Fast and Simple 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 can be iteratively constructed by $n-1$ maximum flow computations. They solve the multiterminal network flow problem, which asks for the all-pairs maximum flow values in a network and at the same time they represent $n-1$ non-crossing, linearly independent cuts that constitute a minimum cut basis of G. 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 a fully-dynamic algorithm that efficiently maintains a cut tree for a changing graph. The algorithm is easy to implement and has a high potential for saving cut computations under the assumption that a local change in the underlying graph does rarely affect the global cut structure. We document the good practicability of our approach in a brief experiment on real world data.

1 Introduction

A *cut tree* is a weighted tree $T(G) = (V,E_T,c_T)$ on the vertices of an undirected (weighted) graph $G = (V,E,c)$ (with edges not necessarily in G) such that each $\{u,v\} \in E_T$ induces a minimum u-v-cut in G (by decomposing $T(G)$ into two connected components) and such that $c_T(\{u,v\})$ is equal to the cost of the induced cut. The cuts induced by $T(G)$ are non-crossing and for each $\{x,y\} \subseteq V$ each cheapest edge on the path $\pi(x,y)$ between x and y in $T(G)$ corresponds to a minimum x-y-cut in G. If G is disconnected, $T(G)$ contains edges of cost 0 between connected components.

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 considering 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 a maximum s-t-flow (or minimum s-t-cut) regarding all possible capacities of the parametric edges. 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 nor in one minimum s-t-cut regarding *all possible* changes of varying edges. Instead we face

⋆ 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.

a concrete 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. To the best of our knowledge no fully-dynamic approach for updating cut trees exists. Coming from sensitivity analysis, Barth et al. [4] state that after the capacity of an edge has increased the path in $T(G)$ between the vertices that define the changing edge in G is the only part of a given cut tree that needs to be recomputed, which is rather obvious. Besides they stress the difficulty for the case of decreasing edge capacities.

In our work we formulate a general condition for the (re)use of given cuts in an (iterative) cut tree construction, which directly implies the result of Barth et al. We further solve the case of decreasing edge capacities showing by an experiment that this has a similar potential for saving cut computations like the case of increasing capacities. In the spirit of Gusfield [2], who simplified the pioneering cut tree algorithm of Gomory and Hu [1], we also allow the use of crossing cuts and give a representation of intermediate trees (during the iteration) that makes our approach very easy to implement.

We give our notational conventions and a first folklore insight in Sec. 1. In Sec. 2 we revisit the static cut tree algorithm [1] and the key for its simplification [2], and construct a first intermediate cut tree by reusing cuts that obviously remain valid after a change in G. We also state several lemmas that imply techniques to find further reusable cuts in this section. Our update approach is described in Sec. 3. In Sec. 4 we finally discuss the performance of our algorithm based on a brief experiment. Proofs omitted due to space constraints can be found in the full paper [7].

Preliminaries and Notation. In this work we consider an undirected, weighted graph $G = (V,E,c)$ with vertices V, edges E and a positive edge cost function c, writing $c(u,v)$ as a shorthand for $c(\{u,v\})$ with $\{u,v\} \in E$. We reserve the term *node* for compound vertices of abstracted graphs, which may contain several basic vertices of a concrete graph; however, we identify singleton nodes with the contained vertex without further notice. *Contracting* a set $N \subseteq V$ in G means replacing N by a single node, and leaving this node adjacent to all former adjacencies u of vertices of N, with an edge cost equal to the sum of all former edges between N and u. Analogously we contract a set $M \subseteq E$ or a subgraph of G by contracting the corresponding vertices.

A *cut* in G is a partition of V into two *cut sides* S and $V \setminus S$. The cost $c(S, V \setminus S)$ of a cut is the sum of the costs of all edges *crossing* the cut, i.e., edges $\{u,v\}$ with $u \in S$, $v \in V \setminus S$. For two disjoint sets $A, B \subseteq V$ we define the cost $c(A,B)$ analogously. Note that a cut is defined by the edges crossing it. Two cuts are *non-crossing* if their cut sides are pairwise nested or disjoint. Two vertices $u, v \in V$ are *separated* by a cut if they lie on different cut sides. A minimum u-v-cut is a cut that separates u and v and is the cheapest cut among all cuts separating these vertices. We call a cut a *minimum separating cut* if there exists an arbitrary vertex pair $\{u,v\}$ for which it is a minimum u-v-cut; $\{u,v\}$ is called a *cut pair* of the minimum separating cut. We further denote the *connectivity* of $\{u,v\} \subseteq V$ by $\lambda(u,v)$, describing the cost of a minimum u-v-cut.

