Karlsruhe Reports in Informatics 2011,25
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Fully-Dynamic Cut Tree Construction

Tanja Hartmann and Dorothea Wagner

2011

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

Fully-Dynamic Cut Tree Construction

Tanja Hartmann and Dorothea Wagner

Department of Informatics, Karlsruhe Institute of Technology (KIT)*
{t.hartmann,dorothea.wagner}@kit.edu

Abstract. A cut tree of an undirected weighted graph $G = (V, E)$ encodes a minimum s-t-cut for each vertex pair $\{s, t\} \subseteq V$, and thus solves the multiterminal network flow problem, which asks for the all-pairs maximum flow values in a network. On the other hand, a cut tree represents a set of $n - 1$ non-crossing and, in particular, linearly independent cuts, which constitute a minimum cut-basis of G and can be constructed by only $n - 1$ maximum flow computations. Hence, cut trees are resident in at least two fundamental fields of network analysis and graph theory, which emphasizes their importance for many applications. In this work we present the first fully-dynamic algorithm that efficiently maintains a cut tree for a changing graph. We give a guarantee for the number of saved cut computations compared to a static algorithm and prove that two consecutive trees strongly resemble in that their cut sets or equivalently minimum cut-bases heavily intersect.

1 Introduction

Cut trees were first introduced by Gomory and Hu [1] in 1961 in the field of multiterminal network flow analysis. Shortly afterwards, in 1964, Elmaghraby [3] already studied how the values of multiterminal flows change if the capacity of an edge in the network varies. Elmaghraby established the *sensitivity analysis of multiterminal flow networks*, which asks for the all-pairs maximum flow values (or all-pairs minimum cut values) in a network, regarding any possible capacity of the varying edge. According to Barth et al. [4] this can be answered by constructing two cut trees. In contrast, the *parametric maximum flow problem* considers a flow network with only two terminals s and t and with several parametric edge capacities. The goal is to give an actual maximum s-t-flow (or minimum s-t-cut) regarding all possible capacities of the parametric edges in order to choose the best flow with respect to some side constraints. Parametric maximum flows were studied, e.g., by Gallo et al. [5] and Scutellà [6].

However, in many applications we are neither interested in all-pairs values (sensitivity analysis) nor in one minimum s-t-cut (parametric flow) regarding *all possible* changes of varying edges in the network $G = (V, E)$. Instead we face a

* This work was partially supported by the DFG under grant WA 654/15-2 and by the Concept for the Future of Karlsruhe Institute of Technology within the German Excellence Initiative.

concrete capacity change on a concrete edge and need all-pairs minimum cuts regarding this single change. This is answered by *dynamic cut trees*, which thus bridge the two sides of sensitivity analysis and parametric maximum flows.

Contribution and Outline. In this work we develop the first algorithm that efficiently and dynamically maintains a cut tree for a changing graph, allowing arbitrary atomic changes and guaranteeing high temporal smoothness, i.e., a large intersection of the cut sets represented by two consecutive trees. To the best of our knowledge no fully-dynamic approach for updating cut trees exists. Coming from sensitivity analysis, Barth et al. [4] introduce a first approach for updating a cut tree after an edge capacity has increased and state the difficulty in the case of decreasing capacity. Giving a concrete cut tree construction that reuses cuts that are still valid after the change, they show that the path between the two vertices of the changing capacity is the only part that needs to be recomputed in order to construct a new cut tree.

In our work, we extend the approach of Barth et al. for increasing capacities by a guaranteed temporal smoothness. However, we even formulate a general condition for the reuse of given cuts in a cut tree construction such that cuts that are known to remain valid can also be exploited for decreasing capacities. We further describe how to detect such cuts, yielding an exact formula for the number of saved cut computations compared to a computation from scratch. Additionally, we provide techniques to determine new cuts that respect cuts that remain valid.

We give our notational conventions and two folklore insights in Sec. 2. In Sec. 3, we revisit the static cut tree algorithm by Gomory and Hu [1] and present Theorem 1, which allows for the reuse of cuts. In Sec. 4 we describe our new update algorithm proving its correctness and the guarantee of smoothness and cut savings, concluding in Sec. 5.

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 cost 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* for compound vertices of abstracted graphs, which may contain several basic vertices; however, we identify singleton nodes with the contained vertex without further notice. Due to space constraints we focus on dynamic modifications of G concerning only edges. We abandon a description of inserting and deleting disconnected vertices, which requires the handling of bridges. The latter retains the cut trees spanning the two sides of a bridge. Thus, we assume G to be connected; otherwise we work on each connected component independently and the results still apply. An edge modification of G always involves an 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. We write $G^{\oplus\ominus}$ as a shorthand for $G^{\oplus}$ *or* $G^{\ominus}$. Decreasing edge costs can be handled by the same method as deletions, the techniques for

edge insertions also apply for increasing costs. We denote by $c^{\oplus}$ and $c^{\ominus}$ the cost functions after inserting and deleting, respectively.

An edge $e_T = \{u, v\}$ of a tree $T(G) = (V, E_T, c_T)$ on V induces a cut in G by decomposing $T(G)$ into two connected components. We sometimes identify e_T with the cut it induces in G. A weighted tree $T(G)$ is called a *cut tree* if edge costs correspond to cut costs and if for any vertex pair $\{u, v\} \in \binom{V}{2}$ the cheapest edge on the unique path γ_{uv} between u and v induces a minimum u-v-cut in G. Neither must this edge be unique, nor $T(G)$. Paths are either represented as a set of edges or vertices/nodes, as convenient. Hence, a cut tree represents for any node pair $\{u, v\} \subseteq V$ a minimum-u-v-cut θ_{uv} in G. For details on cut trees, see the pioneering work by Gomory and Hu [1] or the simplifications by Gusfield [2]. A *contraction* of G by $N \subseteq V$ means replacing the set N in G by a single node, denoted by $[N]$, and leaving this node adjacent to all former adjacencies u of vertices of N, with edge costs equal to the sum of all former edges between N and u. Analogously we can contract by a set $M \subseteq E$. We start by giving some fundamental insights on the behavior of cuts in dynamic graphs. Remark 1 and Lemma 1 are both folklore based on the sensitivity analysis of multiterminal flow networks.

Remark 1. Consider a minimum u-v-cut θ_{uv} in G and let $\{b, d\} \in E$ be modified.

- θ_{uv} remains valid (with respect to u, v) with the previous costs
  in $G^{\oplus}$ iff θ_{uv} does not separate b and d and
  in $G^{\ominus}$ iff additionally there exists no cut θ' with $c(\theta') - \Delta < c(\theta_{uv})$ that separates b and d.
- θ_{uv} remains valid (with respect to u, v) with new costs $c(\theta_{uv}) \mp \Delta$
  in $G^{\ominus}$ iff θ_{uv} separates b and d and
  in $G^{\oplus}$ iff additionally there exists no cut θ' with $c(\theta') < c(\theta_{uv}) + \Delta$ that does not separate b and d.
- θ_{uv} becomes invalid (with respect to u, v) otherwise.

Lemma 1. *Consider $e_T := \{u, v\} \in E_T$ in $T(G)$ and let $\{b, d\}$ be modified in G.*
If $e_T \notin \gamma_{b,d}$ then e_T is still a minimum u-v-cut in $G^{\oplus}$ with costs $c_T(e_T)$.
If $e_T \in \gamma_{b,d}$ then e_T is still a minimum u-v-cut in $G^{\ominus}$ with costs $c_T(e_T) - \Delta$.

3 The Static Cut Tree Algorithm

To provide a self-contained paper we briefly revisit the static construction of a cut tree proposed by Gomory and Hu [1] and simplified by Gusfield [2]. Gomory and Hu presented an algorithm that iteratively constructs $n - 1$ non-crossing minimum u-v-cuts regarding $n - 1$ vertex pairs $\{u, v\} \subseteq V$, which we call *step pairs* in the following. Those step pairs are chosen arbitrarily from the set of pairs not separated by any of the cuts constructed so far. Crossings of the cuts are prevented by contractions. For pseudo-code see App. A. An *intermediate* cut tree $T_*(G) = (V_*, E_*, c_*)$ is initialized as an isolated, edgeless node containing all original vertices. Then, until no node S of $T_*(G)$ contains more than one vertex, a node S is *split*. To this end, nodes $S' \neq S$ are dealt with by contracting in

G whole subtrees N_j of S in $T_*(G)$, connected to S via edges $\{S, S_j\}$, to single nodes $[N_j]$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 minimum u-v-cut (*split cut*) in G_S, which does not cross any of the previously used cuts due to the contraction technique. Afterwards, each N_j is reconnected, again by S_j, to either S_u or S_v depending on which side of the cut $[N_j]$ ended up. Note that this cut in G_S can be proven to induce a min-u-v-cut in G.

The correctness of this Gomory-Hu method bases on Lemma 2, which was formulated and proven within another proof in [1] and rephrased by Gusfield [2]. Lemma 2 guarantees that in each iteration step each edge $\{S, S'\}$ in $T_*(G)$ has a cut pair $\{x, y\}$ with $x \in S$, $y \in S'$. A vertex pair $\{x, y\}$ is called a *cut pair* of a cut θ if θ is a minimum x-y-cut in G.

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

In an intermediate tree $T_*(G)$ we call the cut pair of an edge that appeared most recently according to Lemma 2 the *nearest cut pair* of this edge. A *(partial) execution* $\mathrm{GH} = (G, F, K)$ of Gomory-Hu is characterized by graph G, sequence F of $r \leq n-1$ step pairs and sequence K of r associated split cuts. An execution GH returns an intermediate tree $T_*(G)$.

Remark 2. Each split cut in K is represented by an edge in the final tree. However, the associated step pair might become *hidden* during the execution such that it is not the nearest cut pair in the final tree $T(G)$.

The following theorem follows directly from the correctness of Gomory-Hu and constitutes a general tool for the reuse of cuts that remain valid, dynamically.

Theorem 1. *Let K denote a set of non-crossing cuts in G and F a set of cut pairs such that each cut in K is associated to exact one pair in F and separates no other pair in F. Then $\mathrm{GH} = (G, F, K)$ is well-defined and returns a valid and unique tree $T_*(G)$ independent of the order chosen in F and K.*

The result of Barth et al. [4] for edge insertion, which is that given a cut tree $T(G)$ it suffices to compute $|\gamma_{bd}|$ minimum cuts to construct a new cut tree $T(G^{\oplus})$, now easily results from Theorem 1 together with Lemma 1. Since the edges off γ_{bd} in $T(G)$ identified by Lemma 1 as staying valid conform to the prerequisites of Theorem 1, there exists a valid Gomory-Hu execution that computes only $|\gamma_{bd}|$ new split cuts reusing the remaining cuts. Corollary 1 summarizes this insight for both modification cases.

