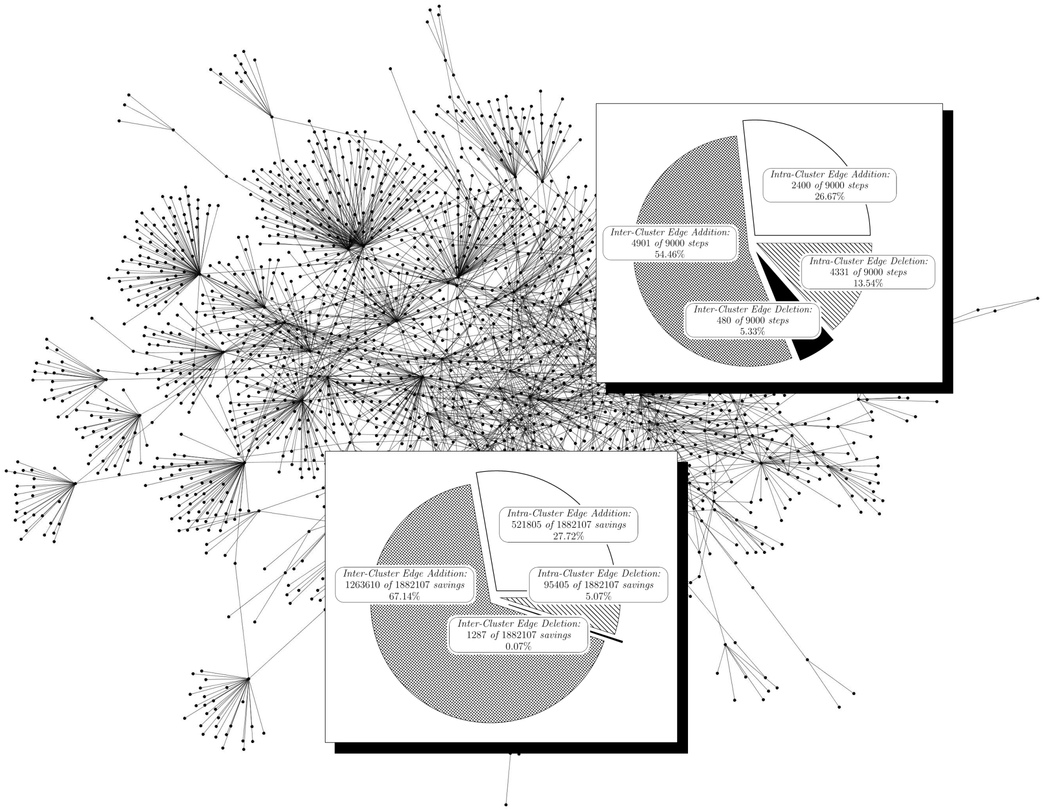
Clustering Dynamic Graphs with Guaranteed Quality

Diplomarbeit von Tanja Hartmann
Thesis of Tanja Hartmann
15th of October 2008



Supervisors: Prof. Dr. Dorothea Wagner and
Dipl.-Math.-tech. Robert Görke

Institute of Theoretical Informatics
Universität Karlsruhe (TH)

The graph shown on the title page in parts models the e-mail correspondence between members of the Fakultät für Informatik at the Universität Karlsruhe (TH). The vertices represent the individuals sending and receiving e-mails, the edges are weighted by the number of exchanged e-mails. This graph illustrates an intermediate instance occurring during the experimental analysis in Chapter 7, Section 7.2.

Acknowledgments

First of all I would like to thank Prof. Dr. Dorothea Wagner for giving me the possibility to create this thesis. She further gave me the chance to visit several conferences, which was a great experience and yielded helpful impulses. I would also like to thank Robert Görke, who continuously supported my work by annotating and discussing my written results. I especially enjoyed to write this work in a foreign language. Thus I gained plenty of experience and insight. Finally I would like to address special thanks to my family for giving me the possibility to spend a second period at university and fulfill a dream.

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und die vorgestellten Ergebnisse ohne die Hilfe Dritter erarbeitet habe. Ich habe auf keine anderen als die angegebenen Quellen und Hilfsmittel zurückgegriffen.

Tanja Hartmann
Karlsruhe, den 15. Oktober 2008

Abstract

This work is aimed at exploring how a clustering with a guaranteed clustering quality can efficiently be realized on fully dynamic graphs. To this end we concentrate on the specific cut-clustering method of Flake et al. [FTT04], as this method guarantees a clustering quality resulting from the structure of minimum-cut trees. In the first part of this work we analyze but disprove an approach of Saha and Mitra [SM06], who tried to dynamically extend the cut-clustering method of Flake et al. As a consequence the second part of this work is aimed at developing new algorithms for updating clusterings in undirected graphs based on the cut-clustering method. As the cut-clustering method uses minimum-cut trees to cluster graphs, we at first give some insights into dynamically extending the Gomory-Hu method [GH61], a method for calculating those trees, and formulate according algorithms. Abbreviating these algorithms, which consider complete minimum-cut trees, finally yields four new, correct and effort saving algorithms for updating clusterings, distinguished by elementary modifications.

Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Clustern dynamischer Graphen und in diesem Zusammenhang mit einem speziellen Algorithmus, vorgestellt von Flake et al. [FTT04]. Dieser Algorithmus, seine Entwickler nennen ihn *cut-clustering Algorithmus*, ist einer von wenigen Clusteralgorithmen, für deren Ergebnis eine Gütegarantie beweisbar ist. Die Gütegarantie des cut-clustering Algorithmus gründet auf speziellen Eigenschaften minimaler Schnittbäume.

Diese Arbeit umfasst zwei übergeordnete Themenblöcke. Der erste Themenblock analysiert einen bereits bestehenden Ansatz zur Dynamisierung des cut-clustering Algorithmus, entwickelt von Saha und Mitra [SM06]. Ergebnis dieser Analyse ist jedoch die Widerlegung dieses Ansatzes, der neben unvollständigen Beweisen vor allem einen weitreichenden methodischen Fehler aufweist. Die vorliegende Arbeit vervollständigt die Beweisführung von Saha und Mitra in Teilen.

Als Konsequenz der Widerlegung des Ansatzes von Saha und Mitra widmet sich der zweite Themenblock der Frage, wie man die Idee des cut-clustering Algorithmus von Flake et al. auf dynamische Graphen übertragen kann. Dabei steht zunächst die Gomory-Hu Methode [GH61] zur Berechnung minimaler Schnittbäume im Vordergrund, da minimale Schnittbäume die Basis des cut-clustering Algorithmus bilden. Diese Arbeit skizziert eine Idee zur Dynamisierung der Gomory-Hu Methode und leitet daraus einen neuen Ansatz zum Clustern dynamischer Graphen unter Beibehaltung der gegebenen Gütegarantie ab.

Die Implementierung der Gomory-Hu Methode ist sehr aufwändig, da diese Methode kreuzungsfreie minimale u-v-Schnitte berechnet und zu diesem Zweck unter großem Aufwand Knoten kontrahiert. Gusfield [Gus90] jedoch zeigt, dass die Kreuzungsfreiheit der Schnitte zur Konstruktion eines minimalen Schnittbaums nicht zwingend notwendig ist. Daraus entwickelt er eine einfacher zu implementierende Technik der Baumkonstruktion. Leider lässt sich Gusfields Vorgehensweise nicht ohne Weiteres auf die in dieser Arbeit vorgestellte dynamische Variante der Gomory-Hu Methode übertragen. Die Verwendung sich kreuzender Schnitte zur dynamischen Aktualisierung von minimalen Schnittbäumen bedarf eines erneuten Beweises des zentralen Theorems in Gusfields Ansatz für eine allgemeinere Situation. Abschnitt 4.2 stellt Gusfields Technik vor und erbringt den nötigen Zusatzbeweis.

Im Rahmen der dynamischen Berechnung von minimalen Schnittbäumen untersucht diese Arbeit außerdem verschiedene Möglichkeiten zur Aktualisierung einzelner minimaler u-v-Schnitte. Die Diskussion der Einsatzmöglichkeiten dieser Aktualisierungstechniken im Zusammenhang mit dynamischen minimalen Schnittbäumen zeigt jedoch Grenzen auf, welche aus der iterativen Konstruktion minimaler Schnittbäume resultieren. Der Großteil der minimalen u-v-Schnitte, die ein minimaler Schnittbaum repräsentiert, wird nicht explizit berechnet, sondern leitet sich aus seiner iterativen Konstruktion ab. Dies wiederum erschwert eine dynamische Aktualisierung jener u-v-Schnitte mit Techniken, die auf einer dynamischen Aktualisierung zugehöriger maximaler Flüsse beruhen.

Zum Clustern von Graphen mit Hilfe des cut-clustering Algorithmus von Flake et al. ist die Berechnung eines kompletten minimalen Schnittbaumes jedoch nicht zwingend notwendig. Meist kann die gewünschte Clusterung bereits vor Ende der Baumberechnung abgelesen werden. Das Ergebnis dieser Arbeit ist daher eine neue Methode zum Clustern dynamischer Graphen basierend auf dem cut-clustering Algorithmus, die im Wesentlichen eine gekürzte Version des zuvor entwickelten dynamischen

Algorithmus zum Aktualisieren minimaler Schnittbäume ist. Vorteil dieser neuen Clustermethode ist, im Vergleich zur kompletten Clusterneuberechnung, das Potential die unnötige Berechnung schon bekannter minimaler u-v-Schnitte oftmals zu vermeiden, und gleichzeitig jene Cluster zu erhalten, die nicht von der dynamischen Veränderung des zugrundeliegenden Graphen betroffen sind. Die aktualisierte Clusterung übernimmt möglichst viele Teile der vorherigen und garantiert so eine gewisse Ähnlichkeit zur vorangegangenen Clusterung. Ein kleines Experiment bestätigt abschließend die theoretisch hergeleiteten Vorteile und Verhaltensweisen des neuen Ansatzes zum Clustern dynamischer Graphen mit Gütegarantie.

Contents

1 Introduction **1**

2 Contradicting Barna Saha and Pabitra Mitra **5**

- 2.1 Review of the Cut-Clustering Algorithm . . . 5
- 2.2 Detecting a Methodical Error . . . 6
- 2.3 Discussion of Inconsistencies . . . 10
  - 2.3.1 The Merging Lemma and *CASE 2* . . . 10
  - 2.3.2 The Unaffect Lemma and *CASE 3* . . . 20
- 2.4 Summary . . . 22

3 Basics Regarding Cuts and Min-Cut Trees **25**

- 3.1 Some Basic Lemmas . . . 25
  - 3.1.1 Canonically Induced Cuts . . . 26
  - 3.1.2 Canonically Induced Cuts in Modified Graphs . . . 28
  - 3.1.3 Canonically Induced Min-Cuts in Modified Graphs . . . 29
- 3.2 The Gomory-Hu Method . . . 31
  - 3.2.1 Gomory-Hu Algorithm for Min-Cut Trees . . . 32
  - 3.2.2 Definitions and Resulting Remarks . . . 34

4 Dynamically Updating Min-Cut Trees **39**

- 4.1 Dynamic Changes of Min-Cut Trees . . . 39
  - 4.1.1 Execution Theorem and Corollaries . . . 39
  - 4.1.2 Algorithm Ideas for Updating Min-Cut Trees . . . 42
- 4.2 Simple Implementation of Update-Algorithms . . . 47
  - 4.2.1 Realizing the Node Splitting (*Phase 1*) . . . 48
  - 4.2.2 Realizing the Subtree Reconnection (*Phase 2*) . . . 48
  - 4.2.3 Specification of Algorithm Ideas . . . 51
- 4.3 Algorithm Engineering . . . 54
  - 4.3.1 Edge-Induced Cuts as Minimum Separating Cuts . . . 54
  - 4.3.2 New Minimum Separating Cuts for Given Vertices . . . 56
  - 4.3.3 Improving the Algorithms . . . 62

5 Dynamically Updating Minimum u-v-Cuts 71
5.1 Adjusting Residual Graphs 71
5.1.1 Flows in Undirected Weighted Graphs 72
5.1.2 The Method of Kohli and Torr 73
5.1.3 Using Dynamic Flows for Updating Min-Cut Trees 77
5.2 Updating a Set of All Minimum u-v-Cuts 78
5.2.1 Representation of All Minimum u-v-Cuts 79
5.2.2 Updating the DAG-Representation 80
5.2.3 Using DAG-Representations for Min-Cut Trees 83

6 Updating Clusterings Based on Min-Cut Trees 89
6.1 Abbreviating the Cut-Clustering Method 89
6.2 Updating Algorithms for Edge Deletions 91
6.2.1 Inter-Cluster Edge Deletion 91
6.2.2 Intra-Cluster Edge Deletion 95
6.3 Updating Algorithms for Edge Additions 100
6.3.1 Intra-Cluster Edge Addition 100
6.3.2 Inter-Cluster Edge Addition 101
6.3.3 Bow to the Approach of Saha and Mitra 105

7 Experimental Analysis 109
7.1 Zachary's Friendship Network 110
7.1.1 Exemplary Inter-Cluster Edge Additions 111
7.1.2 Exemplary Intra-Cluster Edge Deletions 112
7.1.3 Exemplary Inter-Cluster Edge Deletion 113
7.2 Real World E-Mail Graph 114

8 Conclusion 119

List of Algorithms

1 Cut-Clustering 6
2 Inter-Edge-Add 7
3 Gomory-Hu 33
4 Tree-EdgeAdd-1 45
5 Tree-NoBridgeDel 45
6 Tree-BridgeDel 45
7 Tree-EdgeDel-1 46
8 Central-Tree 53
9 Tree-EdgeAdd-2 53
10 Tree-EdgeDel-2 54
11 Tree-EdgeDel-3 63
12 Central-TreeDel 65
13 SplitAndReconnect 66
14 Tree-EdgeAdd-3 69
15 ResidualUpdate 75
16 UpdateDAG 81
17 Inter-Cluster Edge Deletion 92
18 Central-Clus 94
19 Intra-Cluster Edge Deletion 97
20 Best Cut 98
21 Intra-Cluster Edge Addition 100
22 Inter-Cluster Edge Addition 103
23 Cut-Clustering Heuristic 109

Chapter 1

Introduction

Graphs can be found almost everywhere in our everyday life. Basically each kind of network is a graph. The Internet for example naturally induces a graph with webpages corresponding to vertices and links between pages representing edges. In the Internet webpages with a related content are characterized by a strong connectivity, while pages with disparate contents only share a few or even no links. The idea of clustering graphs now is to somehow find groups of related objects in such a given instance. The strength of the relation between two object is often given by an additional weight of the according edge. So a clustering basically decomposes a set of objects into groups regarding some “natural characteristics” depending on the underlying network.

Formally a clustering $\zeta(G)$ of an undirected, weighted graph $G = (V, E, c())$ is a partitioning of the set V of vertices of the graph G. However, the idea of a clustering, in contrast to a partitioning in general, is to bunch those vertices to groups that are highly connected and at the same time to assign weakly connected vertices to different groups. The groups are then called clusters. The trade-off coming up with this idea is described by the paradigm of intra-cluster density versus inter-cluster sparsity: The stricter we choose the condition on which vertices lie in the same cluster the weaker becomes the condition on which vertices are separated into different clusters and vice versa.

Apart from this simple idea of what a clustering intuitively constitutes, there exist further, very sophisticated and specialized formal definitions of clusterings. For an overview of clustering theory see [BE05], Chapter 8. In this work we simply assume the weighted edges in a graph G to represent relations between vertices and say a clustering algorithm tries to find clusters of highly related vertices, while vertices in different clusters have little in common.

Clusterings defined in these terms are used in many different applications. A very common application are recommendation engines used by online stores. Such an engine searches and then recommends products similar to the one a customer is currently interested in. In this application the similarity of products is given by a function $c(\{\text{product A}, \text{product B}\}) \rightarrow \mathbb{R}$, and therefore, modeled as weighted edges between vertices representing the products. How the similarity between products is defined and measured depends on the application. In the case of recommendation engines it is popular to use, roughly spoken, the likelihood of two products being bought by the same customer.

There exist many different approaches and algorithms to find clusterings in graphs. However, only very few of these algorithms guarantee a measurable quality of the resulting clustering. Furthermore, measuring the quality of a clustering is quite involved. For a detailed description of quality measurements see [BE05], Section 8.1. In this work we consider a specific clustering algorithm, namely the *cut-clustering method* introduced by Flake et al. [FTT04], which guarantees a clustering quality depending on a parameter α. This guarantee results from properties of the structure of minimum-cut trees which the algorithm bases on. The quality measurement of the cut-clustering method is similar to *conductance* described in [BE05]. It is thus a measurement of bottlenecks within and between clusters. Although it considerably differs from purely density-based measures, such as *coverage* or *performance* [BE05], and from measures based on statistical significance such as *modularity* [GGW07, NG04], it is also based on the general paradigm of intra-cluster density and inter cluster sparsity.

The aim of this work is to develop a fully dynamic method for updating clusterings with a guaranteed clustering quality. The elementary modifications allowed in this context in the underlying graph are the addition or deletion of an edge as well as the insertion or removal of a vertex. Note, that a vertex is only removable if it is unconnected, i.e., if all incident edges are removed first. The intention of updating a clustering, instead of recalculating a new clustering for the modified graph, is on the one hand to save effort, and on the other hand, we assume an elementary local modification in the underlying graph not to change the clustering too much. So we are looking for an updated clustering as similar as possible to the previous one, which is more difficult to control in the case of a recomputation. The paper of Saha and Mitra [SM06] gives a first proposal how an algorithm for updating clusterings with quality preservation might look like. Unfortunately their approach turns out to be unfeasible. Nevertheless, as their try bases on the cut-clustering method of Flake et al. [FTT04] this work also concentrates on this special algorithm and in parts corrects some inconsistencies of the approach of Saha and Mitra.

Our work can be split into two parts. The first part concentrates on the analysis and disproval of the updating approach given by Saha and Mitra [SM06] and covers Chapter 2. The second part involves the remaining Chapters 3 to 7. In Chapter 3 we start with some basic lemmas and corollaries concerning cuts in graphs which are dynamically modified. Then a method for constructing minimum-cut trees by Gomory and Hu [GH61] is introduced, which facilitates an even better understanding of minimum-cut trees. In Chapter 4 we develop at first, on several levels of detail, an approach for dynamically updating complete minimum-cut trees, as they constitute the basis of the cut-clustering method. Chapter 5 is an additional analysis of the possibilities of dynamically updating minimum separating cuts regarding two fixed vertices. This subject seems to be interesting in the context of updating minimum-cut trees, as it might be reasonable to update the minimum cuts represented in a minimum-cut tree individually. To this end we concentrate on an approach of Kohli and Torr [KT07] combined with some inspirations given by a paper of Fleischer [Fle99]. From the algorithms developed so far we finally deduce in Chapter 6 new correct and feasible algorithms for updating clusterings resulting from the cut-clustering method given by Flake et al. [FTT04]. Chapter 7 additionally summarizes some experiments which affirm the theoretically predicted behavior of our newly developed algorithms.

Preliminaries

A graph constitutes a finite set of objects, also called *vertices*, together with a set of pairs of such vertices. Each pair defines an *edge* between its vertices. This work only considers *undirected* graphs. The edges of undirected graphs are represented by dual-element sets.

Definition: Undirected graph

Definition 1 *Let V denote a set of vertices, E denotes the related set of edges defined on V. With $E \subseteq \binom{V}{2}$ the sets V and E define an* undirected graph *$G = (V, E)$, while $\binom{V}{2}$ denotes the set of all subsets of V exactly containing two elements. The set $\{u, v\} \in E$ represents an* undirected edge.

The cardinality of the set of vertices, and the set of edges respectively, is denoted by $n := |V|$ and $m := |E|$. Two vertices u and v defining an edge $\{u, v\}$ are called *adjacent* to each other. The edge $\{u, v\}$ is called *incident* with vertex u and vertex v.

Apart from the direction of an edge, which we ignore, also the *weight* of an edge is a common property. Edges are commonly *weighted* by a *cost function*.

Definition: Weighted graph

Definition 2 *A graph $G = (V, E, c())$ with a* cost function *$c : E \to X \subseteq \mathbb{R}$ is called a* weighted *graph. The* weight *of an edge $e \in E$ is given by $c(e) \in X$.*

This work only considers positively weighted graphs, i.e., it holds that $X = \mathbb{R}_{>0}$. Furthermore, the number of edges in a graph can be regarded as a property of the whole graph. In this context *complete graphs* play an exceptional role. An undirected graph $G = (V, E)$ is said to be *complete* if it holds that $|E| = \binom{V}{2}$. As this work only considers undirected graphs we omit the definition of complete graphs regarding other graph families. Complete graphs as defined above have $m = |E| = \binom{n}{2} = n(n-1)/2 \in \Theta(n^2)$ edges. This is, each vertex is adjacent to all other vertices.

Definition: Complete graph

However, in general two arbitrary vertices are not connected by an edge. In some graphs two arbitrary vertices are not even connected by a path of several edges. Such a graph is called *unconnected*. A path is defined as a sequence $\{p_0, p_1\}, \ldots, \{p_{i-1}, p_i\}, \{p_i, p_{i+1}\}, \ldots, \{p_{k-1}, p_k\}$ of edges in a graph G. Two vertices u and v are said to be connected by a path if there exists such a sequence with $u = p_1$ and $v = p_k$. Other graphs are *connected*, but only that weak that there exists an edge whose deletion would again disconnect the graph. Such edges are called *bridges*. Finally the *trees* also play an exceptional role. In a tree each pair of vertices is connected by a unique path of edges.

Definition: Connected graph, tree and bridge

Definition 3 *A graph $G = (V, E)$ is* connected *if each pair of vertices is connected by at least one path. A graph $T = (V_T, E_T)$ is a* tree *if each pair of vertices is connected by exactly one path. A* forest *just denotes a set of trees. An edge $e \in E$ is a* bridge *in a graph if its deletion disconnects the graph.*

Definition: Vertex and edge contraction

Each edge in a tree is a bridge. Therefore, a tree T has $m = |E| = n - 1$ edges. A *subtree* of a tree $T = (V_T, E_T)$ is defined by a subset $V'_T \subseteq V_T$ and a valid subset $E'_T \subseteq E_T$ such that both subsets again define a connected graph $T_{\text{sub}} = (V'_T, E'_T)$. A tree $T_s = (V, E')$ defined by all vertices of a graph $G = (V, E)$ and a valid subset $E' \subseteq E$ is called a *spanning tree* of G. Each spanning tree further constitutes a *subgraph* of the graph G. The contraction of subgraphs is a common technique used in many algorithms. In this work we distinguish the *contraction of edges* and the *contraction of vertices*: Two vertices u and v in a graph G are contracted by merging both to a new node ω. Each edge in graph G previously incident with u or v becomes incident with the new node ω instead. Any resulting parallel edges between a vertex $g \notin \{u, v\}$ and ω get merged to one edge of the totaled weight. In case of contracting adjacent vertices, the edge in between is just ignored. A vertex contraction can be regarded as a surjective map from V to $V_\circ$, with $V_\circ$ being the set of nodes in the graph $G_\circ$ resulting from G by the contraction. This map identifies u and v with the new node ω and is the identity for those vertices that are not contracted. The contraction of an edge $e = \{u, v\}$ is equivalent to the contraction of the two adjacent vertices u and v.

In each kind of graph we can further define *cuts*. In this work we only consider cuts in undirected, positively weighted graphs. A cut decomposes the set of vertices into two parts. The *weight* of a cut is given by the totaled weight of all edges crossing the cut.

Definition: Crossing cuts and clusterings

Definition 4 *A* cut $\theta := (U, V \backslash U)$ *in an undirected, weighted graph* $G = (V, E, c())$ *is defined by the decomposition of the set* V *in exactly two parts* U *and* $V \setminus U$ *(the sides of the cut). The* weight *of a cut* $\theta = (U, V \setminus U)$ *is defined by*

$$c(\theta) = \sum_{\substack{\{u,v\} \in E \\ u \in U, v \in (V \setminus U)}} c(\{u, v\}).$$

Two cuts $\theta_1 := (U_1, V \setminus U_1)$ *and* $\theta_2 := (U_2, V \setminus U_2)$ cross *each other if none of the four sets* $U_1 \cap U_2$, $U_1 \cap (V \setminus U_2)$, $U_2 \cap (V \setminus U_1)$ *and* $(V \setminus U_1) \cap (V \setminus U_2)$ *is empty.*

A *clustering* $\zeta(G)$ of an undirected, weighted graph $G = (V, E, c())$ can be regarded as a generalization of a cut in G. A clustering is a partitioning of the set V not into exactly two, but into an arbitrary number of partitions or *clusters*. Edges between vertices within a single cluster are called *intra-cluster* edges, edges between vertices in different clusters are called *inter-cluster* edges. The clusterings considered in this work have an additional quality feature which is introduced in Section 2.1.

According to the definition of the weight of a cut, we can further consider *global minimum cuts* as well as *minimum u-v-cuts* concerning two fixed vertices u and v.

Definition: Minimum u-v-cuts

Definition 5 *A cut* $\theta := (U, V \setminus U)$ *is a u-v-cut if it separates the vertices* u *and* v*, i.e., if it holds that* $u \in U$, $v \in (V \setminus U)$ *or* $v \in U$, $u \in (V \setminus U)$*. A u-v-cut is a minimum u-v-cut in a graph* G *if there exists no cheaper u-v-cut in* G*.*

Analogously to the latter definition we call a *non-trivial* cut in a graph G a *global minimum cut* if there exists no cheaper non-trivial cut in G. A cut is said to be *trivial* if one of the partitions U or $V \setminus U$ is empty. Furthermore, there exists a correspondence between minimum u-v-cuts and *maximum u-v-flows*. Maximum u-v-flows and their correspondence to minimum u-v-cuts are introduced in Chapter 5, as well as the related expression of the *residual graph*.

Definition: Minimum-cut tree and edge-induced cuts

Finally, we present a short description of *minimum-cut trees* and *edge-induced cuts* in an undirected, weighted graph $G = (V, E, c())$. For each undirected graph exists at least one minimum-cut tree (or simply *min-cut tree*). A min-cut tree of a graph G is a weighted tree $T(G)$ defined on V. Note, that $T(G)$ does not need to be a spanning tree. For each pair of vertices $\{u, v\} \subseteq V$ the min-cut tree $T(G)$ represents a minimum u-v-cut in G. We can find this minimum u-v-cut by inspecting the path that connects u and v in $T(G)$. The edge of minimum weight on this path induces the minimum cut, as its removal splits $T(G)$, and G respectively, into two sets of vertices that correspond to the cut sides. The minimum u-v-cut, therefore, is edge-induced and has the same weight as the inducing edge. Gomory and Hu [GH61] describe min-cut trees in more detail and provide an algorithm for the calculation.

Chapter 2

Contradicting Barna Saha and Pabitra Mitra

As this work is aimed at exploring how a clustering with a guaranteed clustering quality can be efficiently realized for dynamic graphs, the paper of Saha and Mitra [SM06], which gives a proposal how an algorithm for dynamic clustering might look like, seems well worth being read. The idea of Saha and Mitra is based on a work of Flake et al. [FTT04] that introduces a static clustering method with a guaranteed clustering quality depending on a parameter α. Thereby, the guarantee of the clustering quality results from properties of the min-cut tree constructed as a preliminary structure. Flake et al. call their method *cut-clustering algorithm*.

Unfortunately a closer look at the paper of Saha and Mitra [SM06] yields a bunch of questions. On the one hand a massive methodical error can be detected, on the other hand, even if we ignore this methodical failure, there are still inconsistencies in some proofs and assertions.

Saha and Mitra [SM06] distinguish four elementary modifications of the graph the current clustering is based on. For each modification they separately develop an updating algorithm. The distinguished modifications are the addition of an edge within a single cluster, the addition of an edge between two different clusters, the deletion of an edge between two different clusters, and finally the deletion of an edge within a single cluster. The removal or insertion of an isolated vertex is considered as the removal or insertion of a single cluster.

In this section we first give a short review of the cut-clustering algorithm of Flake et al. [FTT04] and their definition of clustering quality. Then a description of the methodical error in the approach of Saha and Mitra [SM06] follows. At the end we will point out and in parts correct the inconsistencies of some proofs and assertions given by Saha and Mitra.

2.1 Review of the Cut-Clustering Algorithm

In this section the cut-clustering algorithm, which is proposed by Flake et al. [FTT04] for clustering an edge-weighted graph $G = (V, E, c())$ with a guaranteed clustering quality, is referred to as Algorithm 1. The cut-clustering algorithm adds an artificial sink t to graph G and connects t to each vertex in V by an edge with weight α. For the resulting graph G_α a min-cut tree $T(G_\alpha)$ is calculated. This min-cut tree

Algorithm 1: Cut-Clustering

```
Input: Graph $G = (V, E, c())$, parameter $\alpha$ of clustering quality
Output: Clustering $\zeta(G)$ with clustering quality depending on $\alpha$
$V_\alpha \leftarrow V \cup \{t\}$ %add artifical sink $t$
$E_\alpha \leftarrow E \cup \{\{t, v\} | v \in V\}$ %connect $t$ to each vertex $v$ in $V$
$\forall e \in E : c_\alpha(e) \leftarrow c(e)$
$\forall v \in V : c_\alpha(\{t, v\}) \leftarrow \alpha$ %each edge incident with $t$ gets weight $\alpha$
Calculate a min-cut tree $T(G_\alpha)$ of $G_\alpha = (V_\alpha, E_\alpha, c_\alpha())$
Remove $t$ from $T(G_\alpha)$
$\zeta(G) \leftarrow$ set of connected components resulting from the removal of sink $t$
return $\zeta(G)$
```

decomposes into several connected components after removing the sink t. The resulting set of connected components forms a clustering $\zeta(G)$ of G. The clustering $\zeta(G)$ of G is said to be induced by the min-cut tree $T(G_\alpha)$.

Definition: Clustering quality

Furthermore, Flake et al. [FTT04] define a clustering quality that is guaranteed for any clustering $\zeta(G)$ resulting from Algorithm 1. This clustering quality consists of two aspects: The inter-cluster quality of a single cluster C is high if the connectivity $c(C, V \setminus C)$ to the remaining vertices is low. By contrast, the intra-cluster quality of C is high if the minimum weight $c(P, C \setminus P)$ over all subsets $P \subseteq C$ is high. A clustering $\zeta(G)$ of a graph G respects the clustering quality concerning a fixed parameter α if for all clusters $C \in \zeta(G)$ holds that

$$\begin{array}{lll} \textit{(inter-cluster quality)} & c(C, V \setminus C) \leq \alpha\, |V \setminus C| & \text{and} \\ \textit{(intra-cluster quality)} & c(P, C \setminus P) \geq \alpha \, \min\{|P|, |C \setminus P|\}, \, \forall P \subset C. & \end{array}$$

2.2 Detecting a Methodical Error

We describe the methodical error in the approach of Saha and Mitra [SM06] by means of the *inter-cluster edge addition algorithm* (or simply *inter-edge-add algorithm*) on page four in [SM06]. The inter-edge-add algorithm of Saha and Mitra is referred to as Algorithm 2 in this section. Note, that we give a slightly more detailed formulation of the algorithm using a notation consistent with the remainder of this work. The inter-edge-add algorithm is meant to be used if a dynamic graph $G = (V, E, c())$ is modified by the addition of a weighted edge $e_\oplus = \{b, d\}$ between different clusters $C_b \ni b$ and $C_d \ni d$ of the current clustering $\zeta(G)$. For the input clustering $\zeta(G)$ a clustering quality as given by Flake et al. [FTT04] is assumed. The algorithm then distinguishes three different cases. The result in each case is thought to be a clustering $\zeta(G_\oplus)$ of the modified graph $G_\oplus = (V, E \cup \{e_\oplus\}, c_\oplus())$ still respecting the clustering quality. For the modified cost-function $c_\oplus()$ holds that $c_\oplus(e) = c(e), \forall e \in E$, and $c_\oplus(e_\oplus) := \Delta$. The following fact now constitutes the kernel of the methodical error in the approach of Saha and Mitra.

Fact: Implicit algorithm invariant

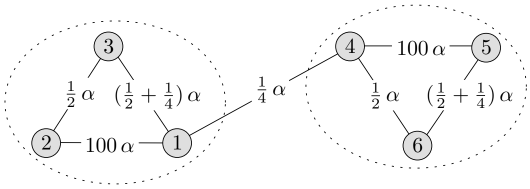
Fact 1 *Some lemmas used by Saha and Mitra [SM06] to prove the correctness of the inter-edge-add algorithm implicitly assume (we will illustrate this in Subsection 2.3.1) that the input clustering $\zeta(G)$ results from the cut-clustering method of Flake et al. [FTT04], i.e., that there exists a min-cut tree $T(G_\alpha)$ such that the removal of the artificial sink t yields $\zeta(G)$. Note, that Saha and Mitra consider the correctness of the inter-edge-add algorithm achieved if the algorithm maintains the clustering quality.*

Algorithm 2: Inter-Edge-Add

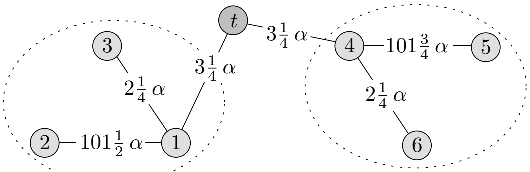
```
Input: Graph $G = (V, E, c())$, clustering $\zeta(G)$ of graph $G$, parameter $\alpha$ of
       clustering quality
Output: Clustering $\zeta(G_\oplus)$ of $G_\oplus = (V, E \cup \{e_\oplus\}, c_\oplus())$ complying with
        clustering quality
%---------- notation ----------
$e_\oplus = \{b, d\}$ new edge added to $E$ with weight $\Delta$
$C_b \ni b, C_d \ni d$ clusters affected by the inter-cluster edge addition
%---------- algorithm ----------
$E_\oplus \leftarrow E \cup \{e_\oplus\}$ %add new weighted edge
Update inter-cluster connectivity $c_\oplus(C_b, V \setminus C_b)$ of $C_b$
Update inter-cluster connectivity $c_\oplus(C_d, V \setminus C_d)$ of $C_d$
%(inter-cluster connectivity of unaffected clusters does not
 change due to edge addition)
%(intra-cluster connectivity does not change due to edge
 addition)
%---CASE 1
if $c_\oplus(C_b, V \setminus C_b) \leq \alpha\,|V \setminus C_b|$ and $c_\oplus(C_d, V \setminus C_d) \leq \alpha\,|V \setminus C_d|$ then
    %(i.e., inter-cluster quality of affected clusters is not
     violated)
    return clustering $\zeta(G_\oplus) = \zeta(G)$
else
    %---CASE 2
    if $\frac{2\,c(C_b, C_d)}{|V|} \geq \alpha$ then
        Cluster $\bar{C} \leftarrow$ Merge $(C_b, C_d)$
        return clustering $\zeta(G_\oplus) = (\zeta(G) \setminus \{C_b, C_d\}) \cup \{\bar{C}\}$
    else
        %---CASE 3
        %(Note that the formulation of this case, taken from Saha
         and Mitra, is quite inscrutable)
        Node $\omega \leftarrow$ Contract $(V \setminus (C_b \cup C_d))$ in $G_\oplus$
        $V_\alpha \leftarrow V \cup \{t\}$ %add artificial sink
        $\forall u \in (C_b \cup C_d)$: connect $t$ to $u$ by an edge of weight $\alpha$
        Connect $t$ to $\omega$ by an edge of weight $\alpha\,|V \setminus (C_b \cup C_d)|$
        %Call the resulting graph $G'_\alpha$
        Tree $T(G'_\alpha) \leftarrow$ Min-Cut Tree $(G'_\alpha)$
        Remove $t$ from $T(G'_\alpha)$
        %The resulting connected components which consist of
         vertices included in $(C_b \cup C_d)$ are called $C_1, \ldots C_z$
        return clustering $\zeta(G_\oplus) = (\zeta(G) \setminus \{C_b, C_d\}) \cup \{C_1, \ldots C_z\}$
```

Methodical error made by Saha and Mitra

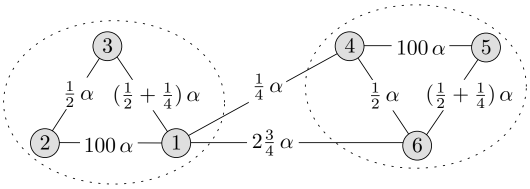
In a dynamic context the inter-edge-add algorithm and all other updating algorithms developed for the remaining modifications are thought to be applied recursively in an arbitrary order. So for this algorithms Fact 1 describes an invariant. Due to this invariant the returned clustering $\zeta(G_\oplus)$ does not only need to respect the clustering quality, but result from the cut-clustering method of Flake et al. [FTT04]. For the correctness of the inter-edge-add algorithm it is hence necessary to prove the invariant for all distinguished cases. However, Saha and Mitra [SM06] only prove the maintenance of the clustering quality, which is not sufficient as the example in Figure 2.1 will show.