Since each edge in a tree $T(G)$ on the vertices of G induces a unique cut in G, we identify tree edges with corresponding cuts without further notice. This allows for saying that a vertex is *incident* to a cut and an edge *separates* a pair of vertices. We consider the path $\pi(u,v)$ between u and v in $T(G)$ as the set of edges or the set of vertices on it, as convenient.

A change in G either involves an edge $\{b,d\}$ or a vertex b. If the cost of $\{b,d\}$ in G descreases by $\Delta > 0$ or $\{b,d\}$ with $c(b,d) = \Delta > 0$ is deleted, the change yields $G^{\ominus}$. Analogously, inserting $\{b,d\}$ or increasing the cost yields $G^{\oplus}$. We denote the cost function after a change by $c^{\ominus}$ and $c^{\oplus}$, the connectivity by $\lambda^{\ominus}$ and $\lambda^{\oplus}$, respectively. We assume that only degree-0 vertices can be deleted from G. Hence, inserting or deleting b changes neither the cost function nor the connectivity. We start with a fundamental insight on the reusability of cuts. Recall that $T(G) = (V, E_T, c_T)$ denotes a cut tree.

Lemma 1. *If $c(b,d)$ changes by $\Delta > 0$, then each $\{u,v\} \in E_T$ remains a minimum u-v-cut (i) in $G^{\oplus}$ with cost $\lambda(u,v)$ if $\{u,v\} \notin \pi(b,d)$, (ii) in $G^{\ominus}$ with cost $\lambda(u,v) - \Delta$ if $\{u,v\} \in \pi(b,d)$.*

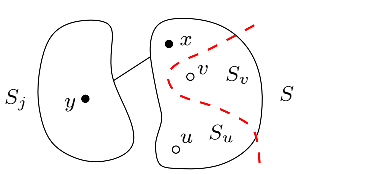
2 The Static Algorithm and Insights on Reusable Cuts

The Static Algorithm. As a basis for our dynamic approach, we briefly revisit the static construction of a cut tree [1,2]. This algorithm iteratively constructs $n-1$ non-crossing minimum separating cuts for $n-1$ vertex pairs, which we call *step pairs*. These pairs are chosen arbitrarily from the set of pairs not separated by any of the cuts constructed so far. Algorithm 1 briefly describes the cut tree algorithm of Gomory and Hu.

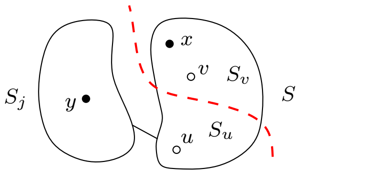
Algorithm 1. CUT TREE

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

The *intermediate* cut tree $T_* = (V_*, E_*, c_*)$ is initialized as an isolated, edgeless node containing all original vertices. Then, until each node of T_* is a singleton node, a node $S \in V_*$ is *split*. To this end, nodes $S' \neq S$ 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 $[N_j]$ before cutting, which yields G_S. The split of S into S_u and 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



(a) If $x \in S_u$, $\{x,y\}$ is still a cut pair of $\{S_u,S_j\}$



(b) If $x \notin S_u$, $\{u,y\}$ is a cut pair of $\{S_u,S_j\}$

Fig. 1. Situation in Lemma 2. There always exists a cut pair of the edge $\{S_u,S_j\}$ in the nodes incident to the edge, independent of the shape of the split cut (dashed).

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 minimum u-v-cut in G.