Corollary 1. *Let K denote the set of cuts and F the set of vertex pairs induced by the edges in $T(G)$ on γ_{bd} (off γ_{bd}). Let further $T_\circ(G^{\ominus})$ ($T_\circ(G^{\oplus})$) denote the tree resulting from $T(G)$ by adjusting the costs of the edges on γ_{bd} according to Lemma 1 and contracting $E_T \setminus \gamma_{bd}$ (γ_{bd}); cp. Fig. 1. Then $\mathrm{GH} = (G^{\ominus}, F, K)$ ($\mathrm{GH} = (G^{\oplus}, E_T \setminus F, E_T \setminus K)$) returns $T_\circ(G^{\ominus})$ ($T_\circ(G^{\oplus})$).*

In the following we will denote by $T_\circ(G^{\oplus\ominus})$ an intermediate tree which serves as a starting point for further Gomory-Hu iterations, and by $T_*(G^{\oplus\ominus})$ a working version. Furthermore, we will avoid contractions during Gomory-Hu according to the simplification introduced by Gusfield [2] basing on the following lemma.

Lemma 3 (Gus. [2], Lem. 1). *Let $(N, V \setminus N)$ be a minimum x-y-cut in G, with $y \in N$. Let $(H, V \setminus H)$ be a minimum u-v-cut, with $u, v \in V \setminus N$ and $y \in H$. Then the cut $(H \cup N, (V \setminus H) \cap (V \setminus N))$ is also a minimum u-v-cut.*

Lemma 3 tells us that any split cut $(H, V \setminus H)$ splitting a subtree N can be bend along one side of N such that N is not injured. The final shape of the split cut, i.e., the side containing N, depends on which side of the split cut the member $y \in N$ of the nearest cut pair of the link between S and N ends up; N and y share the same side. Thus, we say N is sheltered by *pseudo-contraction* and consider arbitrary cuts in G instead of G_S in the following without further notice. We will further use the pseudo-contraction technique in some proofs.

4 The Dynamic Cut Tree Algorithm

In this section we develop our new algorithm for dynamically updating cut trees over time. Our algorithm guarantees the following *smoothness* between two consecutive trees $T(G)$ and $T(G^{\oplus\ominus})$: Let θ denote the cut represented by edge $\{u, v\}$ in $T(G)$. If θ is still a minimum u-v-cut in $G^{\oplus\ominus}$ then θ is again represented in $T(G^{\oplus\ominus})$, possibly with a nearest cut pair different from $\{u, v\}$.

In order to guarantee this smoothness the algorithm needs to find those edges in $T(G)$ that remain valid cuts with respect to their nearest cut pair. Some edges are already known to remain valid due to theoretical deductions (cp. Lemma 1, 4 and 7), the other edges need to be checked by (re)computing a minimum u-v-cut in $G^{\oplus\ominus}$. At the same time we want the update algorithm to be as efficient as possible, i.e., each cut computed for checking an edge in $T(G)$ should be used also for the construction of the new tree, independent of the checking result. We will see that our algorithm also fulfills this efficiency constraint.

Our algorithm is a modified Gomory-Hu method starting the iteration at the intermediate tree $T_\circ(G^{\oplus\ominus})$ constructed in Corollary 1 (see also Fig. 1). It works in two phases. The first phase splits unmarked nodes in $T_*(G^{\oplus\ominus})$, the second phase considers marked nodes resulting from the phase before. In order to provide smoothness and efficiency the step pairs in the first phase are no longer chosen arbitrarily but the intermediate tree $T_*(G^{\oplus\ominus})$ fulfills a structural invariant that tells the algorithm which step pair to choose next. For deletions several non-crossing split cuts are computed in one iteration step, which are all executed at the same time.

The Invariant. We define the invariant for both modifications—deletion and insertion—separately. To this end we introduce the *treetop* of a vertex and the *stem* of a treetop (cp. Figure 5 and Figure 6 in App. B).

Definition 1. *Consider a cut tree $T(G)$, a vertex $u \in V$ and an edge modification between b and d in G.*

Deletion: If $u \notin \gamma_{bd}$ the treetop $\Uparrow_u$ *of u is given by the subtree of $T(G)$ rooted in u not containing γ_{bd}. If $u \in \gamma_{bd}$ the* treetop $\Uparrow_u$ *of u is the union of u with the treetops of all neighbors of u in $T(G)$ that are off γ_{bd}.*

Insertion: Vertex $u \in \gamma_{bd}$ has two orientated treetops*. We denote the subtree of $T(G)$ rooted in u and containing b by $\Uparrow_{u(b)}$; $\Uparrow_{u(d)}$ is defined analogously.*

We call $\{u, v\}$ in $T(G)$, with v the only neighbor of u outside the treetop, the stem *of the treetop of u.*

Edge Deletion. During the first phase of our algorithm the intermediate trees $T_*(G^{\ominus})$ fulfill the following invariant. Each marked node consists of a treetop and constitutes a leave in $T_*(G^{\ominus})$. The root of the treetop is the only member of a nearest cut pair in the node. Let S_b and S_d denote the nodes containing b and d. For unmarked non-singleton nodes S holds:

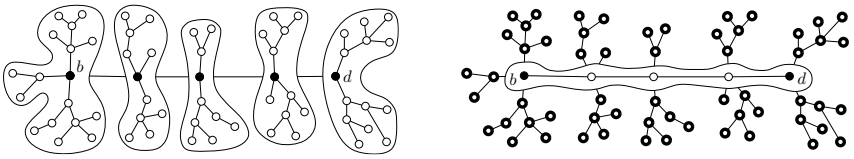
- S lies on the path between S_b and S_d, has at most degree two and contains exactly one vertex that is a member of a nearest cut pair. We call this vertex the *center* of the node.
- S consists of a subset of vertices of an *initial treetop*. The initial treetops are the treetops of the vertices on γ_{bd} in $T(G)$.
- the vertices in S provide an internal tree structure yielding the properties listed below.

Properties of the internal tree structure in unmarked nodes:

a. The structure is rooted in the center c of S.
b. Let u denote a neighbor of c. The internal subtree $\Uparrow^i_u$ rooted in u, with $c \notin \Uparrow^i_u$, consists of $\Uparrow_u$ and possibly more treetops of other vertices in S.
c. If $\Uparrow^i_u$ consists of more than one treetop, there is a minimum u-c-cut θ_u assigned to $\Uparrow^i_u$ (otherwise $\Uparrow^i_u$ *may* have an assigned cut).
d. Assigned cuts are non-crossing. An assigned cut θ_u does not split an internal subtree $\Uparrow^i_{\hat{u}}$ if (i) θ_u separates $\hat{u}$ from c or (ii) $\Uparrow^i_{\hat{u}}$ has an assigned cut.
e. If $\Uparrow^i_u$ consists of $\Uparrow_u$ and there is no minimum u-c-cut assigned, let $\{u, v\}$ denote the stem of $\Uparrow_u$ in $T(G)$. If $\{u, v\}$ is still a minimum u-v-cut after the change, any minimum u-c-cut in $G^{\ominus}$ costs $c_T(u, v)$ and $\{u, c\}$ is a cut pair of $\{u, v\}$ in $G^{\ominus}$.
f. In the situation of (e.) if a minimum u-c-cut in $G^{\ominus}$ is cheaper than the stem $\{u, v\}$ of $\Uparrow_u$ this cut separates b and d.

Edge Insertion. The intermediate trees $T_*(G^{\oplus})$ fulfill the following: Each marked node is either a singleton off γ_{bd} in $T(G)$ or consists of a subpath of γ_{bd}. There are only two unmarked nodes $S_{b'}$ and $S_{d'}$. Let S_b and S_d denote the nodes containing b and d. For the unmarked nodes holds:

- $S_{b'}$ and $S_{d'}$ are adjacent decomposing $T_*(G^{\oplus})$ into two parts. One part contains S_b and $S_{b'}$, the other contains S_d and $S_{d'}$.
- There is at most one vertex b' in $S_{b'}$ that is a member of a nearest cut pair that links $S_{b'}$ to a marked node containing a subpath of γ_{bd} (for $S_{d'}$ the analog assertion holds).



(a) $T_\circ(G^\ominus)$, all nodes unmarked. (b) $T_\circ(G^\oplus)$, singletons off γ_{bd} marked.

Fig. 1. Starting points for update algorithm with internal structure in nodes.

- $S_{b'}$ and $S_{d'}$ consist each of a subset of vertices of γ_{bd} in $T(G)$.
- the vertices in $S_{b'}$ respectively $S_{d'}$ provide an internal path structure yielding the properties listed below.

Properties of the internal path structure in $S_{b'}$ and $S_{d'}$:

a. The vertices are ordered the same as in γ_{bd}.
b. Wlog. consider $S_{b'}$ in b. to d., symmetric assertions hold for $S_{d'}$. Vertex $b' \in S_{b'}$ forms the end of the internal path that is closer to b in γ_{bd}.
c. Let further $\{b', x\}$ denote the unique edge on the internal path incident to b', and $\{v, x\}$ the stem of $\Uparrow_{x(d)}$ in $T(G)$. The cut induced by $\{v, x\}$ does neither cross any edge in $T_*(G^\oplus)$ nor split any node apart from $S_{b'}$. We say $\Uparrow_{x(d)}$ is *closed* in $T_*(G^\oplus)$.
d. If $\{v, x\}$ is still a minimum v-x-cut after the change, any minimum b'-x-cut in $G^\oplus$ costs $c_T(v, x) + \Delta$ and $\{b', x\}$ is a cut pair of $\{v, x\}$ in $G^\oplus$.

Split Cut Construction in the First Phase. The internal structures in the unmarked nodes in $T_*(G^{\oplus\ominus})$ define the step pairs. In the case of edge deletion the step pairs consist of the centers c together with one of their neighbors u, in the case of edge insertion the endings $\{b', x\}$ of the internal paths are considered. Marked nodes are not considered as long as unmarked nodes different from singletons exist (first phase). They are handled in a second phase. In order to preserve the invariant when splitting unmarked nodes the update algorithm adjusts the split cuts as described in the following.

Edge Deletion. The update algorithm starts with $T_\circ(G^\ominus)$ where all nodes are unmarked (cp. Fig. 1(a)). Observe that $T_\circ(G^\ominus)$ fulfills the invariant. In one iteration step the algorithm chooses a non-singleton unmarked node S and computes a whole set Θ of non-crossing split cuts that separate c from all its neighbors u such that none of the subtrees $\Uparrow^i_u$ is split by any cut in Θ. At the beginning of each iteration step Θ is initialized by the cuts already assigned to subtrees and the following *preprocessing* is done: If there is an cut in Θ that separates a neighbor $\hat{u}$ from c but splits $\Uparrow_{\hat{u}}$ that has no cut assigned, this cut is adjusted such that $\Uparrow_{\hat{u}}$ is no longer split according to scenario (c), which will soon be described. By induction the cuts in Θ now fulfill the condition given in (d.) of the invariant. Then the construction of split cuts begins.