3
1/2 α
(1/2 + 1/4) α
2
100 α
1
1/4 α
4
100 α
5
1/2 α
(1/2 + 1/4) α
6

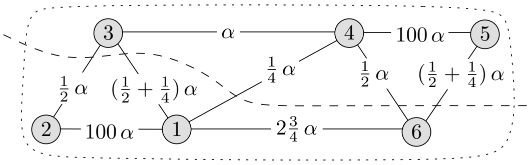
(a) Graph G^0 with clustering $\zeta(G^0)$ resulting from the cut-clustering method


t
3 1/4 α
3
3 1/4 α
2 1/4 α
2
101 1/2 α
1
4
101 3/4 α
5
2 1/4 α
6

(b) Min-cut tree $T(G^0_\alpha)$ inducing the clustering $\zeta(G^0)$ shown above


3
1/2 α
(1/2 + 1/4) α
2
100 α
1
1/4 α
2 3/4 α
4
100 α
5
1/2 α
(1/2 + 1/4) α
6

(c) Graph G^1 with clustering $\zeta(G^1) = \zeta(G^0)$ resulting from *CASE 1* of the inter-edge-add algorithm


3
α
4
100 α
5
1/2 α
(1/2 + 1/4) α
1/4 α
1/2 α
(1/2 + 1/4) α
2
100 α
1
2 3/4 α
6

(d) Graph G^2 with merged cluster $\bar{C}$ resulting from *CASE 2* of the inter-edge-add algorithm and violating intra-cluster quality

Figure 2.1: A clustering violating the clustering quality.

Figure 2.1a considers a weighted graph G^0 with a parametric cost function and a clustering $\zeta(G^0) = \{\{1,2,3\},\{4,5,6\}\}$ resulting from the cut-clustering method of Flake et al. [FTT04]. For all graphs in Figure 2.1 the parametric weights are assigned to the edges. Figure 2.1b shows a min-cut tree $T(G^0_\alpha)$ related to graph G^0 as it induces the clustering $\zeta(G^0)$ shown in Figure 2.1a. The clustering $\zeta(G^0)$, therefore, provides the properties that constitute the implicit invariant. In Figure 2.1c a new weighted edge $e_\oplus = \{1,6\}$ is added to G^0 forming the illustrated graph G^1. As the new inter-cluster edge $\{1,6\}$ does not cause any violation of the inter-cluster quality in $\zeta(G^0)$, the inter-edge-add algorithm applied to this edge addition ends up in *CASE 1* (see Algorithm 2, Line 6) and terminates without any changes to the clustering $\zeta(G^0) = \zeta(G^1)$. Graph G^0 is just updated to graph G^1. Obviously the new clustering $\zeta(G^1)$ still respects the clustering quality. However, the clustering $\zeta(G^1)$ is not necessarily induced by a min-cut tree of G^1_α anymore. So at this point the implicit invariant may be violated.

If we assume the invariant to still be met, as Saha and Mitra [SM06] implicitly do, we again can apply the algorithm to the clustering $\zeta(G^1)$ regarding a further addition of edge $e_\oplus = \{3,4\}$. Figure 2.1d illustrates graph G^2 resulting from this edge addition. A closer look at the inter-cluster weights in the previous clustering $\zeta(G^1)$ shows that the additional weight of the new edge $e_\oplus = \{3,4\}$ causes a violation of the inter-cluster quality in $\zeta(G^1)$: The inter-cluster weight of each cluster $C \in \zeta(G^1)$ after the edge addition is $4\,\alpha$, which is greater than $3\,\alpha = \alpha\,|V \setminus C|$. So for the addition of edge $e_\oplus = \{3,4\}$ the inter-edge-add algorithm will skip *CASE 1* and end up in *CASE 2* instead (see Algorithm 2, Line 9), as in graph G^1 it holds that $c(\{1,2,3\},\{4,5,6\}) = 3\,\alpha \geq \alpha\,|V|/2$. Then *CASE 2* finally returns a clustering $\zeta(G^2)$ of graph G^2 arising out of the clustering $\zeta(G^1)$ of the previous graph by merging the clusters $\{1,2,3\}$ and $\{4,5,6\}$ (see Figure 2.1d).

If the inter-edge-add algorithm by Saha and Mitra [SM06] was correct in the sense that each returned clustering respects the clustering quality, also the clustering $\zeta(G^2)$ in Figure 2.1d would respect the clustering quality. However, by considering the cut drawn as dashed line in Figure 2.1d we realize that the new cluster $\bar{C} = \{1,2,3,4,5,6\}$, which results from merging $\{1,2,3\}$ and $\{4,5,6\}$, violates the intra-cluster quality in $\zeta(G^2)$. This cut $(P, \bar{C} \setminus P) := (\{3,4,5\},\{1,2,6\})$ has weight $c(P, \bar{C} \setminus P) = 11\,\alpha/4$, which is cheaper than $3\,\alpha = \alpha\,\min\{|P|, |\bar{C} \setminus P|\}$.

Maintenance of clustering quality is not sufficient

It follows that either *CASE 2* does not maintain the clustering quality in general or the clustering returned by *CASE 1* does not meet the conditions required by *CASE 2* for returning a correct result or even both. We further see that neither *CASE 2* does return a correct result in general nor is the maintenance of cluster quality sufficient for the correctness of *CASE 2*. Remember that in our example the clustering $\zeta(G^1)$, which serves as input clustering for *CASE 2*, respects the clustering quality. The following fact now states a sufficient condition for the correctness of *CASE 2*.

Fact: Implicit invariant is sufficient for the correctness of *CASE 2*

Fact 2 *If the input clustering $\zeta(G)$ results from the cut-clustering method of Flake et al. [FTT04], i.e., if $\zeta(G)$ provides the properties that constitute the implicit invariant described in Fact 1,* CASE 2 *of the inter-edge-add algorithm (see Algorithm 2, Line 9) returns a clustering $\zeta(G_\oplus)$ that still respects the clustering quality. Note, that this fact does not state anything about the question whether* CASE 2 *respects the implicit invariant.*

The proof of Fact 2 will be given in Section 2.3.1. There may be even weaker conditions than the implicit invariant on which *CASE 2* returns a correct result. Nevertheless, due to the lack of knowledge concerning such conditions, and because Saha and Mitra [SM06] base their ideas on the cut-clustering method of Flake et al. [FTT04] (see Fact 1), we regard the invariant as necessary for the correctness of *CASE 2*, and the inter-edge-add algorithm respectively. In the following we hence explore whether the three cases distinguished by the inter-edge-add algorithm respect the invariant, although Saha and Mitra ignore its need.

CASE 1 does not respect the invariant

The assumption that *CASE 1* (see Algorithm 2, Line 6) respects the implicit invariant is already disproven by the previous example shown in Figure 2.1: As the clustering $\zeta(G^2)$ returned by *CASE 2* is not correct, i.e., $\zeta(G^2)$ does not respect the clustering quality, it follows that none of the conditions for a correct result of *CASE 2* are met. In particular, by Fact 2, there exists no min-cut tree of graph G^1_α that induces the previous clustering $\zeta(G^1)$. Hence the invariant is violated by *CASE 1*. In Section 2.3.1 we will see that *CASE 2* (see Algorithm 2, Line 9) respects the invariant, but implies a very strict property of the input clustering. Therefore, *CASE 2* will turn out to be quite unlikely. Finally *CASE 3* (see Algorithm 2, Line 28), will be proven to be not feasible in general and even for feasible instances it still will not respect the invariant, as illustrated in Section 2.3.2.

In this section we showed that the maintenance of clustering quality is not equivalent to the assertion of the invariant described in Fact 1, which is even stronger. Although Saha and Mitra implicitly assume the invariant to be met by their proposed algorithms, for all algorithms they fail to prove this invariant, which constitutes a massive methodical error through the whole argumentation in their paper. With *CASE 1* they even distinguish a case in their inter-edge-add algorithm for which the invariant can be explicitly disproven. Note further, that the *intra-cluster edge addition algorithm* (or simply *intra-edge-add algorithm*) on page three in [SM06] is feasible despite of the methodical error made by Saha and Mitra [SM06]. In Chapter 6, Subsection 6.3.3, we will see that this algorithm, nevertheless, respects the invariant formulated in Fact 1. A detailed description of the *edge deletion algorithms*, however, is omitted in [SM06].

2.3 Discussion of Inconsistencies

Aside from the methodical error described in the previous section, in this section we discuss further inconsistencies in the approach of Saha and Mitra [SM06], which again regard the inter-edge-add algorithm (see Algorithm 2). The first inconsistency concerns a lemma, namely Lemma 2 in [SM06], which *CASE 2* (see Line 9) of the inter-edge-add algorithm is based on. We will call this lemma *merging lemma* and refer to it as Lemma 1. A second inconsistency necessarily being discussed concerns the formulation of *CASE 3* of the inter-edge-add algorithm (see Algorithm 2, Line 28). This inconsistency further involves a lemma, namely Lemma 4 in [SM06], used by Saha and Mitra to prove the correctness of *CASE 3*. We will call the latter *unaffect lemma* and refer to it as Lemma 6.

2.3.1 The Merging Lemma and *CASE 2*

First of all we introduce the merging lemma as Lemma 1 and explain the used notation. Furthermore, we review two lemmas which are used by Saha and Mitra to prove the merging lemma. From these lemmas we will later deduce the proof of Fact 1 (see the previous Section 2.2). Before proving Fact 1 we spend a closer look at the proof of the merging lemma given by Saha and Mitra [SM06] and show that this proof is not complete at all. Therefore, we will round off the proof and finally conclude that the implicit invariant described in Fact 1 constitutes a sufficient condition for the correctness of *CASE 2*. As this is the assertion of Fact 2 (see the previous Section 2.2), this fact will be proven true, too.

Lemma: Merging lemma

Lemma 1 *(Merging lemma) If it holds that*

$$\frac{2\,c(C_1, C_2)}{|V|} \geq \alpha,$$

then merging cluster C_1 and C_2 maintains the clustering quality.

Lemma 1 considers a given clustering $\zeta(G)$ of a graph $G = (V, E, c())$ for which the clustering quality defined by Flake et al. [FTT04] holds. The notation of C_1 and C_2 denotes two arbitrary clusters in $\zeta(G)$, and the parameter α results from the definition of the clustering quality (see Section 2.1). The merging lemma says that on condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ holds for a pair of clusters C_1 and C_2 in a clustering $\zeta(G)$ which achieves inter- and intra-cluster quality regarding a fixed parameter α, the new clustering $\bar{\zeta}(G)$ resulting from merging C_1 and C_2 also achieves inter- and intra-cluster quality regarding the same parameter α. Note, that the fact that graph G is supposed to be dynamic does not appear in the assertion of the merging lemma. Nevertheless *CASE 2* of the inter-edge-add algorithm (see Algorithm 2, Line 9) bases on the merging lemma: Adding a new edge $e_\oplus = \{b, d\}$ in a merged cluster $\bar{C} = C_b \cup C_d$ does not affect the clustering quality on condition that $\bar{C}$ already respects the intra- and inter-cluster quality before the edge addition. For the proof of the merging lemma Saha and Mitra [SM06] use a lemma which we state as Lemma 2 and call *helping lemma*.

Lemma: Helping lemma

Lemma 2 *(Helping lemma) Let $T(G_\alpha)$ denote a min-cut tree of a graph G after adding the artificial sink t, i.e., $T(G_\alpha)$ denotes a min-cut tree of graph G_α. Then for each nontrivial cut $(P, C \setminus P)$ in a connected component C of $T(G_\alpha)$ (after removing the sink t) it holds that*

$$c(W, C \setminus P) \leq c(P, C \setminus P), \text{ with } W := \{t\} \cup (V \setminus C) \text{ in } G_\alpha.$$

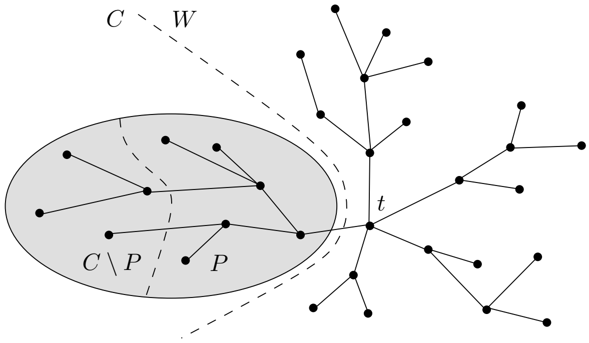


Figure 2.2: Situation described in the helping lemma.

Figure 2.2 illustrates the situation described in the helping lemma. Note, that in the helping lemma Saha and Mitra [SM06] mention objects introduced by Flake et al. [FTT04]; for example the enlarged graph G_α and the min-cut tree $T(G_\alpha)$. As the helping lemma serves to prove the merging lemma, this formulation already allows a guess that Saha and Mitra implicitly assume the clustering $\zeta(G)$ in the merging lemma to result from the cut-clustering method of Flake et al. So this already indicates Fact 1. Furthermore, Saha and Mitra deduce their helping lemma directly from Lemma 3, page 391 in [FTT04], which we call *basic lemma*. In the following we refer to the basic lemma as Lemma 3.

Lemma: Basic lemma

Lemma 3 *(Basic lemma) Let $T(G)$ be a min-cut tree of a graph $G = (V, E, c())$, and let $\{u, v\}$ be an edge of $T(G)$. The edge $\{u, v\}$ induces a cut $(U, V \setminus U)$ in G, with $u \in U$, $v \in V \setminus U$. Now take any cut $(P, U \setminus P)$ of U such that P and $(U \setminus P)$ are nonempty, $u \in P$, $P \cup (U \setminus P) = U$ and $P \cap (U \setminus P) = \emptyset$. Then it holds that*

$$c(V \setminus U, U \setminus P) \leq c(P, U \setminus P).$$

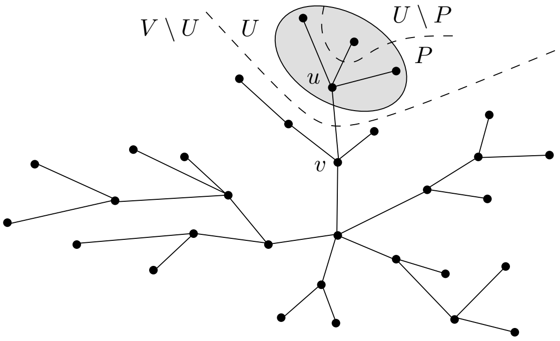


Figure 2.3: Situation described in the basic lemma.

Helping lemma follows from basic lemma

The situation described in the basic lemma is shown in Figure 2.3. The helping lemma (see Lemma 2) follows from the basic lemma (see Lemma 3) by regarding the graph G_α with the artificial sink t included as graph G in Lemma 3. The min-cut tree $T(G_\alpha)$ in Lemma 2 corresponds to T in Lemma 3 and the sink t is implicitly identified with v. Let y denote the vertex in Lemma 2 which links the subtree C to t in $T(G_\alpha)$ via the edge $\{t, y\}$. Then y is related to u in Lemma 3. With $C \ni y$ corresponding to $U \ni u$ and $W \ni t$ corresponding to $(V \setminus U) \ni v$, the cut $(P, C \setminus P)$ of C in Lemma 2 finally can be identified with the cut $(P, U \setminus P)$ of U in Lemma 3. However, the assertion as formulated in Lemma 2 only follows from Lemma 3 if we assume the set P in Lemma 2 to include y, and therefore, to correspond to $P \ni u$ in Lemma 3. Otherwise P and $C \setminus P$ in Lemma 2 swap their roles. As this observation will become important later for understanding why the given proof of the merging lemma by Saha and Mitra [SM06] is not complete, we state it here as Observation 1.

Observation: Feasible deduction of helping lemma

Observation 1 *The assertion of the helping lemma (see Lemma 2) only follows from the basic lemma (see Lemma 3) if we assume the set P in Lemma 2 to include y, and therefore, to correspond to $P \ni u$ in Lemma 3. Otherwise P and $C \setminus P$ in Lemma 2 swap their roles.*

Altogether we remark that the helping lemma considers a special case of the basic lemma, as it regards the special graph G_α and reduces the assertion on edges incident with the sink t in $T(G_\alpha)$.

A closer look at the proof given by Flake et al. [FTT04], page 391, for the basic lemma (see Lemma 3) shows that the condition of $(U, V \setminus U)$ being a minimum u-v-cut for two arbitrary vertices $u, v \in V$ already is sufficient. The existence of a specially shaped min-cut tree $T(G)$ of G is not needed as long as we apply the basic lemma on a single minimum u-v-cut. In the situation of the helping lemma (see Lemma 2) this weaker condition says that it suffices if there exists a vertex y in the considered component C such that $(C, V_\alpha \setminus C)$ is a minimum y-t-cut in $G_\alpha = (V_\alpha, E_\alpha, c_\alpha())$. Again the existence of a specially shaped min-cut tree $T(G_\alpha)$

of G_α is not needed as long as we apply the helping lemma on a single component C defined by a minimum y-t-cut. It further follows that the helping lemma only can be applied on components of a given clustering $\zeta(G)$ that are induced by a minimum y-t-cut in the enlarged graph G_α, with t denoting the designated sink and y a vertex in the component. We will see in Chapter 6 that this condition required simultaneously for each cluster of a given clustering $\zeta(G)$, however, is equivalent to the existence of a parameter α and a specially shaped min-cut tree $T(G_\alpha)$ that induces the clustering $\zeta(G)$ (see Observation 3, Subsection 6.3.3). So Saha and Mitra [SM06] must assume the clustering $\zeta(G)$ in the merging lemma (see Lemma 1) to result from the cut-clustering method, i.e., they must assume the existence of a min-cut tree $T(G_\alpha)$ inducing $\zeta(G)$, as for their proof of the merging lemma they apply the helping lemma on each cluster in $\zeta(G)$. So Fact 1 given in Section 2.2 is proven true, as the merging lemma legitimates *CASE 2* of the inter-edge-add algorithm.

Proof of Fact 1

A Closer Look at the Proof of the Merging Lemma

We now take a closer look at the proof of the merging lemma (see Lemma 1) given by Saha and Mitra [SM06] and show, why this proof suffers from incompleteness. To this end we assume the clustering $\zeta(G)$ of graph G to result from the cut-clustering method, as Saha and Mitra do.

The proof of the merging lemma is divided into one part considering the maintenance of inter-cluster quality after merging two arbitrary clusters C_1 and C_2 in a clustering $\zeta(G)$ and another part related to the intra-cluster quality of all clusters in the new clustering $\bar{\zeta}(G)$ and the new cluster $\bar{C} = C_1 \cup C_2$ in particular. Both parts of the proof given in [SM06] are correct but suffer from imprecise explanation. The part related to the intra-cluster quality additionally suffers from incompleteness. In the following we illustrate why the proof given by Saha and Mitra [SM06] is not complete and try to give a more understandable formulation of this proof.

The first part, which considers the maintenance of inter-cluster quality in $\bar{\zeta}(G)$ after merging C_1 and C_2, is proven by Saha and Mitra very quickly as follows:

Proof of inter-cluster quality

Proof. (*Maintenance of inter-cluster quality*) Let $\bar{C} = C_1 \cup C_2$ denote the result of merging C_1 and C_2. After merging C_1 and C_2 the inter-cluster connectivity $c(C, V \setminus C)$ obviously remains unchanged for all clusters $C \in \zeta(G) \setminus \{C_1, C_2\}$. Note that there is no dynamic effect, i.e., no addition or deletion of edges or vertices, considered yet. As the clustering $\zeta(G)$ in the merging lemma (see Lemma 1) is thought to respect the clustering quality, it holds for the inter-cluster quality of C_1 and C_2 that

$$c(C_1, V \setminus C_1) \leq \alpha |V \setminus C_1| \quad \text{and} \tag{2.1}$$
$$c(C_2, V \setminus C_2) \leq \alpha |V \setminus C_2|. \tag{2.2}$$

For the new cluster $\bar{C} = C_1 \cup C_2$ we get

$$\begin{aligned} c(\bar{C}, V \setminus \bar{C}) &= c(C_1, V \setminus C_1) + c(C_2, V \setminus C_2) - 2\,c(C_1, C_2) \\ &\leq \alpha |V \setminus C_1| + \alpha\,|V \setminus C_2| - 2\,c(C_1, C_2) \quad \text{by (2.1) and (2.2)} \\ &= \alpha\,|V \setminus \bar{C}| + \alpha\,|V| - 2\,c(C_1, C_2). \end{aligned}$$

The condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ yields

$$\alpha\,|V| - 2\,c(C_1, C_2) \leq 0, \tag{2.3}$$

and therefore, it holds that

$$\begin{aligned} c(\bar{C}, V \setminus \bar{C}) &\leq \alpha\, |V \setminus \bar{C}| + \underbrace{\alpha\, |V| - 2\, c(C_1, C_2)}_{\leq 0\text{ , by (2.3)}} \\ &\leq \alpha\, |V \setminus \bar{C}|. \end{aligned}$$

$\square$

Proof of intra-cluster quality

The proof of the second part, which considers the intra-cluster quality in $\bar{\zeta}(G)$ after merging C_1 and C_2, is trivial for all clusters not affected by the addition of $e_\oplus$, i.e., for all clusters apart from $\bar{C} = C_1 \cup C_2$. The intra-cluster quality for $\bar{C}$ can be proven as follows:

Proof. The condition that $2\, c(C_1, C_2)/|V| \geq \alpha$ yields

$$\begin{aligned} c(C_1, C_2) &\geq \frac{|V|}{2}\, \alpha \\ &\geq \alpha\, \min\{|C_1|, |C_2|\} \end{aligned} \tag{2.4}$$

So the cut (C_1, C_2) in $\bar{C}$ does not violate the intra-cluster quality of $\bar{C}$. As the definition of the intra-cluster quality (see Section 2.1) involves all nontrivial cuts in $\bar{C}$, we still need to prove that also an arbitrary nontrivial cut $(P, \bar{C} \setminus P) \neq (C_1, C_2)$ does not violate the intra-cluster quality of $\bar{C}$, i.e., we need to show that it holds that $(P, \bar{C} \setminus P) \geq \alpha\, \min\{|P|, |\bar{C} \setminus P|\}$.

To this end let $(P, \bar{C} \setminus P) \neq (C_1, C_2)$ be a nontrivial cut in $\bar{C} = C_1 \cup C_2$. For a better readability we temporarily rename the set $\bar{C} \setminus P =: Q$. So in the following we consider a cut (P, Q) in $\bar{C}$. The two sets P and Q can be decomposed into $P = P_1 \,\dot{\cup}\, P_2$ and $Q = Q_1 \,\dot{\cup}\, Q_2$, where $P_1, Q_1 \subseteq C_1$ and $P_2, Q_2 \subseteq C_2$, i.e., $P_1 \,\dot{\cup}\, Q_1 = C_1$ and $P_2 \,\dot{\cup}\, Q_2 = C_2$. The union of disjoint sets is represented from now on by the operator "+" in this proof. Figure 2.4 illustrates the decomposition of $\bar{C} = C_1 \cup C_2$ into P_1, P_2, Q_1, Q_2. Without loss of generality we assume the sets P_1 and P_2 to be not empty. This assumption is feasible due to the following consideration: There is at least one of the sets P_1 and P_2 not empty, as $P_1 \cup P_2 = \emptyset$ would yield a trivial cut. If we assume P_2 to be empty, it follows that both sets Q_1 and Q_2 are not empty, as $P_2 \cup Q_1 = \emptyset$ would yield the cut (C_1, C_2), and $P_2 \cup Q_2 = C_2 = \emptyset$ is not feasible. So in the case of P_2 being empty we can choose Q_1 and Q_2 as nonempty sets instead of P_1 and P_2. A symmetric argumentation holds for the assumption of P_1 being empty.

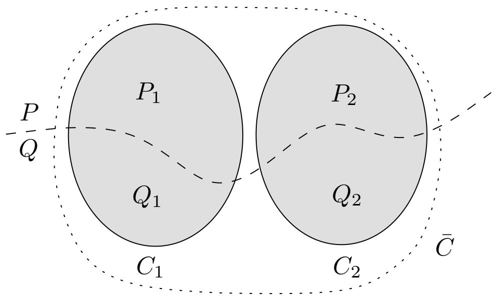


Figure 2.4: Decomposition of $\bar{C} = C_1 \cup C_2$ into P_1, P_2, Q_1, Q_2.

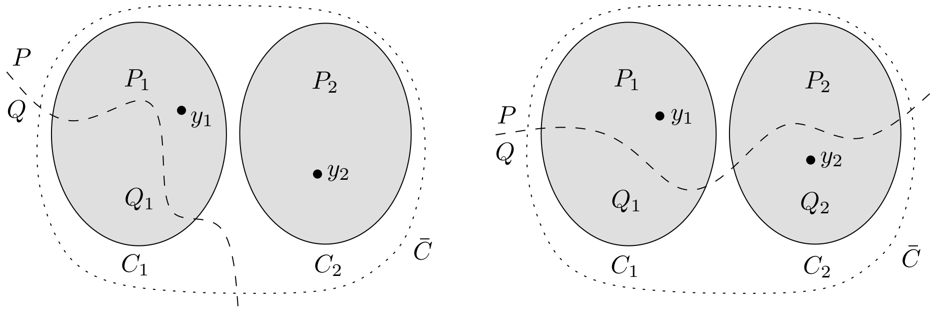
With P_1 and P_2 being not empty it holds for the intra-cluster connectivity $c(P, Q)$, according to Saha and Mitra [SM06], that

$$\begin{aligned} c(P,Q) &= c(P_1 + P_2, Q_1 + Q_2) \\ &= c(P_1,Q_1) + c(P_1,Q_2) + c(P_2,Q_1) + c(P_2,Q_2) \\ &\geq c(P_1,Q_1) + c(P_2,Q_2) && (2.5) \\ &\geq c(W_1,Q_1) + c(W_2,Q_2) && (2.6) \\ &\geq \alpha|Q_1| + \alpha|Q_2| && (2.7) \\ &= \alpha|Q| && (2.8) \\ &\geq \alpha \ \min\{|P|,|Q|\}. && (2.9) \end{aligned}$$

Equation (2.6) can be deduced piecewise from Equation (2.5) with the aid of the helping lemma (see Lemma 2), as (P_1, Q_1) is a nontrivial cut in the connected component C_1, and (P_2, Q_2) is a nontrivial cut in the connected component C_2. Remember that we assumed the clustering $\zeta(G)$ to result from the cut-clustering method and the sets P_1 and P_2 to be not empty. The notation of W_1 in Equation (2.6) denotes the union of $\{t\}$ and $V \setminus C_1$, analogously W_2 denotes the union of $\{t\}$ and $V \setminus C_2$. ⊔⊓

Omission by Saha and Mitra

At this point Saha and Mitra [SM06] consider the merging lemma proven. However, they ignore the small detail mentioned in Observation 1. The conclusion from Equation (2.5) to Equation (2.6) with the aid of the helping lemma (see Lemma 2) only is feasible if we require y_1 to be included in P_1 and y_2 to be included in P_2, with y_1 denoting the vertex connecting the subtree C_1 to t in $T(G_\alpha)$ via the edge $\{t, y_1\}$, and analogously y_2 denoting the connecting vertex of subtree C_2. Figure 2.5a shows a cut (P, Q) that meets this requirement. In the case of Q_1 including y_1 and Q_2 including y_2 the conclusion is also feasible, however, the sets Q_1 and P_1 as well as Q_2 and P_2 swap their roles in the whole proof, by Observation 1. By contrast, if only one pair of sets, either P_1 and Q_1 or P_2 and Q_2, swaps the roles, i.e., if y_1 and y_2 do not lie in the same set P or Q, i.e., if the cut (P, Q) separates y_1 and y_2, then the conclusion fails. Figure 2.5b shows such an example. In the illustrated situation only P_2 and Q_2 swap their roles, as y_2 lies in Q_2. As the sets P_2 and Q_2 are fixed by the cut (P, Q), a simple renaming is impossible. So Equation (2.6) deduced piecewise from Equation (2.5) with the aid of the helping lemma looks as follows:



(a) (P, Q) does not separate y_1 and y_2, with $y_1, y_2 \in P$, Q_2 is empty

(b) (P, Q) separates y_1 and y_2, with $y_1 \in P$ and $y_2 \in Q$, no empty set

Figure 2.5: Different positions of y_1 and y_2.

$$c(P_1,Q_1) + c(P_2,Q_2) \geq c(W_1,Q_1) + c(P_2,W_2).$$

The next conclusion (see Equation (2.7)) results from the construction of G_α and the fact that the artificial sink t lies in W_1 or rather W_2. In the situation of Figure 2.5b this conclusion yields the following assertion:

$$c(W_1, Q_1) + c(P_2, W_2) \geq \alpha|Q_1| + \alpha|P_2|.$$

Now it becomes obvious that merging $|Q_1|$ and $|P_2|$ to $|Q|$ or $|P|$ as in Equation (2.8) is not possible anymore. So the line of estimation breaks here without a reasonable result in the case of (P, Q) separating y_1 and y_2. As Saha and Mitra [SM06] miss to explore these separating cuts, apart from the cut (C_1, C_2), their proof given for the merging lemma is not complete. Remember that the cut (C_1, C_2) separates y_1 and y_2, but respects the intra-cluster quality of $\bar{C}$ due to the condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ (see Equation (2.4)).

Completion of the Proof of the Merging Lemma

In the following we will complete the proof of the merging lemma (see Lemma 1). As illustrated in the previous paragraphs, the proof of the merging lemma given by Saha and Mitra [SM06] is not complete, because in the part concerning the maintenance of intra-cluster quality Saha and Mitra miss to explore the behavior of cuts that separate y_1 and y_2. We will show now that also these separating cuts do not violate the intra-cluster quality of $\bar{C} = C_1 \cup C_2$.

To this end we demonstrate that the condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ in G required by the merging lemma (see Lemma 1) forces any arbitrary y_1-y_2-cut in $\bar{C} = C_1 \cup C_2$ to maintain the intra-cluster quality of $\bar{C}$. More precisely, we prove that on this condition for an arbitrary cut $(P, Q) \neq (C_1, C_2)$ that separates y_1 and y_2 in $\bar{C}$ it holds that $c(P, Q) \geq \alpha \min\{|P|, |Q|\}$. Again we assume the initial clustering $\zeta(G) \ni C_1, C_2$ to result from the cut-clustering method of Flake et al. [FTT04].

Idea of completing the proof

The idea of the following proof is to show that, on condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ in G, any y_1-y_2-cut (P, Q) in $\bar{C}$ which does not respects the intra-cluster quality would induce a y_1-t-cut or a y_2-t-cut in G_α which is cheaper than the previous minimum y_1-t-cut, or the previous minimum y_2-t-cut respectively, represented by the min-cut tree $T(G_\alpha)$ of G_α, with $T(G_\alpha)$ inducing the clustering $\zeta(G) \ni C_1, C_2$. This contradicts the min-cut tree properties. It follows that such a cut does not exist, and vice versa, that each y_1-y_2-cut (P, Q) in $\bar{C}$ respects the intra-cluster quality on condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ in G.

The notation of the following proof adheres to the situation illustrated in Figure 2.6. Figure 2.6 shows the two clusters C_1 and C_2 in a stylized graph G_α. The minimum y_1-t-cut and the minimum y_2-t-cut represented in the min-cut tree $T(G_\alpha)$ are marked as dotted lines. The dashed line represents a cut $(P, Q) \neq (C_1, C_2)$ in $\bar{C}$ separating y_1 and y_2.

Completion of the proof of intra-cluster quality

Proof. Let $G_\alpha = (V_\alpha, E_\alpha, c_\alpha())$ denote the graph resulting from G by adding the artificial sink t. For the definition of V_α, E_α and $c_\alpha()$ see Algorithm 1. The sets P and Q are decomposed as before. We now consider a y_1-y_2-cut $(P, Q) \neq (C_1, C_2)$ in $\bar{C}$ which does not respects the intra-cluster quality. This is, for (P, Q) it holds that

$$c(P, Q) < \alpha \min\{|P|, |Q|\}. \tag{2.10}$$

We want to show that such a y_1-y_2-cut (P, Q) induces a y_1-t-cut or a y_2-t-cut in G_α cheaper than the minimum y_1-t-cut, or the minimum y_2-t-cut respectively, represented by the min-cut tree $T(G_\alpha)$ (see the dotted lines in Figure 2.6). To this end consider the y_1-t-cut $\theta_1 := (P_1, V_\alpha \setminus P_1)$ and the y_2-t-cut $\theta_2 := (Q_2, V_\alpha \setminus Q_2)$

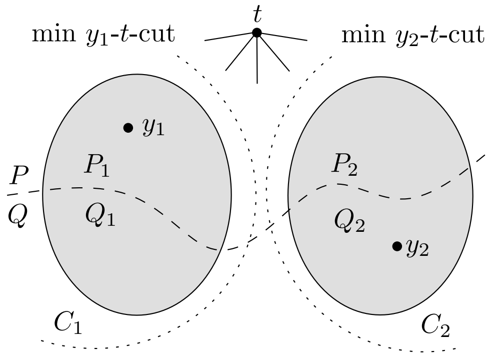


Figure 2.6: Cut (P, Q) separates y_1 and y_2, with $y_1 \in P$ and $y_2 \in Q$.

in G_α, which are defined by (P, Q). Let $\theta_{\min}(1)$ and $\theta_{\min}(2)$ denote the minimum y_1-t-cut and the minimum y_2-t-cut represented by the min-cut tree $T(G_\alpha)$. For the weights in G_α we will show in the following that it holds that $c_\alpha(\theta_1) + c_\alpha(\theta_2) < c_\alpha(\theta_{\min}(1)) + c_\alpha(\theta_{\min}(2))$. So at least one of the minimum cuts $\theta_{\min}(1)$ or $\theta_{\min}(2)$ can be replaced by a cheaper y_1-t-cut, or y_2-t-cut respectively. For $c_\alpha(\theta_1) + c_\alpha(\theta_2)$ we get

$$\begin{aligned}
c_\alpha(\theta_1) &= \alpha\,|P_1| + c(P_1, Q_1) + c(P_1, P_2) + c(P_1, Q_2) + c(P_1, V \setminus (C_1 \cup C_2)) \\
c_\alpha(\theta_2) &= \alpha\,|Q_2| + c(P_2, Q_2) + c(Q_1, Q_2) + c(P_1, Q_2) + c(Q_2, V \setminus (C_1 \cup C_2)) \\
c_\alpha(\theta_1) + c_\alpha(\theta_2) &= \alpha\,(|P_1| + |Q_2|) + c(P_1, Q_1) + c(P_2, Q_2) \\
&\quad + \left\{ \begin{array}{l} c(P_1, P_2) + c(Q_1, Q_2) + 2\,c(P_1, Q_2) + \\ c(P_1, V \setminus (C_1 \cup C_2)) + c(Q_2, V \setminus (C_1 \cup C_2)) \end{array} \right\} := B
\end{aligned}$$

For $c_\alpha(\theta_{\min}(1)) + c_\alpha(\theta_{\min}(2))$ we get

$$\begin{aligned}
c_\alpha(\theta_{\min}(1)) &= \alpha\,|C_1| + c(P_1, P_2) + c(P_1, Q_2) + c(Q_1, Q_2) + c(Q_1, P_2) \\
&\quad + c(P_1, V \setminus (C_1 \cup C_2)) + c(Q_1, V \setminus (C_1 \cup C_2)) \\
c_\alpha(\theta_{\min}(2)) &= \alpha\,|C_2| + c(P_2, P_1) + c(P_2, Q_1) + c(Q_2, Q_1) + c(Q_2, P_1) \\
&\quad + c(Q_2, V \setminus (C_1 \cup C_2)) + c(P_2, V \setminus (C_1 \cup C_2)) \\
c_\alpha(\theta_{\min}(1)) + c_\alpha(\theta_{\min}(2)) &= \alpha\,(|C_1| + |C_2|) + c(P_1, P_2) + c(Q_1, Q_2) \\
&\quad + 2\,c(P_2, Q_1) + c(P_2, V \setminus (C_1 \cup C_2)) + c(Q_1, V \setminus (C_1 \cup C_2)) \\
&\quad + \left\{ \begin{array}{l} c(P_1, P_2) + c(Q_1, Q_2) + 2\,c(P_1, Q_2) + \\ c(P_1, V \setminus (C_1 \cup C_2)) + c(Q_2, V \setminus (C_1 \cup C_2)) \end{array} \right\} = B
\end{aligned}$$

For the final conclusion we need the aid of the following equation:

$$\begin{aligned}
c(P, Q) &= c(P_1, Q_1) + c(P_2, Q_2) + \underbrace{c(P_1, Q_2) + c(P_2, Q_1)}_{:=A} \\
&< \alpha \min\{|P|, |Q|\} \quad \text{by (2.10)} \\
&\leq \frac{|V|}{2}\,\alpha \\
&\leq c(C_1, C_2) \quad \text{by (2.4)} \\
&= c(P_1, P_2) + c(Q_1, Q_2) + \underbrace{c(P_1, Q_2) + c(P_2, Q_1)}_{=A}
\end{aligned}$$

So we get

$$c(P_1, Q_1) + c(P_2, Q_2) < c(P_1, P_2) + c(Q_1, Q_2). \tag{2.11}$$

Now the assertion that $c_\alpha(\theta_1) + c_\alpha(\theta_2) < c_\alpha(\theta_{\min}(1)) + c_\alpha(\theta_{\min}(2))$ can be proven as follows:

$$\begin{aligned} c_\alpha(\theta_1) + c_\alpha(\theta_2) &= \alpha\,(|P_1| + |Q_2|) + c(P_1, Q_1) + c(P_2, Q_2) + B \\ &< \alpha\,(|P_1| + |Q_2|) + c(P_1, P_2) + c(Q_1, Q_2) + B \quad \text{by (2.11)} \\ &\leq \alpha\,(|C_1| + |C_2|) + c(P_1, P_2) + c(Q_1, Q_2) + B \\ &\leq \alpha\,(|C_1| + |C_2|) + c(P_1, P_2) + c(Q_1, Q_2) + B \\ &\quad + 2\,c(P_2, Q_1) + c(P_2, V \setminus (C_1 \cup C_2)) + c(Q_1, V \setminus (C_1 \cup C_2)) \\ &= c_\alpha(\theta_{\min}(1)) + c_\alpha(\theta_{\min}(2)). \end{aligned}$$

Therefore, at least one of the minimum cuts $\theta_{\min}(1)$ or $\theta_{\min}(2)$ represented by the min-cut tree $T(G_\alpha)$ can be replaced by a cheaper cut. This contradicts the min-cut tree properties. It follows that there exists no cut $(P, \bar{C} \setminus P)$ in $\bar{C}$ that separates y_1 and y_2 and simultaneously violates the intra-cluster quality, i.e., meets condition (2.10). So each y_1-y_2-cut $(P, \bar{C} \setminus P)$ does respect the intra-cluster quality of $\bar{C} = C_1 \cup C_2$ on condition that $2\,c(C_1, C_2)/|V| \geq \alpha$, which constitutes the prerequisite of *CASE 2*. ◻

Proof of Fact 2

Now the merging lemma (see Lemma 1), and the correctness of *CASE 2* respectively, is completely proven, as we showed that on condition that $2\,c(C_1, C_2)/|V| \geq \alpha$ all nontrivial cuts $(P, \bar{C} \setminus P)$ in the merged cluster $\bar{C} = C_1 \cup C_2$ respect the intra-cluster quality of $\bar{C}$. Saha and Mitra [SM06] only showed this assertion for cuts $(P, \bar{C} \setminus P)$ that do not separate the vertices $y_1 \in C_1$ and $y_2 \in C_2$ (see the first part of Section 2.3.1). We completed the proof given by Saha and Mitra by also showing the assertion for y_1-y_2-cuts $(P, \bar{C} \setminus P)$ in $\bar{C}$. We see that the whole proof only requires the considered clustering $\zeta(G)$ to result from the cut-clustering method of Flake et al. [FTT04], which constitutes the implicit invariant given in Fact 1. The implicit invariant, therefore, turns out to be a sufficient condition for the correctness of the merging lemma, and *CASE 2* respectively. Hence, Fact 2 in Section 2.2 is proven true.