The correctness of CUT TREE is guaranteed by Lemma 2, which takes care for the *cut pairs* of the reconnected edges. It states that each edge $\{S,S'\}$ in T_* has a cut pair $\{x,y\}$ with $x \in S$, $y \in S'$. An intermediate cut tree satisfying this condition is *valid*. The assertion is not obvious, since the nodes incident to the edges in T_* change whenever the edges are reconnected. Nevertheless, each edge in the final cut tree represents a minimum separating cut of its incident vertices, due to Lemma 2. The lemma was formulated and proven in [1] and rephrased in [2]. See Figure 1.

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

While Gomory and Hu use contractions in G to prevent crossings of the cuts, as a simplification, Gusfield introduced the following lemma showing that contractions are not necessary, since any arbitrary minimum separating cut can be bent along the previous cuts resolving any potential crossings. See Figure 2.

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

We say that $(X,V \setminus X)$ *shelters* X, meaning that each minimum u-v-cut with $u,v \notin X$ can be reshaped, such that it does no longer split X.

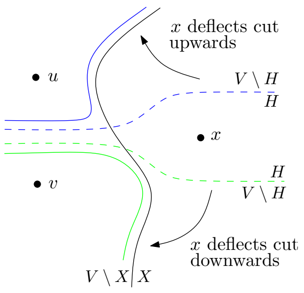


Fig. 2. Depending on x Lem. 3 bends the cut $(H,V \setminus H)$ upwards or downwards

Representation of Intermediate Trees. In the remainder of this work we represent each node in T_*, which consists of original vertices in G, by an arbitrary tree of *thin* edges connecting the contained vertices in order to indicate their membership to the node. An edge connecting two nodes in T_* is represented by a *fat* edge, which we connect to an arbitrary vertex in each incident compound node. Fat edges represent minimum separating cuts in G. If a node contains only one vertex, we color this vertex black. Black vertices are only incident to fat edges. The vertices in non-singleton nodes are colored white. White vertices are incident to at least one thin edge. In this way, T_* becomes a tree on V with two types of edges and vertices. For an example see Figure 3.

Conditions for Reusing Cuts. Consider a set K of $k \leq n-1$ cuts in G for example given by a previous cut tree in a dynamic scenario. The following theorem states sufficient conditions for K, such that there exists a valid intermediate cut tree that represents exactly the cuts in K. Such a tree can then be further processed to a proper tree by CUT TREE, saving at least $|K|$ cut computations compared to a construction from scratch.

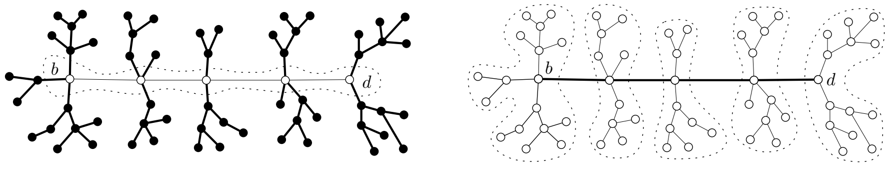
Theorem 1. *Let K denote a set of non-crossing minimum separating cuts in G and let F denote a set of associated cut pairs such that each cut in K separates exactly one pair in F. Then there exists a valid intermediate cut tree representing exactly the cuts in K.*

Proof. Theorem 1 follows inductively from the correctness of CUT TREE. Consider a run of CUT TREE that uses the elements in F as step pairs in an arbitrary order and the associated cuts in K as split cuts. Since the cuts in K are non-crossing each separating exactly one cut pair in F, splitting a node neither causes reconnections nor the separation of a pair that was not yet considered. Thus, CUT TREE reaches an intermediate tree representing the cuts in K with the cut pairs located in the incident nodes. □

With the help of Theorem 1 we can now construct a valid intermediate cut tree from the cuts that remain valid after a change of G according to Lemma 1. These cuts are non-crossing as they are represented by tree edges, and the vertices incident to these edges constitute a set of cut pairs as required by Theorem 1. The resulting tree for an inserted edge or an increased edge cost is shown in Figure 3(a). In this case, all but the edges on $\pi(b,d)$ can be reused. Hence, we draw these edges fat. The remaining edges are thinly drawn. The vertices are colored according to the compound nodes indicated by the thickness of the edges. Vertices incident to a fat edge correspond to a cut pair.