The algorithm iterates the neighbors u not yet separated from c by a cut in Θ. If the stem $\{u, v\}$ of $\Uparrow_u = \Uparrow^i_u$ in $T(G)$ corresponds to an edge in G with

$c(u,v) = c_T(u,v)$, the cut induced by $\{u,v\}$ remains a minimum u-v-cut in $G^{\ominus}$ according to Lemma 4 (for a proof see App. B). Due to (e.) $\{u,v\}$ is also a minimum u-c-cut, and thus, it is marked as a remaining edge and added to Θ. If $\Uparrow_u$ is split by a previous cut in Θ this cut is bent along $\Uparrow_u$ since $\Uparrow_u$ is sheltered by pseudo-contraction. Afterwards, the cuts in Θ are again non-crossing.

Lemma 4. *An edge $\{u,v\}$ in $T(G)$ corresponds to a bridge in G if and only if $\{u,v\}$ is an edge in G with $c(u,v) = c_T(u,v)$ (bridge detection). An edge $\{u,v\}$ in $T(G)$ that corresponds to a bridge in G is still a minimum u-v-cut in $G^{\ominus}$.*

If the stem $\{u,v\}$ of $\Uparrow_u = \Uparrow^i_u$ in $T(G)$ is cheaper than the minimum of the costs of the edges incident to node S in the current intermediate tree, according to (f.), the cut induced by $\{u,v\}$ remains a minimum u-v-cut in $G^{\ominus}$. Due to (e.) $\{u,v\}$ is also a minimum u-c-cut, and thus, it is marked as a remaining edge and added to Θ. If $\Uparrow_u$ is split by a previous cut in Θ this cut is bent along $\Uparrow_u$ since $\Uparrow_u$ is sheltered by pseudo-contraction. Afterwards, the cuts in Θ are again non-crossing.

Otherwise, the algorithm computes a new minimum u-c-cut θ'_{uc} in $G^{\ominus}$. If $c^{\ominus}(\theta'_{uc}) = c_T(u,v)$, the stem $\{u,v\}$ of $\Uparrow_u$ is also a minimum u-c-cut in $G^{\ominus}$ and is marked before added to Θ. In this way, according to (e.), the algorithm reuses $\{u,v\}$ whenever it remains a minimum u-v-cut in $G^{\ominus}$. The cuts in Θ are adjusted non-crossing as described before.

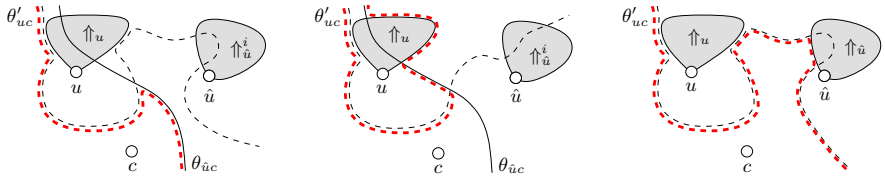
If the new cut θ'_{uc} is cheaper than $\{u,v\}$ it is adjusted such that it does not cut through $\Uparrow_u$, by Lemma 5. For simplicity we define $(V \setminus \Uparrow_u) =: \#_u$. A proof of Lemma 5 can be found in App. B.

Lemma 5. *Given $u \notin \gamma_{b,d}$ in $T(G)$ and $v \notin \Uparrow_u$. Let (A,B) be a cut separating u and v such that (A,B) induces a cut $(\Uparrow_A, \Uparrow_B)$ of $\Uparrow_u$ with $u \in \Uparrow_A$ and a cut $(\#_A, \#_B)$ of $\#_u$ with $v \in \#_B$. Then $c^{\ominus}(\#_A \cup \Uparrow_u, \#_B) \leq c^{\ominus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$.*

Then it is added to Θ and made non-crossing with all previous cuts in Θ such that a subtree $\Uparrow^i_{\hat{u}}$ of a neighbor $\hat{u}$ is not split by any cut in θ if $\hat{u}$ is separated from c by a cut in Θ. This adjustment follows scenario (a) – (c) (cp. Fig. 2).

Scenario (a) [θ'_{uc} (dashed) meets $\theta_{\hat{u}c} \in \Theta$ (solid) and separates $\hat{u}$ from c]: The new cut θ'_{uc} is bent along $\theta_{\hat{u}c}$ due to pseudo-contraction (fat dashed) such that $\Uparrow_u \cup \Uparrow^i_{\hat{u}}$ is not split. In this scenario $\theta_{\hat{u}c}$ is removed from Θ but remains assigned to $\Uparrow^i_{\hat{u}}$. The algorithm further reshapes the internal tree structure such that $\hat{u}$ together with $\Uparrow^i_{\hat{u}}$ becomes a neighbor of u. Note, that $\theta_{\hat{u}c}$ is also a minimum $\hat{u}$-u-cut according to Lemma 2 since we can consider $\theta_{\hat{u}c}$ and θ'_{uc} as the first two split cuts in a Gomory-Hu execution.

Scenario (b) [θ'_{uc} meets $\theta_{\hat{u}c} \in \Theta$ but does not separate $\hat{u}$ from c]: Cut θ'_{uc} is bent along $\theta_{\hat{u}c}$ due to pseudo-contraction. Depending on the shape of $\theta_{\hat{u}c}$ it might split $\Uparrow_u$ after this adjustment. Thus, Lemma 5 is applied restraining θ'_{uc} from splitting $\Uparrow_u$ (fat dashed). Now $\Uparrow_u$ is sheltered by pseudo-contraction and $\theta_{\hat{u}c}$ is also adjusted sparing $\Uparrow_u$ (not depicted in Fig. 2(b)). Finally, neither $\Uparrow_u$ nor $\Uparrow^i_{\hat{u}}$ is split.

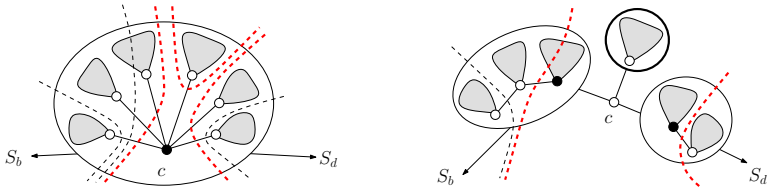


(a) $\theta_{\hat{u}c} \in \Theta$, θ'_{uc} separates $\hat{u}$ and c.

(b) $\theta_{\hat{u}c} \in \Theta$, θ'_{uc} does not separate $\hat{u}$ and c.

(c) θ'_{uc} separates neighbor $\hat{u}$ from c.

Fig. 2. Tree scenarios adjusting a newly computed cut with respect to other cuts in Θ. In any other scenario θ'_{uc} is not adjusted.



(a) Cuts considered in one iteration step, fat dashed cuts are in Θ.

(b) New intermediate tree $T_*(G^{\ominus})$ with assigned cuts in new nodes.

Fig. 3. Previous and new $T_*(G^{\ominus})$ after executing cuts in Θ. Marked nodes bold-framed, black centers of unmarked nodes; new Θs (b) initialized with fat dashed cuts.

Scenario (c) [θ'_{uc} separates $\hat{u}$ from c, no minimum $\hat{u}$-c-cut in Θ yet]: Since θ'_{uc} might split $\Uparrow_{\hat{u}}$ it is adjusted according to Lemma 5 (fat dashed) which is possible as Lemma 5 does not require cut (A, B) to be minimum. Furthermore, $\hat{u}$ together with $\Uparrow_{\hat{u}}$ becomes a neighbor of u.

At the end of each iteration step the adjusted split cuts in Θ are shaped as described in Remark 3. Furthermore, the cuts (and cut pairs) in Θ together with the cuts (and nearest cut pairs) already represented in the previous intermediate tree conform to the requirements of Theorem 1. Thus, the split cuts in Θ are executed in an arbitrary order yielding a new intermediate tree (cp. Fig. 3). Marked edges/cuts inherit their mark to the new node not containing c. The new intermediate tree conforms to all constraints but (d.(i)) given by the invariant. Constraint (d.(i)) becomes true after the preprocessing at the beginning of the next iteration step. Detailed proofs of (c.) – (f.) can be found in App. B.

Remark 3. Due to (f.) and the nesting of cuts in scenario (a) there are at most two cuts in the final set Θ—one separating c from b and one separating c from d—that do not correspond to a stem of a treetop of a neighbor of c.

Edge Insertion. Consider all singletons in $T_\circ(G^{\oplus})$ as marked nodes (cp. Fig. 1(b)). Any cheapest edge e on γ_{bd} in $T(G)$ induces a minimum b-d-cut in $G^{\oplus}$ and the only non-singleton node in $T_\circ(G^{\oplus})$ consists of γ_{bd}. Thus, the update algorithm starts with splitting the non-singleton node by e. The resulting intermediate tree then fulfills the invariant with $b =: b' \in S_{b'}$ and $d =: d' \in S_{d'}$.

The algorithm iteratively considers step pairs in $S_{b'}$. When $S_{b'}$ became a singleton the algorithm continues the same way with $S_{d'}$. In one iteration step

it chooses $\{b', x\}$ as step pair and computes a new minimum b'-x-cut $\theta'_{xb'}$ in $G^{\oplus}$. In order to preserve the invariant θ_x is replaced or adjusted as follows.

If $\theta'_{xb'}$ costs $c_T(v,x) + \Delta$ with $\{v,x\}$ the stem of $\Uparrow_{x(d)}$ in $T(G)$, the cut induced by $\{v,x\}$ is a minimum b'-x-cut in $G^{\oplus}$ since $b' \notin \Uparrow_{x(d)}$. Thus, $\theta'_{xb'}$ is replaced by $\{v,x\}$. In this way, according to (d.), the algorithm reuses $\{v,x\}$ whenever it remains a minimum v-x-cut in $G^{\oplus}$. Note that the shape of $\{v,x\}$ does not change during its execution since $\Uparrow_{x(d)}$ is closed in $T_*(G^{\oplus})$. Afterwards b' is a marked singleton in the new tree. The new node $S_{b'}$ containing x is still adjacent to $S_{d'}$ since e was sheltered by $\Uparrow_{x(d)}$; x becomes b'.

Otherwise, if $\theta'_{xb'}$ costs $c_T(e') + \Delta$, with e' any cheapest edge on the path between b' and x in $T(G)$, the cut induced by e' remains valid. Thus, θ_x is replaced by e'. Note that this new split cut may be reshaped when executed by pseudo-contraction of subtrees linked to $S_{b'}$. However, since e' respects $\Uparrow_{x(d)}$ and $\Uparrow_{x(d)}$ is closed in $T_*(G^{\oplus})$, $\Uparrow_{x(d)}$ shelters e and $\Uparrow_{y(d)} \subset \Uparrow_{x(d)}$, with y the next vertex adjacent to x on the internal path, is even closed in the new tree after the execution; b' becomes a marked singleton, $x \in S_{b'}$ becomes b'.