Recap on the necessity of the invariant

Remember that there may be even weaker conditions than the implicit invariant on which *CASE 2* returns a correct result, i.e., on which the merging lemma holds. It is not feasible to simply deduce the invariant to be necessary from the previous example in Section 2.2 (see Figure 2.1), although in this example it seems that *CASE 2* returns an invalid result because the clustering $\zeta(G^1)$ considered before does not provide the properties representing the invariant, i.e., is not induced by a min-cut tree of G^1_α. However, this interrelation does not necessarily exist. There also may be other initial clusterings detectable which do not provide the required properties, but meet some weaker conditions not met by $\zeta(G^1)$ such that *CASE 2* returns a correct result. The existence of such an initial clustering would disprove the necessity of the invariant regarding the correctness of *CASE 2*, and the merging lemma respectively.

Strict clustering property required by *CASE 2*

In the previous paragraph we gave a proper proof of the merging lemma (see Lemma 1), which legitimates *CASE 2* of the inter-edge-add algorithm (see Algorithm 2, Line 9), but we still do not know if *CASE 2* respects the implicit invariant given in Fact 1 in Section 2.2. However, before answering this final question we first discuss a further aspect of *CASE 2*. In the following we illustrate that the prerequisite of *CASE 2* actually implies a very strict property of the input clustering $\zeta(G)$ which probably makes *CASE 2* occurring very rarely. This is, *CASE 2* turns out to be quite unlikely.

We explore how an input clustering $\zeta(G)$ ending up in *CASE 2* in the inter-edge-add algorithm does look like. To meet a sufficient condition for the correctness of *CASE 2*, by Fact 2, we assume the input clustering $\zeta(G)$ to result from the cut-clustering method of Flake et al. [FTT04]. Therefore, the clustering $\zeta(G)$ in particular respects the clustering quality defined by Flake et al. (see Section 2.1). The clusters C_b and C_d of the input clustering $\zeta(G)$, which are affected by the inter-cluster addition of the new edge $e_\oplus = \{b, d\}$, are shown in a stylized graph G_α in Figure 2.7. As the prerequisite of *CASE 2* (see Algorithm 2, Line 9) considers the input clustering $\zeta(G)$ before the edge addition, the edge $e_\oplus = \{b, d\}$ is indicated by a dashed line in Figure 2.7.

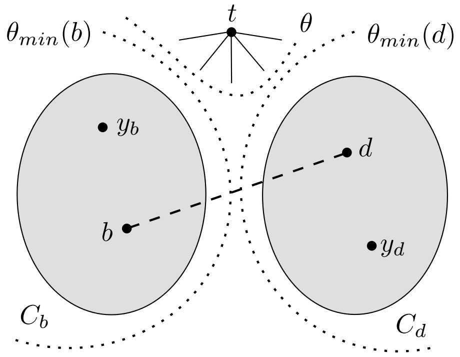


Figure 2.7: Overview of different cuts in graph G_α.

As the inter-edge-add algorithm is supposed to end up in *CASE 2*, for the input clustering $\zeta(G)$ it must hold that $2\,c(C_b, C_d)/|V| \geq \alpha$ which is equivalent to

$$c(C_b, C_d) \geq \frac{|V|}{2}\,\alpha. \tag{2.12}$$

The following lemma now states the strict property of the input clustering $\zeta(G)$ required by *CASE 2* due to condition (2.12).

Lemma: Strict property required by *CASE 2*

Lemma 4 *Let $\theta_{(b,d)} := (C_b \cup C_d, V_\alpha \setminus (C_b \cup C_d))$ denote the cut that separates the merged cluster $\bar{C} = C_b \cup C_d$ from the remaining vertices in graph G_α. Further let $\theta_{min}(b) := (C_b, V_\alpha \setminus C_b)$ and $\theta_{min}(d) := (C_d, V_\alpha \setminus C_d)$ denote the cuts that separate the single clusters C_b and C_d from the rest (see Figure 2.7). Then condition (2.12) implies that for the weights in G_α it holds that*

$$c_\alpha(\theta_{(b,d)}) = c_\alpha(\theta_{min}(b)) = c_\alpha(\theta_{min}(d)).$$

Proof. Consider the cut $\theta_{\min}(b) = (C_b, V_\alpha \setminus C_b)$ in G_α, which is a minimum y_b-t-cut for a vertex $y_b \in C_b$, as the input clustering $\zeta(G)$ is supposed to be induced by a min-cut tree $T(G_\alpha)$. For this cut we get

$$c_\alpha(\theta_{\min}(b)) = \alpha\,|C_b| + c(C_b, V \setminus (C_b \cup C_d)) + \underbrace{c(C_b, C_d)}_{\geq \frac{|V|}{2}\alpha,\text{ by (2.12)}} \tag{2.13}$$

The cut $\theta := (\{t\}, V_\alpha \setminus \{t\})$ (see Figure 2.7) also separates y_b and t and has weight

$$c_\alpha(\theta) = \alpha\,|V|. \tag{2.14}$$

From Equation (2.13) and (2.14) it follows that

$$\alpha\,|C_b| + c(C_b, V \setminus (C_b \cup C_d)) \leq \frac{|V|}{2}\,\alpha \leq c(C_b, C_d), \tag{2.15}$$

as otherwise θ would be a cheaper y_b-t-cut than $\theta_{\min}(b)$ in G_α. By making use of Equation (2.15) we now compare cut $\theta_{(b,d)} = (C_b \cup C_d, V_\alpha \setminus (C_b \cup C_d))$, which separates $\bar{C}$ from the remaining vertices of G_α, with cut $\theta_{\min}(d) = (C_d, V_\alpha \setminus C_d)$, which just cuts off C_d. Thereby cut $\theta_{(b,d)}$ turns out to be cheaper than or of the same weight as cut $\theta_{\min}(d)$.

$$\begin{aligned}
c_\alpha(\theta_{(b,d)}) &= \alpha\,|C_d| + c(C_d, V \setminus (C_b \cup C_d)) + \underbrace{\alpha\,|C_b| + c(C_b, V \setminus (C_b \cup C_d))}_{\leq c(C_b, C_d),\ \text{by (2.15)}} \\
&\leq \alpha\,|C_d| + c(C_d, V \setminus (C_b \cup C_d)) + c(C_b, C_d) \\
&= c_\alpha(\theta_{\min}(d))
\end{aligned}$$

As $\theta_{\min}(d)$ is a minimum y_d-t-cut in G_α for a vertex $y_d \in C_d$, and cut $\theta_{(b,d)}$ also separates y_d and t, the cut $\theta_{(b,d)}$ can not be cheaper than $\theta_{\min}(d)$. A symmetric argumentation holds for the comparison of $\theta_{(b,d)}$ and $\theta_{\min}(b)$. Altogether condition (2.12) of *CASE 2*, hence, implies that

$$c_\alpha(\theta_{\min}(d)) = c_\alpha(\theta_{b,d}) = c_\alpha(\theta_{\min}(d)) \tag{2.16}$$

□

Replacing now condition (2.12) in the merging lemma (see Lemma 1), which founds *CASE 2* in the inter-edge-add algorithm, by the new condition (2.16) yields the following *new merging lemma*.

Lemma:
New merging
lemma

Lemma 5 *(New merging lemma) On the new condition (2.16) merging cluster C_b and C_d of the input clustering $\zeta(G)$ yields a clustering $\zeta(G_\oplus)$ of the modified graph $G_\oplus$ that again results from the cut-clustering method, and therefore, also respects the clustering quality.*

The proof of this new merging lemma will be given later in Chapter 6, Subsection 6.3.3, as for this proof we need some lemmas not stated yet. We will see that Lemma 5 can be proven much shorter than the previous merging lemma.

CASE 2
respects
the implicit
invariant

As the new condition (2.16) is implicated by the previous condition (2.12) of *CASE 2*, by the new merging lemma it follows that *CASE 2* also respects the implicit invariant given in Fact 1. Furthermore, the reverse implication, i.e., the implication of condition (2.12) by condition (2.16), is easy to see. So both conditions turn out to be equivalent.

This paragraph demonstrated that *CASE 2* developed by Saha and Mitra [SM06] can be replaced by a new *CASE 2'* founded by the new merging lemma, which is much easier to prove. Moreover, we realized that the condition of *CASE 2* is very strict. Therefore, *CASE 2* is not expected to occur very often, and hence, to cause a significant saving of time and effort in a dynamic application of the inter-edge-add algorithm.

2.3.2 The Unaffect Lemma and *CASE 3*

In this section we introduce the *unaffect lemma* which is stated by Saha and Mitra [SM06] to prove the correctness of *CASE 3* of the inter-edge-add algorithm (see

Algorithm 2, Line 28). However, before we review the unaffect lemma, which we refer to as Lemma 6, we give a detailed description of what the algorithm does in *CASE 3* according to the formulation given by Saha and Mitra. We will further point out the inconsistencies appearing in the unaffect lemma and finally disprove the correctness of *CASE 3*.

The first two cases of the inter-edge-add algorithm are discussed in Section 2.2 and Subsection 2.3.1. The last case, namely *CASE 3*, serves as default case if none of the previous cases occurs. As before, $\zeta(G)$ denotes the input clustering in the algorithm complying with clustering quality. The input graph $G = (V, E, c())$ is enlarged by adding an edge $e_\oplus = \{b, d\}$ between two clusters C_b and C_d in $\zeta(G)$. The modified graph is called $G_\oplus = (V, E \cup \{e_\oplus\}, c_\oplus())$. Again the inter-edge-add algorithm in *CASE 3* is thought to return a clustering $\zeta(G_\oplus)$ of the modified graph $G_\oplus$ which still respects the clustering quality.

To this end *CASE 3* adds an artificial sink t to $G_\oplus$ as described in the cut-clustering method of Flake et al. [FTT04] and then contracts all vertices not included in the clusters C_b and C_d of graph G (see Line 14 to Line 16). The resulting graph is denoted by G'_α. Note, that this graph does not result from $G_\oplus$ by just adding a sink t as in the cut-clustering method. In this context the construction of G'_α requires an additional contraction. So graph G'_α consists of the vertices included in $C_b \cup C_d$ and a node ω resulting from this contraction. Now a min-cut tree $T(G'_\alpha)$ is calculated for G'_α (see Line 17). After removing the artificial sink t from $T(G'_\alpha)$ the tree decomposes into several connected components (see Line 18). At this point the formulation of *CASE 3* given by Saha and Mitra [SM06] mentions new components $C_1, \ldots, C_z$ consisting of vertices included in $C_b \cup C_d$ and finally constructs an output clustering $\zeta(G_\oplus)$ which assumes that the set $C_b \cup C_d$ decomposes in these components while the node ω forms a singleton after the removal of sink t from $T(G'_\alpha)$. However, this assumption can be disproven, as we will see later in this section. So returning the new clustering $\zeta(G_\oplus)$ as defined in Line 19 is not possible in general.

As the correctness of *CASE 3* can be disproven, there must exist some inconsistencies in the proof given by Saha and Mitra [SM06] for this case. Saha and Mitra state the following lemma to prove the correctness of *CASE 3*.

Lemma: Unaffect lemma

Lemma 6 *(Unaffect lemma) Let C_b and C_d be two clusters in a clustering $\zeta(G)$ resulting from the cut-clustering method of Flake et al. [FTT04]. If there are some insertions and deletions of edges across and within the clusters C_b and C_d of the dynamic graph G, then all clusters in $\zeta(G) \setminus \{C_b, C_d\}$ remain unaffected.*

The inconsistency related to the unaffect lemma concerns the meaning of the expression “remain unaffected” in the context of clusters. This expression is not defined in [SM06]. We can only guess by the formulation of *CASE 3* in Algorithm 2 what the authors deduce from Lemma 6, as they intend to prove the correctness of *CASE 3* with the aid of this lemma. The given proof of the unaffect lemma by Saha and Mitra is meaningless, as it omits a definition of this expression, too. We will not consider this proof any further. However, we will see in Chaper 6, Subsection 6.3.3, that, nevertheless, a reasonable interpretation of the unaffect lemma can be given. Of course this interpretation does not allow the deduction of the approach described in *CASE 3* (which is provably wrong), but of a slightly different approach which is correct, respects the implicit invariant given in Fact 1, and finally reduces the effort of the clustering calculation.

Disproving the correctness of *CASE 3*

We disprove the correctness of *CASE 3* formulated in the inter-edge-add algorithm (see Algorithm 2) with the aid of the following counter-example. Figure 2.8a illustrates a weighted graph G with a parametric cost function and four clusters resulting from the cut-clustering method of Flake et al. [FTT04]. For all graphs in Figure 2.8 the parametric weights are assigned to the edges. A min-cut tree $T(G_\alpha)$ inducing the illustrated clustering $\zeta(G)$ is shown in Figure 2.8b. The dashed line in Figure 2.8a indicates the edge $e_\oplus = \{2, 12\}$, which will be added by the algorithm, but does not belong to graph G yet.

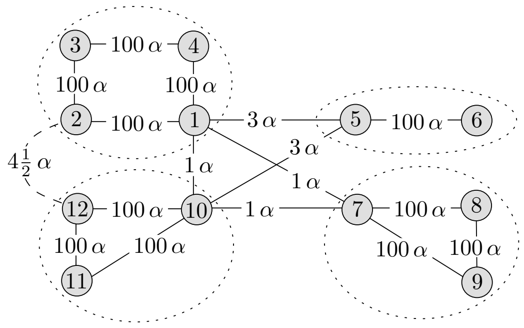
Now we apply Algorithm 2 to graph G. Thereby the new edge $e_\oplus = \{2, 12\}$ with weight $4.5\,\alpha$ is inserted between the clusters $\{1, 2, 3, 4\}$ and $\{10, 11, 12\}$ constructing the modified graph $G_\oplus$. Then the three cases are checked: *CASE 1* is skipped, as $c_\oplus(\{1, 2, 3, 4\}, V \setminus \{1, 2, 3, 4\}) = 8.5\,\alpha > 8\,\alpha = \alpha\,|V \setminus \{1, 2, 3, 4\}|$ holds. *CASE 2* also does not occur due to $c(\{1, 2, 3, 4\}, \{10, 11, 12\}) = 1\,\alpha < 6\,\alpha = \alpha\,|V|/2$. Therefore, the algorithm ends up in *CASE 3*. This case adds an artificial sink t to graph $G_\oplus$ and contracts the vertices of the clusters $\{5, 6\}$ and $\{7, 8, 9\}$ to a node called ω. The resulting graph G'_α is shown in Figure 2.8c. Finally a min-cut tree of G'_α is calculated. The tree $T(G'_\alpha)$ given by Figure 2.8d constitutes a valid min-cut tree of graph G'_α as it might result from this operation. However, in this tree the node ω obviously does not form a singleton after removing the sink t, and the set $\{1, 2, 3, 4\} \cup \{10, 11, 12\}$ does not decompose in disjoint components. Hence, the inter-edge-add algorithm would throw an exception in Line 19.

CASE 3 neither respects invariant nor maintains quality

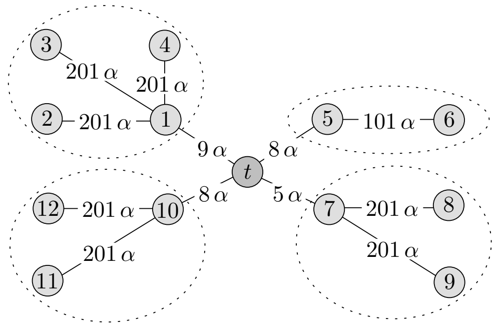
If we modified the algorithm such that it returns the clustering $\bar{\zeta}(G_\oplus)$ induced by the newly calculated min-cut tree $T(G'_\alpha)$ instead of throwing an exception (compare to Figure 2.8d), the returned clustering would not even guarantee the clustering quality. In our example the clustering $\bar{\zeta}(G_\oplus)$ consists of a single cluster containing all vertices of $G_\oplus$. In this cluster for example the cut $(\{7, 8, 9\}, V \setminus \{7, 8, 9\})$ violates the intra-cluster quality, as $c_\oplus(\{7, 8, 9\}, V \setminus \{7, 8, 9\}) = 2\,\alpha < 3\,\alpha = \alpha\,\min\{|\{7, 8, 9\}|, |V \setminus \{7, 8, 9\}|\}$ holds. This violation is rooted in the construction of the graph G'_α in *CASE 3*. At first sight the approach in *CASE 3* seems very similar to the cut-clustering method (see Algorithm 1) indeed, as in *CASE 3* the graph $G_\oplus$ is also enlarged by an artificial sink t and a min-cut tree in calculated. Therefore, one might expect the clustering quality being preserved. However, a closer look shows that in *CASE 3* the sink t in graph G'_α is no more connected to the remaining vertices with the same weight α. This is caused by contracting several vertices after the sink t was inserted. In our example the edge $\{\omega, t\}$ in graph G'_α has weight $5\,\alpha$. The graph G'_α constructed in *CASE 3* hence does not provide the properties required by the cut-clustering method before calculating the min-cut tree.

2.4 Summary

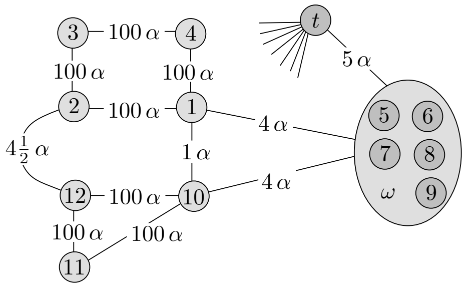
In this chapter we stated an invariant (see Fact 1 in Section 2.2) that Saha and Mitra [SM06] implicitly assume for their updating algorithms, but miss to prove. Therefore, we studied whether the three cases distinguished by the inter-edge-add algorithm of Saha and Mitra (see Algorithm 2) do respect this invariant. In doing so the example given in Section 2.2 showed that *CASE 1* does not meet the invariant. By contrast, on condition that the input clustering already provides the necessary properties, in Subsection 2.3.1 *CASE 2* (see Line 9) turned out to maintain those clustering properties defining the invariant. At the same time we showed that the occurrence of *CASE 2* is very unlikely. Furthermore, we proved the invariant to


3
4
2
1
5
6
12
10
7
8
11
9
100 α
3 α
1 α
4½ α

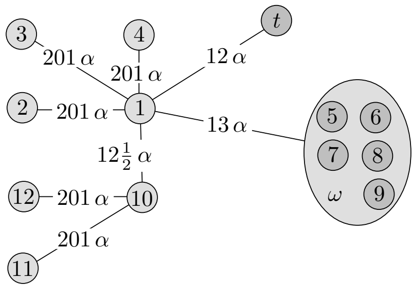
(a) Graph G with clustering $\zeta(G)$ resulting from the cut-clustering method


3
4
2
1
t
5
6
12
10
7
8
11
9
201 α
101 α
9 α
8 α
5 α

(b) Min-cut tree $T(G_\alpha)$ inducing the clustering $\zeta(G)$ shown above


t
5 α
3
4
2
1
12
10
11
100 α
4½ α
1 α
4 α
5
6
7
8
ω
9

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


3
4
t
2
1
12
10
11
201 α
12 α
13 α
12½ α
5
6
7
8
ω
9

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

Figure 2.8: Counter-example for the correctness of *CASE 3*.

be sufficient for the correctness of *CASE 2*, but it is still open whether it also constitutes a necessary condition. Nevertheless, we know at least one example of an input clustering that does not provide the properties required by the invariant and consequently yields an invalid result by applying *CASE 2*. The default case in Section 2.3.2, namely *CASE 3* (see Line 28), not even maintains the clustering quality, although, the approach given in *CASE 3* can be modified such that it yields a correct algorithm which can be proven to meet the invariant.

Chapter 3

Basics Regarding Cuts and Min-Cut Trees

In Chapter 2 we showed that the approach of Saha and Mitra [SM06] for dynamically updating a given clustering with a guaranteed clustering quality is not feasible. Nevertheless, the idea of using the cut-clustering method of Flake et al. [FTT04] (see Algorithm 1, Section 2.1), which guarantees a clustering quality depending on a parameter α, seems to be reasonable for a dynamic extension. So, as the cut-clustering method in based on the calculation of a min-cut tree, in Chapter 4 we will at first explore, independently from any clusterings, how a min-cut tree $T(G)$ gets affected by a dynamic modification of the considered graph G. To this end it is not necessary to distinguish dynamic inter- and intra-cluster modifications. In Chapter 4 we will just analyze the four elementary modifications of adding and deleting an edge and inserting and removing a vertex.

However, to get familiar with the min-cut tree as the mainly considered structure in the following, this chapter afore states some basic lemmas about the impact of dynamic modifications on cuts and describes the method of Gomory and Hu [GH61] to construct a min-cut tree.

3.1 Some Basic Lemmas

This section defines canonically induced cuts (see Subsection 3.1.1) and gives a collection of lemmas and corollaries about such cuts in differently modified graphs (see Subsection 3.1.2 and 3.1.3). The modifications considered in this section are the addition and deletion of an edge.

Note, that the addition and deletion of an edge in an undirected, positively weighted graph $G = (V, E, c())$ can be regarded as a special case of increasing and decreasing weights of edges in the related undirected, non-negatively weighted, complete graph $G_c = (V, E_c, c_c())$, with $E_c = E \cup \{\{u, v\} \subseteq V | \{u, v\} \notin E\}$ and $c_c(\{u, v\}) = c(\{u, v\})$ if $\{u, v\} \in E$, and $c_c(\{u, v\}) = 0$ otherwise. Adding a new positively weighted edge $\{b, d\}$ to E then corresponds to increasing the weight of edge $\{b, d\}$ in E_c from zero to the new weight of $\{b, d\}$ in G. Analogously the deletion of an edge $\{b, d\}$ from E corresponds to the reduction of the weight of edge $\{b, d\}$ in E_c to zero. Therefore, it is reasonable to call the vertices b and d which are incident with an added or deleted edge *modified vertices*.

Definition: Modified vertices

In the remainder of this work we mostly talk about the addition and deletion of edges, instead of increasing and decreasing weights. Hoever, note that all assertions made in this context will also be feasible if we regard the addition and deletion of edges as a special case of increasing and decreasing weights. If nothing else is mentioned we consider undirected, positively weighted, connected graphs and nontrivial cuts, i.e, cuts with positive weights.

3.1.1 Canonically Induced Cuts

In this subsection we define and explore canonically induced cuts in general. Subsection 3.1.2 and 3.1.3 then will state some assertions about the impact of dynamic modifications on the weight of canonically induced cuts and the maintenance of minimality.

Definition: Canonically induced cuts

Definition 6 *Let* $G_1 = (V_1, E_1, c_1())$ *and* $G_2 = (V_2, E_2, c_2())$ *denote two undirected, weighted graphs. Let* $\Phi : V_1 \longrightarrow V_2$ *denote a surjective map. Further let* $(U_1, V_1 \setminus U_1)$ *denote a cut in* G_1 *and* $(U_2, V_2 \setminus U_2)$ *a cut in* G_2. *Cut* $(U_1, V_1 \setminus U_1)$ canonically induces *cut* $(U_2, V_2 \setminus U_2)$ *(under map* Φ*) if it holds that*

$$\Phi^{-1}(W) \subseteq U_1 \quad \forall\, W \subseteq U_2 \quad \textit{and}$$
$$\Phi^{-1}(W) \subseteq (V_1 \setminus U_1) \quad \forall\, W \subseteq (V_2 \setminus U_2).$$

Vice versa, cut $(U_1, V_1 \setminus U_1)$ *is said to be* re-induced *by cut* $(U_2, V_2 \setminus U_2)$ *if it induces cut* $(U_2, V_2 \setminus U_2)$.

The only maps considered in this work in the context of canonically induced cuts are the identity and the vertex contraction defined in the Preliminaries. The following lemma considers canonically induced cuts under vertex contraction. Assertion and proof of this lemma are quite intuitive.

Lemma: Canonically induced cuts under vertex contraction

Lemma 7 *Let* $G = (V, E, c())$ *denote an undirected, weighted graph. The graph formed by contracting* $W \subseteq V$ *is called* $G_\circ = (V_\circ, E_\circ, c_\circ())$. *Let* $\omega \in V_\circ$ *denote the contraction of* W.

Then each cut $(U, V \setminus U)$ *in* G *with* $W \subseteq (V \setminus U)$ *canonically induces a cut* $(U, V_\circ \setminus U)$ *in* $G_\circ$, *while it holds that* $c_\circ(U, V_\circ \setminus U) = c(U, V \setminus U)$.

Vice versa, each cut $(U, V_\circ \setminus U)$ *in* $G_\circ$ *with* $\omega \in (V_\circ \setminus U)$ *is canonically induced by (or re-induces) a cut* $(U, V \setminus U)$ *in* G, *while it holds that* $c(U, V \setminus U) = c_\circ(U, V_\circ \setminus U)$.

Proof. Let $\Phi : V \longrightarrow V_\circ$ denote the surjective map defined by the vertex contraction, this is, $\Phi(u) = u, \forall u \in (V \setminus W)$, and $\Phi(v) = \omega, \forall v \in W$. Now consider an arbitrary cut $(U, V \setminus U)$ in G with $W \subseteq (V \setminus U)$. So, as the contracted set W is supposed to be a subset of $(V \setminus U)$, i.e., the cut $(U, V \setminus U)$ in G does not separate any vertices in W, the cut $(U, V_\circ \setminus U)$ is well defined in $G_\circ$ and it holds that $\Phi^{-1}(D) \subseteq U$, $\forall\, D \subseteq U$, as $\Phi|_U$ is the identity. It holds further that $\Phi^{-1}(D) \subseteq (V \setminus U)$, $\forall\, D \subseteq (V_\circ \setminus U)$, because otherwise there would exist an $x \in (V_\circ \setminus U)$ with $\Phi^{-1}(x) \in U$ which contradicts the assumption that $\Phi|_U$ is the identity.

Vice versa, considering an arbitrary cut $(U, V_\circ \setminus U)$ with $\omega \in (V_\circ \setminus U)$ the cut $(U, V \setminus U)$ is well defined in G and does not separate any vertices in W. Therefore $(U, V \setminus U)$ canonically induces $(U, V_\circ \setminus U)$ (as just shown above). So we just have to prove that it holds that $c_\circ(U, V_\circ \setminus U) = c(U, V \setminus U)$:

$$\begin{aligned} c_\circ(U, V_\circ \setminus U) &= \sum_{\substack{\{u,v\} \in E_\circ \\ u \in U, v \in (V_\circ \setminus U)}} c_\circ(\{u, v\}) \\ &= \sum_{\substack{\{u,v\} \in E_\circ \\ u \in U, v \in V_\circ \setminus (U \cup \{\omega\})}} c_\circ(\{u, v\}) + \sum_{\substack{\{u,\omega\} \in E_\circ \\ u \in U}} c_\circ(\{u, \omega\}) \\ &= \sum_{\substack{\{u,v\} \in E \\ u \in U, v \in V \setminus (U \cup W)}} c(\{u, v\}) + \sum_{\substack{\{u,v\} \in E \\ u \in U, v \in W}} c(\{u, v\}) \\ &= \sum_{\substack{\{u,v\} \in E \\ u \in U, v \in (V \setminus U)}} c(\{u, v\}) \quad = \quad c(U, V \setminus U) \end{aligned}$$

❐

Remark: Canonically induced cuts under identity

Remark 1 In Lemma 7 the map Φ is defined by a vertex contraction. As a vertex contraction causes a reduction of the set V of graph G, the map $\Phi : V \longrightarrow V_\circ$ is only surjective, never injective. So under a map Φ defined by a vertex contraction each cut in $G_\circ$ is canonically induced by a unique cut in G, but there exist cuts in G (namely all cuts that separate some vertices in W) which do not canonically induce a cut in $G_\circ$. If now we consider the identity, instead of map Φ defined by the vertex contraction, the set V of graph G is not changed, and hence the assertion of Lemma 7 becomes trivial. Under the identity (as a bijective map Φ between sets of vertices of two graphs G_1 and G_2) each cut in G_1 canonically induces a unique cut in G_2 and each cut in G_2 is canonically induced by a unique cut in G_1.

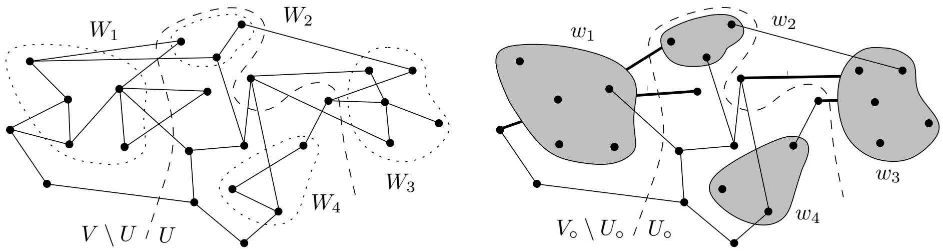
We consider such cuts, which canonically induce and re-induce one another under vertex contraction or identity, to be equivalent. Hence, we address such cuts as only one cut in different graphs (with either a vertex contraction or the identity implicitly assumed as map Φ between these graphs). In contrast to Lemma 7, we will see in Section 3.1.2 that cuts which are equivalent in this terms, however, may have different weights if we assume graph G to be modified by the addition or deletion of an edge.

However, at first we extend the assertion of Lemma 7 concerning several contractions. The next corollary considers an undirected, weighted graph $G = (V, E, c())$ and mutually disjoint contraction sets $W_1, \dots, W_k \subseteq V$ in G. The graph formed by contracting each subset W_j $(j = 1, \dots, k)$ in G is called $G_\circ = (V_\circ, E_\circ, c_\circ())$. Let $\omega_j \in V_\circ$ denote the contraction of W_j. Then the corollaries assertion follows by induction on the sets $W_1, \dots, W_k$ and Lemma 7. Figure 3.1 illustrates the corollaries assertion.

Corollary: Canonically induced cuts under several vertex contractions

Corollary 1 *Each cut $(U, V \setminus U)$ in G which does not separate any vertices $\{u, v\} \subseteq W_j$ $(j = 1, \dots, k)$ canonically induces a cut $(U_\circ, V_\circ \setminus U_\circ)$ in $G_\circ$ of the same weight, with $U_\circ = \Phi(U)$ regarding the map Φ defined by the vertex contractions.*

Vice versa, each cut $(U_\circ, V_\circ \setminus U_\circ)$ in $G_\circ$ is canonically induced by (or re-induces) a cut $(U, V \setminus U)$ in G of the same weight, with $U = \Phi^{-1}(U_\circ)$ regarding the map Φ defined by the vertex contractions. The re-induced cut $(U, V \setminus U)$ does not separate any vertices $\{u, v\} \subseteq W_j$ $(j = 1, \dots, k)$.



(a) Graph G with cut $(U, V \setminus U)$ **(b)** Graph $G_\circ$ with cut $(U_\circ, V_\circ \setminus U_\circ)$

Figure 3.1: Canonically induced cuts under several vertex contractions.

3.1.2 Canonically Induced Cuts in Modified Graphs

This section states some assertions about weights of canonically induced (or equivalent) cuts changing in modified graphs. The considered modifications in this subsection are the addition and deletion of an edge. The following corollaries result from the coherences explored in the previous Section and the definition of the weight of a cut given in the preliminaries.

In the first corollary we consider the same situation as in Corollary 1, but additionally assume the graph $G_\circ$ to be enlarged (reduced) by the addition (deletion) of an edge $\{b, d\}$, with weight $\Delta > 0$. Note, that the vertices b and d are regarded as vertices in V, which may also be contracted. Then, by Corollary 1, a cut $(U, V \setminus U)$ in G which does not separate any vertices $\{u, v\} \subseteq W_j$ $(j = 1, \ldots, k)$ canonically induces a cut $(U_\circ, V_\circ \setminus U_\circ)$ in $G_\circ$, with $U_\circ = \Phi(U)$.

Corollary: Canonically induced cuts under several vertex contractions

Corollary 2 *The induced cut* $(U_\circ, V_\circ \setminus U_\circ)$ *in* $G_\circ$ *is of the same weight if* $(U, V \setminus U)$ *does not separate* b *and* d*, and has weight*

$$c_\circ(U_\circ, V_\circ \setminus U_\circ) = c(U, V \setminus U) \pm \Delta$$

otherwise. Vice versa, each cut $(U_\circ, V_\circ \setminus U_\circ)$ *in* $G_\circ$ *is canonically induced by (or re-induces) a cut* $(U, V \setminus U)$ *in* G*, with* $U = \Phi^{-1}(U_\circ)$*. The re-induced cut* $(U, V \setminus U)$ *is of the same weight if* $(U_\circ, V_\circ \setminus U_\circ)$ *does not separate* b *and* d*, and otherwise has weight*

$$c(U, V \setminus U) = c_\circ(U_\circ, V_\circ \setminus U_\circ) \mp \Delta.$$

We can state the same assertion for canonically induced cuts under the identity, instead of map Φ defined by the vertex contraction. Again $G = (V, E, c())$ denotes an undirected, weighted graph and $G_{\oplus(\ominus)} = (V, E_{\oplus(\ominus)}, c_{\oplus(\ominus)}())$ the graph enlarged (reduced) by the addition (deletion) of an edge $e_{\oplus(\ominus)} = \{b, d\}$, with weight $\Delta > 0$. Then, by Remark 1, a cut $(U, V \setminus U)$ in G canonically induces a cut $(U, V \setminus U)$ in graph $G_{\oplus(\ominus)}$.