For a deleted edge or a decreased edge cost, the edges on $\pi(b,d)$ are fat, while the edges that do not lie on $\pi(b,d)$ are thin (cp. Figure 3(b)). Furthermore, the costs of the fat edges decrease by Δ, since they all cross the changing edge $\{b,d\}$ in G. Compared to a construction from scratch, starting the CUT TREE routine from these intermediate trees already saves $n-1-|\pi(b,d)|$ cut computations in the first case and $|\pi(b,d)|$ cut computations in the second case, where $|\pi(b,d)|$ counts the edges on $\pi(b,d)$. Hence, in scenarios with only little varying path lengths and a balanced number of increasing and decreasing costs, we can already save about half of the cut computations. We further remark that the result of Barth et. al. [4], who costly prove the existence of the intermediate cut tree in Figure 3(a), easily follows by Theorem 1 applied to the cuts in Lemma 1 as seen above. In the following we want to use even more information from the previous cut tree $T(G)$ when executing CUT TREE unfolding the intermediate tree to a proper cut tree of $(n-1)$ fat edges. The next section lists further lemmas that allow the reuse of cuts already given by $T(G)$.

Further Reusable Cuts. In this section we focus on the reuse of those cuts that are still represented by thin edges in Figure 3. If $\{b,d\}$ is inserted or the cost increases, the following corollary obviously holds, since $\{b,d\}$ crosses each minimum b-d-cut.



(a) Intermediate cut tree for $G^{\oplus}$. (b) Intermediate cut tree for $G^{\ominus}$.

Fig. 3. Intermediate cut trees in dynamic scenarios. Fat edges represent valid minimum cuts, thin edges indicate compound nodes. Contracting the thin edges yields nodes of white vertices (indicated by dotted lines). Black vertices correspond to singletons.

Corollary 1. *If $\{b,d\}$ is newly inserted with $c^{\oplus}(b,d) = \Delta$ or $c(b,d)$ increases by Δ, any minimum b-d-cut in G remains valid in $G^{\oplus}$ with $\lambda^{\oplus}(b,d) = \lambda(b,d) + \Delta$.*

Note that reusing a valid minimum b-d-cut as split cut in CUT TREE separates b and d such that $\{b,d\}$ cannot be used again as step pair in a later iteration step. This is, we can reuse only one minimum b-d-cut, even if there are several such cuts represented in $T(G)$. Together with the following corollary, Corollary 1 directly allows the reuse of the whole cut tree $T(G)$ if $\{b,d\}$ is an existing bridge in G (with increasing cost).

Corollary 2. *An edge $\{u,v\}$ is a bridge in G iff $c(u,v) = \lambda(u,v) > 0$. Then $\{u,v\}$ is also an edge in $T(G)$ representing the cut that is given by the two sides of the bridge.*

While the first part of Corollary 2 is obvious, the second part follows by the fact that a bridge induces a minimum separating cut for all vertices on different bridge sides, while it does not cross any minimum separating cut of vertices on a common side. If G is disconnected and $\{b,d\}$ is a new bridge in $G^{\oplus}$, reusing the whole tree is also possible by replacing a single edge. Such bridges can be easily detected having the cut tree $T(G)$ at hand, since $\{b,d\}$ is a new bridge if and only if $\lambda(b,d) = 0$. New bridges particularly occur if newly inserted vertices are connected for the first time.

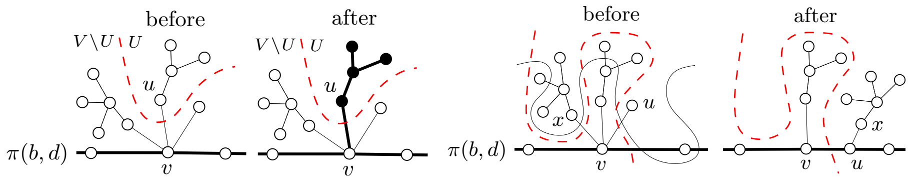
Lemma 4. *Let $\{b,d\}$ be a new bridge in $G^{\oplus}$. Then replacing an edge of cost 0 by $\{b,d\}$ with cost $c^{\oplus}(b,d)$ on $\pi(b,d)$ in $T(G)$ yields a new cut tree $T(G^{\oplus})$.*

If $\{b,d\}$ is deleted or the cost decreases, handling bridges (always detectable by Corollary 2) is also easy.