If $\theta'_{xb'}$ is cheaper than $c_T(e') + \Delta$ it does not separate b and d and needs to be adjusted such that after its execution (c.) holds again. This adjustment bases on the following lemma applied in two different scenarios. For simplicity we define $(V \setminus \Uparrow_{y(d)}) =: \#_{y(d)}$. Lemma 6 also holds after replacing d by b.

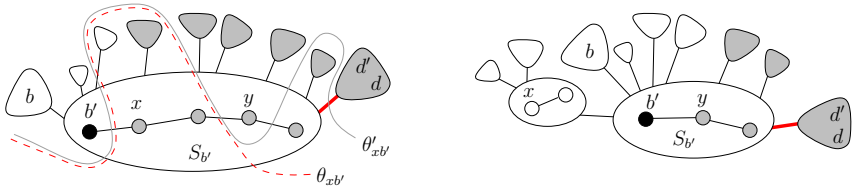
Lemma 6. *Given two vertices $u, y \in \gamma_{b,d}$ in $T(G)$ with $u \notin \Uparrow_{y(d)}$. Let (A,B) be a cut separating u and y such that (A,B) induces a cut $(\Uparrow_A, \Uparrow_B)$ of $\Uparrow_{y(d)}$ with $y \in \Uparrow_A$ and a cut $(\#_A, \#_B)$ of $\#_{y(d)}$ with $u \in \#_B$. Furthermore let b, d, y share the same side of cut $(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$. Then $c_{\oplus}(\#_A \cup \Uparrow_{y(d)}, \#_B) \leq c_{\oplus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$.*

Scenario (a): If $\theta'_{xb'}$ does not separate x from $\{b,d\}$, according to Lemma 6, it can be adjusted such that it does not split $\Uparrow_{y(d)} \subset \Uparrow_{x(d)}$, with y the next vertex adjacent to x on the internal path (cp. Fig. 7 in App. C). In this case b' becomes a marked singleton, $x \in S_{b'}$ becomes b' and the unmarked nodes are again adjacent.

Scenario (b): If $\theta'_{xb'}$ separates x and $\{b,d\}$, let $y \neq b'$ denote the vertex closest to x on the internal path that shares the cut side with $\{b,d\}$. If such an y does not exist $\theta'_{xb'}$ separates all vertices of the internal path from b' and $S_{b'}$ becomes an unmarked singleton containing b'; e is reconnected to b'. Otherwise, applying Lemma 6 to y prevents treetop $\Uparrow_{y(d)}$ from splitting (cp. Fig. 4) and e is sheltered by $\Uparrow_{y(d)}$.

After splitting $S_{b'}$ by the replaced or adjusted split cut the new intermediate tree $T_*(G^{\oplus})$ conforms to all constraints given by the invariant. A detailed proof of (d.) can be found in App. C.

Handling Marked Nodes in the Second Phase. Marked nodes are considered when all unmarked nodes in $T_*(G^{\oplus\ominus})$ became singletons. The shape of the marked nodes conforms to the invariant.



(a) Cut $\theta'_{xb'}$ (solid) is adjusted to dashed cut $\theta_{xb'}$ according to Lemma 6.

(b) New intermediate tree after executing $\theta_{xb'}$.

Fig. 4. Previous and new $T_*(G^{\oplus})$ for scenario (b); $\Uparrow_{x(d)}$ (left) and $\Uparrow_{y(d)}$ (right) are gray colored, black end b' of internal path; fat edge denotes e linking $S_{b'}$ and $S_{d'}$.

Edge Deletion. Each non-singleton marked node S consists of a treetop $\Uparrow_u$ and was formed when the stem $\{u, v\}$ of $\Uparrow_u$ turned out to remain a valid minimum u-v-cut in $G^{\ominus}$. Thus, Lemma 7 together with Theorem 1 allows for just unfolding $\Uparrow_u$ in order to complete the cut tree. For a proof of Lemma 7 see App. D.

Lemma 7. *In $G^{\ominus}$, let $(U, V \setminus U)$ be a minimum u-v-cut not separating $\{b, d\}$, with $\gamma_{b,d} \cap U = \emptyset$. Then, a cut induced by an edge $\{g, h\}$ in the old tree $T(G)$, with $g, h \in U$, remains a minimum separating cut in $G^{\ominus}$ for all its previous cut pairs within U, and a minimum g-h-cut in particular.*

Edge Insertion. Wlog. consider marked nodes resulting from splitting $S_{b'}$. Each non-singleton marked node S consists of a subpath of γ_{bd} and was formed by scenario (b) when x was separated from b' by a split cut θ that separated x from $\{b, d\}$ (cp. Fig. 4). Note that x became the end of the subpath in S that is closer to b and b' is between b and x on γ_{bd}. The following assertion holds for an edge $\{g, h\} \subseteq S$. A proof is given in App. D.

Lemma 8. *There exists no pair $\{u, n\} \in \binom{V}{2}$ such that edge $\{g, h\}$ is a valid minimum u-n-cut in $G^{\oplus}$. In particular, $\{g, h\}$ is no minimum g-h-cut in $G^{\oplus}$.*

Thus, there is no need for searching for remaining edges in marked nodes. Consequently, the update algorithm just continues Gomory-Hu choosing arbitrary step pairs in marked nodes until $T_*(G^{\oplus})$ is a tree of singletons.

5 Summary of the Algorithm's Performance

The smoothness guarantee given in Theorem 2 holds for both modification cases and is due to the invariant. The latter guarantees that for each edge in $T(G)$ it is checked whether it remains valid with respect to its nearest cut pair and allows for reusing the found cuts by managing the choice of step pairs and reshaping newly computed cuts in order to guarantee the right shape of $T_*(G^{\oplus\ominus})$.

Theorem 2. *The minimum cut-basis given by $T(G^{\oplus\ominus})$ contains each cut represented in $T(G)$ that remains valid with respect to its nearest cut pair.*

Due to the order in which the step pairs are considered, in the case of edge deletion, further results are applicable, these are Lemma 4 for detecting bridges and Lemma 7 for locating valid treetops. Together with Lemma 1, they yield the following guarantee on saved cut computations compared to a static algorithm.

Theorem 3. *The update algorithm saves* $n-1-(|\gamma_{bd}|-1)$ *cut computations in the case of edge insertion (cp. to Barth et al. [4]), and* $|\gamma_{bd}|+|B|+\sum_{\Uparrow\in X}|\Uparrow|$ *cut computations in the case of edge deletion.*

In the above theorem $|\gamma_{bd}|$ denotes the number of edges in γ_{bd}, $|\Uparrow|$ the number of edges in a treetop $\Uparrow$, X the set of marked non-singleton nodes in $T_*(G^{\ominus})$ after the first phase, and B the set of edges/stems that are marked during the first phase (B particularly contains all bridges not in $\gamma_{bd}\cup\bigcup_{\Uparrow\in X}\Uparrow$). The invariant finally takes care that each cut computation corresponds to the splitting of a node. This justifies Theorem 4. In the case of deletion this becomes possible due to the storage of cuts by assigning them to treetops.

Theorem 4. *The update algorithm calculates at most* $n-1$ *cuts.*

We further ask for the efficiency of our update algorithm, which we define by the ratio of necessary cut computations to calculated cuts. Unfortunately, the set of necessary cut computations is difficult to predict. We consider a set A of cuts as necessary if there exists both a cut tree $\hat{T}(G^{\oplus\ominus})$ providing a maximum intersection with $T(G)$ in terms of represented cuts, and a Gomory-Hu execution returning $\hat{T}(G^{\oplus\ominus})$ such that the split cuts consist of A and the intersection of $\hat{T}(G^{\oplus\ominus})$ and $T(G)$. From this point of view, we can express bounds for the number of necessary cuts dependent on the shape of $T_*(G^{\oplus\ominus})$ after the first phase of our algorithm. This allows for estimating the efficiency at least for an actual update. A high efficiency indicates that $T(G^{\oplus\ominus})$ is close to an optimum tree $\hat{T}(G^{\oplus\ominus})$. In contrast, low efficiency does not necessarily mean that $T(G^{\oplus\ominus})$ is far from $\hat{T}(G^{\oplus\ominus})$. Note that any tree $\hat{T}(G^{\oplus\ominus})$ represents at least those cuts in $T(G)$ that remain valid with respect to their nearest cut pair. Thus, Theorem 2 constitutes an upper bound for the distance between $T(G^{\oplus\ominus})$ and $\hat{T}(G^{\oplus\ominus})$.

Furthermore, in the case of deletion, low efficiency indicates that many reusable cuts form many small treetops linked to a short path γ_{bd} in $T(G)$, in particular, the vertices on γ_{bd} have high degrees. In the case of insertion, low efficiency only occurs if γ_{bd} is long and contains many reusable cuts. Both cases seem to constitute rather special trees. Thus, we conjecture our algorithm to achieve high efficiency on most instances. See App. F for a further discussion.

Conclusion. We introduced the first algorithm that dynamically updates a cut tree of G, guaranteeing a high temporal smoothness and providing strong savings of cut computations in both cases, edge insertion and deletion. Future work includes further improvement of this algorithm by also exploiting cost limits in order to avoid cut computations, the analysis of batch updates and a systematic experimental runtime evaluation.

References

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

3. S. E. Elmaghraby. Sensitivity Analysis of Multiterminal Flow Networks. *Operations Research*, 12(5):680–688, October 1964.
4. D. Barth, P. Berthomé, M. Diallo and A. Ferreira. Revisiting parametric multiterminal problems: Maximum flows, minimum cuts and cut-tree computations. *Discrete Optimization*, 3(3):195–205, July 2006.
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. M. G. Scutellà. A note on the parametric maximum flow problem and some related reoptimization issues. *Annals of Operations Research*, 150(1):231–244, 2006.