Corollary: Canonically induced cuts under identity

Corollary 3 *The induced cut* $(U, V \setminus U)$ *in* $G_{\oplus(\ominus)}$ *is of the same weight if* $(U, V \setminus U)$ *in* G *does not separate* b *and* d*, and has weight*

$$c_{\oplus(\ominus)}(U, V \setminus U) = c(U, V \setminus U) \pm \Delta$$

otherwise. Vice versa, each cut $(U, V \setminus U)$ *in* $G_{\oplus(\ominus)}$ *is canonically induced by (or re-induces) a cut* $(U, V \setminus U)$ *in* G*. The re-induced cut* $(U, V \setminus U)$ *in* G *is of the same weight if* $(U, V \setminus U)$ *in* $G_{\oplus(\ominus)}$ *does not separate* b *and* d*, and otherwise has weight*

$$c(U, V \setminus U) = c_{\oplus(\ominus)}(U, V \setminus U) \mp \Delta.$$

Note, that in Corollary 3 the modification that points from $G_\ominus$ to G constitutes an edge addition, while the modification that points from $G_\oplus$ to G corresponds to an edge deletion. Due to this observation Lemma 8 and Lemma 9 in the following subsection consider the modifications in only one direction.

In this section we have seen that cuts which are equivalent in the sense that they canonically induce and re-induce one another under vertex contraction or identity, however, may have different weights in a graph G modified by the addition or deletion of an edge.

3.1.3 Canonically Induced Min-Cuts in Modified Graphs

In this section we additionally suppose the considered cuts to be minimum u-v-cuts for a pair of vertices $\{u, v\}$. The following lemmas then explore whether the equivalent cuts in modified graphs are also minimum u-v-cuts concerning the same pair $\{u, v\}$. The considered modifications in this subsection are again the addition and deletion of an edge. In the context of edge deletion we additionally consider the special case when the deletion yields a disconnected graph, i.e., when the deleted edge was a bridge in G.

The lemmas of this section are also quite intuitive but we will need them in later proofs. That is why we sum them here. The first lemma considers the identity as map Φ and an undirected, weighted graph $G = (V, E, c())$ modified by the addition of an edge $e_\oplus = \{b, d\}$, with weight $\Delta > 0$. The resulting graph we denote by $G_\oplus = (V, E \cup \{e_\oplus\}, c_\oplus())$. Let $u, v \in V$ denote two arbitrary vertices of G. Then, by Remark 1, a minimum u-v-cut $\theta_{\min} := (U, V \setminus U)$ in G canonically induces a u-v-cut $(U, V \setminus U)$ in $G_\oplus$.

Lemma: Canonically induced min-cuts under identity and edge addition

Lemma 8 *If the cut θ_{min} in G does* not *separate b and d, the induced cut $(U, V \setminus U)$*

(A1) *is of the same weight (by Corollary 3) and*

(A2) *is also a minimum u-v-cut in $G_\oplus$ (not proven yet).*

Otherwise, if θ_{min} in G separates b and d, the induced cut $(U, V \setminus U)$

(A3) *has weight $c_\oplus(U, V \setminus U) = c(\theta_{min}) + \Delta$ (by Corollary 3), and*

(A4) *may be a minimum u-v-cut in $G_\oplus$ on several conditions (not proven yet):*

- *It is* no *minimum u-v-cut in $G_\oplus$ if there exists a u-v-cut in G that does not separate b and d and is cheaper than $c(\theta_{min}) + \Delta$.*
- *It is a minimum u-v-cut in $G_\oplus$ if all u-v-cuts in G that do not separate b and d are at least of weight $c(\theta_{min}) + \Delta$.*

Proof. By Corollary 3, cuts that separate b and d in G become more expensive in $G_\oplus$ by addition of the weight $\Delta > 0$, and cuts that do not separate b and d in G are of the same weight in $G_\oplus$. It follows that the weight $lb := c(\theta_{\min})$ of a minimum u-v-cut $\theta_{\min}$ in G constitutes a lower bound for the weight of a minimum u-v-cut in $G_\oplus$. Proof of (A2): By (A1), it follows that each minimum u-v-cut in G that does not separate b and d meets this lower bound and, therefore, is also a minimum u-v-cut in $G_\oplus$. Proof of (A4-1): Furthermore, if there exists a u-v-cut that is cheaper than $lb + \Delta$ in G and does not separate b and d (i.e., is of the same weight in $G_\oplus$ by Corollary 3), then a minimum u-v-cut in G that separates b and

d is never a minimum u-v-cut in $G_\oplus$, as it has weight $lb + \Delta$ in $G_\oplus$ by (A3). Proof of (A4-2): If otherwise all u-v-cuts that do not separate b and d (i.e., that do not become more expensive in $G_\oplus$) are at least of weight $lb + \Delta$ in G, then a minimum u-v-cut in G that separates b and d (and therefore, has weight $lb + \Delta$ in $G_\oplus$ by (A3)) is also a minimum u-v-cut in $G_\oplus$. □

The next lemma again considers the identity as map Φ, but a graph $G = (V, E, c())$ modified by the deletion of an edge $e_\ominus = \{b, d\}$ which is no bridge in G and has weight $\{b, d\} > 0$. The resulting graph we call $G_\ominus = (V, E \setminus \{e_\ominus\}, c_\ominus())$. Let $u, v \in V$ denote two arbitrary vertices of G. Then, by Remark 1, a minimum u-v-cut $\theta_{\min} := (U, V \setminus U)$ in G canonically induces a u-v-cut $(U, V \setminus U)$ in $G_\ominus$.

Lemma: Canonically induced min-cuts under identity and edge deletion

Lemma 9 *If the cut θ_{min} in G does not separate b and d, the induced cut $(U, V \setminus U)$*

(A1) *is of the same weight (by Corollary 3) and*

(A2) *may be a minimum u-v-cut in $G_\ominus$ on several conditions:*

- *It is* no *minimum u-v-cut in $G_\ominus$ if there exists a u-v-cut in G that separates b and d and is cheaper than $c(\theta_{min}) + \Delta$ (not proven yet).*
- *It is a minimum u-v-cut in $G_\ominus$ if all u-v-cuts in G that separate b and d are at least of weight $c(\theta_{min}) + \Delta$ (not proven yet).*

Otherwise, if θ_{min} in G separates b and d, the induced cut $(U, V \setminus U)$

(A3) *has weight $c_\ominus(U, V \setminus U) = c(\theta_{min}) - \Delta$ (by Corollary 3), and*

(A4) *is also a minimum u-v-cut in $G_\ominus$ (not proven yet).*

Proof. By Corollary 3, cuts that separate b and d in G become cheaper in $G_\ominus$ by subtraction of the weight $\Delta > 0$, and cuts that do not separate b and d in G are of the same weight in $G_\ominus$. It follows that the weight $lb := c(\theta_{\min}) - \Delta$, with $\theta_{\min}$ a minimum u-v-cut in G, constitutes a lower bound for the weight of a minimum u-v-cut in $G_\ominus$. Proof of (A4): By (A3) it follows that each minimum u-v-cut in G that separates b and d meets this lower bound and, therefore, is also a minimum u-v-cut in $G_\ominus$. Proof of (A2-1): Furthermore, if there exists a u-v-cut that is cheaper than $c(\theta_{\min}) + \Delta$ in G and separates b and d (i.e., whose weight is reduced by Δ in $G_\ominus$ by Corollary 3), then a minimum u-v-cut that does not separate b and d in G is never a minimum u-v-cut in $G_\ominus$, as it has weight $c(\theta_{\min})$ in $G_\ominus$ by (A1). Proof of (A2-2): If otherwise all u-v-cuts that separate b and d (i.e., that become cheaper in $G_\ominus$) are at least of weight $c(\theta_{\min}) + \Delta$ in G, then a minimum u-v-cut that does not separate b and d in G (and therefore, has weight $c(\theta_{\min})$ in $G_\ominus$ by (A1)) is also a minimum u-v-cut in $G_\ominus$. □

Lemma: Canonically induced min-cuts under identity and bridge deletion

Now we additionally consider the case when the edge $e_\ominus = \{b, d\}$, with weight $\Delta > 0$, deleted from graph G was a bridge in G, i.e., when the deletion yields a disconnected graph $G_\ominus$. The two connected components of $G_\ominus$ we call $G_b = (V_b, E_b, c_b())$ and $G_d = (V_d, E_d, c_d())$, with $V_b \cup V_d = V$ and $E_b \cup E_d = E \setminus \{e_\ominus\}$. Let $u, v \in V_j$ ($j \in \{b, d\}$) denote two arbitrary vertices of G_j. Then, by Remark 1 and Corollary 3, each minimum u-v-cut $\theta_{\min} := (U, V \setminus U)$ in G canonically induces a u-v-cut $(U_j, V_j \setminus U_j)$ in G_j, with $U_j = V_j \cap U$ and weight $c(\theta_{\min})$ ($-\Delta$ if $\theta_{\min}$ separates b and d).

Lemma 10 *The induced cut $(U_j, V_j \setminus U_j)$ in G_j is a minimum u-v-cut in G_j.*

Proof. An arbitrary minimum u-v-cut $\theta_{\min} := (U, V \setminus U) = (U_b \cup U_d, V \setminus (U_b \cup U_d))$ in G has weight $c(\theta_{\min}) = c_b(U_b, V_b \setminus U_b) + c_d(U_d, V_d \setminus U_d)$ ($+\Delta$ if $\theta_{\min}$ separates b and d), because there exist no edges (apart from $e_\ominus$) between V_b and V_d in G.

If there existed a u-v-cut $(U_c, V_j \setminus U_c)$ cheaper than $c_j(U_j, V_j \setminus U_j)$ in G_j, with $j \in (V_j \setminus U_c)$ $(i := \{b, d\} \setminus \{j\})$, then cut $\theta_c := (U_c \cup U_i, V \setminus (U_c \cup U_i))$ does not separate b and d and therefore, would have weight $c(\theta_c) = c_j(U_c, V_j \setminus U_c) + c_i(U_i, V_i \setminus U_i)$ in G. So cut θ_c would be cheaper than $\theta_{\min}$. As cut θ_c also separates u and v, this contradicts the assumption that $\theta_{\min}$ is a minimum u-v-cut in G. $\square$

The last lemma states an assertion similar to the assertions of Lemma 8 and 9 for canonically induced minimum u-v-cuts under vertex contraction. As the map Φ defined by a vertex contraction is not injective, the question whether equivalent cuts are also minimum u-v-cuts in a contracted graph $G_\circ$ already appears in the absence of additional modifications. So we consider minimum u-v-cuts in the graphs G and $G_\circ$, without any additions or deletions of edges (similar to the situation in Corollary 1).

Then, by Corollary 1, each minimum u-v-cut $\theta_{\min} := (U, V \setminus U)$ in G which does not separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$ canonically induces a cut $(U_\circ, V_\circ \setminus U_\circ)$ in $G_\circ$ of the same weight $c(\theta_{\min})$, with $U_\circ = \Phi(U)$. Vice versa, by Corollary 1, each minimum u-v-cut $\theta_{\min} := (U_\circ, V_\circ \setminus U_\circ)$ in $G_\circ$ (with $u, v \in V_\circ$) is canonically induced by (or re-induces) a cut $(U, V \setminus U)$ in G of the same weight $c_\circ(\theta_{\min})$, with $U = \Phi^{-1}(U_\circ)$.

Lemma: Canonically induced min-cuts under vertex contractions

Lemma 11 *The induced cut $(U_\circ, V_\circ \setminus U_\circ)$ is also a minimum u-v-cut in $G_\circ$. However, the re-induced cut $(U, V \setminus U)$ is not necessarily a minimum u-v-cut in G. It is a minimum u-v-cut in G if and only if all u-v-cuts in G that separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$ are at least of weight $c_\circ(\theta_{min})$.*

Proof. By Corollary 1 each cut that does not separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$ in G is well defined in $G_\circ$ with the same weight, and each cut in $G_\circ$ is well defined in G with the same weight and does not separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$. Proof of (A): Therefore, the existence of a u-v-cut in $G_\circ$ cheaper than the one canonically induced by a minimum u-v-cut $\theta_{\min}$ in G (which does not separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$) contradicts the assumption that $\theta_{\min}$ is a minimum u-v-cut in G. Proof of (B): Consider a minimum u-v-cut $\theta_{\min}$ in $G_\circ$. If $\theta_{\min}$ is also a minimum u-v-cut in G, it follows that there exists no u-v-cut in G cheaper than $c_\circ(\theta_{\min})$, in particular there exists no u-v-cut cheaper than $c_\circ(\theta_{\min})$ that separates any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$ in G. Vice versa, if all u-v-cuts in G that separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$ are at least of weight $c_\circ(\theta_{\min})$, there exists no other u-v-cut in G cheaper than $c_\circ(\theta_{\min})$, as all remaining u-v-cuts in G (namely those which do not separate any vertices $\{g, h\} \subseteq W_j$ $(j = 1, \ldots, k)$) are u-v-cuts in $G_\circ$ with the same weight. Therefore, those remaining u-v-cuts are at least of weight $c_\circ(\theta_{\min})$ already. $\square$

3.2 The Gomory-Hu Method

The Gomory-Hu method, which was first introduced by Gomory and Hu [GH61], describes an algorithm to construct a min-cut tree $T(G)$ of an undirected, positively weighted, connected graph $G = (V, E, c())$ by calculating $n - 1$ pairwise noncrossing minimum u-v-cuts. Note, that the Gomory-Hu method explicitly requires the calculated minimum u-v-cuts to be noncrossing. As a consequence the method uses operations of vertex contraction to meet the noncrossing condition. Due to these vertex contractions it becomes fairly involved and nontrivial to implement. Gusfield [Gus90] shows that it is possible to modify the Gomory-Hu method such that vertex contractions are not necessary anymore, as also crossing cuts can be used to construct a min-cut tree. We will explain and use this method later in Chapter 4, Section 4.2. The discussion of the Gomory-Hu method in this section aims at a better understanding of the structure of min-cut trees. For a more detailed analysis of this method see [GH61].

3.2.1 Gomory-Hu Algorithm for Min-Cut Trees

The following description of the Gomory-Hu method is based for the most part on the one given in [Gus90]. Algorithm 3 additionally gives a more compact illustration. The Gomory-Hu method takes an undirected, positively weighted, connected graph $G = (V, E, c())$ as input, with $|V| = n$, and returns a min-cut tree $T(G) = (V, E_T, c_T())$ of G. Note, that the cuts calculated in this method are nontrivial, as G is supposed to be connected, i.e., the cuts are positively weighted.

The Gomory-Hu algorithm is an iterative method, which adjusts step by step an intermediate min-cut tree $T_\star(G) = (V_\star, E_\star, c_\star())$ to get the min-cut tree $T(G) = (V, E_T, c_T())$ in the end. Figure 3.2 shows an exemplary iteration.

Start: Define $V_\star := \{V\}$ to be the single node containing all vertices of G. Then iterate the following step until each node $S \in V_\star$ contains only one vertex g of G. Identify each singleton S with the vertex $g \in V$ it consists of and call the resulting tree $T(G) = (V, E_T, c_T())$.

Definition:
Split node
and
step pair

Iteration: Pick a node $S \in V_\star$ (called *split node*) containing more than one vertex of G, and pick two vertices $u, v \in S$, called *step pair*. Note, that a single vertex may appear in different step pairs. Find all connected components of $T_\star(G)$ after removing the current node S. These components correspond to all subtrees of S in $T_\star(G)$. So let $N(j) \subseteq V$ denote the set of vertices contained in the j-th subtree N_j of S. Let further $\bar{S}_j \in V_\star$ denote the node in the j-th subtree N_j which is effectively connected to S in $T_\star(G)$. Now successively contract the vertices in each set $N(j)$ in G (call the contraction η_j), and let $G(S)$ denote the resulting graph. Note that the vertices in S are not contracted, i.e., the graph $G_\circ(S)$ has vertices $V(S) = \{\eta_1, \ldots, \eta_z\} \cup S$.

Definition:
Split cut

Phase 1: Compute a minimum u-v-cut $(U(S), V(S) \setminus U(S))$ in $G(S)$, with $u \in U(S)$, called *split cut*. The split cut has weight $c(U(S), V(S) \setminus U(S))$ in $G(S)$ as well as in G, by Corollary 1. Define $S_u := S \cap U(S)$ and $S_v := S \cap (V(S) \setminus U(S))$. Modify $T_\star(G)$ by replacing node S by S_u and S_v connected by an edge $e = \{S_u, S_v\}$ of weight $c_\star(e) = c(U(S), V(S) \setminus U(S))$ (This edge e is said to be created by the step pair $\{u, v\}$).

Node splitting
and subtree
reconnecting

Phase 2: Each edge $e_j = \{S, \bar{S}_j\} \in E_\star$ which connects the subtree N_j to S in $T_\star(G)$, is moved to be incident with S_u (instead of S) if the contraction η_j in $G_\circ(S)$ is a node in $U(S)$, and is moved to be incident with S_v (instead of S) if the contraction η_j in $G(S)$ is a node in $(V(S) \setminus U(S))$. We call such an edge $e_j = \{S, \bar{S}_j\}$, which connects the subtree N_j to S, the *connection edge* of N_j. The weights of all edges in $T_\star(G)$ remain unchanged in this phase, including those edges which were moved.

Definition:
Connection
edge

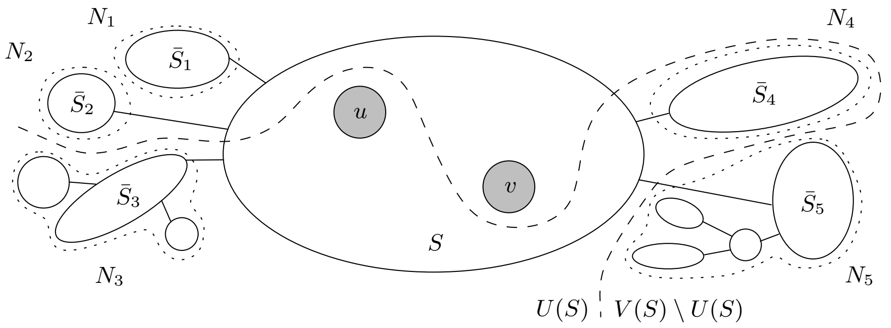
Figure 3.2 illustrates *Phase 1* and *Phase 2* of an exemplary iteration of the Gomory-Hu algorithm. Figure 3.2a shows an intermediate min-cut tree $T_\star(G)$ with eleven nodes (displayed as ovals). The current split node $S \in V_\star$ has five subtrees

Algorithm 3: GOMORY-HU

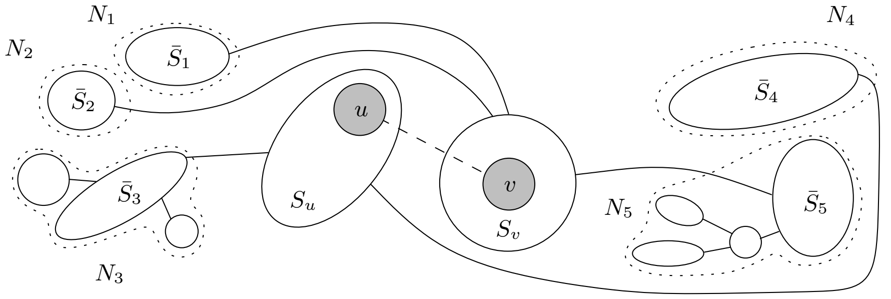
```
Input: Graph $G = (V, E, c())$
Output: Min-cut tree $T(G) = (V, E_T, c_T())$ of $G$
%---------- notation ----------
$T_\star(G) = (V_\star, E_\star, c_\star())$ intermediate min-cut tree
$S \in V_\star$ node in intermediate min-cut tree, split node
$N_j$ j-th subtree of node $S$ in intermediate min-cut tree
$N(j) \subseteq V$ set of vertices in j-th subtree of node $S$
$\bar{S}_j \in V_\star$ node in j-th subtree connected to node $S$ in intermediate min-cut tree
$G(S)$ result from the contraction of subtrees in $G$
$V(S)$ set of vertices of $G(S)$
$\eta_j \in V(S)$ contraction of j-th subtree of $S$ in $G$
%---------- algorithm ----------
Initialize $V_\star \leftarrow \{V\}$ and $E_\star \leftarrow \emptyset$ and $c_\star()$ accordingly
Initialize $E_T \leftarrow \emptyset$ and $c_T()$ accordingly
while $\exists S \in V_\star$ with $|S| > 1$ do
    $\{u, v\} \leftarrow$ arbitrary pair of different vertices in $S$
    Calculate set $\{N_1, \ldots, N_z\}$ of subtrees of $S$ in $T_\star(G)$
    Calculate $G(S)$ by contracting in $G$ the vertices of each subtree
    %---Phase 1
    $(U(S), V(S) \setminus U(S)) \leftarrow$ minimum $u$-$v$-cut in $G(S)$, $u \in U(S)$
    $\Delta \leftarrow$ weight of cut $(U(S), V(S) \setminus U(S))$
    $S_u \leftarrow S \cap U(S)$
    $S_v \leftarrow S \cap (V(S) \setminus U(S))$
    $V_\star \leftarrow (V_\star \setminus \{S\}) \cup \{S_u, S_v\}$
    $e \leftarrow \{S_u, S_v\}$
    $E_\star \leftarrow E_\star \cup \{e\}$
    Update cost function $c_\star()$ by $c_\star(e) \leftarrow \Delta$
    %---Phase 2
    forall edges $e_j = \{S, \bar{S}_j\} \in E_\star$ incident with $S$ in $T_\star(G)$ do
        if $\eta_j \in U(S)$ then
            $e_j \leftarrow \{S_u, \bar{S}_j\}$
        else
            $e_j \leftarrow \{S_v, \bar{S}_j\}$
forall edges $e = \{\{g\}, \{h\}\} \in E_\star$ do
    $\Delta \leftarrow c_\star(e)$
    $e \leftarrow \{g, h\}$
    $E_T \leftarrow E_T \cup \{e\}$
    Update cost function $c_T()$ by $c_T(e) \leftarrow \Delta$
return $T(G) = (V, E_T, c_T())$
```

Phase 1

$N_1, \ldots, N_5$ (bordered by dotted lines) in $T_\star(G)$. The nodes $\bar{S}_1, \ldots, \bar{S}_5$, which are effectively connected to S are accordingly labeled. The vertices of the contracted graph $G(S)$ hence add up to $V(S) = \{\eta_1, \ldots, \eta_5\} \cup S$. The vertices of the step pair $u, v \in S$ are displayed as light gray bullets. The split cut $(U(S), V(S) \setminus U(S))$ in $G(S)$ calculated in *Phase 1*, with $u \in U(S)$, is shown as a dashed line. In Figure 3.2b *Phase 1* is continued. The node S gets split by the split cut $(U(S), V(S) \setminus U(S))$ in $G(S)$. The newly defined nodes $S_u := S \cup U(S)$ and $S_v := S \cup (V(S) \setminus U(S))$ are accordingly labeled and S is replaced in the intermediate min-cut tree $T_\star(G)$ by S_u and S_v connected by a new edge.



(a) Intermediate min-cut tree $T_\star(G)$ with step pair $\{u, v\}$ and related split cut (dashed)



(b) Step pair $\{u, v\}$ splits S into S_u and S_v connected by a new edge (dashed) in $T_\star(G)$

Figure 3.2: Splitting node S by step pair $\{u, v\}$.

Phase 2

Furthermore, the connection edges $e_1, \ldots, e_5$ of the five subtrees $N_1, \ldots, N_5$ of S are reconnected as described in *Phase 2*. The contracted sets η_1, η_2 and η_5 (which correspond to the subtrees N_1, N_2 and N_5) are included in $U(S)$. So the connection edges $e_1 = \{S, \bar{S}_1\}$, $e_2 = \{S, \bar{S}_2\}$ and $e_5 = \{S, \bar{S}_5\}$ are reconnected to S_u instead of S. The contracted sets η_3 and η_4 (which correspond to the subtrees N_3 and N_4) are included in $(V(S) \setminus U(S))$. So the connection edges $e_3 = \{S, \bar{S}_3\}$ and $e_4 = \{S, \bar{S}_4\}$ are reconnected to S_v instead of S.

3.2.2 Definitions and Resulting Remarks

The following definitions, lemmas and remarks explore the structure of min-cut trees with the aid of the Gomory-Hu algorithm as constructional method.

Remark 2 A step pair $\{u, v\}$ is explicitly defined during an execution of the Gomory-Hu algorithm. As the final min-cut tree $T(G)$ of G is a tree that connects all vertices of G (by definition), $T(G)$ has exactly $n - 1$ edges. In each iteration the Gomory-Hu algorithm defines a new step pair and adds a new edge to the intermediate min-cut tree $T_\star(G)$ (see *Phase 1*). So an entire execution of the Gomory-Hu algorithm defines $n - 1$ step pairs and related split cuts.

Remark:
Number of
step pairs

Definition 7 *A specific execution of the Gomory-Hu algorithm is characterized by the considered graph G, a specific sequence F of $n - 1$ step pairs and a related sequence K of split cuts. So a specific* Gomory-Hu execution *GH is denoted by $GH = (G, F, K)$.*

Definition:
Gomory-Hu
execution

In addition to step pairs we define *cut pairs* as follows:

Definition 8 *A pair of vertices $\{u, v\} \subseteq V$ of graph G is called a* cut pair *regarding an edge e of an intermediate min-cut tree $T_\star(G)$ if the cut induced in G by the edge e is a minimum u-v-cut in G.*

Definition:
Cut pair

The following Lemma 12 we took from [Gus90] where it is referred to as Lemma 4. Originally this lemma is stated and proven by Gomory and Hu [GH61].

Lemma 12 considers an intermediate min-cut tree $T_\star(G)$ produced by the Gomory-Hu algorithm with $e_j = \{S, \bar{S}_j\}$ an edge in $E_\star$. If we assume S to be split next, the edge e_j constitutes the connection edge of the subtree N_j of S containing $\bar{S}_j$. Let $\{x, y\} \subseteq V$ be a cut pair regarding the edge e_j, with $x \in S$ and $y \in \bar{S}_j$. Let u and v be any vertices in S (serving as next step pair), and let $(U(S), V(S) \setminus U(S))$ denote a minimum u-v-cut in the contracted graph $G(S)$ (serving as next split cut). Let S_u and S_v denote the new nodes created from S by splitting (compare to Figure 3.3).

Lemma 12 *The step pair $\{u, v\} \subseteq V$ is a cut pair regarding the newly created edge $e = \{S_u, S_v\}$ in the intermediate min-cut tree $T_\star(G)$ after splitting the node S. Assume $x \in U(S)$ (the case when $x \in (V(S) \setminus U(S))$ is symmetric).*

If the connection edge e_j of subtree N_j of S containing $\bar{S}_j$ is reconnected to S_u after splitting S, i.e., if $\{\bar{S}_j, S_u\}$ is an edge in the intermediate min-cut tree $T_\star(G)$ after splitting S, then $\{x, y\}$ is still a cut pair regarding the reconnected edge $e_j = \{\bar{S}_j, S_u\}$.

Otherwise, if $\{\bar{S}_j, S_v\}$ is an edge in the intermediate min-cut tree $T_\star(G)$ after splitting S, then $\{v, y\}$ is a cut pair (additionally to $\{x, y\}$) regarding the reconnected edge $e_j = \{\bar{S}_j, S_v\}$.

Lemma:
Step pairs
remain
cut pairs

The following simpler version of Lemma 12 we also took from [Gus90]. It follows by induction on the iterations of the Gomory-Hu algorithm.

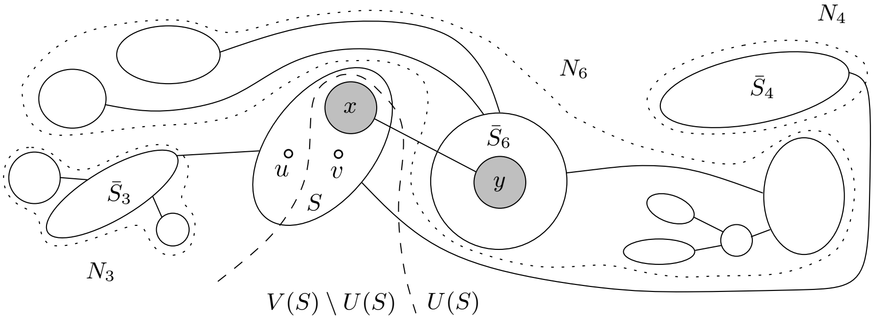
Corollary 4 *Consider an arbitrary intermediate min-cut tree $T_\star(G)$ during a Gomory-Hu execution. Let $e = \{S, \bar{S}\}$ denote an edge of the tree $T_\star(G)$. Then there exists at least one pair of vertices $\{u, v\} \subseteq V$ in G, with $u \in S$ and $v \in \bar{S}$, such that $\{u, v\}$ is a cut pair regarding the edge $e = \{S, \bar{S}\}$.*

Corollary:
Cut pairs
regarding
intermediate
tree edges

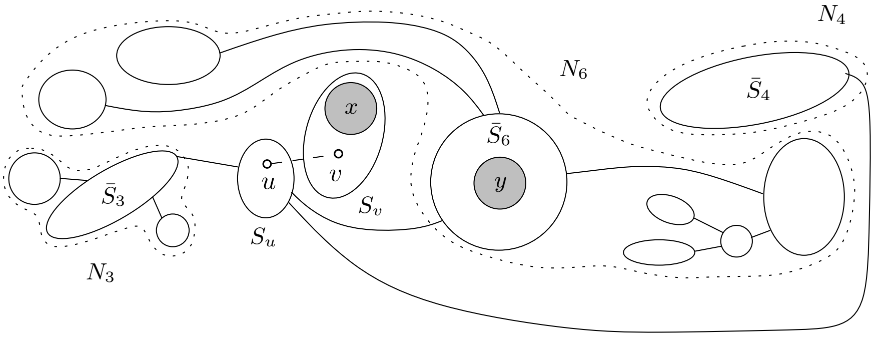
With Corollary 4 calling the intermediate tree $T_\star(G)$ in the Gomory-Hu algorithm ”‘min-cut tree”’ becomes reasonable, as in fact each edge $e = \{S, \bar{S}\}$ in $T_\star(G)$ induces a minimum S-$\bar{S}$-cut in G. Otherwise, a cheaper S-$\bar{S}$-cut in G would also constitute a cheaper separating cut for the vertices $\{u, v\}$ of the cut pair mentioned in Corollary 4. By induction on the iterations the correctness of the Gomory-Hu algorithm finally follows.

Step Pairs, Hidden Step Pairs and Cut Pairs

The last paragraph in this section is due to illustrate the difference between a step pair and a cut pair. To this end Figure 3.3 shows a situation as described in the context of Lemma 12. The situation in Figure 3.3a is actually the same as in Figure 3.2b, but with $\{u, v\}$ renamed as $\{x, y\}$ and the previous node S_u now regarded as the next split node S. Therefore, the new split node S now has tree subtrees N_3, N_4 and N_6 (bordered by dotted lines), with subtree N_6 consisting of the previous subtrees N_1, N_2 and N_5 and the previous node S_v in Figure 3.2b, which now constitutes node $\bar{S}_6$ (connected to S by the connection edge e_6 of subtree N_6). By Lemma 12 the pair $\{x, y\}$ in Figure 3.3a previously introduced as step pair $\{u, v\}$ in Figure 3.2b is a cut pair regarding the edge $e_6 = \{S, \bar{S}_6\}$, with $x \in S$ and $y \in \bar{S}_6$. The vertices $\{u, v\} \subseteq S$ serve as next step pair with the related split cut $(U(S), V(S) \setminus U(S))$ (dashed line). In Figure 3.3b the node S is already split and replaced by S_u and S_v connected by a new edge $e = \{S_u, S_v\}$. The vertex x now lies in $(V(S) \setminus U(S))$ (and in S_v particularly), but the connection edge e_6 of subtree N_6 of S containing $\bar{S}_6$ is reconnected to S_u after splitting S, i.e., $\{\bar{S}_6, S_u\}$ is an edge in the intermediate min-cut tree $T_\star(G)$ after splitting S. So by Lemma 12 the vertices $\{u, y\}$ constitute a cut pair (additionally to $\{x, y\}$) regarding the reconnected edge $e_6 = \{\bar{S}_6, S_u\}$.



(a) Minimum u-v-cut in $G_\circ(S)$ also separates x and y



(b) Previous step pair $\{x, y\}$ gets hidden

Figure 3.3: About step pairs, cut pairs and hidden step pairs.

Definition:
Nearest
cut pair

Definition 9 *We call such a cut pair that occurs last of all cut pairs regarding an edge e_j in an intermediate min-cut tree as just described and that corresponds either* *to the step pair that creates the edge e_j or consists of vertices from two different step pairs, as the pair $\{u, y\}$ does, the* nearest cut pair *regarding the edge e_j.*

In the resulting intermediate min-cut tree $T_\star(G)$ in Figure 3.3b the nodes S_v and $\bar{S}_6$ which contain x and y are not adjacent anymore. So the step pair $\{x, y\}$ does not define an edge in $T_\star(G)$ anymore. We call such a step pair *hidden step pair*. As a hidden step pair never gets *visible* again, once hidden the step pair $\{x, y\}$ does not define an edge in the final min-cut tree $T(G)$. However, the hidden step pair $\{x, y\}$ still is a cut pair for the edge $e_6 = \{\bar{S}_6, S_u\}$ originally created the step pair $\{x, y\}$.

Definition: Hidden step pair

The step pair $\{x, y\}$ in Figure 3.3a gets hidden, because the edge $e_6 = \{S, \bar{S}_6\}$, which was originally created by $\{x, y\}$, is reconnected to S_u after splitting S in Figure 3.3b. So e_6 then connects nodes which do not both contain either x or y anymore (as x lies in S_v). This corresponds to the case in Lemma 12 which defines an additional cut pair for the reconnected edge. This case only occurs if there occurs a split cut that separates the node $\bar{S}_6$ containing y from the cut side containing x. This yields the following observation.

Observation 2 *A step pair $\{u, v\}$ in a Gomory-Hu execution only gets hidden if there occurs a split cut that separates u and v. Note, that such a split cut may only occur by splitting one of the nodes containing u or v. Otherwise, if the current split node S contains neither u nor v (we assume $\{u, v\}$ not to be hidden yet), both vertices u and v belong to the same subtree N_j of the current split node S, and therefore, are contracted in η_j in $G(S)$.*

Observation: Condition for hidden step pairs

According to Lemma 12 and based on the structure of min-cut trees we can state three further remarks, which we will use later.

Remark 3 Each step pair of a Gomory-Hu execution is also a cut pair regarding at least one edge in the final min-cut tree $T(G)$, namely the edge which is created by the step pair. By contrast, there may exist cut pairs regarding some edges in $T(G)$ which never served as step pairs. For each edge e in $T(G)$ there exists at least one cut pair regarding this edge e, namely the step pair which created e. Furthermore, the vertices u and v incident with e always constitute a cut pair regarding e, as they form the nearest cut pair regarding e.

Remark: Step pairs and cut pairs

Remark 4 A cut in an undirected, weighted $G = (V, E, c())$ induced by an edge e of a min-cut tree $T(G)$ separates the vertices u and v in V if and only if e is an edge on the unique path γ from u to v in $T(G)$.

Remark: Separation properties of edge-induced cuts

Remark 5 Let $\{u, v\} \in E$ denote a bridge in graph $G = (V, E, c())$ and $G_u = (V_u, E_u, c_u())$ and $G_v = (V_v, E_v, c_v())$ the two connected components induced by $\{u, v\}$, with $u \in V_u$ and $v \in V_v$. Then the cut (V_u, V_v) is the only minimum u-v-cut in G, as each u-v-cut at least crosses the bridge $\{u, v\}$. Therefore, in each min-cut tree $T(G)$ of G there exists an edge e that induces the cut (V_u, V_v). Removing e from $T(G)$ then yields a disconnected tree with connected components $T_u = T(G)|_{G_u}$ and $T_v = T(G)|_{G_v}$ (min-cut tree $T(G)$ reduced to component G_v).