Lemma 5. *If $\{b,d\}$ is a bridge in G and the cost decreases by Δ (or $\{b,d\}$ is deleted), decreasing the edge cost on $\pi(b,d)$ in $T(G)$ by Δ yields a new cut tree $T(G^{\ominus})$.*

If $\{b,d\}$ is no bridge, at least other bridges in G can still be reused if $\{b,d\}$ is deleted or the edge cost decreases. Observe that a minimum separating cut in G only becomes invalid in $G^{\ominus}$ if there is a cheaper cut in $G^{\ominus}$ that separates the same vertex pair. Such a cut necessarily crosses the changing edge $\{b,d\}$ in G, since otherwise it would have been already cheaper in G. Hence, an edge in E_T corresponding to a bridge in G cannot become invalid, since any cut in $G^{\ominus}$ that crosses $\{b,d\}$ besides the bridge would be more expensive. In particular, this also holds for zero-weighted edges in E_T.

Corollary 3. *Let $\{u,v\}$ denote an edge in $T(G)$ with $c_T(u,v) = 0$ or an edge that corresponds to a bridge in G. Then $\{u,v\}$ is still a minimum u-v-cut in $G^{\ominus}$.*



(a) Edges in U remain valid, cp. Lemma 6. (b) Reshaping new cut by reconnecting edges.

Fig. 4. (a) cut $\{u,v\}$ remains valid, subtree U can be reused. (b) new cheaper cut for $\{u,v\}$ (black) can be reshaped by Theo. 2, Lem. 3 (dashed), $\{u,v\}$ becomes a fat edge.

Lemma 6 shows how a cut that is still valid in $G^{\ominus}$ may allow the reuse of all edges in E_T that lie on one cut side. Figure 4(a) shows an example. Lemma 7 says that a cut that is cheap enough, cannot become invalid in $G^{\ominus}$. Note that the bound considered in this context depends on the current intermediate tree.

Lemma 6. *Let $(U, V \setminus U)$ be a minimum u-v-cut in $G^{\ominus}$ with $\{b,d\} \subseteq V \setminus U$ and $\{g,h\} \in E_T$ with $g,h \in U$. Then $\{g,h\}$ is a minimum separating cut in $G^{\ominus}$ for all its previous cut pairs within U.*

Lemma 7. *Let $T_* = (V, E_*, c_*)$ denote a valid intermediate cut tree for $G^{\ominus}$, where all edges on $\pi(b,d)$ are fat and let $\{u,v\}$ be a thin edge with v on $\pi(b,d)$ such that $\{u,v\}$ represents a minimum u-v-cut in G. Let N_π denote the set of neighbors of v on $\pi(b,d)$. If $\lambda(u,v) < \min_{x \in N_\pi}\{c_*(x,v)\}$, then $\{u,v\}$ is a minimum u-v-cut in $G^{\ominus}$.*

3 The Dynamic Cut Tree Algorithm

In this section we introduce one update routine for each type of change: inserting a vertex, deleting a vertex, increasing an edge cost or inserting an edge, decreasing an edge cost or deleting an edge. These routines base on the static iterative approach but involve the lemmas from Sec. 2 in order to save cut computations. We again represent intermediate cut trees by fat and thin edges, which simplifies the reshaping of cuts.

We start with the routines for vertex insertion and deletion, which trivially abandon cut computations. We leave the rather basic proofs of correctness to the reader. A vertex b inserted into G forms a connected component in $G^{\oplus}$. Hence, we insert b into $T(G)$ connecting it to the remaining tree by an arbitrary zero-weighted edge. If b is deleted from G, it was a single connected component in G before. Hence, in $T(G)$ b is only incident to zero-weighted edges. Deleting b from $T(G)$ and reconnecting the resulting subtrees by arbitrary edges of cost 0 yields a valid intermediate cut tree for $G^{\ominus}$.