Appendix

A Omitted Pseudo-Code of Static Cut Tree Algorithm

Algorithm 1: GOMORY-HU (CUT TREE)

Input: Graph $G = (V, E, c)$
Output: 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 nodes`
  $\{u, v\} \leftarrow$ arbitrary pair from $\binom{S}{2}$
  forall the $S_j \sim S$ *in* $T_*(G)$ **do** $N_j \leftarrow$ subtree of S in $T_*(G)$ with $S_j \in N_j$
  $G_S = (V_S, E_S, c_S) \leftarrow$ in G contract each N_j to $[N_j]$ `// contraction`
  $(U, V_S \setminus U) \leftarrow$ min-u-v-cut in G_S, costs δ, $u \in U$
  $S_u \leftarrow S \cap U$ and $S_v \leftarrow S \cap (V_S \setminus U)$ `// split` $S = S_u \dot\cup S_v$
  $V_* \leftarrow (V_* \setminus \{S\}) \cup \{S_u, S_v\}$, $E_* \leftarrow E_* \cup \{\{S_u, S_v\}\}$, $c_*(S_u, S_v) \leftarrow \delta$
  forall the *former edges* $e_j = \{S, S_j\} \in E_*$ **do**
    if $[N_j] \in U$ **then** $e_j \leftarrow \{S_u, S_j\}$; `// reconnect` S_j `to` S_u
    else $e_j \leftarrow \{S_v, S_j\}$; `// reconnect` S_j `to` S_v
return $T_*(G)$

B Omitted Proofs on Split Cut Construction for Deletion

Lemma 4. *An edge $\{u, v\}$ in $T(G)$ corresponds to a bridge in G if and only if $\{u, v\}$ is an edge in G with $c(u, v) = c_T(u, v)$ (bridge detection). An edge $\{u, v\}$ in $T(G)$ that corresponds to a bridge in G is still a minimum u-v-cut in $G^{\ominus}$.*

Proof. Bridge detection ($\Rightarrow$): Assume $\{u, v\}$ to be a bridge in G. Any minimum u-v-cut in G separates u and v, and thus, is at least as expensive as $c(u, v)$. Furthermore, the cut decomposing G into the two parts indicated by the bridge costs exactly $c(u, v)$, and thus, is a minimum u-v-cut and it holds $c_T(u, v) = c(u, v)$.

Bridge detection ($\Leftarrow$): Assume $\{u, v\}$ to be an edge in G with $c(u, v) = c_T(u, v)$. Any minimum u-v-cut in G separates u and v, and thus, is at least as expensive as $c(u, v)$. Thus, there is no cycle in G including $\{u, v\}$, i.e., $\{u, v\}$ is a bridge in G. Since we assume G to be connected, the cut decomposing G into the two parts indicated by the bridge is the only minimum u-v-cut in G, and thus, $\{u, v\}$ is an edge in $T(G)$.

Second assertion: Assume $\{u, v\}$ to be a bridge in G. If $\{u, v\}$ in $T(G)$ is on γ_{bd}, the assertion holds according to Lemma 1. Otherwise, since $\{u, v\}$ is not

involved in the edge deletion/cost reduction it is still a bridge and still the only minimum u-v-cut in $G^{\ominus}$. □

Lemma 5. *Given $u \notin \gamma_{b,d}$ in $T(G)$ and $v \notin \Uparrow_u$. Let (A, B) be a cut separating u and v such that (A, B) induces a cut $(\Uparrow_A, \Uparrow_B)$ of $\Uparrow_u$ with $u \in \Uparrow_A$ and a cut $(\#_A, \#_B)$ of $\#_u$ with $v \in \#_B$. Then $c^{\ominus}(\#_A \cup \Uparrow_u, \#_B) \leq c^{\ominus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$.*

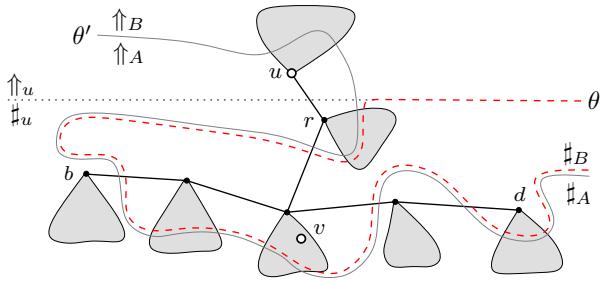


Fig. 5. Special parts of $G^{\ominus}$: $\gamma_{b,d}$ (black) connects b and d, stem $\{u, r\}$ of $\Uparrow_u$ induces cut $(\Uparrow_u, \#_u)$; $\#_u$ and $\Uparrow_u$, both cut by θ' (solid), adjusted to θ (dashed) by Lemma 5.

Proof. Using the fact that in $T(G)$ the stem $\{u, r\}$ of $\Uparrow_u$ represents a minimum u-r-cut, we prove Lemma 5 by contradiction. We show that cut $(\Uparrow_A, V \setminus \Uparrow_A)$ would have been cheaper than the edge-induced minimum u-r-cut $(\Uparrow_u, \#_u) = (\Uparrow_u, V \setminus \Uparrow_u)$ in G if $c^{\ominus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$ was cheaper than $c^{\ominus}(\#_A \cup \Uparrow_u, \#_B)$ in $G^{\ominus}$. We express the costs of $(\Uparrow_A, V \setminus \Uparrow_A)$ and $(\Uparrow_u, V \setminus \Uparrow_u)$ with the aid of $(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$ and $(\#_A \cup \Uparrow_u, \#_B)$ considered in Lemma 5. Note that $(\Uparrow_A, V \setminus \Uparrow_A)$ and $(\Uparrow_u, V \setminus \Uparrow_u)$ do not separate b and d. Thus, their costs are unaffected by the deletion. We get

$$\begin{aligned}
&\textit{(i)}\ c(\Uparrow_A, V \setminus \Uparrow_A) = c^{\ominus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) - c^{\ominus}(\#_A, \#_B \cup \Uparrow_B) + c^{\ominus}(\#_A, \Uparrow_A)\\
&\textit{(ii)}\ c(\Uparrow_u, V \setminus \Uparrow_u) = c^{\ominus}(\#_A \cup \Uparrow_u, \#_B) - c^{\ominus}(\#_A, \#_B) + c^{\ominus}(\#_A, \Uparrow_u)
\end{aligned}$$

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

$$\begin{aligned}
c(\Uparrow_A, V \setminus \Uparrow_A) - c(\Uparrow_u, V \setminus \Uparrow_u) = {} & [c^{\ominus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) - c^{\ominus}(\#_A \cup \Uparrow_u, \#_B)]\\
& - [c^{\ominus}(\#_A, \#_B \cup \Uparrow_B) - c^{\ominus}(\#_A, \#_B)]\\
& + [c^{\ominus}(\#_A, \Uparrow_A) - c^{\ominus}(\#_A, \Uparrow_u)] < 0
\end{aligned}$$

This contradicts the fact that the edge-induced cut $(\Uparrow_u, V \setminus \Uparrow_u)$ is a minimum u-r-cut in G. □

Proposition 1. *After each iteration step during the first phase $T_*(G^{\ominus})$ fulfills constraint (c.) of the invariant:*

If $\Uparrow_u^i$ consists of more than one treetop, there is a minimum u-c-cut θ_u assigned to $\Uparrow_u^i$.

Proof. We prove Proposition 1 by induction. At the beginning $T_\circ(G^\ominus)$ fulfills the invariant. Suppose further that the previous intermediate tree fulfills the invariant. Now consider a neighbor u of the center c in an unmarked node S of a new intermediate tree $T_*(G^\ominus)$, and assume $\Uparrow_u^i \neq \Uparrow_u$. We show that $\Uparrow_u^i$ has a minimum u-c-cut assigned.

Since $\Uparrow_u^i \neq \Uparrow_u$, u must have been already considered in a previous iteration step as a neighbor of another center c' catching an internal subtree by a minimum u-c'-cut θ_u according to scenario (a) or (c). Since u is not a center but still a neighbor in $T_*(G^\ominus)$, u must have been further caught itself by a minimum $\hat{u}$-c'-cut according to scenario (a) where θ_u is also a minimum u-$\hat{u}$-cut. If $\hat{u} = c$ the assertion holds. Otherwise, the latter argument iteratively holds for all reconnections of u according to scenario (a) that have occurred in previous iteration steps finally yielding the link between u and c. □

Proposition 2. *After each iteration step during the first phase $T_*(G^\ominus)$ fulfills all but (i) of constraint (d.) of the invariant. Constraint (d.(i)) holds after the preprocessing at the beginning of the next iteration step:*

Assigned cuts are non-crossing. An assigned cut θ_u does not split an internal subtree $\Uparrow_{\hat{u}}^i$ if (i) θ_u separates $\hat{u}$ from c or (ii) $\Uparrow_{\hat{u}}^i$ has an assigned cut.

Proof. We prove Proposition 2 by induction. At the beginning $T_\circ(G^\ominus)$ fulfills the invariant. Suppose further that the previous intermediate tree fulfills the invariant. Now consider a subtree $\Uparrow_u^i$ and its assigned minimum u-c-cut θ_u in an unmarked node S with center c of a new intermediate tree $T_*(G^\ominus)$. We show that θ_u does not split any internal subtree $\Uparrow_{\hat{u}}^i$ that has a cut $\theta_{\hat{u}}$ assigned; in particular if θ_u separates $\hat{u}$ from c. Furthermore, we will see that θ_u and $\theta_{\hat{u}}$ are non-crossing. After the preprocessing, according to scenario (c), θ_u also spares any internal subtree $\Uparrow_{\hat{u}}^i$ that has no assigned cut but whose root $\hat{u}$ is separated from c by θ_u.

Let $\Uparrow_{\hat{u}}^i$ denote the internal subtree of neighbor $\hat{u}$ and $\theta_{\hat{u}}$ the assigned cut. Wlog. let θ_u be initially computed before $\theta_{\hat{u}}$. Now consider the initial computation of $\theta_{\hat{u}}$ with respect to a center c'. In this iteration step all assigned cuts—and θ_u in particular—fulfill (d.) by induction hypothesis, and $\theta_{\hat{u}}$ does not split $\Uparrow_{\hat{u}}^i$ due to the application of Lemma 5.

If u is a neighbor of c' when $\theta_{\hat{u}}$ is initially computed, then $\theta_{\hat{u}}$ either separates u and c' according to scenario (a), or scenario (b) occurs. In both scenarios θ_u and $\theta_{\hat{u}}$ do not cross. In scenario (a) $\Uparrow_u^i$ becomes part of $\Uparrow_{\hat{u}}^i$, and u and $\hat{u}$ will never be neighbors of the same center in a subsequent iteration step. In scenario (b) θ_u and $\theta_{\hat{u}}$ are adjusted such that neither $\Uparrow_u^i$ nor $\Uparrow_{\hat{u}}^i$ are split.

If u is not a neighbor of c' when $\theta_{\hat{u}}$ is initially computed, $\Uparrow_u^i$ is part of a subtree $\Uparrow_x^i$ where x is a neighbor of c' and θ_x the assigned cut. Thus θ_u and θ_x are nested as shown in scenario (a). If $\theta_{\hat{u}}$ separates x and c' according to scenario (a) $\Uparrow_x^i$ becomes part of $\Uparrow_{\hat{u}}^i$, and x and $\hat{u}$ (and in particular u and $\hat{u}$) will never

be neighbors of the same center in a subsequent iteration step. If scenario (b) occurs θ_x and $\theta_{\hat{u}}$ are adjusted such that neither $\Uparrow_x^i$ nor $\Uparrow_{\hat{u}}^i$ are split and θ_x and $\theta_{\hat{u}}$ do not cross. Since θ_u is nested in θ_x, also θ_u and $\theta_{\hat{u}}$ do not cross. Note that a possible adjustment of θ_x in this scenario indirectly reshapes θ_u such that both cuts remain nested.