Remark: Bridges in min-cut trees

Chapter 4

Dynamically Updating Min-Cut Trees

As the cut-clustering method of Flake et al. [FTT04] bases on the calculation of a min-cut tree, in the previous Chapter 3 we stated some basic lemmas about the impact of dynamic modifications on cuts and described the method of Gomory and Hu [GH61] to construct a min-cut tree. In this Chapter now we explore, independently from any clusterings, how a min-cut tree $T(G)$ is affected by a dynamic modification of the considered graph G. To this end it is not necessary to distinguish dynamic inter- and intra-cluster modifications. We just analyze the two general modifications of addition and deletion of an edge and develop an idea for an algorithm which dynamically updates an initial min-cut tree. An approach for the addition and deletion of a vertex will also be described.

4.1 Dynamic Changes of Min-Cut Trees

To get an idea how a min-cut tree can be updated dynamically, we explore how a min-cut tree $T(G)$ is affected by a dynamic modification of the graph G. To this end we consider the construction of min-cut trees as described in the Gomory-Hu method (see Algorithm 3). The following theorem and the additional corollaries base on this method.

4.1.1 Execution Theorem and Corollaries

The following theorem constitutes the base of all further conclusions in this section. It allows us to construct a Gomory-Hu execution $GH = (G, F, K)$ where a specific intermediate min-cut tree $T_\star(G)$ occurs in. Thus, we call it *execution theorem.* Note, that the execution theorem does not consider any dynamic modification of the graph G yet.

Theorem: Constructing Gomory-Hu execution from min-cut tree

Theorem 1 *(Execution theorem) Let $T(G) = (V, E_T, c_T())$ denote a min-cut tree of an undirected, weighted graph $G = (V, E, c())$. Consider further a set $M \subseteq E_T$ of edges in $T(G)$ and let $T_\circ(G) = (V_\circ, M, c_\circ())$ denote the tree that results from contracting the edges of $E_T \setminus M$ in the min-cut tree $T(G)$.*

With f an arbitrary sequence of all edges in M and f' an arbitrary sequence of all edges in $E_T \setminus M$ and k and k' the related sequences of edge-induced cuts in G, *the Gomory-Hu execution* $GH = (G, f \cdot f', k \cdot k')$ *has* $T_\circ(G)$ *as intermediate min-cut tree (after the application of sequence* f*). The notation* $f \cdot f'$*, and* $k \cdot k'$ *respectively, describes the concatenation.*

Proof. We prove Theorem 1 by induction on the $n-1$ edges in $f \cdot f'$. The edges are regarded as step pairs in a Gomory-Hu execution GH. The set M' denotes the step pairs already applied in the execution.

Induction base case

Induction base case: The algorithm starts with a single node S containing all vertices of G, i.e., for the intermediate min-cut tree $T_\star(G)$ it holds at the beginning that $V_\star = \{V\}$ and $E_\star = \emptyset$. The contracted graph $G(S)$ corresponds to G. Therefore, the current intermediate min-cut tree $T_\star(G)$ corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$ in $T(G)$, with $M' = \emptyset$ (as there was no step pair applied yet). Now take the first pair $\{u,v\}_1$ in $f \cdot f'$ as step pair for the algorithm. As it holds that $S = V$, $\{u,v\}_1$ is a valid step pair in the current split node S. At the same time $\{u,v\}_1$ represents an edge in $T(G)$ and therefore, induces a minimum u-v-cut $(U, V \setminus U)$ in $G = G(S)$ as a valid split cut, with $u \in U$. By splitting and replacing $S = V$ by $S_u = U$ and $S_v = V \setminus U$ connected with a new edge, we get an intermediate min-cut tree $T_\star(G)$ with $V_\star = \{S_u, S_v\} = \{U, (V \setminus U)\}$ and $E_\star = \{\{S_u, S_v\}\}$. The only edge in $E_\star$, created by the step pair $\{u,v\}_1$, has weight $c_T(\{u,v\}_1) = c(U, V \setminus U)$. So after one iteration the intermediate min-cut tree $T_\star(G)$ corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$ in $T(G)$, with $M' = \{\{u,v\}_1\}$. Note, that the step pair $\{u,v\}_1$ is not hidden until now.

Induction hypothesis

Induction hypothesis: We now assume the pairs $\{u,v\}_2, \ldots, \{u,v\}_z$ in $f \cdot f'$ to be valid step pairs regarding the various split nodes S in $z-1$ further iterations and the related edge-induced cuts in G to be valid split cuts regarding the various contracted graphs $G(V)$. The current intermediate min-cut tree $T_\star(G)$ after z iterations is supposed to correspond to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$, with $M' = \{\{u,v\}_1, \ldots, \{u,v\}_z\}$ being the set of the first z step pairs in $f \cdot f'$. This implies that none of the step pairs in M' is hidden until now.

Induction step

Induction step: By the induction hypothesis the edge represented by the next pair $\{u,v\}_{z+1}$ in $f \cdot f'$ is still contracted in the previous intermediate min-cut tree $T_\star(G)$, i.e., in the intermediate min-cut tree after z iterations. So the vertices u and v lie in the same node S of $T_\star(G)$ and $\{u,v\}_{z+1}$ is a valid step pair in the current iteration regarding S as current split node. The related cut $(U, V \setminus U)$ induced by the edge $\{u,v\}_{z+1}$ in G, with $u \in U$, is supposed to serve as the current split cut. So we need show, that this cut is also a minimum u-v-cut in the current contracted graph $G(S)$.

Let $N(j)$ denote the set of vertices in a subtree N_j of the current split node S. Then the current contracted graph $G(S)$ results from G by contracting the set $N(j)$ in G for all subtrees of S. The cut $(U, V \setminus U)$ induced by the edge $\{u,v\}_{z+1}$ in G is a minimum u-v-cut in G. Moreover, cut $(U, V \setminus U)$ does not separate any two vertices g and h lying in the same set $N(j)$, as otherwise the edge $\{u,v\}_{z+1}$ would lie on the unique path γ from g to h in $T(G)$ (by Remark 4). This contradicts the assumption that g and h belong to the same subtree N_j of S while it holds that $\{u,v\} \subseteq S$. So, by Lemma 11, the cut $(U, V \setminus U)$ is also a minimum u-v-cut in the contracted graph $G(S)$ and hence is a valid split cut for the $(z+1)$-th iteration.

We finally have to prove that after splitting and replacing the current split node S and after reconnecting the subtrees of S the resulting intermediate min-cut tree $T_\star(G)$, i.e., the intermediate min-cut tree after $z+1$ iterations, again corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T \setminus M'$ in $T(G)$, with $M' = \{\{u,v\}_1, \ldots, \{u,v\}_{z+1}\}$ being the set of the first $z+1$ step pairs in $f \cdot f'$.

To this end we show that none of the step pairs $\{u,v\}_1,\dots,\{u,v\}_z$, which created the edges of the previous intermediate min-cut tree, gets hidden by the splitting of S. To get hidden, by Observation 2 a previous step pair needs to be separated by the current split cut in the contracted graph $G(S)$. This further implies the separation of this step pair in G (by Corollary 1). However, it follows directly from the tree structure of $T(G)$ that no edge in $T(G)$ gets separated by a cut induced by another edge of $T(G)$ in G. As the new edge $\{S_u,S_v\}$ in $T_\star(G)$ is created by the step pair $\{u,v\}_{z+1}$, which represents an edge in $T(G)$, and as all other step pairs in $M'=\{\{u,v\}_1,\dots,\{u,v\}_z\}$ also represent edges in $T(G)$ as well as in $T_\star(G)$ (by the induction hypothesis), none of the step pairs $\{u,v\}_1,\dots,\{u,v\}_z$ gets separated by the split cut related to $\{u,v\}_{z+1}$. Therefore, after $z+1$ iterations, the new intermediate min-cut tree $T_\star(G)$ corresponds to $T_\circ(G)$ formed by contracting all edges of $E_T\setminus M'$ in $T(G)$, with $M'=\{\{u,v\}_1,\dots,\{u,v\}_{z+1}\}$ being the set of the first $z+1$ step pairs in $f\cdot f'$. The assertion of Theorem 1 follows with $M'=M$. □

In addition to Theorem 1 we state a corollary which, in contrast to the former, considers a dynamic modification of graph G. With the aid of Remark 4 this corollary restricts the assertions of Lemma 8 (A2) and Lemma 9 (A4) to cuts induced by edges of a min-cut tree $T(G)$. As we identify edge-induced cuts with their inducing edges, we can say that Corollary 5 describes which cut $\{u,v\}\in E_T$ remains a minimum u-v-cut in G after a dynamic modification of G. Let $G=(V,E,c())$ denote an undirected, weighted graph and $T(G)=(V,E_T,c_T())$ a min-cut tree of G. The graph enlarged by the addition of an edge $e_\oplus=\{b,d\}$ is called $G_\oplus=(V,E\cup\{e_\oplus\},c_\oplus())$. The graph reduced by the deletion of an edge $e_\ominus=\{b,d\}$ (which is no bridge in G) is called $G_\ominus=(V,E\setminus\{e_\ominus\},c_\ominus())$.

Corollary: Canonically edge-induced min-cuts after edge addition and edge deletion

Corollary 5 *By Remark 3, each cut $\{u,v\}\in E_T$ is a minimum u-v-cut in G. Let γ denote the unique path from b to d in $T(G)$.*

Edge addition: *Each cut $\{u,v\}\in E_T$ that* does not lie on *path γ remains a minimum u-v-cuts in $G_\oplus$, by Remark 4 and Lemma 8 (A2). A cut $\{u,v\}\in E_T$ that* lies on *path γ may be a minimum u-v-cut in $G_\oplus$ on several conditions given in Lemma 8 (A4).*

Edge deletion: *Each cut $\{u,v\}\in E_T$ that* lies on *path γ remains a minimum u-v-cut in $G_\ominus$, by Remark 4 and Lemma 9 (A4). A cut $\{u,v\}\in E_T$ that* does not lie on *path γ may be a minimum u-v-cut in $G_\ominus$ on several conditions given in Lemma 9 (A2).*

We further deduce Corollary 6, which considers the special case when a bridge is deleted. The two connected components induced by the deletion of a bridge $e_\ominus=\{b,d\}$ in G are called $G_b=(V_b,E_b,c_b())$ and $G_d=(V_d,E_d,c_d())$, with $b\in V_b$ and $d\in V_d$. Then, by Remark 5, the min-cut tree $T(G)$ decomposes into two connected components $T_b=T(G)|_{G_b}$ and $T_d=T(G)|_{G_d}$.

Corollary: Reduced min-cut trees

Corollary 6 *The component T_b is a min-cut tree of G_b and the component T_d is a min-cut tree of G_d.*

Corollary 6 is correct as each pair $\{u,v\}\subseteq V_b$ (the case when $\{u,v\}\subseteq V_d$ is symmetric) is a cut pair regarding the cheapest edge e on the path from u to v in $T(G)$, and in T_b respectively by Lemma 10. Note, that, vice versa, the addition of a bridge between two components not connected yet is equivalent to the insertion of a vertex, which is described in the following subsection.

4.1.2 Algorithm Ideas for Updating Min-Cut Trees

In this subsection we introduce a first idea of an algorithm dynamically updating min-cut trees. The idea results from Corollary 5, Corollary 6 and the execution theorem (see Theorem 1). We firstly analyze the modification of an edge and afterwards shortly describe an approach for the insertion and removal of a vertex, too. Again let $G = (V, E, c())$ denote a connected, undirected, weighted graph and $T(G) = (V, E_T, c_T())$ a min-cut tree of G. Let $\{b, d\} \in E$ denote the modified edge in G. The graph resulting from G by increasing the weight of $\{b, d\}$ is denoted by $G_\oplus$, and $G_\ominus$ denotes the graph resulting from G by decreasing the weight of $\{b, d\}$ to zero. Remember that $G_\ominus$ becomes disconnected if $\{b, d\}$ is a bridge in G. The unique path from b to d in $T(G)$ is called γ, and the set of all cuts, and edges respectively, on the path is called Γ.

Updating After Edge Addition and Edge Deletion

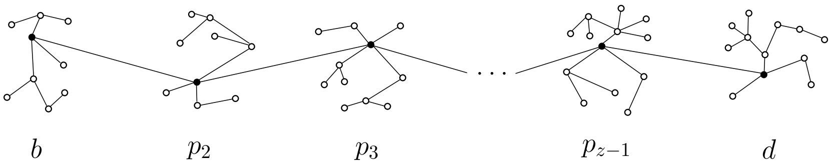
By Corollary 5 we know some edges $\{u, v\}$ in $T(G)$ which remain minimum u-v-cuts for sure due to the modification of G. The residual edges may only remain minimum u-v-cuts on several conditions which are as costly to check as the calculation of a new minimum u-v-cut. So the rough idea for updating min-cut trees is to omit at least the calculation of the minimum u-v-cuts that we already know as they remain the same for sure.

Illustration of the basic idea

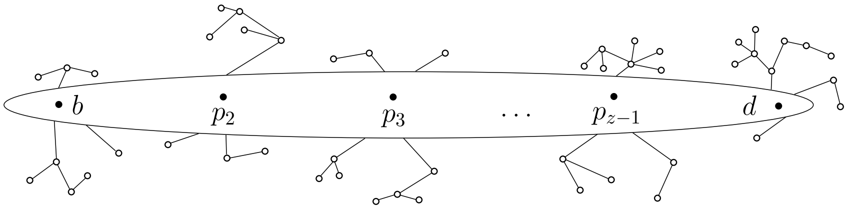
To this end let $T_\circ(G)$ denote the tree formed by contracting all edges $\{u, v\}$ in $T(G)$ for which it is doubtful whether they remain minimum u-v-cuts due to the modification of G. This is, the tree $T_\circ(G)$ is formed by contracting all edges of Γ (by Corollary 5) in the case of edge addition, i.e., weight increasing. In the case of edge deletion, i.e., weight decreasing, with $\{b, d\}$ no bridge in G, the tree $T_\circ(G)$ is formed by contracting all edges of $E_T \setminus \Gamma$. Figure 4.1 shows a min-cut tree $T(G)$ and the related tree $T_\circ(G)$ in both cases. In the case of edge deletion, with $\{b, d\}$ a bridge in G, there are no edges contracted in $T(G)$. Instead, by Corollary 6 the tree $T(G)$ decomposes into two min-cut trees $T(G_b) = T(G)|_{G_b}$ and $T(G_d) = T(G)|_{G_d}$ of the components G_b and G_d induced by the bridge $\{b, d\}$ in G. So in this case we already know a min-cut tree $T(G_\ominus)$ of the disconnected graph $G_\ominus$.

Therefore, in the following we only consider the two cases not concerning bridges. The idea resulting from the execution theorem is now to show that the contracted tree $T_\circ(G)$ can be taken as an initial intermediate min-cut tree for the Gomory-Hu algorithm applied to the enlarged graph $G_\oplus$, and the reduced graph $G_\ominus$ respectively. With this approach, to get an entire min-cut tree of $G_\oplus$, and $G_\ominus$ respectively, we only need to process the iterations indicated by step pairs that are still contracted in nodes of $T_\circ(G)$. So in the case of edge deletion (when the edges on γ induce minimum u-v-cuts) there are still $(n-1) - |\gamma|$ iterations to process, as the number of edges in each min-cut tree corresponds to the number of iterations of a Gomory-Hu execution (by Remark 2). In the case of edge addition (when the edges off γ induce minimum u-v-cuts) there are $|\gamma|$ iterations left. So the effort for updating an initial min-cut tree $T(G)$ with this approach in both cases depends on the length of path γ.

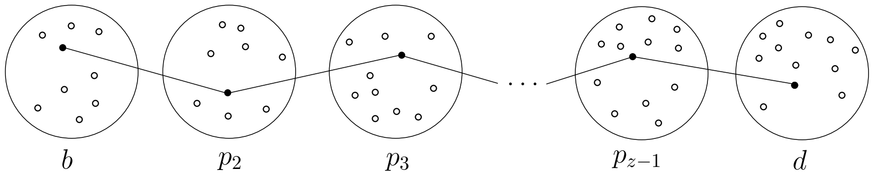
To prove this approach of an updating algorithm to be feasible, we state and prove the following lemma with the aid of the execution theorem (see Theorem 1). Thereby we consider both cases, edge addition and edge deletion, at the same time, so in the following the tree $T_\circ(G)$ results from contracting all edges of either Γ or $E_T \setminus \Gamma$ in $T(G)$. In the formulation of Theorem 1 this means that it either holds that $M = E_T \setminus \Gamma$ or $M = \Gamma$. Let f again denote an arbitrary sequence of all



(a) Min-cut tree $T(G)$ of G with path $\gamma = (b, p_2, \dots, p_{z-1}, d)$



(b) Tree $T_\circ(G)$ formed by contracting all edges of Γ in $T(G)$



(c) Tree $T_\circ(G)$ formed by contracting all edges of $E_T \setminus \Gamma$ in $T(G)$

Figure 4.1: Min-cut tree $T(G)$ and related tree $T_\circ(G)$ in both cases.

edges in M and f' an arbitrary sequence of all edges in $E_T \setminus M$ and k and k' the related sequences of edge-induced cuts in G. Then, by Theorem 1, the Gomory-Hu execution $GH = (G, f \cdot f', k \cdot k')$ reaches $T_\circ(G)$ as intermediate min-cut tree after the application of f. We still need to show that there also exists a Gomory-Hu execution regarding the modified graph $G_\oplus$, or $G_\ominus$ respectively, that has $T_\circ(G)$ as intermediate min-cut tree.

Lemma: Gomory-Hu execution for modified graphs

Lemma 13 *In the situation described above the Gomory-Hu execution* $GH_{\oplus(\ominus)} = (G_{\oplus(\ominus)}, f \cdot f_{\oplus(\ominus)}, k \cdot k_{\oplus(\ominus)})$ *is feasible regarding the graph* $G_\oplus$ *(or* $G_\ominus$ *respectively) and has* $T_\circ(G)$ *as intermediate min-cut tree. The sequence* $f_{\oplus(\ominus)}$ *denotes an arbitrary feasible sequence of step pairs remaining after the application of* f*, and* $k_{\oplus(\ominus)}$ *denotes a feasible sequence of related split cuts.*

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

Induction base case

Induction base case: The execution $GH_{\oplus(\ominus)}$ starts with the first split cut induced by the first edge $\{u, v\}_1$ in f. As the first split cut is applied to the contracted graph $G_{\oplus(\ominus)}(S) = G_{\oplus(\ominus)}$, and $\{u, v\}_1 \in M$ induces a minimum u-v-cut in $G_{\oplus(\ominus)}$ (by the choice of M and Corollary 5), the first split cut is feasible.

Induction hypothesis

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

Induction step

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

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

If it holds that $\{b,d\} \subseteq S$, the contracted graph $G_{\oplus(\ominus)}(S)$ results from $G(S)$ by the addition of the edge $e_\oplus = \{b,d\}$ and, as the edge $\{u,v\}_{z+1}$ does not lie on the path γ, by Remark 4, the current split cut does not separate b and d. So, as the current split cut is a minimum u-v-cut in $G(S)$, by Lemma 8 the current split cut also represents a minimum u-v-cut in $G_\oplus(S)$ and hence is feasible.

Edge deletion ($M = \Gamma$): As all split cuts considered so far separate the modified vertices b and d, the current intermediate min-cut tree is a path of nodes with b included in the first and d included in the last node. So if the current split node S includes b (the case when it includes d is symmetric), then S has only one subtree, which includes d. If S includes neither b nor d, then S has exactly two subtrees, with b and d in different subtrees. In both cases the graph $G_\ominus(S)$ results from $G(S)$ by the deletion of the edge $e_\ominus = \{b,d\}$. Furthermore, by Remark 4 the current split cut separates b and d, as the edge $\{u,v\}_{z+1}$ lies on path γ. So, as the current split cut is a minimum u-v-cut in $G(S)$, by Lemma 9 the current split cut also represents a minimum u-v-cut in $G_\ominus(S)$ and hence is feasible.

As the remaining step pairs and split cuts in $f_{\oplus(\ominus)}$ and $k_{\oplus(\ominus)}$ are defined as arbitrary valid sequences, and as such sequences always exist, the assertion of Lemma 13 is proven. □

Summary of algorithms

The following roughly formulated algorithms finally sum up the ideas for dynamically updating initial min-cut trees introduced in this section. Algorithm 4 considers the modification by addition of an edge, Algorithm 5 regards the edge deletion (not concerning bridges), and Algorithm 6 describes the case of deleting a bridge.

A question still open is how to recognize a bridge in graph G. A very involved approach would be the use of an algorithm as introduced in [Tar74] concerning the connectivity problem and to check if the modified edge $\{b,d\}$ connects two bridge-connected components. An easier way is to calculate a minimum b-d-cut, as one can show that the edge $\{b,d\}$ is a bridge if and only if the minimum b-d-cut is of the same weight as the edge $\{b,d\}$ in G. However, we already know a min-cut tree $T(G)$ of G which allows us to check the modified edge $\{b,d\}$ along the way, by the following lemma.

Lemma: Bridge detection

Lemma 14 *Let $G = (V,E,c())$ denote an undirected, weighted graph with $T(G) = (V,E_T,c_T())$ a min-cut tree of G. An edge $\{b,d\} \in E$, with weight Δ, is a bridge in G if and only if $\{b,d\}$ also constitutes an edge in $T(G)$ with $c_T(\{b,d\}) = \Delta$.*

Algorithm 4: TREE-EDGEADD-1

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, enlarged graph
       $G_\oplus = (V, E \cup \{\{b, d\}\}, c_\oplus())$ with modified vertices $b$ and $d$
Output: Min-cut tree $T(G_\oplus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
if path $\gamma$ spans $V$ then
    return GOMORY-HU ($G_\oplus$)
else
    Calculate tree $T_\circ(G)$ by contracting all edges lying on path $\gamma$ in $T(G)$
    Initialize Gomory-Hu algorithm with $T_\circ(G)$ as intermediate min-cut tree
    %instead of $T_\star(G_\oplus) = (\{V\}, \emptyset, c_\star())$
    Iterate Gomory-Hu algorithm until it stops
    %i.e., until no more step pairs exist
    return result of this Gomory-Hu execution
```

Algorithm 5: TREE-NOBRIDGEDEL

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, reduced graph
       $G_\ominus = (V, E \setminus \{\{b, d\}\}, c_\ominus())$ with modified vertices $b$ and $d$
       (inducing no bridge in $G$)
Output: Min-cut tree $T(G_\ominus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
if path $\gamma$ spans $V$ then
    return path $\gamma$
else
    Calculate tree $T_\circ(G)$ by contracting all edges not lying on path $\gamma$ in $T(G)$
    Initialize Gomory-Hu algorithm with $T_\circ(G)$ as intermediate min-cut tree
    %instead of $T_\star(G_\ominus) = (\{V\}, \emptyset, c_\star())$
    Iterate Gomory-Hu algorithm until it stops
    %i.e., until no more step pairs exist
    return result of this Gomory-Hu execution
```

Algorithm 6: TREE-BRIDGEDEL

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, reduced graph
       $G_\ominus = (V, E \setminus \{\{b, d\}\}, c_\ominus())$ with modified vertices $b$ and $d$
       (inducing a bridge in $G$)
Output: Min-cut tree $T(G_\ominus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
Find cheapest edge $e$ on $\gamma$ inducing minimum $b$-$d$-cut
Remove $e$ from $T(G)$
return resulting components $T(G)|_{G_b}$ and $T(G)|_{G_d}$
```

Proof. (⇒): Let $G_b = (V_b, E_b, c_b())$ and $G_d = (V_d, E_d, c_d())$ denote the two connected components induced by the bridge $\{b, d\}$ in G, with $b \in V_b$ and $d \in V_d$. By Remark 5 there exists an edge e in $T(G)$ that induces the unique minimum b-d-cut (V_b, V_d) in G and hence has weight Δ. This edge e lies on path γ, by Remark 4. If, however, path γ was longer than just edge e, then, by Remark 5, there would exist an edge $e' = \{u, v\}$ (with $\{u, v\} \neq \{b, d\}$ and $\{u, v\} \subseteq V_j$ $(j \in \{b, d\})$) that induces a minimum u-v-cut $(U, V \setminus U)$ with weight $c_T(e')$ in G, which also separates b and d. So, by Lemma 10, the cut $(U_j, V_j \setminus U_j)$, with $U_j = V_j \cap U$, would be a minimum u-v-cut in G_j, with weight $c_T(e') - \Delta$ (as e' separates b and d). We assume $j \in V_j \setminus U_j$ (the case when $j \in U_j$ is symmetric). Then the cut $\theta := (U_j, V \setminus U_j)$ in G would also have weight $c_T(e') - \Delta$, because there are no edges (apart from the bridge $\{b, d\}$) between V_b and V_d in G, and θ does not separate b and d. As cut θ separates u and v and would be cheaper than the cut induced by $e' = \{u, v\}$, this contradicts the assumption that cut e' is a minimum u-v-cut in G. Finally it follows that the path γ only consists of the edge $e = \{b, d\}$ with weight Δ.

($\Leftarrow$): Assume the edge $\{b, d\}$ to be an edge in G with weight Δ and also to constitute an edge $e = \{b, d\}$ of the min-cut tree $T(G)$ with weight $c_T(e) = \Delta$. By Remark 3, the edge e induces a minimum b-d-cut (V_b, V_d) in G, with $b \in V_b$ and $d \in V_d$. As it holds that this minimum b-d-cut has exactly the same weight as the edge $\{b, d\}$ in G, it follows that there are no more edges (apart from $\{b, d\}$) between V_b and V_d. Therefore, $\{b, d\}$ is a bridge between V_b and V_d in G. □

With the aid of Lemma 14 we can join the Algorithms 5 and 6 to a new updating Algorithm 7 in the case of deleting an edge.

Algorithm 7: TREE-EDGEDEL-1

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, reduced graph
    $G_\ominus = (V, E \setminus \{\{b, d\}\}, c_\ominus())$, modified edge $\{b, d\}$ with weight $\Delta$
Output: Min-cut tree $T(G_\ominus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
if $\gamma = \{b, d\}$ and $c_T(\{b, d\}) = \Delta$ then
    Remove edge $\{b, d\}$ from $T(G)$
    return resulting components $T(G)|_{G_b}$ and $T(G)|_{G_d}$
else
    if path $\gamma$ spans $V$ then
        return path $\gamma$
    else
        Calculate tree $T_\circ(G)$ by contracting all edges
          not lying on path $\gamma$ in $T(G)$
        Initialize Gomory-Hu algorithm with $T_\circ(G)$ as
          intermediate min-cut tree
        %instead of $T_\star(G_\ominus) = (\{V\}, \emptyset, c_\star())$
        Iterate Gomory-Hu algorithm until it stops
        %i.e., until no more step pairs exist
        return result of this Gomory-Hu execution
```

Updating After Vertex Insertion and Vertex Removal

For the sake of completeness we further describe two short algorithms regarding the modifications of inserting and removing a single vertex. A vertex $d \in V$ is only removable from graph $G = (V, E, c())$ if it is unconnected, i.e., if all incident edges were removed first. A new vertex d is at first inserted as unconnected vertex, before the edges incident with d are successively added. So the following algorithms base on the updating algorithms regarding edge addition and edge deletion.

Approach for vertex removal

We first consider the removal of a vertex. With Algorithm 7 the approach for updating a min-cut tree after the removal of a vertex d form G is very simple. We just apply Algorithm 7 to all edges incident with vertex d until the current modified graph $G_\ominus$ decomposes into two connected components $G_1 = (\{d\}, \emptyset, \emptyset)$

and $G_2 = (V \setminus \{d\}, E_\ominus, c_\ominus())$. Then Algorithm 7 returns two min-cut trees $T(G_1)$ and $T(G_2)$. We just delete G_1 and the related min-cut tree and keep G_2 and $T(G_2)$ as result.

Approach for vertex insertion

After inserting a vertex d in graph G the modified graph $G_\oplus = (V \cup \{d\}, E, c())$ firstly is disconnected. However, to apply Algorithm 4 we need a connected initial graph and a related min-cut tree. Therefore, we additionally add an edge $\{b, d\}$ to graph $G_\oplus$ which has weight Δ and is incident with b. At the same time we enlarge the min-cut tree $T(G)$ of G by the vertex d and an edge $e = \{b, d\}$ with the same weight Δ. We call the resulting tree $T_\oplus$. As $\{b, d\}$ in $G_\oplus$ is a bridge, the cut $(\{d\}, V)$ is the unique minimum b-d-cut in $G_\oplus$ and has weight Δ. So the edge $e = \{b, d\}$ in $T_\oplus$ induces a minimum b-d-cut in $G_\oplus$. Furthermore, the edge $e = \{b, d\}$ in does not affect any path between vertices of V in $T(G)$, and $T_\oplus$ respectively. So $T_\oplus$ turns out to be a min-cut tree of the connected graph $G_\oplus$. Any further edges incident with d can now be added with the aid of Algorithm 4.

4.2 Simple Implementation of Update-Algorithms

In the previous section we introduced an approach for dynamically updating min-cut trees concerning the elementary modifications of adding and deleting an edge. This approach bases on the idea of using parts of the initial min-cut tree of a graph to abbreviate the Gomory-Hu execution that calculates a new min-cut tree of the modified graph.

As the Gomory-Hu method explicitly requires non-crossing minimum u-v-cuts and, to meet this non-crossing condition, uses operations of vertex contraction, it is fairly involved and nontrivial to implement. Gusfield [Gus90] shows that it is possible to modify the Gomory-Hu method such that vertex contractions are not necessary anymore, as also crossing cuts can be used to construct a min-cut tree. More precisely, Gusfield introduces some ideas which makes the implementation of the Gomory-Hu method much easier by avoiding the calculation of the contracted graph $G(S)$ in the iterations.

Adaptability of Gusfield's ideas

In this subsection we will explore how the ideas of Gusfield [Gus90] are adaptable to our approach for dynamically updating min-cut trees. Unfortunately Gusfield's ideas can not be applied in a natural way, as his proofs and conclusions require a "closed form" of the Gomory-Hu execution. This is, the step pairs are not arbitrarily selectable during the execution, but the sequence of step pairs is required to be of a special shape. However, our updating algorithms (see Algorithm 4 and 7) initialize the Gomory-Hu algorithm with an intermediate min-cut tree $T_\circ(G)$ resulting from a sequence of step pairs that might not meet the required shape. There is no guarantee that there even exists a validly shaped sequence of step pairs which yields the initial intermediate min-cut tree $T_\circ(G)$ used in our algorithms. So we need to adjust Gusfield's ideas to the situation given in our approach.

Gusfield [Gus90] considers the two phases of the Gomory-Hu method (see Algorithm 3) separately. His idea of a simplier realization of *Phase 1* we can adopt without any changes. Remember, *Phase 1* splits and replaces the current split node S by S_u and S_v, with $\{u, v\}$ the related step pair in S. *Phase 2* decides how to reconnect the subtrees of S to the new nodes S_u and S_v. However, to simplify the implementation of this second phase, we need to prove one of the theorems introduced by Gusfield in a more general situation. We start with a review of Gusfield's idea concerning the splitting of the current node S.

4.2.1 Realizing the Node Splitting (*Phase 1*)

In the first phase of an iteration of the Gomory-Hu algorithm the current node S is split by a minimum u-v-cut (the split cut) in the contracted graph $G(S)$ concerning the step pair $\{u, v\}$ in S. Gusfield's idea now bases on the following theorem, which is Theorem 2 in [Gus90] and allows us to use any minimum u-v-cut in G, instead of $G(S)$, to split S.

Theorem: Splitting current node

Theorem 2 *Let $\{u, v\}$ denote the step pair in the current split node S of an intermediate min-cut tree $T_\star(G)$ during a Gomory-Hu execution. If $(U, V \setminus U)$ is any minimum u-v-cut in $G = (V, E, c())$, then there exists a minimum u-v-cut $(U(S), V(S) \setminus U(S))$ in the contracted graph $G(S)$ such that $S \cap U = S \cap U(S)$ and $S \cap (V \setminus U) = S \cap (V(S) \setminus U(S))$ and such that the weights of the two cuts are the same.*

Theorem 2 says that there always exists a minimum u-v-cut in $G(S)$ that splits S the same way as $(U, V \setminus U)$ does. It is proven in [Gus90] and in this work later together with Theorem 3. The proof of Theorem 3 will further illustrate the construction of cut $(U(S), V(S) \setminus U(S))$ in $G(S)$ based on cut $(U, V \setminus U)$ in G. However, to realize the splitting of node S, for the present we just keep in mind that there exists a minimum u-v-cut in $G(S)$ that splits S exactly the same way as the cut $(U, V \setminus U)$ in G does.

4.2.2 Realizing the Subtree Reconnection (*Phase 2*)

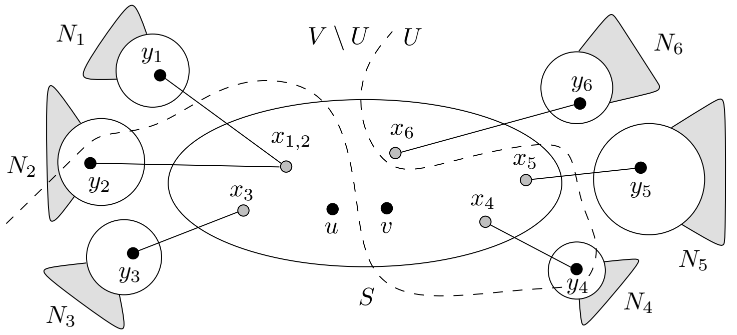
As long as we do not know how to construct cut $(U(S), V(S) \setminus U(S))$ in $G(S)$ based on $(U, V \setminus U)$ in G, the problem that comes with splitting S by Theorem 2 is that we do also not know how to reconnect the subtrees of S after the splitting. Note, that the minimum u-v-cut $(U, V \setminus U)$ in G may, beside the current split node S, also split the sets $N(j)$ corresponding to the subtrees N_j of S. By contrast, in $G(S)$ the subtrees N_j are contracted to nodes η_j and therefore, can not get split by the cut $(U(S), V(S) \setminus U(S))$. So what we need is a rule that allows us to decide by the knowledge of $(U, V \setminus U)$ in G on which side of $(U(S), V(S) \setminus U(S))$ in $G(S)$ a contracted subtree η_j lies.

Gusfield's assertions concerning a "closed form"

Gusfield [Gus90] shows, in his Theorem 3 and Corollary 5, that if for an arbitrary graph G the step pairs are chosen in a special order, what also influences the development of the subtrees, a designated vertex of each subtree N_j lies in $U \subseteq V$ if and only if the contracted node η_j lies in $U(S) \subseteq V(S)$. However, this designated vertices, Gusfield calls them *representatives*, and the special sequence of step pairs are defined iteratively what causes the "closed form" of the Gomory-Hu method required by Gusfield's appoach.

Our assertions concerning a general situation

In this subsection we show that even in an arbitrary intermediate min-cut tree $T_\star(G)$ there exist representatives which tell us how to reconnect the subtrees. To this end we need to reformulate and prove Gusfield's Theorem 3 for this general situation. In the following reformulation, which is referred to as Theorem 3, let S denote the current split node in an intermediate min-cut tree $T_\star(G)$ of graph G. Let further $e_j = \{S, \bar{S}_j\}$ denote the connection edge of a subtree N_j of S, and the nearest cut pair regarding the edge $e_j = \{S, \bar{S}_j\}$, as defined in Definition 9, is called $\{x_j, y_j\}$, with $x_j \in S$ and $y_j \in \bar{S}_j$. An example of such an situation is shown in Figure 4.2. Now suppose $\{u, v\}$ to be the next step pair in S with an arbitrary minimum u-v-cut $(U, V \setminus U)$ in G as split cut.



(a)

Figure 4.2: Intermediate min-cut tree $T_\star(G)$ with subtrees $N_1, \dots, N_6$ and nearest cut pairs $\{x_2, y_1\}, \dots, \{x_6, y_6\}$.

Theorem 3 *A subtree contracted to a node η_j lies in $U(S) \subseteq V(S)$ if and only if the vertex y_j lies in $U \subseteq V$. Note, that this assertion does not depend on the vertex x_j.*

Theorem: Reconnecting subtrees

Theorem 3 constitutes the key for reconnecting subtrees after splitting the current node S with the aid of an arbitrary minimum u-v-cut in G as split cut, concerning a step pair $\{u, v\}$.

Proof of Theorem 3, and Theorem 2

Our proof of Theorem 3 is similiar to Gusfield's proof of his Theorem 3, which considers a special case of the situation in our theorem. So for our proof of Theorem 3 we also use Lemma 1 in [Gus90], which here is reviewed as Lemma 15, and Corollary 1 in [Gus90], which is referred to as Corollary 7.