The routine for increasing an edge cost or inserting an edge first checks if $\{b,d\}$ is a (maybe newly inserted) bridge in G. In this case, it adapts $c_T(b,d)$ according to Corollary 1 if $\{b,d\}$ already exists in G, and rebuilds $T(G)$ according to Lemma 4 otherwise. Both requires no cut computation. If $\{b,d\}$ is no bridge, the routine constructs the intermediate cut tree shown in Figure 3(a), reusing all edges that are not on $\pi(b,d)$. Furthermore, it chooses one edge on $\pi(b,d)$ that represents a minimum b-d-cut in $G^{\oplus}$ and draws this edge fat (cp. Corollary 1). The resulting tree is then further processed by CUT TREE, which costs $|\pi(b,d)| - 1$ cut computations and is correct by Theorem 1.

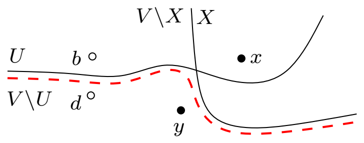
Algorithm 2. Decrease or Delete

```
   Input: $T(G)$, $b,d$, $c(b,d)$, $c^{\ominus}(b,d)$, $\Delta := c(b,d) - c^{\ominus}(b,d)$
   Output: $T(G^{\ominus})$
1  $T_* \leftarrow T(G)$
2  if $\{b,d\}$ is a bridge then apply Lemma 5; return $T(G^{\ominus}) \leftarrow T_*$
3  Construct intermediate tree according to Figure 3(b)
4  $Q \leftarrow$ thin edges non-increasingly ordered by their costs
5  while $Q \neq \emptyset$ do
6      $\{u,v\} \leftarrow$ most expensive thin edge with $v$ on $\pi(b,d)$
7      $N_\pi \leftarrow$ neighbors of $v$ on $\pi(b,d)$; $L \leftarrow \min_{x \in N_\pi}\{c_*(x,v)\}$
8      if $L > \lambda(u,v)$ or $\{u,v\} \in E$ with $\lambda(u,v) = c(u,v)$ then    // Lem. 7 and Cor. 3
9          draw $\{u,v\}$ as a fat edge
10         consider the subtree $U$ rooted at $u$ with $v \notin U$,    // Lem. 6 and Fig. 4(a)
11         draw all edges in $U$ fat, remove fat edges from $Q$
12         continue loop
13     $(U, V \setminus U) \leftarrow$ minimum $u$-$v$-cut in $G^{\ominus}$ with $u \in U$
14     draw $\{u,v\}$ as a fat edge, remove $\{u,v\}$ from $Q$
15     if $\lambda(u,v) = c^{\ominus}(U, V \setminus U)$ then goto line 10    // old cut still valid
16     $c_*(u,v) \leftarrow c^{\ominus}(U, V \setminus U)$    // otherwise
17     $N \leftarrow$ neighbors of $v$
18     forall the $x \in N$ do    // bend split cut by Theo. 2 and Lem. 3
19         if $x \in U$ then reconnect $x$ to $u$
20 return $T(G^{\ominus}) \leftarrow T_*$
```

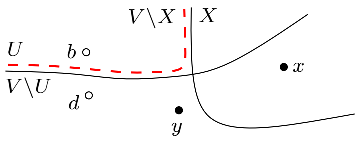
The routine for decreasing an edge cost or deleting an edge is given by Algorithm 2. We assume G and $G^{\ominus}$ to be available as global variables. Whenever the intermediate tree T_* changes during the run of Algorithm 2, the path $\pi(b,d)$ is implicitly updated without further notice. Thin edges are weighted by the old connectivity, fat edges by the new connectivity of their incident vertices. Whenever a vertex is reconnected, the newly occurring edge inherits the cost and the thickness from the disappearing edge.