Subsequent steps after the initial computation of $\theta_{\hat{u}}$ (which finally link u and $\hat{u}$ both to c) respect the properties of θ_u and $\theta_{\hat{u}}$ proven above. □

Proposition 3. *After each iteration step during the first phase $T_*(G^{\ominus})$ fulfills constraint (e.) of the invariant:*

> *If $\Uparrow_u^i$ consists of $\Uparrow_u$ and there is no minimum u-c-cut assigned, let $\{u, v\}$ denote the stem of $\Uparrow_u$ in $T(G)$. If $\{u, v\}$ is still a minimum u-v-cut after the change, any minimum u-c-cut in $G^{\ominus}$ costs $c_T(u, v)$ and $\{u, c\}$ is a cut pair of $\{u, v\}$ in $G^{\ominus}$.*

Proof. We prove Proposition 3 by induction. At the beginning $T_\circ(G^{\ominus})$ fulfills the invariant. Suppose further that the previous intermediate tree fulfills the invariant. Now let c denote the center in an unmarked node S of a new intermediate tree $T_*(G^{\ominus})$ and let $\Uparrow_u^i$ consists of $\Uparrow_u$ with u a neighbor of c. Let further $\{u, v\}$ denote the stem of $\Uparrow_u$ in $T(G)$ and assume that $\{u, v\}$ is still a minimum u-v-cut after the change. We show that any minimum u-c-cut θ in $G^{\ominus}$ costs $c_T(u, v)$. Since the cut induced by $\{u, v\}$ in $G^{\ominus}$ exactly cuts off $\Uparrow_u$ this cut also separates u and c. Thus, θ is at most as expensive as $\{u, v\}$. In case of equality $\{u, c\}$ is a cut pair of $\{u, v\}$ in $G^{\ominus}$.

If $c = v$ or θ separates u and v, the assertion obviously holds. Otherwise, θ does not separate u and v but v and c. Since $\Uparrow_u$ is linked to $c \neq v$, v must have been considered as a node center in a previous iteration step, i.e., $v \notin S$. Thus, v is in a subtree $[N]$ linked to S by a nearest cut pair $\{c, n\}$ in $T_*(G^{\ominus})$. As θ does not separate u and v, $[N]$ and u are on the same side of θ and θ separates n and c. However, the minimum c-n-cut θ_n induced by the link $\{c, n\}$ also separates $v \in [N]$ and $u \in S$. Thus, $c^{\ominus}(\theta) \geq c^{\ominus}(\theta_n) \geq c_T(u, v)$. □

Proposition 4. *After each iteration step during the first phase $T_*(G^{\ominus})$ fulfills constraint (f.) of the invariant:*

> *In the situation of (e.) if a minimum u-c-cut in $G^{\ominus}$ is cheaper than the stem $\{u, v\}$ of $\Uparrow_u$ this cut separates b and d.*

Proof. We prove Proposition 4 by induction. At the beginning $T_\circ(G^{\ominus})$ fulfills the invariant. Suppose further that the previous intermediate tree fulfills the invariant. Now let c denote the center in an unmarked node S of a new intermediate tree $T_*(G^{\ominus})$ and let $\Uparrow_u^i$ consists of $\Uparrow_u$ with u a neighbor of c. Let further $\{u, v\}$ denote the stem of $\Uparrow_u$ in $T(G)$ and θ a minimum u-c-cut in $G^{\ominus}$ that is cheaper than the stem $\{u, v\}$ of $\Uparrow_u$. We show that θ separates b and d. Assume that θ does not separate b and d.

If $c = v$ it obviously holds $c^{\ominus}(\theta) = c_T(u, v)$, and the assumption that θ is cheaper than stem $\{u, v\}$ fails.

Otherwise, if $c \neq v$, u must have become a neighbor of c due to the occurrence of scenario (c) in a previous iteration step, i.e., u and c have previously been neighbors of a common center c', c has caught u by a minimum c-c'-cut θ_c, and $\Uparrow_u$ and $\Uparrow_c$ are thus disjoint in $T(G)$. Furthermore, since θ is supposed to not separate b and d, any cheapest edge e on the path between u and c in $T(G)$ still represents a valid minimum u-c-cut after the change. Note that e separates $\Uparrow_u$ and $\Uparrow_c$. Now let $x \notin \gamma_{bd}$ denote the vertex in $T(G)$ with the smallest treetop $\Uparrow_x$ such that $(\Uparrow_u \cup \Uparrow_c) \subset \Uparrow_x$. Vertex x exists since S consists of a subset of an initial treetop; x it is on the path from u to c.

If e is between x and u it follows by Lemma 7 that $\{u, v\}$ is still a valid minimum u-v-cut and according to Proposition 3 θ costs $c_T(u, v)$. Thus the assumption that θ is cheaper than stem $\{u, v\}$ fails.

If e is between x and c it follows by Lemma 7 that the stem $\{c, y\}$ of $\Uparrow_c$ in $T(G)$ is still a valid minimum c-y-cut. However, due to (e.), when c was initially considered as a neighbor of c', the cut induced by $\{c, y\}$ would have been chosen as split cut instead of θ_c. Thus, u would have never become a neighbor of c. Thus, this case does not occur. □

C Omitted Proofs on Split Cut Construction for Insertion

Lemma 6. *Given two vertices $u, y \in \gamma_{b,d}$ in $T(G)$ with $u \notin \Uparrow_{y(d)}$. Let (A, B) be a cut separating u and y such that (A, B) induces a cut $(\Uparrow_A, \Uparrow_B)$ of $\Uparrow_{y(d)}$ with $y \in \Uparrow_A$ and a cut $(\#_A, \#_B)$ of $\#_{y(d)}$ with $u \in \#_B$. Furthermore let b, d, y share the same side of cut $(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$. Then $c_\oplus(\#_A \cup \Uparrow_{y(d)}, \#_B) \leq c_\oplus(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$.*

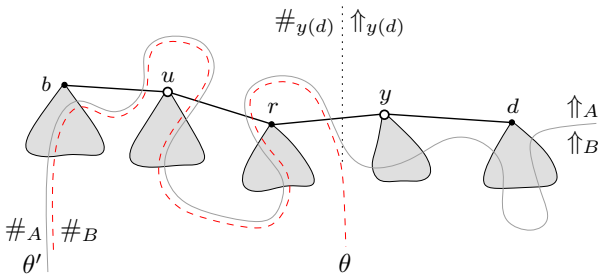


Fig. 6. Special parts of $G^\oplus$: $\gamma_{b,d}$ (black) connects b and d, stem $\{r, y\}$ treetop $\Uparrow_{y(d)}$ induces cut $(\Uparrow_{y(d)}, \#_{y(d)})$; $\#_{y(d)}$ and $\Uparrow_{y(d)}$ both cut by θ' (solid), adjusted to θ (dashed) by Lemma 6.

Proof. The Proof of Lemma 6 bases on the same idea as the proof of Lemma 5. Using the fact that in $T(G)$ the stem $\{r, y\}$ with $r \in \#_{y(d)}$ represents a minimum r-y-cut, we prove Lemma 6 by contradiction. We show that cut $(\Uparrow_A, V \setminus \Uparrow_A)$ would have been cheaper than the edge-induced minimum r-y-cut

$(\Uparrow_{y(d)}, \#_{y(d)}) = (\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)})$ in G if $c^{\oplus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$ was cheaper than $c^{\oplus}(\#_A \cup \Uparrow_{y(d)}, \#_B)$ in $G^{\oplus}$. We express the costs of $(\Uparrow_A, V \setminus \Uparrow_A)$ and $(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)})$ with the aid of $(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B)$ and $(\#_A \cup \Uparrow_{y(d)}, \#_B)$ considered in Lemma 6. Note that $c(\Uparrow_A, V \setminus \Uparrow_A) = c^{\oplus}(\Uparrow_A, V \setminus \Uparrow_A) - \Delta$ and $c(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)}) = c^{\oplus}(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)}) - \Delta$. Thus, for our contradiction, it will do to show that $c^{\oplus}(\Uparrow_A, V \setminus \Uparrow_A)$ would have been cheaper than $c^{\oplus}(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)})$. We get

$$\begin{aligned}
&(i)\ c^{\oplus}(\Uparrow_A, V \setminus \Uparrow_A) &&= c^{\oplus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) &&- c^{\oplus}(\#_A, \#_B \cup \Uparrow_B) \\
& & & & &+ c^{\oplus}(\#_A, \Uparrow_A) \\
&(ii)\ c^{\oplus}(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)}) &&= c^{\oplus}(\#_A \cup \Uparrow_{y(d)}, \#_B) &&- c^{\oplus}(\#_A, \#_B) \\
& & & & &+ c^{\oplus}(\#_A, \Uparrow_{y(d)})
\end{aligned}$$

Again we observe two inequalities: $c^{\oplus}(\#_A, \#_B) \leq c^{\oplus}(\#_A, \#_B \cup \Uparrow_B)$ and $c^{\oplus}(\#_A, \Uparrow_A) \leq c^{\oplus}(\#_A, \Uparrow_{y(d)})$; together with the assumption that $c^{\oplus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) < c^{\oplus}(\#_A \cup \Uparrow_{y(d)}, \#_B)$, by subtracting (ii) from (i), we get:

$$\begin{aligned}
c^{\oplus}(\Uparrow_A, V \setminus \Uparrow_A) - c^{\oplus}(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)}) = {} & [c^{\oplus}(\#_A \cup \Uparrow_A, \#_B \cup \Uparrow_B) \\
& - c^{\oplus}(\#_A \cup \Uparrow_{y(d)}, \#_B)] \\
& - [c^{\oplus}(\#_A, \#_B \cup \Uparrow_B) - c^{\oplus}(\#_A, \#_B)] \\
& + [c^{\oplus}(\#_A, \Uparrow_A) - c^{\oplus}(\#_A, \Uparrow_{y(d)})] < 0
\end{aligned}$$

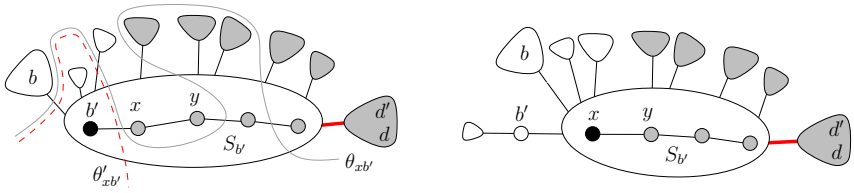
This contradicts the fact that the edge-induced cut $(\Uparrow_{y(d)}, V \setminus \Uparrow_{y(d)})$ is a minimum r-y-cut in G. □

Proposition 5. *After each iteration step during the first phase $T_*(G^{\oplus})$ fulfills constraint (d.) of the invariant:*

If $\{v, x\}$ is still a minimum v-x-cut after the change, any minimum b'-x-cut in $G^{\oplus}$ costs $c_T(v, x) + \Delta$ and $\{b', x\}$ is a cut pair of $\{v, x\}$ in $G^{\oplus}$.