Lemma 15 *Let $(Y, V \setminus Y)$ denote a minimum y-x-cut in G, with $y \in Y$ and $x \in V \setminus Y$. Let u and v denote two vertices in $V \setminus Y$. Now consider an arbitrary minimum u-v-cut in G and let H denote the side including y. Then the cut $(Y \cup H, (V \setminus Y) \cap (V \setminus H))$ is also a minimum u-v-cut in G.*

Lemma: New minimum u-v-cut

Note, that by Lemma 15 the related sides of the previous u-v-cut and the new u-v-cut only differ in the set Y, i.e., $H \setminus Y = (Y \cup H) \setminus Y = H$ and $(V \setminus H) \setminus Y = ((V \setminus Y) \cap (V \setminus H)) \setminus Y = (V \setminus Y) \cap (V \setminus H)$.

Corollary 7 *Consider the cuts $(Y, V \setminus Y)$ and $(Y \cup U, (V \setminus Y) \cap (V \setminus H))$ from Lemma 15. Then the minimum u-v-cut $(Y \cup U, (V \setminus Y) \cap (V \setminus H))$ does not cross $(Y, V \setminus Y)$, but splits $V \setminus Y$ exactly the same way as the previous arbitrary minimum u-v-cut did.*

Corollary: Non-crossing minimum u-v-cuts

Now the proof of Theorem 2 and Theorem 3 follows. It uses induction on the subtrees of a split node S in an intermediate min-cut tree $T_\star(G)$ and shows constructively that there always exists a split cut $(U(S), V(S) \setminus U(S))$ in $G(S)$ as described in Theorem 2, which is by the way also a minimum u-v-cut in G and does not split any subtree of S. Furthermore, the proof shows that the two sides of this split cut pick the subtrees as describes by Theorem 3.

Proof of Theorem 3

Proof. We consider a situation as described for Theorem 3. For each subtree N_j of the split node S the connection edge $e_j = \{S, \bar{S}_j\}$ induces the minimum y_j-x_j-cut $\theta_j := (N(j), V \setminus N(j))$ in G, with $y_j \in Y$. As it holds that $S \subset V \setminus N(j)$, for each subtree N_j the step pair $\{u, v\}$ lies on the $V \setminus N(j)$-side of the minimum y_j-x_j-cut θ_j induced by the connection edge $e_j = \{S, \bar{S}_j\}$ (see Figure 4.2). Now let $(U, V \setminus U)$ denote an arbitrary minimum u-v-cut in G, with $u \in U$.

Induction base case

Induction base case: We apply Lemma 15 to θ_1 and $(U, V \setminus U)$ and get a minimum u-v-cut $(U_1, V \setminus U_1)$, with $u \in U_1$, that does not separate any vertices in $N(1)$ and splits $V \setminus N(1)$ the same way as $(U, V \setminus U)$ does, by Corollary 7. So, as it holds that $S \subseteq V \setminus N(1)$, also S gets split the same way, and we get

$$\begin{aligned} S \cap U_1 &= S \cap U \\ \text{and} \quad S \cap V \setminus U_1 &= S \cap V \setminus U. \end{aligned}$$

With $y_1 \in N(1)$, by Lemma 15, we further get

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

and therefore, it holds that $N(1) \subseteq U_1$ if and only if $y_1 \in U$. By the remark to Lemma 15 this induces that the related sides of $(U_1, V \setminus U_1)$ and $(U, V \setminus U)$ only differ in $N(1)$, i.e., $U_1 \setminus N(1) = U \setminus N(1)$ and $(V \setminus U_1) \setminus N(1) = (V \setminus U) \setminus N(1)$.

Induction hypothesis

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

$$\begin{aligned} S \cap U_z &= S \cap U \\ \text{and} \quad S \cap V \setminus U_z &= S \cap V \setminus U. \end{aligned}$$

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

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

Induction step

Induction step: We apply Lemma 15 to cut $\theta_{z+1} = (N(z+1), V \setminus N(z+1))$, which is induces by the connection edge $e_{z+1} = \{S, \bar{S}_{z+1}\}$ of subtree N_{z+1}, and cut $(U_z, V \setminus U_z)$. So we get a minimum u-v-cut $(U_{z+1}, V \setminus U_{z+1})$, with $u \in U_{z+1}$, that does not separate any vertices in $N(z+1)$ and splits $V \setminus N(z+1)$ the same way as $(U_z, V \setminus U_z)$ does, by Corollary 7. So, as it holds that $S \subseteq V \setminus N(z+1)$, also S gets split the same way, and we get

$$\begin{aligned} S \cap U_{z+1} &= S \cap U_z \underset{\text{induction hypothesis}}{=} S \cap U \\ \text{and} \quad S \cap V \setminus U_{z+1} &= S \cap V \setminus U_z \underset{\text{induction hypothesis}}{=} S \cap V \setminus U. \end{aligned} \tag{4.1}$$

With $y_{z+1} \in N(z+1)$, by Lemma 15, we further get

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

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

$$N(z+1) \subseteq U_{z+1} \text{ if and only if } y_{z+1} \in U. \tag{4.2}$$

Furthermore, by the remark to Lemma 15 it holds that the related sides of $(U_{z+1}, V \setminus U_{z+1})$ and $(U_z, V \setminus U_z)$ only differ in $N(z+1)$, i.e., $U_{z+1} \setminus N(z+1) = U \setminus N(z+1)$ and $(V \setminus U_{z+1}) \setminus N(z+1) = (V \setminus U_z) \setminus N(z+1)$. So for all sets $N(j)$, $j = 1, \ldots, z$, it follows that $N(j) \subseteq U_{z+1}$ if and only if $N(j) \subseteq U_z$. By induction hypothesis and (4.2) we finally get for $j = 1, \ldots, z+1$

$$N(j) \subseteq U_{z+1} \text{ if and only if } y_j \in U. \tag{4.3}$$

So with Assertion (4.1) and Assertion (4.3) we finally proved the existence of a minimum u-v-cut in G that splits S the same way as $(U, V \setminus U)$ does, and that does not separate any vertices of any subtree of S. By Lemma 11, such a minimum u-v-cut is also a minimum u-v-cut in graph $G(S)$, which results from G by contracting all subtrees of S. So Theorem 2 and Theorem 3 are both proven true. □

4.2.3 Specification of Algorithm Ideas

In the previous subsections we showed that it is possible to adopt Gusfield's idea of considering split cuts not in the contracted graph $G(S)$, but just in graph G instead, to our approach, although our approach initializes the Gomory-Hu method with the special intermediate min-cut tree $T_\circ(G)$, and therefore, does not provide the closed form originally required in Gusfield's paper. With the aid of Gusfield's idea we now specify Algorithm 4 and 7 introduced in Subsection 4.1.2 such that the implementation becomes more easy by omitting the calculation of vertex contractions. Furthermore, considering split cuts just in graph G will make several argumentations and conclusions less involved in the remainder of this work.

For the specification of our algorithms according to Gusfield [Gus90], we define representatives of the nodes in the intermediate min-cut trees during a Gomory-Hu execution. These allow us to decide how to reconnect the subtrees of the current split node S. We distinguish the definition of representatives in the special initial intermediate min-cut tree $T_\circ(G)$ and the iterative definition of representatives of newly occurring nodes during an execution.

Different definitions for representatives

The definition of representatives in the initial intermediate min-cut tree $T_\circ(G)$ again distinguishes the cases of edge addition and edge deletion, as the initial intermediate min-cut tree differs in structure depending on these cases. So we first consider the initial intermediate min-cut tree $T_\circ(G)$ in the case of the addition of an edge $e_\oplus = \{b, d\}$ in graph G. Remember that this intermediate min-cut tree results from an entire min-cut tree $T(G)$ of G by contracting all edges lying on the unique path γ from b to d in $T(G)$ (compare to Lemma 13). So in this case the intermediate min-cut tree $T_\circ(G)$ has only one node, namely the contraction of path γ, that is no singleton. All other nodes in $T_\circ(G)$ consist of exactly one vertex $g \in V$ of G. Furthermore, the contraction of path γ is the node in $T_\circ(G)$ that is considered by the first iteration of the Gomory-Hu execution in Algorithm 4. So the first step pair chosen by Algorithm 4 lies in this node. The representatives in $T_\circ(G)$, in the case of edge addition, are now defined as follows:

Definition: Representatives regarding edge addition

Definition 10 *(Representatives in the case of edge addition) For each singleton $S = \{g\}$ in $T_\circ(G)$ the vertex g is defined as representative $r(S)$. For the contraction of path γ, which is no singleton in $T_\circ(G)$, we choose one of the vertices of the first step pair $\{u, v\}$ as representative.*

In contrast, the initial intermediate min-cut tree $T_\circ(G)$ in the case of the deletion of an edge $e_\ominus = \{b, d\}$ which is no bridge in G results from contracting all edges not lying on the unique path γ from b to d in $T(G)$ (compare to Lemma 13). So in this case the intermediate min-cut tree $T_\circ(G)$ can be regarded as a path of nodes, while each node consists of exactly one vertex $g \in V$ that lies on path γ and several vertices of G not lying on γ. The representatives in $T_\circ(G)$ are then defined as follows:

Definition: Representatives regarding edge deletion

Definition 11 *(Representatives in the case of edge deletion) For each node S in $T_\circ(G)$ the unique vertex g included in S that lies on path γ is defined as representative $r(S)$.*

According to Definition 10 and 11 each node of the initial intermediate min-cut tree $T_\circ(G)$ has a representative. We now illustrate that this representatives correctly decide how to reconnect the subtrees of a split node S.

Correctness of representatives regarding edge addition

In the case of edge addition the contraction of path γ constitutes the split node S in the first iteration of the Gomory-Hu execution in Algorithm 4, and each subtree N_j of S in $T_\circ(G)$ is connected to S by its connection edge $e_j = \{S, \bar{S}_j\}$. As e_j corresponds to an edge in the entire min-cut tree $T(G)$ of G with $\bar{S}_j$ a singleton, the vertex y_j in $\bar{S}_j$ belongs to the nearest cut pair of e_j regarding the Gomory-Hu execution introduced in Lemma 13. So by Theorem 3 the representative $r(\bar{S}_j) := y_j$ correctly decides how to reconnect the subtree N_j. Note, that singletons that belong to connection edges of subtrees of other singletons are never considered as representatives, as singletons do not get split any further.

Correctness of representatives regarding edge deletion

In the case of edge deletion the first split node S in Algorithm 7 is just one of the nodes of $T_\circ(G)$. As the tree $T_\circ(G)$ constitutes is a path of nodes, the split node S has exactly two subtrees N_1 and N_2 connected by connection edges $e_1 = \{S, \bar{S}_1\}$ and $e_2 = \{S, \bar{S}_2\}$. As the edges e_1 and e_2 correspond to edges lying on path γ in the entire min-cut tree $T(G)$ of G, the unique vertices $y_1 \in \bar{S}_1$ and $y_2 \in \bar{S}_2$ that lie on path γ belong to the nearest cut pairs of e_1 and e_2 regarding the Gomory-Hu execution introduced in Lemma 13. So by Theorem 3 the representative $r(\bar{S}_1) := y_1$ and $r(\bar{S}_2) := y_2$ correctly decide how to reconnect the subtrees N_1 and N_2 after splitting S.

Finally we still need to define the representatives of nodes newly occurring during the Gomory-Hu execution in Algorithm 4 and Algorith 7. According to Gusfield [Gus90] these representatives are defined iteratively by the following rule.

Definition: Iteratively defined representatives

Definition 12 *(Representatives of newly occurring nodes) Considering the current split node S, the next step pair is required to include the representative $r(S)$ of S, i.e., the next step pair is $\{r(S), v\}$, with $v \neq r(S)$ an arbitrary vertex in S. After splitting S the new node $S_{r(S)}$, which contains $r(S)$, has again $r(S)$ as representative, for the other node S_v, which contains v, we define $r(S_v) := v$.*

Correctness of iterative representatives

The representatives in Definition 12 correctly decide how to reconnect the subtrees of the current split node S by Theorem 3, as, by Lemma 12 and the fact that each step pair consists of later representatives, a representative y_j is included in the node $\bar{S}_j$ of a connection edge $e_j = \{S, \bar{S}_j\}$ if and only if y_j belongs to the nearest cut pair of e_j regarding the Gomory-Hu execution in Algorithm 4, or Algorithm 7 respectively.

Now we can give the formal description of the specification of TREE-EDGEADD-1 (Algorithm 4) and TREE-EDGEDEL-1 (Algorithm 7). To this end we introduce Algorithm 8, which considers the central issue of executing the Gomory-Hu method

Algorithm 8: CENTRAL-TREE

```
Input: Graph $G_{\oplus(\ominus)}$, initial intermediate min-cut tree $T_\circ(G)$ with
       representatives of nodes, designated split node $S$ in $T_\circ(G)$
Output: Intermediate min-cut tree $T_\star(G_{\oplus(\ominus)})$ with $S$ entirely unfolded
$M \leftarrow S$
$T_\star(G_{\oplus(\ominus)}) \leftarrow T_\circ(G)$
while $\exists$ split node $S \subseteq M$ in $T_\star(G_{\oplus(\ominus)})$ with $|S| > 1$ do
    %compare the following to Definition 12
    Choose arbitrary vertex $v \in S$, with $v \neq r(S)$
    Calculate a minimum $r(S)$-$v$-cut $(U, V \setminus U)$ in $G_{\oplus(\ominus)}$, $r(S) \in U$
    Split $S$ into $S_{r(S)} = S \cap U$ and $S_v = S \cap (V \setminus U)$
    $r(S_{r(S)}) \leftarrow r(S)$
    $r(S_v) \leftarrow v$
    Create new edge $\{S_{r(S)}, S_v\}$ in $T_\star(G_{\oplus(\ominus)})$
    forall edges $e_j = \{S, \bar{S}_j\}$ previously incident with $S$ in $T_\star(G_{\oplus(\ominus)})$ do
        if $r(\bar{S}_j) \in U$ then
            Reconnect subtree $N_j$ with $S_{r(S)}$ in $T_\star(G_{\oplus(\ominus)})$
        else
            Reconnect subtree $N_j$ with $S_v$ in $T_\star(G_{\oplus(\ominus)})$
return $T_\star(G_{\oplus(\ominus)})$
```

for a single node S in the initial intermediate min-cut tree $T_\circ(G)$. More precisely, Algorithm 8 executes the Gomory-Hu method by choosing step pairs from the initial set S as long as there exist valid step pairs and then stops.

With CENTRAL-TREE (Algorithm 8) the specification of TREE-EDGEADD-1 (Algorithm 4) and TREE-EDGEDEL-1 (Algorithm 7) can be given as follows in Algorithm 9 and 10. In the case of the addition of an edge, where the initial intermediate min-cut tree $T_\circ(G)$ has only one node that is no singleton, only this node needs to be unfolded by the application of CENTRAL-TREE. In the case of the deletion of an edge, where the initial intermediate min-cut tree $T_\circ(G)$ is a path of nodes, CENTRAL-TREE is applied to each node in $T_\circ(G)$ that is no singleton.

Algorithm 9: TREE-EDGEADD-2

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, enlarged graph
       $G_\oplus = (V, E \cup \{\{b, d\}\}, c_\oplus())$ with modified vertices $b$ and $d$
Output: Min-cut tree $T(G_\oplus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
if path $\gamma$ spans $V$ then
    return GOMORY-HU ($G_\oplus$)
else
    Calculate tree $T_\circ(G)$ by contracting all edges lying on path $\gamma$ in $T(G)$
    $S \leftarrow$ contraction of $\gamma$
    Calculate representatives of nodes in $T_\circ(G)$
    %as described in Definition 10
    $T_\star(G_\oplus) \leftarrow T_\circ(G)$
    $T_\star(G_\oplus) \leftarrow$ CENTRAL-TREE ($G_\oplus, T_\star(G_\oplus), S$)
    $T(G_\oplus) \leftarrow T_\star(G_\oplus)$ with singletons replaced by vertices
    return $T(G_\oplus)$
```

Algorithm 10: Tree-EdgeDel-2

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, reduced graph
       $G_\ominus = (V, E \setminus \{\{b, d\}\}, c_\ominus())$, modified edge $\{b, d\}$ with weight $\Delta$
Output: Min-cut tree $T(G_\ominus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
if $\gamma = \{b, d\}$ and $c_T(\{b, d\}) = \Delta$ then
    Remove edge $\{b, d\}$ from $T(G)$
    return resulting components $T(G)|_{G_b}$ and $T(G)|_{G_d}$
else
    if path $\gamma$ spans $V$ then
        return path $\gamma$
    else
        Calculate tree $T_\circ(G)$ by contracting all edges
          not lying on path $\gamma$ in $T(G)$
        Calculate representatives of nodes in $T_\circ(G)$
        %as described in Definition 11
        $T_\star(G_\ominus) \leftarrow T_\circ(G)$
        while $\exists$ split node $S$ in $T_\star(G_\ominus)$ with $|S| > 1$ do
            $T_\star(G_\ominus) \leftarrow$ Central-Tree $(G_\ominus, T_\star(G_\ominus), S)$
        $T(G_\ominus) \leftarrow T_\star(G_\ominus)$ with singletons replaced by vertices
        return $T(G_\ominus)$
```

4.3 Algorithm Engineering

In the previous Sections 4.1 and 4.2 we introduced an approach for dynamically updating min-cut trees by avoiding the calculation of minimum u-v-cuts that for sure do not change due to the modification of graph G. However, especially in the case of edge deletion the set of cuts known not to change is expected to be rather small, as so far we only regard the edges lying on path γ as not changing. By contrast, in the case of edge addition the presumably larger part of a min-cut tree $T(G)$ consisting of edges not lying on γ does not change for sure.

In this section we explore whether there are even more minimum u-v-cuts whose calculation can be avoided, or for which at least can be checked with little effort whether they may change. To this end we watch the cuts from two different angles. The first view concentrates on a given cut and asks for cut pairs in a given graph. The second view considers pairs of vertices in a given graph and asks for minimum separating cuts. The following Subsection 4.3.1 explores edge-induced cuts in the modified graph from the first angle, while Subsection 4.3.2 tries to find minimum u-v-cuts for given pairs $\{u, v\}$ with less effort compared to a recalculation. In Section 4.3.3 we then improve Tree-EdgeAdd-2 (Algorithm 9) and Tree-EdgeDel-2 (Algorithm 10) with the aid of the insight we gained in the former two sections. As for the case of edge deletion less effort saving is expected we mainly concentrate on the improvement of this subject.

4.3.1 Edge-Induced Cuts as Minimum Separating Cuts

In this subsection we consider cuts induced in the modified graph $G_{\oplus(\ominus)}$ by edges of the previous min-cut tree $T(G)$. We just give a loose collection of observations. The first observation stated as Lemma 16 is quite simple. It regards the minimum weighted edge on the path γ in $T(G)$ in the case of edge addition.

Lemma 16 *In the case of the addition of an edge $e_\oplus = \{b, d\}$, with weight Δ, in graph G the minimum weighted edge e_{min} on the unique path γ from b to d in the previous min-cut tree $T(G) = (V, E_T, c_T())$ also induces a minimum b-d-cut θ_{min} in the modified graph $G_\oplus$.*

Lemma: Minimum b-d-cut remains

Proof. The cut $\theta_{\min}$ in graph $G_\oplus$ has weight $c_\oplus(\theta_{\min}) = c_T(e_{\min}) + \Delta$. If there was a cheaper b-d-cut in $G_\oplus$, this cut would also separate b and d, and therefore, would have already been cheaper in G. $\square$

The next lemma considers the case of edge deletion. It assumes the calculation of at least one new minimum u-v-cut in the modified graph $G_\ominus$, but depending on the shape of this cut Lemma 17 states an assertion about edge-induced minimum separating cuts not changing due to the modification. Note, that an analog assertion in the case of edge addition does not exist.

Lemma 17 *Let $(U, V \setminus U)$ denote an arbitrary minimum u-v-cut in the modified graph $G_\ominus$ for arbitrary vertices u and v. If $(U, V \setminus U)$ does not separate the modified vertices b and d, each cut induced by an edge $\{g, h\}$ of the previous min-cut tree $T(G)$ that lies on the side of $(U, V \setminus U)$ not including b and d also remains a minimum g-h-cuts in $G_\ominus$.*

Furthermore, each such edge $\{g, h\}$ remains a minimum separating cut in $G_\ominus$ for each of its previous cut pairs in G that lies on the side of $(U, V \setminus U)$ not including b and d.

Lemma: Known cut side

Proof. Consider the minimum u-v-cut in $G_\ominus$ to be the first split cut, concerning the step pair $\{u, v\}$, in a Gomory-Hu execution. Then, after splitting V by this cut, consider the side of $(U, V \setminus U)$ which does not include b and d as next split node S. So regarding S both modified vertices b and d lie in the same subtree, and therefore, are contracted in $G_\ominus(S)$. Hence, each minimum g-h-cut in $G_\ominus(S)$ with $\{g, h\} \subseteq S$ does not separate b and d and is also a minimum g-h-cut in $G_\ominus$, by Lemma 12. So from Lemma 8 (A1, A2) it follows that any previous minimum g-h-cut in G that does not separate b and d is also a minimum g-h-cut in $G_\ominus$. $\square$

The next corollary, which considers *treetops* in the previous min-cut tree $T(G)$, follows from Lemma 17. For an introduction of treetops see the following Definition 13. Figure 4.3 shows an example of a treetop $\Uparrow_e$.

Definition 13 *Let $e = \{u, v\}$ denote an edge not lying on the path γ between the modified vertices b and d in the previous min-cut tree $T(G)$, and let θ_e denote the cut induced by e in G. Then the treetop $\Uparrow_e$ of the edge e is defined as the set of all vertices lying on that side of θ_e which does not include the path γ.*

Definition: Treetop of an edge

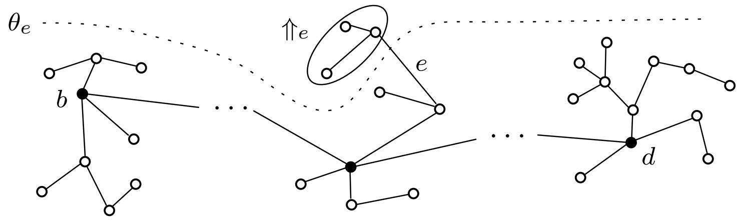


Figure 4.3: Treetop $\Uparrow_e$ of an edge e.

Corollary: Known treetop

Corollary 8 *Let* $e = \{u, v\}$ *denote an edge in the previous min-cut tree* $T(G)$ *that does not lie on the path* γ *between the modified vertices* b *and* d*, and let* θ_e *denote the cut induced by* e *in graph* G*. If* θ_e *also constitutes a minimum* u-v*-cut in the modified graph* $G_\ominus$*, then each edge* $\{g, h\}$ *included in the treetop* $\Uparrow_e$ *also induces a minimum* g-h*-cut in graph* $G_\ominus$*.*

The last lemma now states an assertion which at the first glance seems quite different from the one in Lemma 16, but follows by a similar argumentation. It considers the deletion of an edge $e_\ominus = \{b, d\}$, with weight Δ, from graph G and an edge $\{g, h\}$ with only vertex g lying on the unique path γ from b to d in the previous min-cut tree $T(G)$ (compare to Figure 4.4).

Lemma: Condition for remaining minimum u-v-cuts

Lemma 18 *Let* e_{min} *denote the cheaper of the two edges on path* γ *incident with vertex* g*, and let* $\{u, v\}$ *denote an edge with* u *and* v *included in the treetop of edge* $\{g, h\}$ *in* $T(G)$*. If the edge* $\{u, v\}$ *is cheaper than* $c_T(e_{min}) - \Delta$ *in the previous min-cut tree* $T(G)$*, it also induces a minimum* u-v*-cut in the modified graph* $G_\ominus$*.*

Proof. Consider the initial intermediate min-cut tree $T_\circ(G)$ resulting from contracting all edges not lying on the unique path γ in $T(G)$. In $T_\circ(G)$ the vertex g then constitutes the representative $r(S)$ of a node S, and S also contains the vertices h, u and v. Furthermore, the two edges on path γ incident with vertex g in $T(G)$ correspond to the connection edges $e_b := \{y_b, g\}$ and $e_d := \{y_d, g\}$ of the two subtrees N_b and N_d of S in $T_\circ(G)$ (see Figure 4.4). Remember that these two edges also induce minimum separating cuts of at least weight $c_T(e_{\min}) - \Delta$ in the modified graph $G_\ominus$. Lemma 9 now says that each minimum u-v-cut θ in the contracted graph $G_\ominus(S)$ that is cheaper than the previous minimum u-v-cut in G separates the modified vertices b and d. This is, such a cut θ either separates the nearest cut pair $\{y_b, g\}$ of edge e_b or the nearest cut pair $\{y_d, g\}$ of edge e_d.

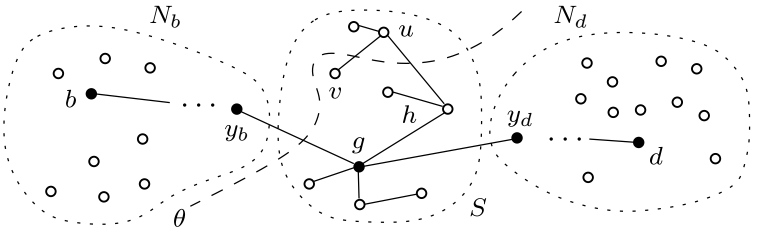


Figure 4.4: Illustration to the proof of Lemma 18.

Now we assume the previous minimum u-v-cut induced in G by edge $\{u, v\}$ to be cheaper than $c_T(e_{\min}) - \Delta$, with $e_{\min}$ the cheaper of the two edges e_b and e_d. If there existed a new minimum u-v-cut θ in $G_\ominus(S)$ cheaper than the previous minimum u-v-cut in G, this cut θ would separate one of the nearest cut pairs $\{y_b, g\}$ or $\{y_d, g\}$. As cut θ, however, would be cheaper than $c_T(e_{\min}) - \Delta$, this contradicts the fact that both edges e_b and e_d also induce minimum separating cuts of at least weight $c_T(e_{\min}) - \Delta$ in the modified graph $G_\ominus$. $\square$

4.3.2 New Minimum Separating Cuts for Given Vertices

In this subsection we consider given pairs of vertices and ask for the updated minimum separating cuts in the modified graph. We mainly explore the case of the deletion of an edge. To this end we independently regard two different aspects which will be combined later in Subsection 4.3.3 to an improvement of TREE-EDGEDEL-2 (Algorithm 10). The first aspect concentrates on the contracted graph $G_\ominus(S)$, with S a node of the initial intermediate min-cut tree $T_\circ(G)$ resulting from the entire min-cut tree $T(G)$ of G by contracting the edges not lying on path γ, and asks for properties of cuts in $G_\ominus(S)$ which separate the modified vertices. Remember that $T_\circ(G)$ is a path of nodes in the case of edge deletion. The second aspect considers minimum separating cuts in a more general situation, which is described later. We start with the more concrete issue of graph $G_\ominus(S)$.

First Aspect

As mentioned above this aspect considers the contracted graph $G_\ominus(S)$, with S a node of the initial intermediate min-cut tree $T_\circ(G)$ resulting from the entire min-cut tree $T(G)$ of G by contracting the edges not lying on path γ. To achieve a skimpy formulation of the lemmas and facts in this paragraph, we characterize some special parts of the graph $G_\ominus(S)$. Figure 4.5 shows the terms defined in the following.

Definition: Subtree-sets and wood of an edge

Definition 14 *In the contracted graph* $G_\ominus(S)$ *we denote the set of vertices included in the subtree of* S *that contains the modified vertex* b *by* $N(b)$*, and analogously* $N(d)$ *denotes the set of vertices in the subtree containing* d*. With the aid of the structure of the previous min-cut tree* $T(G)$ *the set* S *can be split regarding a designated edge* $e = \{u, v\} \subseteq S$ *into the treetop* $\upuparrows_e$ *and the remaining vertices in* $S \setminus \upuparrows_e$*. We call this set of remaining vertices the wood* $\sharp_e := S \setminus \upuparrows_e$ *of the edge* e*.*

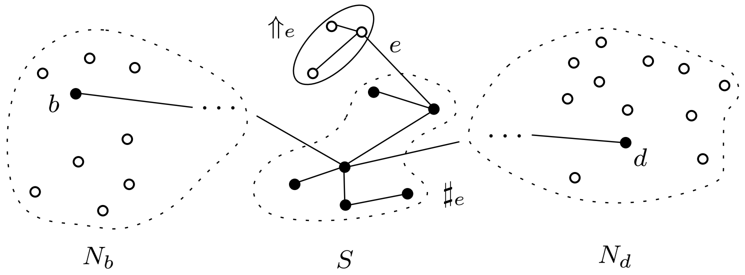


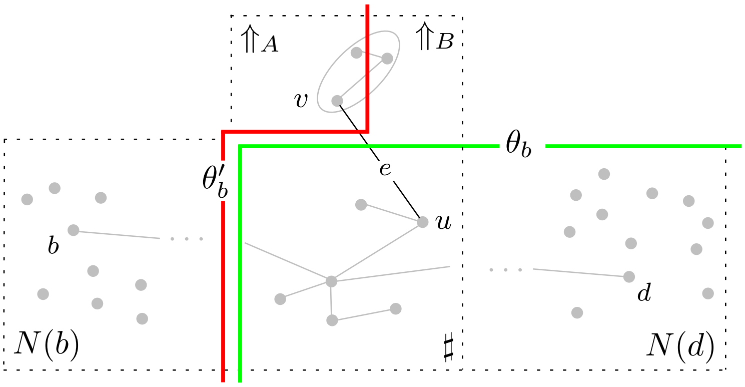
Figure 4.5: Special parts of the contracted graph $G_\ominus(S)$.

The following Lemma 19 actually serves to introduce the more general Lemma 20 and to state the main fact in this subsection, which is given later as Fact 3. To this end Lemma 19 states an assertion about specially defined cuts in the contracted graph $G_\ominus(S)$. It compares, regarding an arbitrary edge $e = \{u, v\} \subseteq S$, a cut that separates u and v and additionally splits the treetop $\upuparrows_e$ and one of the subtrees of S from the rest of $G_\ominus(S)$ with a cut also separating u and v and additionally splitting the same subtree, but only a part of the treetop from the rest of $G_\ominus(S)$. Note, that the cuts considered in this lemma are not yet required to be minimum separating cuts, although they both separate u and v. For a better readability we omit the indices regarding the edge e in the formulas.

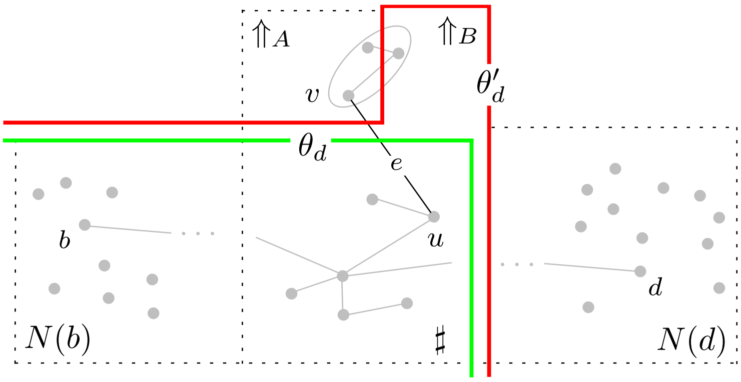
Lemma: Comparing special cuts in $G_\ominus(S)$

Lemma 19 *In the situation described above and shown in Figure 4.6 let* $e = \{u, v\}$ *denote an edge in* S *and let* $(\upuparrows_A, \upuparrows_B)$ *denote an arbitrary cut of the treetop* $\upuparrows_e$*, with* $v \in \upuparrows_A$*. Then in* $G_\ominus(S)$*, and* $G_\ominus$ *respectively, it holds that*

$$\begin{aligned} c_\ominus(\theta_b) &:= c_\ominus(N(b) \cup \upuparrows,\ N(d) \cup \sharp) \\ &\le c_\ominus(N(b) \cup \upuparrows_A,\ N(d) \cup \sharp \cup \upuparrows_B) =: c_\ominus(\theta'_b) \end{aligned}$$



(a) Cut θ_b is cheaper than or of the same weight as cut θ'_b



(b) Cut θ_d is cheaper than or of the same weight as cut θ'_d

Figure 4.6: Specially defined cuts compared by Lemma 19.

regarding the subtree N_b*, and analogously, regarding the subtree* N_d*, it holds that*

$$\begin{aligned} c_\ominus(\theta_d) &:= c_\ominus(N(d) \cup \Uparrow,\ N(b) \cup \sharp) \\ &\leq c_\ominus(N(d) \cup \Uparrow_A,\ N(b) \cup \sharp \cup \Uparrow_B) =: c_\ominus(\theta'_d). \end{aligned}$$

Proof. We prove Lemma 19 regarding the subtree N_b by contradiction. The proof regarding the subtree N_d is symmetric. We show that the cut $\theta := (\Uparrow_A, N(b) \cup N(d) \cup \sharp \cup \Uparrow_B)$, which differs from θ'_b in the set $N(b)$, would be cheaper in G than the edge-induced minimum u-v-cut $\theta_{\min} := (\Uparrow,\ N(b) \cup N(d) \cup \sharp)$ in G, which differs from θ_b in the set $N(b)$, if θ'_b was cheaper than θ_b.

So we assume that $c_\ominus(\theta_b) > c_\ominus(\theta'_b)$. As the cuts θ and $\theta_{\min}$ both do not separate the modified vertices b and d, each of them is of the same weight in $G_\ominus(S)$, $G_\ominus$ and G, by Corollary 1 and Lemma 8 (A1). Here we consider the weights in $G_\ominus$ and get

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

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

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

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

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

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

Lemma 20 now considers more general cuts by also allowing the wood $\sharp_e$ of edge e to be split (see Figure 4.7). Note, that the cuts considered in this lemma still are not required to be minimum separating cuts, although they both separate u and v. For a better readability we again omit the indices regarding the edge e in the formulas.

Lemma: Comparing more general cuts in $G_\ominus(S)$ by cutting the wood as well

Lemma 20 *In the situation of Lemma 19 we additionally consider an arbitrary cut* $(\sharp_A, \sharp_B)$ *of the wood* $\sharp_e$, *with* $u \in \sharp_B$. *Then in* $G_\ominus(S)$, *and* $G_\ominus$ *respectively, it holds that*

$$\begin{aligned} c_\ominus(\theta_{bb}) :=\; & c_\ominus(N(b) \cup \Uparrow \cup \sharp_A,\ N(d) \cup \sharp_B) \\ \leq\; & c_\ominus(N(b) \cup \Uparrow_A \cup \sharp_A,\ N(d) \cup \sharp_B \cup \Uparrow_B) =: c_\ominus(\theta'_{bb}) \end{aligned}$$

regarding the subtree N_b, *and analogously, regarding the subtree* N_d, *it holds that*

$$\begin{aligned} c_\ominus(\theta_{dd}) :=\; & c_\ominus(N(d) \cup \Uparrow \cup \sharp_A,\ N(b) \cup \sharp_B) \\ \leq\; & c_\ominus(N(d) \cup \Uparrow_A \cup \sharp_A,\ N(b) \cup \sharp_B \cup \Uparrow_B) =: c_\ominus(\theta'_{dd}). \end{aligned}$$

Proof. Again we prove Lemma 20 regarding the subtree N_b. The proof regarding the subtree N_d is symmetric. The assertion of Lemma 20 follows by Lemma 19. We express the cuts θ_{bb} and θ'_{bb} with the aid of the cuts θ_b and θ'_b considered in Lemma 19, which just differ in the set $\sharp_A$. So we get

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

So with $c_\ominus(\theta_b) \leq c_\ominus(\theta'_b)$, by Lemma 19, we finally get

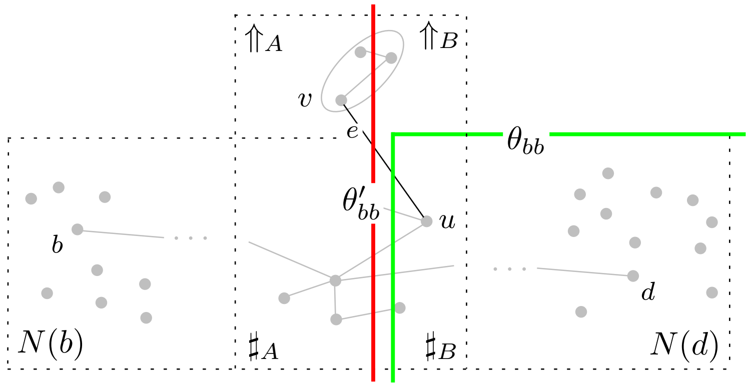
$$\begin{aligned} c_\ominus(\theta'_{bb}) - c_\ominus(\theta_{bb}) = \; & [c_\ominus(\theta'_b) - c_\ominus(\theta_b)] \\ & - [c_\ominus(\sharp_A, N(b) \cup \Uparrow_A) - c_\ominus(\sharp_A, N(b) \cup \Uparrow)] \\ & + [c_\ominus(\sharp_A, N(d) \cup \sharp_B \cup \Uparrow_B) - c_\ominus(\sharp_A, N(d)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

□

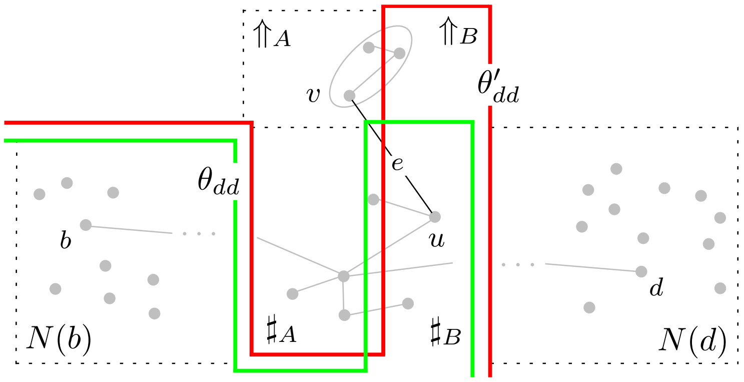
At this point we can finally deduce the following main fact of this subsection from Lemma 20. Remember that we still assume the graph $G_\ominus(S)$ to be the contracted graph in the first iteration of a Gomory-Hu execution in Algorithm 10, i.e., the considered split node S is also a node in the initial intermediate min-cut tree $T_\circ(G)$ resulting by contracting all edges not lying on the path γ in the entire min-cut tree $T(G)$ of G.