Algorithm 2 starts by checking if $\{b,d\}$ is a bridge (line 2) and reuses the whole cut tree $T(G)$ with adapted cost $c_T(b,d)$ (cp. Lemma 5) in this case. Otherwise (line 3), it constructs the intermediate tree shown in Figure 3(b), reusing all edges on $\pi(b,d)$ with adapted costs. Then it proceeds with iterative steps similar to Cut Tree. However, the difference is, that the step pairs are not chosen arbitrarily, but according to the edges in $T(G)$, starting with those edges that are incident to a vertex v on $\pi(b,d)$ (line 6). In this way, each edge $\{u,v\}$ which is found to remain valid in line 8 or line 15 allows to retain a maximal subtree (cp. Lemma 6), since $\{u,v\}$ is as close as possible to $\pi(b,d)$. The problem however is that cuts that are no longer valid, must be replaced by new cuts, which not necessarily respect the tree structure of $T(G)$. This is, a new cut possibly separates adjacent vertices in $T(G)$, which hence cannot be used as a step pair in a later step. Thus, we potentially miss valid cuts and the chance to retain further subtrees.

We solve this problem by reshaping the new cuts in the spirit of Gusfield. Theorem 2 shows how arbitrary cuts in $G^{\ominus}$ (that separate b and d) can be bend along old minimum separating cuts in G without becoming more expensive (see Figure 5).



(a) Deflected by x, Theorem 2(i) bends $(U,V\setminus U)$ downwards along X.



(b) Deflected by x, Theorem 2(ii) bends $(U,V\setminus U)$ upwards along X.

Fig. 5. Situation of Theorem 2. Reshaping cuts in $G^{\ominus}$ along previous cuts in G.

Theorem 2. *Let $(X,V\setminus X)$ denote a minimum x-y-cut in G with $x\in X$, $y\in V\setminus X$ and $\{b,d\}\subseteq V\setminus X$. Let further $(U,V\setminus U)$ denote a cut that separates b,d. If (i) $(U,V\setminus U)$ separates x,y with $x\in U$, then $c^{\ominus}(U\cup X,V\setminus(U\cup X))\le c^{\ominus}(U,V\setminus U)$. If (ii) $(U,V\setminus U)$ does not separate x,y with $x\in V\setminus U$, then $c^{\ominus}(U\setminus X,V\setminus(U\setminus X))\le c^{\ominus}(U,V\setminus U)$.*

Since any new cheaper cut found in line 13 needs to separate b and d, we can apply Theorem 2 to this cut regarding the old cuts that are induced by the other thin edges $\{x,v\}$ incident to v. As a result, the new cut gets reshaped without changing its cost such that each subtree rooted at a vertex x is completely assigned to either side of the reshaped cut (line 19), depending on if the new cut separates x and v (cp. Figure 4(b)). Furthermore, Lemma 3 allows the reshaping of the new cut along the cuts induced by the fat edge on $\pi(b,d)$ that are incident to v. This ensures that the new cut does not cross parts of T_* that are beyond these flanking fat edges. Since after the reshaping exact one vertex adjacent to v on $\pi(b,d)$ ends up on the same cut side as u, u finally becomes a part of $\pi(b,d)$.

It remains to show that after the reconnection the reconnected edges are still incident to one of their cut pairs in $G^{\ominus}$ (for fat edges) and G (for thin edges), respectively. For fat edges this holds according to Lemma 2. For thin edges the order in line 4 guarantees that an edge $\{x,v\}$ that will be reconnected to u in line 19 is at most as expensive as the current edge $\{u,v\}$, and thus, also induces a minimum u-x-cut in G. This allows applying Lemma 6 and 7 as well as the comparison in line 15 to reconnected thin edges, too. Observe that an edge corresponding to a bridge never crosses a new cheaper cut, and thus, gets never reconnected. In the end all edges in T_* are fat, since each edge is either a part of a reused subtree or was considered in line 6. Note that reconnecting a thin edge makes this edge incident to a vertex on $\pi(b,d)$ and decrements the hight of the related subtree.