Proof. We prove Proposition 5 by induction. At the beginning $T_{\circ}(G^{\oplus})$ fulfills the invariant after the execution of e. Suppose further that the previous intermediate tree fulfills the invariant. Now consider the step pair $\{b', x\}$ in a new intermediate tree $T_*(G^{\oplus})$. Let $\{v, x\}$ denote the stem of $\Uparrow_{x(d)}$ in $T(G)$ and assume that $\{v, x\}$ is still a minimum v-x-cut after the change. We show that any minimum b'-x-cut θ in $G^{\oplus}$ costs $c_T(v, x) + \Delta$. Since the cut induced by $\{v, x\}$ exactly cuts off $\Uparrow_{x(d)}$ this cut also separates $b' \notin \Uparrow_{x(d)}$ and x. Thus, θ costs at most $c_T(v, x) + \Delta$. In case of equality $\{b', x\}$ is a cut pair of $\{v, x\}$ in $G^{\oplus}$.

If $b' = v$ or θ separates x and v, the assertion obviously holds. Otherwise, θ does not separate x and v but v and b'. Since x is adjacent to $b' \neq v$, v must have been cut out of $S_{b'}$ in a previous iteration step, i.e., $v \notin S_{b'}$; v is in a subtree $[N]$ linked to S by a nearest cut pair $\{b', n\}$ in $T_*(G^{\oplus})$. Note that the subtree can be linked only b' in $S_{b'}$ since $\Uparrow_{x(d)} \not\ni v$ is closed in $T_*(G^{\oplus})$. Thus, $[N]$ and x are on the same side of θ and θ separates n and b'. However, the minimum b'-n-cut θ_n induced by the link $\{b', n\}$ separates $v \in [N]$ and $x \in S_{b'}$. Thus, $c^{\oplus}(\theta) \geq c^{\oplus}(\theta_n) \geq c_T(v, x) + \Delta$. □



(a) Cut $\theta'_{xb'}$ (solid) is adjusted to dashed cut $\theta_{xb'}$ according to Lemma 6.

(b) New intermediate tree after executing $\theta_{xb'}$.

Fig. 7. Previous and new intermediate tree $T_*(G^{\oplus})$ for scenario (a); $\Uparrow_{x(d)}$ in the previous and $\Uparrow_{y(d)}$ in the new tree are colored in gray. Black vertex denotes end of internal path closer to b, fat edge denotes e linking $S_{b'}$ and $S_{d'}$.

D Omitted Proofs on Handling Marked Nodes

Lemma 7. *In $G^{\ominus}$, let $(U, V \setminus U)$ be a minimum u-v-cut not separating $\{b, d\}$, with $\gamma_{b,d} \cap U = \emptyset$. Then, a cut induced by an edge $\{g, h\}$ in the old tree $T(G)$, with $g, h \in U$, remains a minimum separating cut in $G^{\ominus}$ for all its previous cut pairs within U, and a minimum g-h-cut in particular.*

Proof. Using $(U, V \setminus U)$ as the first split cut of a Gomory-Hu execution for $G^{\ominus}$ yields two adjacent nodes $\{U\}$ and $\{V \setminus U\}$ with $b, d \in \{V \setminus U\}$. From the view of an arbitrary step pair $\{x, y\}$ in $\{U\}$ b and d are sheltered by pseudo-contraction in a common subtree of $\{U\}$. This is, there exists a minimum x-y-cut in $G^{\ominus}$ that does not separate b and d and thus is as expensive as in G, which implies that it was already a minimum x-y-cut in G. It follows that none of the previous minimum x-y-cuts separates b and d and each such cut remains valid in $G^{\ominus}$. In particular, the cut induced by edge $\{g, h\} \subseteq U$ remains a minimum g-h-cut and a minimum separating cut for all its previous cut pairs within U. □

Lemma 8. *There exists no pair $\{u, n\} \in \binom{V}{2}$ such that edge $\{g, h\}$ is a valid minimum u-n-cut in $G^{\oplus}$. In particular, $\{g, h\}$ is no minimum g-h-cut in $G^{\oplus}$.*

Proof. Assume $\{g, h\}$ to be still valid with respect to any pair $\{u, n\}$. Consider the iteration step that formed S. By executing split cut $\{g, h\}$ first ($\{g, h\}$ does not separate b' and x since $\{g, h\}$ is in $\Uparrow_{x(d)}$, which is closed in $T_*(G^{\oplus})$), the minimum b'-x-cut θ would have been bend along $\{g, h\}$ by pseudo-contraction resulting in a minimum b'-x-cut that separates b and d. Thus, the previous minimum b'-x-cut on γ_{bd} would have remained valid and used as split cut by the update algorithm. This contradicts the existence of S. □

E Discussion on Performance

We call cuts in $T(G)$ that remain valid in $G^{\oplus\ominus}$ with respect to their nearest cut pair *stable edges* and all other cuts in $T(G)$ *unstable edges*. We call cuts in $T(G)$

that remain valid in $G^{\oplus\ominus}$ with respect to any cut pair *stable cuts* and all other cuts in $T(G)$ *unstable cuts*. It is

$$\begin{aligned} \textit{stable edges} &\subseteq \textit{stable cuts}, \\ \textit{unstable cuts} &\subseteq \textit{unstable edges}. \end{aligned}$$

The set of *stable cuts*$\backslash$*stable edges* does not necessarily conform to the requirements of Theorem 1. Let M denote a maximum subset of *stable cuts*$\backslash$*stable edges* such that there exists an order of cut pairs that allows for the execution of all cuts in M at the beginning of Gomory-Hu. In the resulting intermediate tree $T_*(G^{\oplus\ominus})$ the stable edges are then executable in an arbitrary order according to Theorem 1. In particular, the cardinality of M does not depend on the stable edges. This is, any optimal cut tree $\hat{T}(G^{\oplus\ominus})$ for $G^{\oplus\ominus}$ providing maximum intersection with $T(G)$ in terms of represented cuts contains all stable edges.

Assuming that we know an optimum tree $\hat{T}(G^{\oplus\ominus})$, thus, the number of cuts that can be carried over from $T(G)$ to $\hat{T}(G^{\oplus\ominus})$ is $|M| + |\textit{stable edges}| \leq |\textit{stable cuts}|$.

Vice versa, the number of cut computations still necessary in a Gomory-Hu execution returning $\hat{T}(G^{\oplus\ominus})$ is $n - 1 - (|M| + |\textit{stable edges})| \geq n - 1 - |\textit{stable cuts}| = |\textit{unstable cuts}|$. The number of necessary cut computations is at most $n-1-|\textit{stable edges}| = |\textit{unstable edges}|$. We define the efficiency of updating a cut tree $T(G)$ yielding $T(G^{\oplus\ominus})$ as

$$\eta(T(G), T(G^{\oplus\ominus})) := \frac{\#\text{ necessary cut computations}}{\#\text{ calculated cuts}} \in [0,1], \text{ with}$$

$$\frac{\#\text{ unstable edges}}{\#\text{ calculated cuts}} \geq \frac{\#\text{ necessary cut computations}}{\#\text{ calculated cuts}} \geq \frac{\#\text{ unstable cuts}}{\#\text{ calculated cuts}}$$

If there are no cuts calculated by the update algorithm, the efficiency is set to 1. We express the number of calculated cuts as well as the number of unstable edges and unstable cuts dependent on the shape of the intermediate tree $T_*(G^{\oplus\ominus})$ after the first phase. This yields an upper and lower bound of the efficiency of an actual update. To this end we introduce the following notions.

- X denotes the set of all marked non-singleton nodes in $T_*(G^{\oplus\ominus})$ after the first phase of the update algorithm.
- H denotes the set of inclusion maximum treetops over all treetops in $T(G)$ apart from initial treetops that are again treetops with respect to the same root in $T(G^{\ominus})$.
- B denotes the set of edges/stems that are marked during the first phase in the case of edge deletion (B particularly contains all bridges in $T(G)$ that are not in $\gamma_{bd} \cup \bigcup_{\Uparrow \in X} \Uparrow$).
- $|\gamma_{bd}|$ counts the edges, $|\gamma_{bd}|_\circ$ the stable edges on path γ_{bd} in $T(G)$.
- $|\Uparrow|$ counts the edges in a treetop in $T(G)$.
- $|P|$ counts the edges in a subpath P of γ_{bd} in $T(G)$.

In the case of edge deletion we get

$$\begin{aligned}
\#\text{ calculated cuts} &= n-1-(|\gamma_{bd}|+|B|+\sum_{\Uparrow\in X}|\Uparrow|)\\
\#\text{ unstable edges} &= n-1-(|\gamma_{bd}|+|B|+\sum_{\Uparrow\in X}|\Uparrow|+|X|)\\
\#\text{ unstable cuts} &\geq n-1-(|\gamma_{bd}|+|B|+\sum_{\Uparrow\in X}|\Uparrow|+|X|)-|H|
\end{aligned}$$

Thus, the number of unnecessarily computed cuts is bounded by the interval $[\,|X|\,,\,|X|+|H|\,]$ with $|H|\in[\,1\,,\,|X|\,]$. In the case of edge insertion we get

$$\begin{aligned}
\#\text{ calculated cuts} &= |\gamma_{bd}|-1\\
\#\text{ unstable edges} &= |\gamma_{bd}|-|\gamma_{bd}|_\circ\\
\#\text{ unstable cuts} &\geq \sum_{P\in X}|P|
\end{aligned}$$

The number of unnecessarily computed cuts is in $[\,|\gamma_{bd}|_\circ-1\,,\,|\gamma_{bd}|-\sum_{P\in X}|P|\,]$ In the following we exemplarily analyze the cases of efficiency 1 for updates after edge deletion.

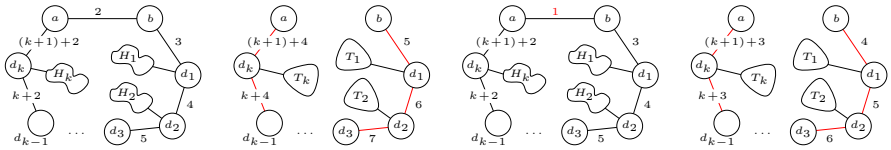
Efficiency 1 *for Edge Deletion.* Efficiency 1 occurs if either (a) no cut is calculated or (b) each calculated cut is necessary.

(a) [no calculated cut] In this case $T(G)$ is a tree where the stem of each subtree linked to γ_{bd} is a bridge. The new tree $T(G^{\ominus})$ equals $T(G)$ apart from adjusted costs on γ_{bd}, i.e., all cuts in $T(G)$ are carried over to $T(G^{\ominus})=\hat{T}(G^{\ominus})$, which is an optimum tree. Fig. 8 shows an example.