Fact: Adjustable cuts

Fact 3 *In the case of the deletion of an edge* $e_\ominus = \{b, d\}$ *in graph* G, *consider* $G_\ominus(S)$ *as described above and let* $e = \{u, v\}$ *denote an edge of the previous complete min-cut tree* $T(G)$ *lying in* S. *Let further* $\theta' := (U, V(S) \setminus U)$ *denote a cut in* $G_\ominus(S)$ *that separates* b *and* d *as well as* u *and* v *and additionally splits the treetop* $\Uparrow_e$, *with* $v \in \Uparrow_e \cap (V(S) \setminus U)$. *Then the cut* $\theta := (U \setminus \Uparrow_e, (V(S) \setminus U) \cup \Uparrow_e)$



(a) Cut θ_{bb} is cheaper than or of the same weight as cut θ'_{bb}



(b) Cut θ_{dd} is chaper than or of the same weight as cut θ'_{dd}

Figure 4.7: Specially defined cuts compared by Lemma 20.

(F1) *does not split the treetop* $\Uparrow_e$ *of edge* e,

(F2) *but splits the remaining set* $V \setminus \Uparrow_e$ *(concerning graph* $G_\ominus$*) the same way as cut* θ' *does and*

(F3) *is at most as expensive as cut* θ' *in* $G_\ominus(S)$*, and graph* $G_\ominus$ *respectively, and also separates* u *and* v.

Implied vertex property

Fact 3 does not consider minimum separating cuts in $G_\ominus(S)$, but just any cuts separating the modified vertices b and d as well as an edge $e = \{u, v\}$ not lying on path γ in $T(G)$. So Fact 3 can be regarded as a property of the designated vertex v which allows to adjust each cut that separates $e = \{u, v\}$ and the modified vertices b and d such that it does not split the treetop $\Uparrow_e$ of $e = \{u, v\}$. As each minimum u-v-cut in $G_\ominus(S)$ that is cheaper than the previous minimum u-v-cut induced by edge $e = \{u, v\}$ in G needs to separate the modified vertices b and d by Lemma 8 (A1), Fact 3 particularly holds for such minimum u-v-cuts in $G'_\ominus(S)$.

Second Aspect

The second aspect considered in this subsection explores specially shaped minimum separating cuts in the following more general situation. Let $H = (V, E, c())$ denote an undirected, weighted graph and $\{r, v_1, \ldots, v_z\}$ a set of designated vertices in H. We call the vertex r the *center*, the remaining vertices $v_1, \ldots, v_z$ are

Definition: Partition-property

called *cut-vertices* regarding the center r. Furthermore, let $V_{\subset}$ denote a subset of V containing all cut-vertices, but not containing the center r. For $V_{\subset}$ a partitioning $\Pi := \{P_1, \ldots, P_z\}$ is defined such that each vertex v_j lies in P_j. Now we assume for each cut-vertex v_j the property that for each v_j-r-cut $\theta'_j := (R_j, V \setminus R_j)$ in H, with $r \in R_j$, the cut $\theta_j := (R_j \setminus P_j, (V \setminus R_j) \cup P_j)$ is of at most the same weight in H. Note, that θ_j does not split P_j, but splits $V \setminus P_j$ the same way as θ'_j does and separates v_j and r (compare to Fact 3). We call this property *partition-property.*

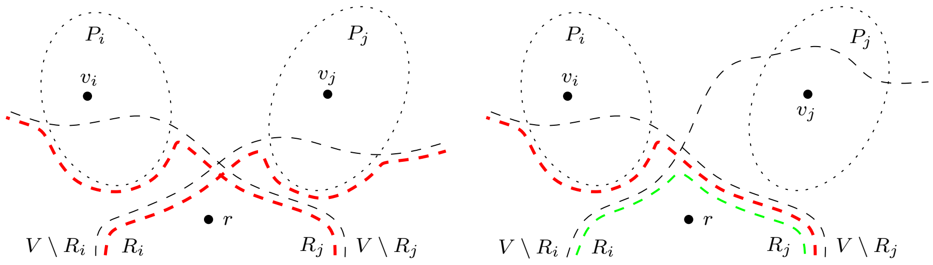
Three different cases

Now consider an arbitrary minimum v_i-r-cut $\theta'_i := (R_i, V \setminus R_i)$, with $r \in R_i$, that does not split P_i and an arbitrary minimum v_j-r-cut $\theta'_j := (R_j, V \setminus R_j)$, with $r \in R_j$, that does not split P_j, with $v_i \neq v_j$ (such cuts exist by the partition-property assumed for v_i and v_j). We distinguish the following three cases (compare to Figure 4.8) concerning the positions of θ'_i and θ'_j:

1) Cut θ'_i separates v_j and r, and θ'_j separates v_i and r.

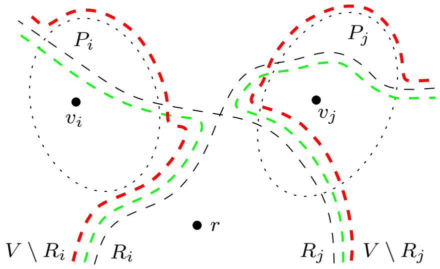
2) Cut θ'_i does not separate v_j and r, but cut θ'_j separates v_i and r.

3) Cut θ'_i does not separate v_j and r, and cut θ'_j does not separate v_i and r.



(a) Positions of case 1

(b) Positions of case 2



(c) Positions of case 3

Figure 4.8: Three different cases concerning the positions of θ'_i and θ'_j.

The analysis of these cases yield the following:

Analyzing case 1 (see Figure 4.8a) As cut θ'_i separates v_j and r, and as v_j satisfies the partition-property, the cut $\theta_i := (R_i \setminus P_j, (V \setminus R_i) \cup P_j)$ (red dashed) is a minimum v_i-r-cut, which does not split the union $P_i \cup P_j$. Analogously, the cut $\theta_j := (R_j \setminus P_i, (V \setminus R_j) \cup P_i)$ (red dashed) is a minimum v_j-r-cut, which does not split the union $P_i \cup P_j$.

Analyzing case 2 (see Figure 4.8b) As cut θ'_j separates v_i and r and v_i satisfies the partition-property, the cut $\theta_j := (R_j \setminus P_i, (V \setminus R_j) \cup P_i)$ (red dashed) is a minimum v_j-r-cut, which does not split the union $P_i \cup P_j$.

Furthermore, by Lemma 15 the cut $\theta_{\text{new(j)}} := (R_i \cap R_j, (V \setminus R_i) \cup (V \setminus R_j))$ (green dashed) is a minimum v_j-r-cut, which does not split the union $P_i \cup P_j$. This corresponds to the following fact: If we regard cut θ'_i as the first split cut of a Gomory-Hu execution, we get an intermediate min-cut tree consisting of two nodes R_i and $V \setminus R_i$ connected by an edge. If now $\{v_j, r\}$ is chosen as the next step pair in R_i, cut θ'_j induces cut $\theta_{\text{new(j)}}$ as split cut in $H(R_i)$, by Theorem 2 and 3. By Lemma 12 the previous split cut θ'_i turns out to be also a minimum v_i-v_j-cut, as $\theta_{\text{new(j)}}$ separates v_i and r.

Analyzing case 3 (see Figure 4.8c) By Lemma 15 the cut $\theta_{\text{new(i)}} := ((V \setminus R_j) \cup R_i, (V \setminus R_i) \cap R_j)$ (green dashed) is a minimum v_i-r-cut, and the cut $\theta_{\text{new(j)}} := ((V \setminus R_i) \cup R_j, (V \setminus R_j) \cap R_i)$ (green dashed) is a minimum v_j-r-cut. Both cuts $\theta_{\text{new(i)}}$ and $\theta_{\text{new(j)}}$ do not cross. So as v_i and v_j both satisfy the partition-property, cut $\theta_i := (((V \setminus R_j) \cup R_i) \setminus P_i, ((V \setminus R_i) \cap R_j) \cup P_i)$ and $\theta_j := (((V \setminus R_i) \cup R_j) \setminus P_j, ((V \setminus R_j) \cap R_i) \cup P_j)$ (both red dashed) are non-crossing minimum separating cuts, which do neither split partition P_i nor P_j.

4.3.3 Improving the Algorithms

Now we combine the two aspects previously explored in Subsection 4.3.2 and several Lemmas stated in Subsection 4.3.1 to a better algorithmic solution for the deletion of an edge. The case of adding an edge can also get slightly improved with the aid of Lemma 16. However, at first we concentrate on the edge deletion. Consider Algorithm 11, which is actually the same as the previous Algorithm 10, but calls a different method to solve the central problem in Line 19 and stores the minimum weight of the edges on path γ incident with the current representative in $w(\min)$ (Line 18) for a later improvement according to Lemma 18. Furthermore, some initialization takes place in Line 11 to Line 15 which will be explained later.

Idea of improved approach

The idea of our improved algorithmic approach in the case of edge deletion is to construct split cuts which preserve known partitions during the Gomory-Hu execution, according to the fact that some vertices in $G_\ominus(S)$ meet the partition-property introduced in the second aspect in Subsection 4.3.2. The considered partitions will turn out to correspond to treetops, which sometimes may remain unchanged by the edge deletion in $G_\ominus$, by Corollary 8. So the aim of this improved appraoch is to find such remaining treetops as soon as possible and to use each newly calculated minimum separating cut as split cut in a Gomory-Hu execution.

Handling remaining treetops

As soon as a remaining treetop, or partition, is found (see if-clause in Line 11 in Central-TreeDel (Algorithm 12)), it gets marked as “already known” in Line 16. When there are no unmarked nodes, apart from singletons, left in the intermediate min-cut tree $T_\star(G_\ominus)$, the marked nodes can easily get unfolded according to the Gomory-Hu method (see Line 21, Algorithm 11), as all split cuts are already known by the previous min-cut tree $T(G)$.

In the following we illustrate how the modified graph $G_\ominus$, concerning $G_\ominus(S)$ for a node S in the initial intermediate min-cut tree $T_\circ(G)$, corresponds to graph H described in the second aspect in Subsection 4.3.2. Afterwards we exemplarily apply Central-TreeDel (Algorithm 12) to a node in the initial intermediate min-cut tree $T_\circ(G)$ to show that the nodes in the resulting intermediate min-cut tree $T_\star(G_\ominus)$ again induce a correspondence between $G_\ominus$ and H, which allows us to apply Central-TreeDel iteratively.

Algorithm 11: TREE-EDGEDEL-3

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, reduced graph
       $G_\ominus = (V, E \setminus \{\{b, d\}\}, c_\ominus())$, modified edge $\{b, d\}$ with weight $\Delta$
Output: Min-cut tree $T(G_\ominus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
if $\gamma = \{b, d\}$ and $c_T(\{b, d\}) = \Delta$ then
    Remove edge $\{b, d\}$ from $T(G)$
    return resulting components $T(G)|_{G_b}$ and $T(G)|_{G_d}$
else
    if path $\gamma$ spans $V$ then
        return path $\gamma$
    else
        Calculate tree $T_\circ(G)$ by contracting all edges not lying on path $\gamma$
        Calculate representatives of nodes in $T_\circ(G)$
        %as described in Definition 11
        %---------- initialization ----------
        forall vertices $v$ in $V$ do
            $L(v) \leftarrow \emptyset$
            $l(v) \leftarrow \emptyset$
            $P_v \leftarrow \emptyset$
            $D(v) \leftarrow \emptyset$
        %-----------------------------------
        $T_\star(G_\ominus) \leftarrow T_\circ(G)$
        forall unmarked nodes $S$ in $T_\star(G_\ominus)$ with $|S| > 1$ do
            $w(\min) \leftarrow$ minimum weight of the two edges on $\gamma$
            incident with $r(S)$
            $T_\star(G_\ominus) \leftarrow$CENTRAL-TREEDEL $(G_\ominus, T(G), T_\star(G_\ominus), S, w(\min))$
        forall marked nodes $S$ in $T_\star(G_\ominus)$ with $|S| > 1$ do
            Unfold $S$
        $T(G_\ominus) \leftarrow T_\star(G_\ominus)$ with singletons replaced by vertices
        return $T(G_\ominus)$
```

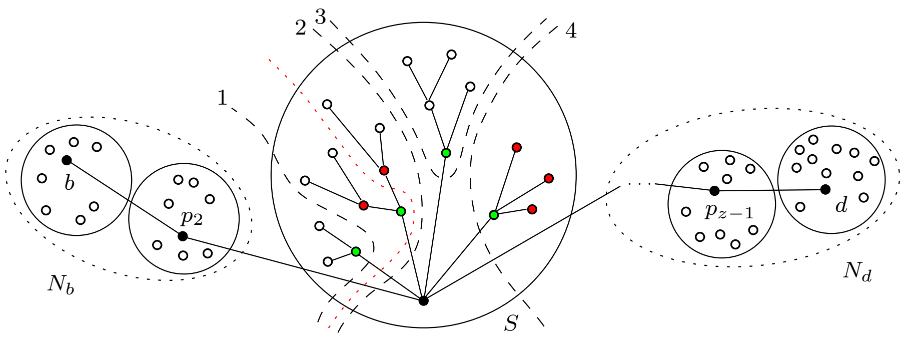
Correspondence between graph $G_\ominus$ and graph H

Detecting second aspect in $G_\ominus$

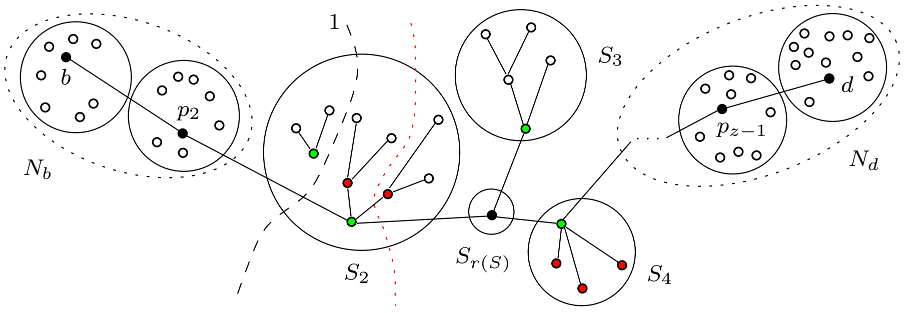
The modified graph $G_\ominus$, concerning $G_\ominus(S)$ for a node S in the initial intermediate min-cut tree $T_\circ(G)$, corresponds to the graph H described in the second aspect in Subsection 4.3.2 as follows: Regard the representative $r(S)$ of node S, as given in Definition 11, as the designated center r (see black vertex in S in Figure 4.9a), and such vertices in S which are adjacent to $r(S)$ in $T(G)$ as cut-vertices $v_1, \dots, v_z$ (see green vertices in Figure 4.9a). The treetops of the edges $\{v_i, r(S)\}$, $i = 1, \dots, z$, correspond to the partitions $P_1, \dots, P_z$.

Detecting first aspect in $G_\ominus$

Furthermore, by Fact 3 each cut-vertex v meets the partition-property regarding any v-r-cut that separates the modified vertices b and d. So a cut-vertex v particularly meets the partition-property for minimum v-r-cuts in $G_\ominus$ that are cheaper than a previous minimum v-r-cut in G, as such minimum v-r-cuts always separate the modified vertices b and d, by Lemma 9. Therefore, each newly calculated minimum v-r-cut in $G_\ominus$ that is cheaper than the previous one in G can get adjusted, before it may be used as split cut in Line 35 (Algorithm 12), such that it does not split the treetop, i.e., the partition, related to the cut-vertex v. This happens in Line 30, Algorithm 12. In this situation the newly occurring node S_v after split-



(a) Situation at the first call of Central-TreeDel



(b) Situation after splitting S by the non-crossing cuts $2, 3$ and 4 shown above

Figure 4.9: Exemplary application of Central-TreeDel

ting S is not marked as "already known", and so its representative, which is vertex v, becomes a center in a later iteration. A newly calculated minimum v-r-cut that is not cheaper in $G_\ominus$ induces a remaining treetop by Corollary 8, and therefore, is replaced by the previous edge-induced cut, which is used as split cut instead (see if-clause in Line 11 to Line 20, Algorithm 12). In this case the newly occurring node S_v after splitting S is marked as "already known" and so never considered by Central-TreeDel (Algorithm 12).

Additional information

So, as a cut-vertex v becomes either a new center or a part of a remaining treetop, we additionally store some information regarding v. This information is initialized for all vertices in V before the first call of Central-TreeDel (see Algorithm 11, Line 11 to Line 15) and is updated for the current representative $r(S)$ at the beginning of Central-TreeDel (Algorithm 12). In the case that v becomes a center we need information I(1) and I(2), in case that v induces a remaining treetop we store I(3) and I(4).

I(1) list $L(v)$ of cut-vertices regarding v as center r

I(2) list $l(v)$ of related minimum $\bar{v}$-v-cuts $\theta'_{\bar{v}}$ in $G_\ominus$, $\bar{v} \in L(v)$

I(3) partition P_v defined as the treetop of $e = \{v, r\}$ in $T(G)$

I(4) set $D(v)$ of cut-vertices separated from center r by the minimum v-r-cut θ'_v in $l(r)$

For the current center $r(S)$ the list $L(r(S))$ of cut-vertices changes during the execution of Central-TreeDel (Algorithm 12). In the end it contains several cut-

Algorithm 12: Central-TreeDel

```
Input: Graph $G_\ominus$, complete min-cut tree $T(G)$, current intermediate
       min-cut tree $T_\star(G_\ominus)$ with representatives of nodes, current node $S$,
       minimum edge weight $w(\min)$ on $\gamma$
Output: In parts unfolded intermediate min-cut tree $T_\star(G_\ominus)$
%Side-effect:  Updating information I(1), I(2) and I(4)
Add all vertices in $S$ that are adjacent to $r(S)$ regarding $T(G)$ to $L(r(S))$
forall vertices $v$ in $L(r(S))$ do
    $P_v \leftarrow \Uparrow_{\{v,r(S)\}}$
while $L(r(S))$ has next element $v$ do
    if $l(r(S))$ does not yet contain related cut $\theta_v$ then
        if $c_T(\{v, r(S)\}) < w(min) - \Delta$ then
            $\theta_v \leftarrow \theta_{\min}$ induced by edge $\{v, r(S)\}$    %by Lemma 18
        else
            $\theta_v \leftarrow$ GoldbergTarjan $(v, r(S), G_\ominus)$
        Add $\theta_v$ to list $l(r(S))$    %in a valid relation to $v$
    if $c_\ominus(\theta_v) = c(\theta_{min})$ then
        %$\theta_{\min}$ denotes previous edge-induced minimum $v$-$r(S)$-cut
        Delete $v$ from list $L(r(S))$
        Delete $\theta_v$ from list $l(r(S))$
        %---Split partition $P_v$ (treetop) from $S$
        $T_\star(G_\ominus) \leftarrow$ SplitAndReconnect $(T_\star(G_\ominus), S, v, \theta_{\min})$
        $S \leftarrow S_{r(S)}$
        Mark $S_v$ as "already known"
        if $|S| > 1$ then
            return Central-TreeDel $(G_\ominus, T(G), T_\star(G_\ominus), S, w(\min))$
        else
            return $T_\star(G_\ominus)$
    else
        while $L(r(S))$ has next element $\bar{v}$ do
            if $\theta_v$ separates $\bar{v}$ and $r(S)$ then
                Delete $\bar{v}$ from $L(r(S))$
                if $l(r(S))$ already contains related cut $\theta_{\bar{v}}$ then
                    Add $\bar{v}$ to set $D(v)$
                    Add $\bar{v}$ to list $L(v)$
                    Move related cut $\theta_{\bar{v}}$ from list $l(r(S))$ to list $l(v)$
%---Now $L(r(S))$ has at most two elements
while $L(r(S))$ has next element $v$ do
    %---Calculate partition-preserving cut for $v$
    %$\theta_v = (R, V \setminus R)$, $r(S) \in R$, denotes cut in $l(r(S))$ related to $v$
    $\theta_v \leftarrow (R \setminus P_v, (V \setminus R) \cup P_v)$    %by partition-property
    forall vertices $\bar{v}$ in $D(v)$ do
        $\theta_v \leftarrow (R \setminus P_{\bar{v}}, (V \setminus R) \cup P_{\bar{v}})$   %by case 1 and case 2
    forall vertices $\bar{v} \neq v$ in $L(r(S))$ do
        $\theta_v \leftarrow (R \cup P_{\bar{v}}, (V \setminus R) \setminus P_{\bar{v}})$   %by case 3
    $T_\star(G_\ominus) \leftarrow$ SplitAndReconnect $(T_\star(G_\ominus), S, v, \theta_v)$
    $S \leftarrow S_{r(S)}$
return $T_\star(G_\ominus)$
```

Introducing set M of split cuts

vertices v related to a set M of non-crossing minimum v-r-cuts in $G_\ominus$ such that none of the minimum v-r-cuts splits its related partition P_v, none of the minimum v-r-cuts separates another cut-vertex $\bar{v}$ (whose minimum $\bar{v}$-r-cut is in M) from $r(S)$, and finally the center $r(S)$ gets isolated (compare to the cuts $2, \ldots, 4$ in Figure 4.9a). More precisely, in Line 29 the cut-vertices that induce remaining treetops (compare to cut 3 in Figure 4.9a) are not included in $L(r(S))$ anymore, as the related cuts are already used as split cuts in Line 14, and therefore, the vertices are deleted from $L(r(S))$ in Line 12. (Note, that one can prove that there are at most two cuts left in $l(r(S))$ in Line 29.)

Construction of split cuts in M

Based on the second aspect in Subsection 4.3.2 such a set M of split cuts always exists, as the two subtrees N_b and N_d of S in $T_\star(G_\ominus)$ are contracted in $G_\ominus(S)$ and the remaining vertices of $G_\ominus$, apart from the center r, are partitioned into treetops (compare to Figure 4.9a). The construction of M works a follows: At the beginning we assume for each cut-vertex $\bar{v}$ one minimum $\bar{v}$-r-cut to be in M (see Line 1). Then consider a fixed cut-vertex v (see Line 4) and a minimum v-r-cut $\theta'_v = (R, V \setminus R)$ in $G_\ominus$, with $r \in R$, that is cheaper than the previous one in G (see Line 21). As v meets the partition-property, we can consider cut $\theta_v = (R \setminus P_v, (V \setminus R) \cup P_v)$ instead, which preserves partition P_v (see Line 30). Now for each other cut-vertex $\bar{v}$ still in M compare θ_v to a minimum $\bar{v}$-r-cut $\theta_{\bar{v}}$, which preserves $P_{\bar{v}}$ (see Line 22).

Considering separated cut-vertices

By the analysis of case 1 and case 2 in Subsection 4.3.2, we are allowed to transfer the partition $P_{\bar{v}}$ to the v-side of cut θ_v for each cut-vertex $\bar{v}$ that is separated from r by cut θ_v (see Line 32). As we suppose θ_v to be in M, it follows that the related cuts to these vertices are not in M anymore (see Line 24 to Line 26). However, each cut $\theta_{\bar{v}}$ whose cut-vertex $\bar{v}$ is separated from r by cut θ_v and that does not separate v from r vice versa, by the analysis of case 2, also constitutes a minimum $\bar{v}$-v-cut in $G_\ominus$ which still can be used as later split cut (see Line 27 and Line 28).

Considering cut-vertices not separated

Considering the cut-vertices $\bar{v}$ in M that are not separated from the center r by θ_v, we distinguish two situations. If there is any minimum $\bar{v}$-r-cut $\theta'_{\bar{v}}$ that separates v from r, it follows that θ_v is not included in M, as we suppose $\theta'_{\bar{v}}$ to be in M (see Line 24). Otherwise, by the analysis of case 3 in Subsection 4.3.2, we are allowed to transfer the partition $P_{\bar{v}}$ to the r-side of cut θ_v for each cut-vertex $\bar{v}$ that is not separated from r by cut θ_v (see Line 28).

Algorithm 13: SplitAndReconnect

```
Input: Intermediate min-cut tree $T_\star(G_\ominus)$ with representatives of nodes,
       designated split node $S$ in $T_\star(G_\ominus)$, second vertex $v \neq r(S)$ of step
       pair, split cut $(U, V \setminus U)$ in $G_\ominus$ with $r(S) \in U$
Output: Intermediate min-cut tree $T_\star(G_\ominus)$ with $S$ split
Split $S$ into $S_{r(S)} = S \cap U$ and $S_v = S \cap (V \setminus U)$
$r(S_{r(S)}) \leftarrow r(S)$
$r(S_v) \leftarrow v$
Create new edge $\{S_{r(S)}, S_v\}$ in $T_\star(G_\ominus)$
forall edges $e_j = \{S, \bar{S}_j\}$ previously incident with $S$ in $T_\star(G_\ominus)$ do
    if $r(\bar{S}_j) \in U$ then
        Reconnect subtree $N_j$ with $S_{r(S)}$ in $T_\star(G_\ominus)$
    else
        Reconnect subtree $N_j$ with $S_v$ in $T_\star(G_\ominus)$
return $T_\star(G_\ominus)$
```

Exemplary Application of Central-TreeDel

Now we exemplarily apply Central-TreeDel (Algorithm 12) to a node in the initial intermediate min-cut tree $T_\circ(G)$ to show that the nodes in the resulting intermediate min-cut tree $T_\star(G_\ominus)$ again induce a correspondence between $G_\ominus$ and H, which allows us to apply Central-TreeDel iteratively. Figure 4.9a shows an initial intermediate min-cut tree $T_\circ(G)$ with a designated node S. Additionally we know the structure given by the previous entire min-cut tree $T(G)$ for the vertices in S. So this is the situation before the first call of Central-TreeDel.

At the beginning all nodes in $T_\circ(G)$ are unmarked and in our example the node S includes more than one vertex. So in Line 19 of Algorithm 11 Central-TreeDel is applied to S, with its representative $r(S)$ shown as black vertex in Figure 4.9a. According to Line 1 in Algorithm 12 the list $L(r(S))$ contains all green vertices in Figure 4.9a. We assume these vertices to be labeled with $v_1, \dots, v_4$ from left to right. The list $l(r(S))$ of related cuts is empty or filled with dummy-cuts, depending on the implementation (It just needs to relate each cut to the right cut-vertex in $L(r(S))$.

Initialization

Now the set M induced by the cut-vertices in $L(r(S))$ is calculated. The first while-loop in Line 4, Algorithm 12, starts with $v = v_1$. As we do not know yet a minimum v_1-$r(S)$-cut in $G_\ominus$, but there exists one that is cheaper than the previous minimum v_1-$r(S)$-cut in G, we calculate a new minimum v_1-$r(S)$-cut θ'_1 in Line 9 (for example with the preflow-push method by Goldberg and Tarjan [GT88]). As cut θ'_1 does not separate any other cut-vertices from $r(S)$, set $D(v_3)$ remains empty in the while-loop in Line 22 all cut-vertices remain in $L(r(S))$. The cut 1 shown in Figure 4.9a that separates v_1 from $r(S)$ is already adjusted. The original cut θ'_1 may cross several partitions, i.e., treetops, of the other (green) cut-vertices. However, we will see in the following that cut θ'_1, and therefore, cut 1 shown in Figure 4.9a is also a minimum v_1-v_2-cut in $G_\ominus$.

First while-loop for vertex v_1

To this end consider the second while-loop in Line 4. This iteration regards $v = v_2$. In Line 9 we get a new minimum v_2-$r(S)$-cut θ'_2 in $G_\ominus$, which we add to list $l(r(S))$, as it is not of the same weight as the previous minimum v_2-$r(S)$-cut $\theta_{\min}$ in G. As cut θ'_2 separates cut-vertex v_1 from $r(S)$, but θ'_1 did not separate v_2 from $r(V)$ before, by the analysis of case 2 in Subsection 4.3.2 we see that θ'_1 is also a minimum v_1-v_2-cut in $G_\ominus$. By using later the adjusted cut 2 shown in Figure 4.9a as split cut, θ'_1 induces the minimum v_1-v_2-cut 1 shown in Figure 4.9b, which is the same as shown in Figure 4.9a. So it is feasible to add v_1 to set $D(v_2)$ in the while-loop in Line 22, at the same time to deleted it from $L(r(S))$, and to add it instead to $L(v_2)$ in case v_2 becomes a center in a later iteration (see node S_2 in Figure 4.9b).

Second while-loop for vertex v_2

Now we consider the third while-loop in Line 4. This iteration regards $v = v_3$. As do not know yet a minimum v_3-$r(S)$-cut in $G_\ominus$, but the new cut is supposed to have the same weight as the previous one in G, we either calculate a new minimum v_3-$r(S)$-cut θ'_3 in Line 9 or we notice that there exists no cheaper cut in $G_\ominus$ by the condition in Line 6. In this case we consider the previous edge-induced cut as related cut to v_3 in $l(r(S))$. Then the if-clause in Line 11 becomes true, and by Corollary 8 the minimum separating cuts for all pairs of vertices in partition P_3 remain the same, i.e., are already given by the previous min-cut tree $T(G)$. So it is feasible to delete v_3 from $L(r(S))$ in Line 12 (analogously θ'_3 is deleted from $l(r(S))$) and to use the previous minimum v_3-$r(S)$-cut $\theta_{\min}$ as split cut instead in Line 14. As the minimum separating cuts in the newly occurring node S_3 in Figure 4.9b are already known, we accordingly mark this node in Line 16. The rest after splitting node S

Third while-loop for vertex v_3

is again considered as the current node S in Line 15. Note, that the additional information stored in several lists and sets remains unchanged.

Last while-loop for vertex v_4

The last while-loop in Line 4 regards $v = v_4$ whose related minimum v_4-$r(S)$-cut θ'_4 is again cheaper in $G_\ominus$. As θ'_4 does not separate any previous cut-vertex from $r(S)$ we finally get the adjusted cut 4 as shown in Figure 4.9a as split cut. In this example the list $l(r(S))$ in Line 29 finally contains cut 2 and cut 4. Cut 1 was deleted, as it is "absorbed" by cut 2, and cut 3 preserves a treetop, and therefore, was unfolded immediately as a new subtree of S.

Iterating Central-TreeDel is feasible

After applying the remaining split cuts in $l(r(S))$ in an adjusted form, Figure 4.9b shows the resulting intermediate min-cut tree $T_\star(G_\ominus)$. Now node S_3 is marked as "already known", node $S_{r(S)}$ became a singleton only containing the isolated center $r(S)$, and to node S_2 and S_4 Central-TreeDel will be applied in the next iterations. This is feasible, as for example in node S_2 again the representative v_2 can be regarded as the new center with its related treetop also lying in S_2, which is guaranteed by the fact that the previous split cuts preserve their related partitions. New cut-vertices are all previous cut-vertices which were separated from the previous center by the split cut θ_2 that induces S_2 (they are already stored in $L(v_2)$) and again all vertices adjacent to the center v_2 in S_2 (red vertices in Figure 4.9b). To regard the vertex v_1, which was already added to $L(v_2)$ by the previous iteration, as cut-vertex of the new center v_2 is feasible, as we already know a minimum v_1-v_2-cut, and the related partition P_1 is also preserved by the construction of θ_2. Furthermore, each so defined cut-vertex v for the new center v_2 again meets the partition-property by Fact 3 regarding any v-v_2-cut that separates the modified vertices b and d (see for example the red dotted cut in Figure 4.9b and Figure 4.9a). Finally also a new set M of split cuts (as introduced before) can be constructed, as, concerning S_2, both subtrees again are contracted in $G_\ominus(S_2)$ and the remaining vertices of $G_\ominus$, apart from the center v_2, are again partitioned into treetops (compare to Figure 4.9b).

Summary for Improved Edge Deletion Algorithm

Points of improvement

The improvement in our new algorithm Tree-EdgeDel-3 (Algorithm 11) bases on the fact that this algorithm reduces the number of calculations of minimum u-v-cuts in $G_\ominus$ in a high degree, compared to Tree-EdgeDel-2 (Algorithm 10), which just unfolds the independent nodes of the initial intermediate min-cut tree $T_\circ(G)$ by randomly choosing valid step pairs for the Gomory-Hu execution in Central-Tree (Algorithm 8). Our new algorithm knows a condition given by Lemma 18 which allows to decide in some special cases, before calculating a new cut, that a step pair has no cheaper minimum separating cut in $G_\ominus$. So in these cases the algorithm avoids a needless minimum u-v-cut calculation. Furthermore, if there are any remaining treetops, mutually disjoint, in the previous min-cut tree $T(G)$, our improved algorithm finds them by recalculating only one minimum u-v-cut per treetop.

Best and worst case effort

Finally, as the path γ between the two modified vertices b and d consists of at least one edge, we always know at least one minimum separating cut in $G_\ominus$. This is, in the worst case, when path γ consists of only one edge and the new min-cut tree $T(G_\ominus)$ develops to a path from b to d spanning all vertices in V, we still save one minimum u-v-cut calculation compared to a recalculation of the whole min-cut tree, which would take $n-1$ cut calculations. In the best case, when path γ spans all vertices in V, there is no calculation of any minimum u-v-cut necessary at all. In case that the previous min-cut tree $T(G)$ is also a valid min-cut tree $T(G_\ominus)$ for $G_\ominus$, we need to calculate at most as many new minimum u-v-cuts as there are subtrees remaining after deleting path γ from $T(G)$.

Improved Edge Addition Algorithm

Points of improvement

For the sake of completeness we also review algorithm TREE-EDGEADD-2 (Algorithm 9), which considers the case of edge addition, slightly improved with the aid of Lemma 16. Lemma 16 just says that the minimum b-d-cut induced by the edge of minimum weight on path γ in $T(G)$ also constitutes a minimum b-d-cut in graph $G_\oplus$. So we transform Algorithm 9 by changing Line 4 and Line 6 into Algorithm 14.

Algorithm 14: TREE-EDGEADD-3