4 Performance of the Algorithm

Unfortunately we cannot give a meaningful guarantee on the number of saved cut computations. The saving depends on the length of the path $\pi(b,d)$, the number of $\{u,v\}\in E_T$ for which the connectivity $\lambda(u,v)$ changes, and the shape of the cut tree. In a star, for example, there exist no subtrees that could be reused by Lemma 6 (see Figure 6 (left) for a bad case example for edge deletion). Nevertheless, a first experimental proof of concept promises high practicability, particularly on graphs with less regular cut structures. The instance we use is a network of e-mail communications within the

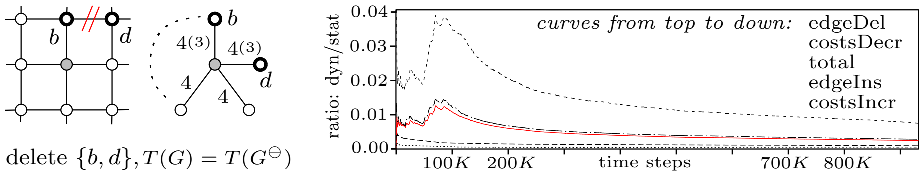


Fig. 6. left: $T(G)$ could be reused (new cost on $\pi(b,d)$ in brackets), but Alg. 2 computes $n-3$ cuts. right: Cumulative ratio of dynamic and static cut computations.

Department of Informatics at KIT [8]. Vertices represent members, 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 924 900 elementary changes, which indicate the time steps in Figure 6 (right), and 923 031 of which concern edges. We start with an empty graph, constructing the network from scratch. Figure 6 shows the ratio of cuts computed by the update algorithm and cuts needed by the static approach until the particular time step. The ratio is shown in total, and broken down to edge insertions (151 169 occurrences), increasing costs (310473), edge deletions (151 061) and decreasing costs (310 328). The trend of the curves follows the evolution of the graph, which slightly densifies around time step 100 000 due to a spam-attack; however, the update algorithm needs less than 4% of the static computations even during this period. We further observe that for decreasing costs, Theorem 2 together with Lemma 3 allows to contract all subtrees incident to the current vertex v on $\pi(b,d)$, which shrinks the underlying graph to $deg_*(v)$ vertices, with $deg_*(v)$ the degree of v in T_*. Such contractions could further speed up the single cut computations. Similar shrinkings can obviously be done for increasing costs, as well.

5 Conclusion

We introduced a simple and fast algorithm for dynamically updating a cut tree for a changing graph. In a first prove of concept our approach allowed to save over 96% of the cut computations and it provides even more possibilities for effort saving due to contractions. A more extensive experimental study is given in the full paper [7]. Recently, we further succeeded in improving the routine for an inserted edge or an increased cost such that it guarantees that each cut that remains valid is also represented by the new cut tree. This yields a high temporal smoothness, which is desirable in many applications. Note that the routine for a deleted edge or a decreased cost as presented in this work already provides this temporal smoothness.

References

1. Gomory, R.E., Hu, T.: Multi-terminal network flows. Journal of the Society for Industrial and Applied Mathematics 9(4), 551–570 (1961)
2. Gusfield, D.: Very simple methods for all pairs network flow analysis. SIAM Journal on Computing 19(1), 143–155 (1990)

3. Elmaghraby, S.E.: Sensitivity Analysis of Multiterminal Flow Networks. Operations Research 12(5), 680–688 (1964)
4. Barth, D., Berthomé, P., Diallo, M., Ferreira, A.: Revisiting parametric multi-terminal problems: Maximum flows, minimum cuts and cut-tree computations. Discrete Optimization 3(3), 195–205 (2006)
5. Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A fast parametric maximum flow algorithm and applications. SIAM Journal on Computing 18(1), 30–55 (1989)
6. Scutellà, M.G.: A note on the parametric maximum flow problem and some related reoptimization issues. Annals of Operations Research 150(1), 231–244 (2006)
7. Hartmann, T., Wagner, D.: Fast and Simple Fully-Dynamic Cut Tree Construction. Karlsruhe Reports in Informatics 2012-18, KIT Karlsruhe Institute of Technology (2012), `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000030004`
8. Görke, R., Holzer, M., Hopp, O., Theuerkorn, J., Scheibenberger, K.: Dynamic network of email communication at the Department of Informatics at Karlsruhe Institute of Technology, KIT (2011), `http://i11www.iti.kit.edu/projects/spp1307/emaildata`