(b) [each calculated cut necessary] In this case $T(G)$ is a tree in which no stable cuts exist apart from the stable edges on γ_{bd} and possibly subtrees linked to γ_{bd} by bridges. The new tree $T(G^{\ominus})$ consists of a path between b and d where the subtrees linked to this path correspond to the subtrees linked to γ_{bd} by bridges in $T(G)$. Since $T(G)$ contains no real stable cuts (which are no stable edges) the new tree $T(G^{\ominus})=\hat{T}(G^{\ominus})$ is optimum. Fig. 9 shows an example with a maximum number of saved cut computations ($=n-2$). Fig. 10 shows an example with a minimum number of saved cut computations ($=1$).

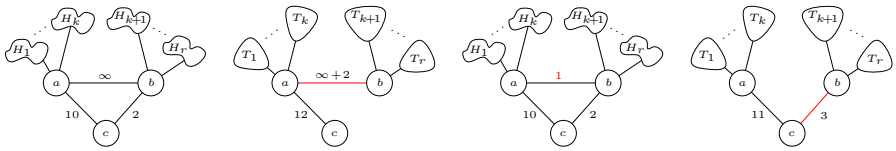
In both cases the number of saved cut computations compared to a construction from scratch is $|\gamma_{bd}|+|B|$ ($=n-1$ in (a)).

Maximum Efficiency <1 *for Edge Deletion.* The maximum possible efficiency different from 1 is reached if there is one more cut calculated than necessary. Figure 11 shows that for each $\epsilon>0$ exists a graph with efficiency at least $1-\epsilon$.



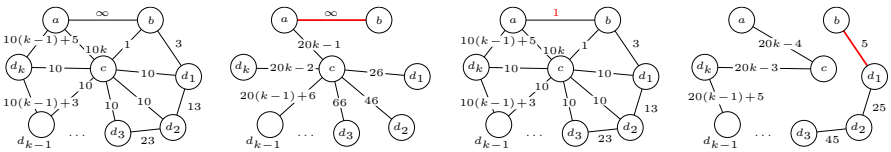
(a) Graph G. (b) Cut tree $T(G)$, γ_{ab} red colored. (c) Graph $G^{\ominus}$. (d) Cut tree $T(G^{\ominus})$, stable edges in red.

Fig. 8. Deletion: efficiency 1, no calculated cuts; subgraphs H_i linked by bridges.



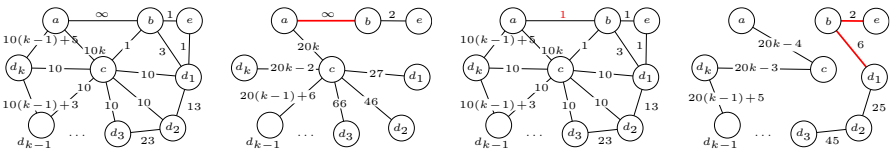
(a) Graph G. (b) Cut tree $T(G)$, γ_{ab} red colored. (c) Graph $G^{\ominus}$. (d) Cut tree $T(G^{\ominus})$, stable edges in red.

Fig. 9. Deletion: efficiency 1, each calculated cut (edge $\{a, c\}$ in $T(G)$) necessary, maximum number of saved cut computations $= n - 2$; subgraphs H_i linked by bridges.



(a) Graph G. (b) Cut tree $T(G)$, γ_{ab} red colored. (c) Graph $G^{\ominus}$. (d) Cut tree $T(G^{\ominus})$, stable edges in red.

Fig. 10. Deletion: efficiency 1, each calculated cut (all but $\{a, b\}$ in $T(G)$) necessary, minimum number of saved cut computations $= 1$.



(a) Graph G. (b) Cut tree $T(G)$, γ_{ab} red colored. (c) Graph $G^{\ominus}$. (d) Cut tree $T(G^{\ominus})$, stable edges in red.

Fig. 11. Deletion: efficiency $\frac{k+1}{k+2}$. All cuts in $T(G) \setminus \gamma_{ab}$ apart from $\{b, e\}$ are unstable cuts, one cut computations saved, one unnecessary cut computation if we consider only bridges when marking edges. Otherwise, the algorithm would have found $c(b, e) = 2$ in $T(G)$ being cheaper than the minimum a-b-cut in $G^{\ominus}$ (which costs 6), and thus, $\{b, e\}$ would have been marked according to (f.) instead of a new cut computation.

F Some Pseudo Code

Algorithm 2: EDGE DELETION

Input: $T(G)$, $G^{\ominus}$, edge $\{b,d\}$, costs Δ
Output: $T(G^{\ominus})$

Calculate path γ_{bd} in $T(G)$
if $\gamma_{b,d} = \{b,d\}$ with $c_T(\{b,d\}) = \Delta$ **then** // $\{b,d\}$ is a bridge in G
    $c_T(\{b,d\}) \leftarrow 0$
    return $T(G)$ // tree decomposes by deletion of zero-weighted edge
if path γ_{bd} spans V **then** // $T(G)$ is path from b to d
    forall the edges e in $T(G)$ **do**
        $c_T(e) \leftarrow c_T(e) - \Delta$ // edges on γ_{bd} remain, Lemma 1
    return $T(G)$
$T_*(G^{\ominus}) \leftarrow$calculate tree $T_\circ(G^{\ominus})$ by contracting edges *off* γ_{bd} // see Fig. 1(a)
Consider nodes S in $T_*(G^{\ominus})$ as unmarked
Consider vertices on γ_{bd} as *centers* $c(S)$ of their ambient nodes S in $T_*(G^{\ominus})$
forall the vertices u *off* γ_{bd} **do**
    $L(u)$, $\Theta(u) \leftarrow \emptyset$, $D(u) \leftarrow \{u\}$ // global variables
    Calculate and store $\Uparrow_u$ in $T(G)$ // as global variable
while there are unmarked nodes S in $T_*(G^{\ominus})$ with $|S| > 1$ **do**
    $\delta \leftarrow$ minimum costs of edges incident to S on $\gamma_{S_b S_d}$ in $T_*(G^{\ominus})$
    $T_*(G^{\ominus}) \leftarrow$ checkStar$(G^{\ominus}, T_*(G^{\ominus}), S, \delta)$ // checkStar changes $T_*(G^{\ominus})$
forall the marked nodes S in $T_*(G^{\ominus})$ with $|S| > 1$ **do** // S remains treetop
    Replace S by the corresponding treetop // by Lemma 7
return $T_*(G^{\ominus})$ with singletons replaced by vertices

Procedure checkStar

Input: $G^{\ominus}$, $T_*(G^{\ominus})$ with current centers, S, δ
Output: $T_*(G^{\ominus})$ after unfolding lowest star in S

Add vertices adjacent to $c(S)$ in S to $L(c(S))$ `// build star, centered at` c
while $L(c(S))$ has next element u **do** `//` $L(c(S))$ `may change in loop`
  if $\Theta(c(S))$ does not yet contain cut θ_u **then**
    if *stem of* $\Uparrow_u$ *is bridge in* G **then**
      $\theta_u \leftarrow \theta_u^{\text{old}}$ `// retain old cut by Lemma 4`
    else
      if $c(\theta_u^{\text{old}}) < \delta$ **then**
        $\theta_u \leftarrow \theta_u^{\text{old}}$ `// retain old cut according to (f.)`
      else
        $\theta_u \leftarrow$ minimum u-$c(S)$-cut given by FlowAlgo $(u, c(S), G^{\ominus})$
    Add θ_u to $\Theta(c(S))$ pointed at by u
  if $c^{\ominus}(\theta_u) = c(\theta_u^{\text{old}})$ **then** `// retain cut and treetop` $\Uparrow_u$ `acc. to (e.)`
    Remove u from $L(c(S))$, remove θ_u from $\Theta(c(S))$
    $T_*(G^{\ominus}) \leftarrow$ `splitAndReconnect`$(T_*(G^{\ominus}), S, u, \theta_u^{\text{old}})$
    Mark S_u as remaining treetop, $S \leftarrow S_{c(S)}$
  else
    while $L(c(S))$ *has next element* $\bar{u} \neq u$ **do** `// test vs. other cuts`
      if θ_u *separates* $\bar{u}$ *and* $c(S)$ **then** `// cp. to Scenario (a,c)`
        Remove $\bar{u}$ from $L(c(S))$, $D(u) \leftarrow D(u) \cup D(\bar{u})$
        Add $\bar{u}$ to $L(u)$ `//` $\{\bar{u}, u\}$ `becomes edge that is no stem`
        if $\Theta(c(S))$ already contains cut $\theta_{\bar{u}}$ **then** `// pointed at by` $\bar{u}$
          Move $\theta_{\bar{u}}$ from $\Theta(c(S))$ to $\Theta(u)$ `// by Lemma 2`
forall the $u \in L(c(S))$ **do** `// make new cuts treetop-preserving`
  set $(R, V \setminus R) := \theta_u$, with $c(S) \in R$ for $\theta_u \in \Theta(c(S))$ pointed at by u
  forall the $\bar{u}$ *in* $D(u)$ **do** `// handle shadowed cuts ...`
    $\theta_u \leftarrow (R \setminus \Uparrow_{\bar{u}}, (V \setminus R) \cup \Uparrow_{\bar{u}})$ `// ...with Lemma 5 and Scenario (a,c)`
  forall the $\bar{u} \neq u$ *in* $L(c(S))$ **do** `// handle other cuts ...`
    $\theta_u \leftarrow (R \cup \Uparrow_{\bar{u}}, (V \setminus R) \setminus \Uparrow_{\bar{u}})$ `// ...with Scenario (b)`
  $T_*(G^{\ominus}) \leftarrow$ `splitAndReconnect`$(T_*(G^{\ominus}), S, u, \theta_u)$
  $S \leftarrow S_{c(S)}$
return $T_*(G^{\ominus})$

Procedure splitAndReconnect

Input: $T_*(G^{\ominus})$ with current centers, S, u in step pair $\{u, c(S)\}$, cut $(U, V \setminus U)$ in $G^{\ominus}$ with $u \in U$

Output: $T_*(G^{\ominus})$ after splitting S

Split S into $S_{c(S)} = S \cap V \setminus U$ and $S_u = S \cap U$
$c(S_{c(S)}) \leftarrow c(S)$
$c(S_u) \leftarrow u$
Create new edge $\{S_{c(S)}, S_u\}$ in $T_*(G^{\ominus})$ with costs $c(U, V \setminus U)$ in $G^{\ominus}$
forall the edges $e_j = \{S, S_j\}$ previously incident to S in $T_*(G^{\ominus})$ **do**
    if $c(S_j) \in U$ **then**
        Reconnect subtree N_j with S_u in $T_*(G^{\ominus})$
    else
        Reconnect subtree N_j with $S_{c(S)}$ in $T_*(G^{\ominus})$
return $T_*(G^{\ominus})$