 Performance of the Algorithm

Temporal Smoothness. Our secondary criterion—which we left unformalized—to preserve as much of the previous clustering as possible, in parts synergizes with effort-saving, an observation foremost reflected in the usage of $T_\circ$. Lemmas 5 and 6, using *first cuts* and Observation 1 nicely enforce temporal smoothness. However, in some

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

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

cases we must cut back on this issue, e.g., when we examine which other cut-vertex candidates are shadowed by another one, as in line 7 of Alg. 3. Here it entails many more cut-computations and a combinatorially non-trivial problem to find an ordering of $L(t)$ to optimally preserve old clusters. Still we can state the following lemma:

Lemma 7. *Let $\mathscr{C}(G)$ fulfill the* invariant *for $G_\ominus$, i.e., let the old clustering be valid for $G_\ominus$. In the case of an inter-cluster deletion, Alg 2 returns $\mathscr{C}(G)$. For an intra-cluster deletion we return a clustering $\mathscr{C}(G_\ominus) \supseteq \mathscr{C}(G) \setminus C_{b,d}$, i.e., only $C_{b,d}$ might become fragmented. Intra-cluster addition retains a valid old clusterings.*

Running Times. We express running times of our algorithms in terms of the number of max-flow computations, leaving open how these are done. A summary of tight bounds is given in Tab. 1 (for an in-depth discussion thereof see the full version). 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. Note that a computation from scratch entails a tight upper bound of $|V|$ max-flow computations for all four cases, in the worst case.

Further Speed-Up. For the sake of brevity we leave a few ideas and lemmas for effort-saving to the full version. One heuristic is to decreasingly order vertices in the list $L(t)$, e.g., in line 2 of Alg. 2; for their static algorithm Flake et al. [3] found that this effectively reduces the number of cuts necessary to compute before t is isolated. Since individual min-u-v-cuts are constantly required, another dimension of effort-saving lies in dynamically maintaining max-u-v-flows. We detail two approaches based on dynamic residual graphs in the full version.

Experiments. In this brief section, we very roughly describe some experiments we made with an implementation of the update algorithms described above, just for a first proof of concept. The instance we use is a network of e-mail communications within the Fakultät für Informatik at Universität Karlsruhe. Vertices represent members and edges correspond to e-mail contacts, weighted by the number of e-mails sent between two individuals during the last 72 hours. We process a queue of 12560 elementary modifications, 9000 of which are actual edge modifications, on the initial graph G ($|V| = 310, |E| = 450$). This queue represents about one week, starting on Saturday (21.10.06); a spam-attack lets the graph slightly grow/densify over the course.

We delete zero-weight edges and isolated nodes. Following the recommendations of Flake et al. [3] we choose $\alpha = 0.15$ for the initial graph, yielding 45 clusters. For the $9K$ proper steps, static computation needed $\approx 2M$ max-flows, and our dynamic update needed $\approx 2K$, saving more than 90% max-flows, such that in 96% of all modifications, the dynamic algorithm was quicker. Surprisingly, inter-additions had the greatest impact on effort-saving, followed by the trivial intra-additions. Out of the $9K$ operations, 49 of the inter-, and 222 of the intra-cluster deletions were the only ones, where the static algorithm was quicker. See the full versions [5,7] for details on these results.

5 Conclusion

We have proven a number of results on the nature of min-u-v-cuts in changing graphs, which allow for feasible update algorithms of a minimum-cut tree. In particular we have presented algorithms which efficiently update specific parts of such a tree and thus fully dynamically maintain a graph clustering based on minimum-cut trees, as defined by Flake et al. [3] for the static case, under arbitrary atomic changes. The striking feature of graph clusterings computed by this method is that they are guaranteed to yield a certain expansion—a bottleneck measure—within and between clusters, tunable by an input parameter α. As a secondary criterion for our updates we encourage temporal smoothness, i.e., changes to the clusterings are kept at a minimum, whenever possible. Furthermore, we disprove an earlier attempt to dynamize such clusterings [9]. Our experiments on real-world dynamic graphs affirm our theoretical results and show a significant practical speedup over the static algorithm of Flake et al. [3]. Future work on dynamic minimum-cut tree clusterings will include a systematic comparison to other dynamic clustering techniques and a method to dynamically adapt the parameter α.

References

1. Brandes, U., Delling, D., Gaertler, M., Görke, R., Höfer, M., Nikoloski, Z., Wagner, D.: On Modularity Clustering. IEEE TKDE 20(2), 172–188 (2008)
2. Brandes, U., Erlebach, T. (eds.): Network Analysis. LNCS, vol. 3418. Springer, Heidelberg (2005)
3. Flake, G.W., Tarjan, R.E., Tsioutsiouliklis, K.: Graph Clustering and Minimum Cut Trees. Internet Mathematics 1(4), 385–408 (2004)
4. Gomory, R.E., Hu, T.: Multi-terminal network flows. Journal of the Society for Industrial and Applied Mathematics 9(4), 551–570 (1961)
5. Görke, R., Hartmann, T., Wagner, D.: Dynamic Graph Clustering Using Minimum-Cut Trees. Technical report, Informatics, Universität Karlsruhe (2009)
6. Gusfield, D.: Very simple methods for all pairs network flow analysis. SIAM Journal on Computing 19(1), 143–155 (1990)
7. Hartmann, T.: Clustering Dynamic Graphs with Guaranteed Quality. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik (October 2008)
8. Kannan, R., Vempala, S., Vetta, A.: On Clusterings - Good, Bad and Spectral. In: Proc. of FOCS 2000, pp. 367–378 (2000)
9. Saha, B., Mitra, P.: Dynamic Algorithm for Graph Clustering Using Minimum Cut Tree. In: Proc. of the, SIAM Int. Conf. on Data Mining, pp. 581–586 (2007)