```
Input: Min-cut tree $T(G)$ of $G = (V, E, c())$, enlarged graph
       $G_\oplus = (V, E \cup \{\{b, d\}\}, c_\oplus())$ with modified vertices $b$ and $d$
Output: Min-cut tree $T(G_\oplus)$
Calculate path $\gamma$ from $b$ to $d$ in $T(G)$
$e_{\min} \leftarrow$ edge of minimum weight lying on $\gamma$
if path $\gamma$ spans $V$ then
    return GOMORY-HU ($G_\oplus$), with first split cut $e_{\min}$ already known
else
    Calculate tree $T_\circ(G)$ by contracting all edges lying to the left of $e_{\min}$
      and contracting all edges lying to the right of $e_{\min}$ on path $\gamma$ in $T(G)$
    Calculate representatives of nodes in $T_\circ(G)$
    %according to Definition 10
    $S_1 \leftarrow$ left contraction of $\gamma$
    $S_2 \leftarrow$ right contraction of $\gamma$
    $T_\star(G_\oplus) \leftarrow T_\circ(G)$
    for $i$ = 1,2 do
        $T_\star(G_\oplus) \leftarrow$ CENTRAL-TREE ($G_\oplus, T_\star(G_\oplus), S$)
    $T(G_\oplus) \leftarrow T_\star(G_\oplus)$ with singletons replaced by vertices
    return $T(G_\oplus)$
```

Best and worst case effort

In the worst case, when path γ spans all vertices in V, we again save one cut calculation compared to a recalculation of the whole min-cut tree. In the best case, when path γ consists of only one edge, there is no calculation of any minimum u-v-cut necessary at all. In case that the previous min-cut tree $T(G)$ is also a valid min-cut tree $T(G_\oplus)$ for $G_\oplus$, we need to calculate one minimum u-v-cut less than edges lying on path γ.

Chapter 5

Dynamically Updating Minimum u-v-Cuts

In the previous chapter our approach for dynamically updating min-cut trees concentrated on the minimization of the number of recalculated minimum u-v-cuts. Another idea is to update the minimum u-v-cuts, which are given in the min-cut tree, individually. Unfortunately there is little related work on this subject. Most of the publications we found concentrate on dynamically updating global minimum cuts instead of minimum separating cuts concerning two fixed vertices (see for example Nagamochi [Nag06]). An approach of Kaplan and Shafir [KS08] considers edges of minimum weight along paths in incremental (dynamic) trees, which seems quite similar to the representation of minimum u-v-cuts in a min-cut tree. Unfortunately, due to the special shape of min-cut trees and problems similar to those illustrated later in Subsection 5.1.3, this approach can not be adapted for updating min-cut trees.

However, talking about minimum u-v-cuts means talking about maximum flows from a source u to a target v, according to the Max-Flow-Min-Cut Theorem by Ford and Fulkerson [FF56]. So in Section 5.1 we shortly introduce u-v-flows and an approach of Kohli and Torr [KT07] for dynamically updating minimum u-v-cuts by adjusting residual graphs of given maximum flows.

Furthermore, a maximum u-v-flow not only represents one minimum u-v-cut, but all minimum u-v-cuts in the underlying graph, by Picard and Queyranne [PQ80]. This yields another approach, which does not really update minimum u-v-cuts, but tries to preserve any of the known cuts as long as possible while the underlying graph is repeatedly modified. This idea is presented in Section 5.2.

5.1 Adjusting Residual Graphs

In this section we shortly introduce flows in undirected weighted graphs and review the approach of Kohli and Torr [KT07] of dynamically adjusting residual graphs of given maximum u-v-flows. We will further outline the problems coming with this approach applying for updating min-cut trees.

5.1.1 Flows in Undirected Weighted Graphs

We consider an undirected positively weighted graph $G = (V, E, c())$. As flows are originally defined in directed networks, we identify each edge $e = \{g, h\} \in E$ with two opposedly directed edges $\overrightarrow{e} = (g, h) \in \overrightarrow{E}$ and $\overleftarrow{e} = (h, g) \in \overleftarrow{E}$, both of the same weight as the original edge in G, i.e., $c((g, h)) = c((h, g)) = c(\{g, h\})$. The weight of an edge can also be regarded as capacity. So in the remainder of this section we talk about capacities in the context of flows. A flow in graph G is then defined as follows:

Definition: Flow in an undirected graph

Definition 15 *Let $N(g)$ denote the set of vertices adjacent to vertex g in an undirected graph $G = (V, E, c())$, and let u and v denote two fixed vertices. Then, a u-v-flow f in G from source u to target v is a function $f : \overrightarrow{E} \cup \overleftarrow{E} \to \mathbb{R}$ on the (directed) edges in G that meets the following conditions:*

- $f((g, h)) = -f((h, g))$, $\forall$ *undirected edges* $\{g, h\} \in E$ *(Asymmetry)*
- $f((g, h)) \leq c((g, h))$, $\forall$ *directed edges* $(g, h) \in \overrightarrow{E} \cup \overleftarrow{E}$ *(Capacity)*
- $\sum_{h \in N(g)} f((g, h)) = 0$, $\forall$ *vertices* $g \in V \setminus \{u, v\}$ *(Mass balance)*

Definition: Maximum flow and value of a flow

A flow is a maximum u-v-flow if there exists no other u-v-flow in G that has a higher value. The value $c(f)$ of a u-v-flow f is defined as $c(f) := \sum_{h \in N(u)} f((u, h)) = \sum_{h \in N(v)} f((h, v))$.

Figure 5.1a shows an example of a maximum u-v-flow with value $c(f) = 20$. For a better perceptibility only directed edges with non-negative flow are shown. The edges are labeled with flow values and capacity values in brackets. The bottlenecks consisting of saturated edges $e \in \overrightarrow{E} \cup \overleftarrow{E}$, i.e., $f(e) = c(e)$ (drawn as dashed (red) arrows), represent minimum u-v-cuts in G (We will see this later by a theorem of Picard and Queyranne [PQ80]). Such bottlenecks can be detected more easily in the residual graph $G(f)$, which represents the residual edge capacities left open by the flow f.

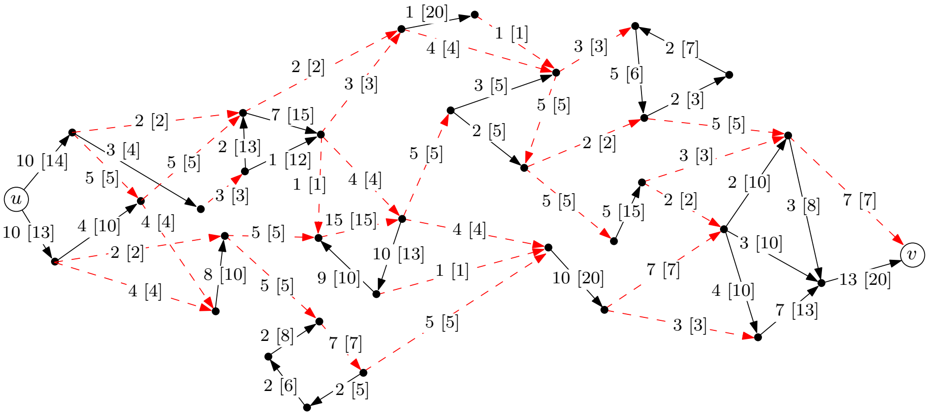
Definition: Residual graph of a flow

Definition 16 *For a flow f in an (undirected) graph $G = (V, E, c())$ the residual graph is given by $G(f) = (V, E(f), r())$ with residual capacity $r(e) := c(e) - f(e)$, $\forall e \in \overrightarrow{E} \cup \overleftarrow{E}$ and $E(f) := \{e \in \overrightarrow{E} \cup \overleftarrow{E} | r(e) > 0\}$*

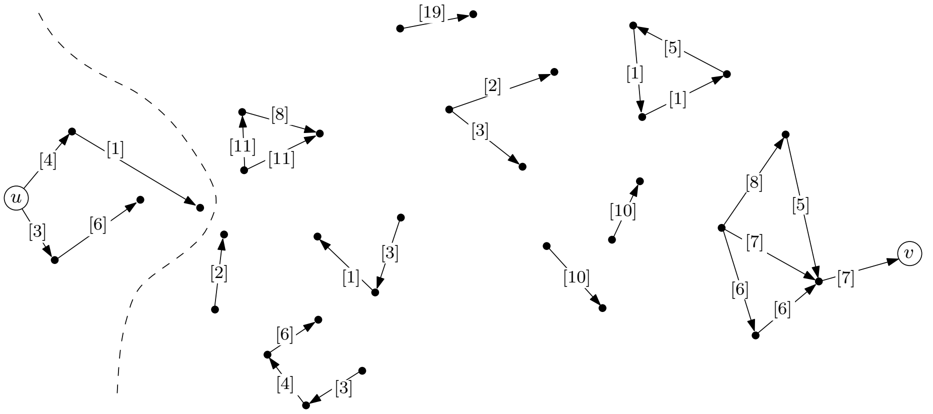
Definition: First minimum u-v-cut

Reversing flows

Figure 5.1b shows the residual graph $G(f)$ of flow f given in Figure 5.1a. Again only directed edges with non-negative flow are shown. The edges are now labeled with residual capacities in brackets. The minimum u-v-cut drawn as a dashed line can be found by a breadth-first search starting at u and visiting all vertices that can be reached over an augmenting path from the source. As this is the first minimum u-v-cut found by such a search, we simply refer to it as the *first* minimum u-v-cut given by a maximum u-v-flow. Note, that the first minimum u-v-cut given by a maximum u-v-flow minimizes the cut side containing the source and does not cross any other minimum u-v-cut in G. In contrast to minimum separating cuts, maximum flows are directed from the source to the target. A maximum u-v-flow in an (undirected) graph can be reversed, i.e., it becomes a v-u-flow, by reversing all directed edges while the flow on each edge remains the same.


1 [20]
1 [1]
4 [4]
3 [3]
2 [7]
2 [2]
3 [3]
5 [6]
3 [5]
5 [5]
2 [3]
2 [2]
7 [15]
2 [5]
5 [5]
2 [2]
5 [5]
10 [14]
3 [4]
2 [13]
1 [12]
5 [5]
3 [3]
2 [10]
5 [5]
1 [1]
4 [4]
3 [3]
3 [8]
7 [7]
u
5 [5]
5 [15]
2 [2]
4 [4]
15 [15]
10 [13]
4 [10]
10 [13]
5 [5]
4 [4]
3 [10]
2 [2]
v
9 [10]
1 [1]
8 [10]
10 [20]
7 [7]
13 [20]
4 [4]
5 [5]
4 [10]
7 [13]
5 [5]
3 [3]
2 [8]
7 [7]
2 [6]
2 [5]

(a) Maximum u-v-flow f in a graph G, with flow weight $c(f) = 20$


[19]
[5]
[2]
[1]
[1]
[8]
[3]
[11]
[11]
[4]
[1]
[8]
u
[5]
[6]
[10]
[3]
[7]
[2]
[1]
[10]
v
[7]
[3]
[6]
[6]
[6]
[4]
[3]

(b) Residual graph $G(f)$ of the maximum flow f shown above

Figure 5.1: Example of a maximum u-v-flow f in a graph G and its residual graph $G(f)$.

5.1.2 The Method of Kohli and Torr

For updating a minimum u-v-cut after an elementary modification of the underlying graph G, Kohli and Torr [KT07] require the knowledge of a maximum u-v-flow f, which represents a minimum u-v-cut in graph G. Their idea is to adjust the residual graph of the known flow f such that neither the capacity constraint nor the mass balance constraint is violated due to the modification. Then a max-flow algorithm regarding the adjusted residual graph is applied to calculate a maximum flow which represents a minimum u-v-cut in the modified graph $G_{\oplus(\ominus)}$. Hence, the method of Kohli and Torr can be split into two phases. The first phase adjusts the residual graph, while the second phase applies a max-flow algorithm and so calculates a new maximum flow, and a new minimum u-v-cut respectively.

Idea of Kohli and Torr

Max-flow algorithms (for solving the max-flow problem) are classified into two categories: Augmenting path and preflow-push algorithms. Augmenting path algorithms repeatedly find augmenting paths in a residual graph $G(f)$ and push the maximum possible flow f' through this path resulting in the totaled flow $(f + f')$ and the residual graph $G(f + f')$. Preflow-push algorithms flood the graph and create excess flow at the vertices. This excess flow is then incrementally drained

About max-flow algorithms

out by sending it from its vertex toward the target or the source. For more detailed information about preflow-push algorithms see [GT88] or [Gol08]. An efficient augmenting path algorithm is introduced in [Din70].

Savings by updating residual graphs

Kohli and Torr [KT07] use an augmenting path algorithm in their approach. The number of augmenting paths found by such an algorithm depends on the value of the total maximum flow pushed from the source to the target. Initializing an augmenting path algorithm with an updated residual graph instead of the modified graph $G_{\oplus(\ominus)}$ saves the recalculation of some augmenting paths, as only the difference between the value of the adjusted flow and the value of a maximum flow in $G_{\oplus(\ominus)}$ needs to be pushed through. Furthermore, Kohli and Torr dynamically adapt a technique introduced by Boykov et al. [BK04], which builds and reuses search trees to find augmenting paths even faster and with less effort. As Boykov et al. claim this technique, based on their experiments, to outperform the best known augmenting path and push-relabel algorithms on graphs commonly used in computer vision, we assume the updating method of Kohli and Torr also to be "good" in general.

Two situations of adjusting residual graphs

As elementary modifications Kohli and Torr not only regard the addition and deletion of an edge, but also the more general increasing and decreasing of edge capacities. According to this view in the first phase of their method they distinguish two situations, as described in the following. Algorithm 15 additionally outlines the phases of the method of Kohli and Torr. For a fixed pair $\{u, v\}$ this algorithm takes a tuple $(G, [f, G_a, G_a(f)])$ consisting of the original graph G, a u-v-flow f in a graph G_a, the graph G_a and the residual graph $G_a(f)$ for the flow f in G_a. The graph G_a is not necessarily equivalent to the original graph G, as it may have additional edges. However, in the initial tuple, before the first updating, it holds that $G = G_a$. The flow f in G_a induces the same first minimum u-v-cut as a maximum u-v-flow in graph G would do. Now assume the capacity of an edge e in graph G, which also exists in graph G_a, to change. The new capacity is denoted by $c'(\overrightarrow{e}) = c'(\overleftarrow{e})$.

First situation

In the first situation the capacity constraint of the previous flow f in G_a is not violated due to the modification, i.e., it holds that $f(\overrightarrow{e}) \leq c'(\overrightarrow{e})$ and $f(\overleftarrow{e}) \leq c'(\overleftarrow{e})$ (see Line 2, Algorithm 15). Therefore, we just set the capacities of the modified edges $\overrightarrow{e}$ and $\overleftarrow{e}$ in graph G_a to $c'(\overrightarrow{e})$. The resulting graph then is denoted by G'_a. The flow f does not get adjusted. We define $f' := f$. At this point f' is a valid u-v-flow in graph G'_a. So the residual graph $G'_a(f')$ is well defined. In the first situation *Phase 1* ends here (see Line 4, Algorithm 15). The only thing still to be done is the calculation of a maximum u-v-flow $f^{\max}$ in graph G'_a, as the flow f' is not necessarily maximal (see Line 29). Kohli and Torr proved that then the flow $f^{\max}$ also represents a minimum u-v-cut in the modified graph $G_{\oplus(\ominus)}$.

Second situation

The second situation considers the violation of the capacity constraint, i.e., in this situation the modification is a weight reduction and it holds that $f(\overrightarrow{e}) > c'(\overrightarrow{e})$ or $f(\overleftarrow{e}) > c'(\overleftarrow{e})$. Then the further steps depend on the position of the modified edges in graph G_a (see Line 7, Algorithm 15).

Incidence with source or target

Assume the modified edge $\overrightarrow{e}$, which is basically the same as the edge $\overleftarrow{e}$, to be incident with the source, i.e., $\overrightarrow{e} = (u, b)$ and $\overleftarrow{e} = (b, u)$ for a vertex $b \in V$ (the case of incidence with the target is symmetric, see Figure 5.2a and the example in the next paragraph). Note, that each flow can be modified such that there is no negative flow leaving the source or entering the target. So in the following we concentrate on edges with non-negative flow. To resolve the violation of the capacity constraint, which we assume in this case, the capacities of the modified edges $\overrightarrow{e} = (u, b)$ and $\overleftarrow{e} = (b, u)$ are both set to the positive flow value $f(\overrightarrow{e}) = f((u, b))$ in graph G_a. Hence there is no need to bypass or backtrack the excess flow on the modified edges. We just define $f' := f$ (see Line 12). Additionally a new pair of opposedly directed edges $\overrightarrow{e}' = (b, v)$ and $\overleftarrow{e}' = (v, b)$ incident with the target is added to graph G_a,

Algorithm 15: ResidualUpdate

```
   Input: Tuple $(G, [f, G_a, G_a(f)])$ with $f$ in $G_a$ representing
          minimum $u$-$v$-cut in $G$, modified edge $e$ with new capacity $c'(e)$
   Output: Tuple $(G_{\oplus(\ominus)}, [f^{\max}, G'_a, G'_a(f^{\max})])$ with $f^{\max}$ in $G'_a$ representing
           minimum $u$-$v$-cut in $G_{\oplus(\ominus)}$
   %---Phase 1
1  $G_{\oplus(\ominus)} \leftarrow$ modification applied to $G$
2  if $f(\overrightarrow{e}) \leq c'(\overrightarrow{e})$ and $f(\overleftarrow{e}) \leq c'(\overleftarrow{e})$ then
3      $G'_a \leftarrow$ set capacities of $\overrightarrow{e}$ and $\overleftarrow{e}$ in $G_a$ to $c'(\overrightarrow{e}) = c'(\overleftarrow{e})$
4      $f' \leftarrow f$
5      Calculate residual graph $G'_a(f')$
6      factor $\leftarrow 0$
7  else
       %---distinguish positions of modified edge
8      if $e$ is incident with source or target then
9          $G'_a \leftarrow$ set capacities of $\overrightarrow{e}$ and $\overleftarrow{e}$ in $G_a$ to $|f(\overrightarrow{e})|$
10         Add shortcuts $\overrightarrow{e}'$ and $\overleftarrow{e}'$ incident with target or source in $G'_a$,
            with capacity zero   %if not existing yet
11         Add $\mu := |f(\overrightarrow{e})| - c'(\overrightarrow{e})$ to capacities of $\overrightarrow{e}'$ and $\overleftarrow{e}'$ in $G'_a$
12         $f' \leftarrow f$
13         Calculate residual graph $G'_a(f')$
14         factor $\leftarrow 1$
15     else
           %---edge $\overrightarrow{e} = (b, d)$ denotes direction with positive flow
16         $G'_a \leftarrow$ set capacities of $\overrightarrow{e}$ and $\overleftarrow{e}$ in $G_a$ to $c'(\overrightarrow{e}) = c'(\overleftarrow{e})$
17         $\mu \leftarrow |f(\overrightarrow{e})| - c'(\overrightarrow{e})$
18         Add shortcuts $\overrightarrow{e_b}'$ and $\overleftarrow{e_b}'$ incident with target in $G'_a$,
            with capacity zero   %if not existing yet
19         if capacity of $\overrightarrow{e_b}' < \mu$ then
20             Process Line 9 to Line 11,
                with $\overrightarrow{e} := \overrightarrow{e_b}'$, $G_a := G'_a$, $|f(\overrightarrow{e})| := \mu$, $c'(\overrightarrow{e}) := 0$
21         Add shortcuts $\overrightarrow{e_d}'$ and $\overleftarrow{e_d}'$ incident with source in $G'_a$,
            with capacity zero   %if not existing yet
22         if capacity of $\overrightarrow{e_d}' < \mu$ then
23             Process Line 9 to Line 11,
                with $\overrightarrow{e} := \overrightarrow{e_d}'$, $G_a := G'_a$, $|f(\overrightarrow{e})| := \mu$, $c'(\overrightarrow{e}) := 0$
24         $f' \leftarrow$ set flow on $\overrightarrow{e}$ to $c'(\overrightarrow{e})$ and flow on $\overleftarrow{e}$ to $-2\,c'(\overleftarrow{e})$
25         $f'(\overrightarrow{e_b}') \leftarrow f'(\overrightarrow{e_b}') + \mu$, $f'(\overleftarrow{e_b}') \leftarrow f'(\overleftarrow{e_b}') - \mu$
26         $f'(\overrightarrow{e_d}') \leftarrow f'(\overrightarrow{e_d}') + \mu$, $f'(\overleftarrow{e_d}') \leftarrow f'(\overleftarrow{e_d}') - \mu$
27         Calculate residual graph $G'_a(f')$
28         factor $\leftarrow 2$
   %---Phase 2
29 $f^{\max} \leftarrow$ FlowAlgo $(u, v)$ regarding residual graph $G'_a(f')$
30 Calculate residual graph $G'_a(f^{\max})$
31 Reduce value of $f^{\max}$ by factor $\cdot\, \mu$
32 return $(G_{\oplus(\ominus)}, [f^{\max}, G'_a, G'_a(f^{\max})])$
```

with an initial capacity of zero (if it does not exist yet). The capacities of these two edges are then adjusted by the addition of the difference $\mu := f(\overrightarrow{e}) - c'(\overrightarrow{e}) = f((u, g)) - c'((u, g))$, which characterizes the violation of the capacity constraint. This increases the according residual capacities in $G'_a(f')$ by μ. So the edges $\overrightarrow{e}'$ and

$\overleftarrow{e}'$ serve as shortcuts connecting vertex b to the target and allow additional flow of value μ to bypass the previous saturated bottlenecks in *Phase 2*. The residual capacity of the modified edge $\overrightarrow{e} = (u, b)$ in the adjusted residual graph $G'_a(f')$ is zero, while edge $\overleftarrow{e} = (b, u)$ has a residual capacity of $2\, f((u, b))$. In *Phase 2* the value of the updated maximum u-v-flow $f^{\max}$ resulting from the application of a max-flow algorithm to the adjusted graph $G'_a(f')$ finally needs to be reduced again by μ.

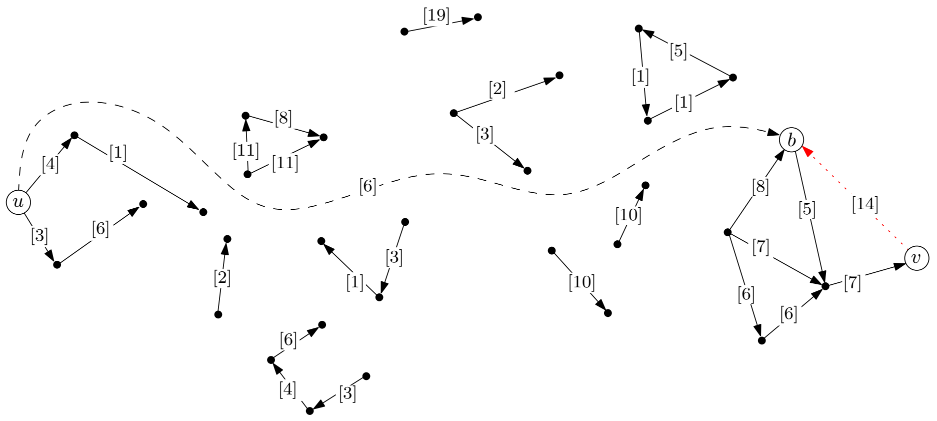
Example for incidence with target

Figure 5.2a shows an example for the modification of an edge $\overrightarrow{e}$ incident with the target v, regarding the flow and residual graph given in Figure 5.1a (we consider the initial case of $G_a := G$). Assume the capacity of edge $\overrightarrow{e} = (b, v)$ to change from $c((b, v)) = 7$ to $c'((b, v)) = 1$. As edge $\overrightarrow{e} = (b, v)$ is already saturated in the residual graph $G_a(f)$ (compare to Figure 5.1a), it remains saturated due to the capacity reduction. Therefore, only the opposed edge $\overleftarrow{e} = (v, b)$ appears in the adjusted residual graph $G'_a(f')$ in Figure 5.2a, drawn as dotted (red) arrow. It has a residual capacity of $2\, f((b, v)) = 14$. Furthermore, a new pair of edges $\overrightarrow{e}' = (u, b)$ and $\overleftarrow{e}' = (b, u)$ is added in G'_a with an initial capacity of zero in both directions. However, after increasing the capacities by $\mu = f((b, v)) - c'((b, v)) = 7 - 1 = 6$, this pair of edges appears in the adjusted residual graph $G'_a(f')$ as a dashed arrow. Instead of bypassing or backtracking the excess flow stored in vertex b to the target or the source, this updated residual graph $G'_a(f')$ thus allows additional flow of value $\mu = 6$ to be pushed through as far as possible by a max-flow algorithm applied in *Phase 2*. Finally this additional amount of $\mu = 6$ needs to be subtracted again from the value of the resulting maximum u-v-flow $f^{\max}$.

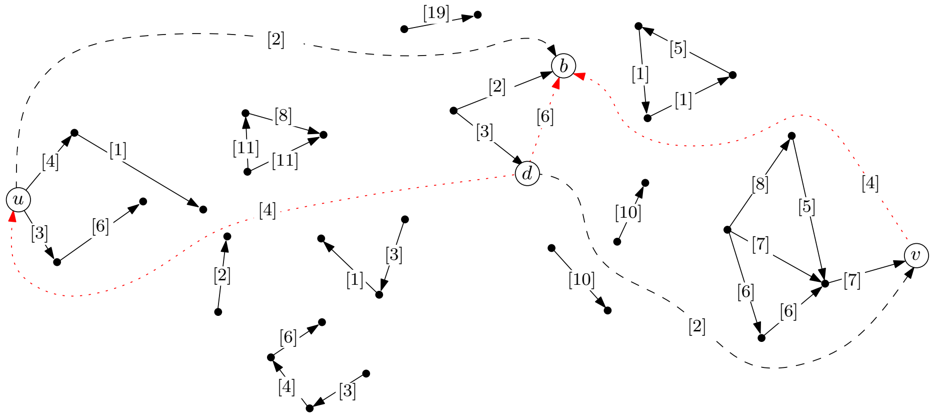
No incidence with source or target

The remaining positions regarding a modified edge $\overrightarrow{e}$ are characterized by the fact that $\overrightarrow{e}$ is neither incident with the source nor the target, i.e., that $\overrightarrow{e} = (b, d)$ and $\overleftarrow{e} = (d, b)$, with $b, d \notin \{u, v\}$. In this case we assume $\overrightarrow{e} = (b, d)$ to be the directed edge with positive flow. Figure 5.2b shows an example of capacity reduction for such a modified edge $\overrightarrow{e} = (b, d)$. As underlying graph we again consider the graph $G = G_a$ given by Figure 5.1a. The capacity of $\overrightarrow{e}$ changes from $c(\overrightarrow{e}) = 5$ to $c'(\overrightarrow{e}) = 3$. To resolve the resulting violation of the capacity constraint, we set the flow on the modified edge $\overrightarrow{e} = (b, d)$ to the new reduced capacity $c'(\overrightarrow{e}) = c'(\overleftarrow{e})$ and the flow on edge $\overleftarrow{e} = (d, b)$ to $-2\, c'(\overleftarrow{e})$ in G_a (see Line 24, Algorithm 15). Accordingly, the capacities of $\overrightarrow{e}$ and $\overleftarrow{e}$ in graph G_a are both set to $c'(\overrightarrow{e}) = c'(\overleftarrow{e})$ (see Line 16). In the adjusted residual graph $G'_a(f')$ in Figure 5.2b this yields a residual capacity of 6 for edge $\overleftarrow{e} = (d, b)$, while edge $\overrightarrow{e} = (b, d)$ remains saturated (compare to the dotted (red) arrow). In contrast to the previous position case, in this case the flow f changes, and therefore, we get an actual excess flow of value $\mu := f((b, d)) - c'((b, d))$ in vertex b and the same amount of flow missing in vertex d (see Line 17). To deal with this excess flow stored in vertex b, in Line 18 we add a new pair of opposedly directed edges $\overrightarrow{e_b}' = (b, v)$ and $\overleftarrow{e_b}' = (v, b)$ incident with the target to graph G_a as shortcuts, with an initial capacity of zero (if it does not exist yet). Analogously, a second pair of opposedly directed edges $\overrightarrow{e_d}' = (u, d)$ and $\overleftarrow{e_d}' = (d, u)$ incident with the source is added (see Line 21). Now we bypass the excess flow stored in b through the edge (b, v) to the target, while the source provides the flow missing in d over edge (u, d) (the negative flow on the opposed edges is adjusted accordingly, compare to the dotted (red) arrows in Figure 5.2b). However, this step may now cause a violation of the capacity constraint on these edges incident with the source, and the target respectively, which needs to be resolved as described in the former case (see Line 20 and Line 23).

According to the former case, in Figure 5.2b the edges $\overrightarrow{e_b}'$ and $\overrightarrow{e_d}'$ become saturated in graph G'_a, and therefore, do not appear in the adjusted residual graph $G'_a(f')$. The opposedly directed edges (v, b) and (d, u) (drawn as dotted (red) arrows) get a residual capacity of $2\, f'((b, v)) = 2\, f'((u, d)) = 2\, \mu = 4$. Furthermore,



(a) Adjusted residual graph $G'_a(f')$ after the reduction of $c((b,v)) = 7$ to $c'((b,v)) = 1$



(b) Adjusted residual graph $G'_a(f')$ after the reduction of $c((b,d)) = 5$ to $c'((b,d)) = 3$

Figure 5.2: Examples of adjusted residual graphs in different cases.

the approach of the former case requires the addition of a new pair of opposedly directed edges (b, v) and (v, b) for vertex b to graph G'_a, incident with the target and with an initial capacity of zero (if not existing yet); analogously for the vertex d (drawn as dashed arrows). The capacities of such edges, and the residual capacities as well, are adjusted by the addition of $\mu = 2$. Finally, after the application of a max-flow algorithm to $G'_a(f')$ in *Phase 2*, the additional amount of $\mu = 2$ needs to be subtracted twice from the value of the resulting flow $f^{\max}$.

5.1.3 Using Dynamic Flows for Updating Min-Cut Trees

The method of Kohli and Torr for dynamically updating minimum u-v-cuts in graph G requires maximum u-v-flows to be known in G, or graph G_a respectively. If we want to use such a method in our algorithms for updating min-cut trees (see Algorithm 14 and 11), the following problem occurs:

The construction of a min-cut tree $T(G)$ according to the Gomory-Hu method calculates exactly $n - 1$ maximum u-v-flows related to the step pairs used by Algorithm 3, as the split cuts corresponding to these flows are the $n - 1$ minimum

Problem caused by hidden step pairs

separating cuts that are directly calculated with the aid of a max-flow algorithm. All other minimum separating cuts represented in $T(G)$ are constructed indirectly by hiding step pairs and reconnecting subtrees (compare to Lemma 12). So, due to the possibility of step pairs to get hidden, the knowledge about a corresponding maximum u-v-flow is neither guaranteed for the edges $\{u, v\}$ in $T(G)$ nor any arbitrary pair of vertices, apart from the step pairs. On the other hand, there is also no guarantee that there exists a min-cut tree $T(G_{\oplus(\ominus)})$ regarding the modified graph $G_{\oplus(\ominus)}$ that can be constructed with the same set of step pairs as the previous min-cut tree $T(G)$. If we found a way to calculate a set of $n-1$ non-hiding step pairs in advance, this set would already define a unique structure of an updated min-cut tree $T(G_{\oplus(\ominus)})$, by Theorem 1.

However, our Algorithms 14 and 11, developed in Chapter 4 for updating min-cut trees, choose their step pairs according to the structure of the previous min-cut tree $T(G)$. So updating a minimum u-v-cut by updating the corresponding maximum u-v-flow is only possible if this maximum u-v-flow is known, i.e., if the considered step pair $\{u, v\}$ is also a step pair regarding the construction of the previous min-cut tree $T(G)$. Basically each updating approach that bases on the knowledge of a maximum flow only supports step pairs which have already occurred during the calculation of the previous min-cut tree $T(G)$.

Open problem of flow construction

In the case of a step pair $\{x, y\}$ getting hidden the construction of a valid maximum v-y-flow by the knowledge of a maximum u-v-flow and a maximum x-y-flow (as Lemma 12 does for minimum separating cuts) constitutes an open problem. Each of our tries ended up in somehow backtracking flow through the graph, which is as involved as the standard computation of a new maximum v-y-flow.

Note, that updating u-v-cuts individually by adjusting residual graphs of flows further requires the adjustment of all known residual graphs, even those whose flow does do not change for sure after the current modification. As it might happen that these flows get affected by a later modification, they need an adjusted residual graph at all times.

5.2 Updating a Set of All Minimum u-v-Cuts

As already mentioned at the beginning of this chapter a maximum u-v-flow not only represents one, but all minimum u-v-cuts in a graph G, according to a theorem of Picard and Queyranne [PQ80], which is referred to as Theorem 4 in this work.

Main idea of updating sets of cuts

Assume the set $\Theta(\{u, v\})$ of all minimum u-v-cuts in G to be known. Then updating these cuts is very easy in some cases: In the case of adding an edge $e_\oplus = \{b, d\}$ such that there exists at least one minimum u-v-cut in G that does not separate the modified vertices b and d, the set $\Theta_\oplus(\{u, v\}) := \Theta(\{u, v\}) \setminus \{\theta \in \Theta(\{u, v\}) | \theta \text{ separates } b \text{ and } d\}$ represents all minimum u-v-cuts in the modified graph $G_\oplus$ (by Lemma 8). In the case of deleting an edge $e_\ominus = \{b, d\}$ such that there exists at least one minimum u-v-cut in G that separates the modified vertices b and d, the set $\Theta_\ominus(\{u, v\}) := \Theta(\{u, v\}) \setminus \{\theta \in \Theta(\{u, v\}) | \theta \text{ does not separate } b \text{ and } d\}$ represents all minimum u-v-cuts in the modified graph $G_\ominus$ (by Lemma 9).

In following we review the theorem of Picard and Queyranne [PQ80] and deduce a compact representation of all minimum u-v-cuts in graph G. In Subsection 5.2.2 we will illustrate how such a representation can be updated, and we show the relation between this approach and the approach of Kohli and Torr introduced in the former section. Subsection 5.2.3 finally analyzes the conditions on which this approach can get used for updating min-cut trees according to Algorithm 14 and 11.

5.2.1 Representation of All Minimum u-v-Cuts

The following Theorem, originally given by Picard and Queyranne [PQ80], page 9, describes how a maximum u-v-flow represents all minimum u-v-cuts in a graph G.

Theorem: Flow-cut correspondence

Theorem 4 *There exists a one-to-one correspondence between all minimum u-v-cuts in a graph G and all closed sets of vertices containing the source u in the residual graph of a maximum u-v-flow in G.*

Definition: Closed set and strongly connected vertices

A closed set of vertices in a directed graph is defined as a set without outgoing edges. Two vertices in a directed graph are strongly connected if they are connected by a directed path in both directions.

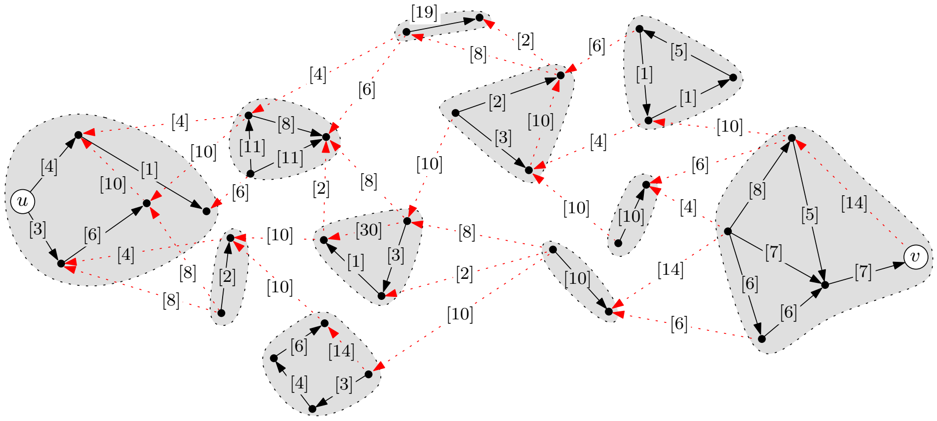
Based on Theorem 4 Fleischer [Fle99] modifies an approach of Hao and Orlin [HO94] to calculate a cactus-model that represents all global minimum cuts of a graph G with the same asymptotic effort a standard preflow-push algorithm (Goldberg and Tarjan [GT88]) needs to find a maximum u-v-flow in G. Note further, that the set $\Theta(\{u, v\})$ of all minimum separating cuts regarding two fixed vertices u and v can not be represented in a cactus-model. This follows by a Theorem of Dinitz and Nutov [DN95], which characterizes all sets of cuts that can be modeled by a cactus. For a detailed introduction of cacti see [DKL76].

However, we can model the set $\Theta(\{u, v\})$ of all minimum u-v-cuts in G by a directed acyclic graph (DAG). This DAG results from contracting all mutually strongly connected vertices in the residual graph of a maximum u-v-flow in G. The closed sets containing the source u in the residual graph are then given by cuts in the DAG that separate the source from the target and are only crossed by entering edges regarding the side containing the source. So these cuts in the DAG correspond to the minimum u-v-cuts in the underlying graph G, by Theorem 4. Reversing the edges in such a DAG yields a compact DAG-representation of all minimum u-v-cuts in G, with edges now directed from the source to the target, called $DAG(u, v)$.

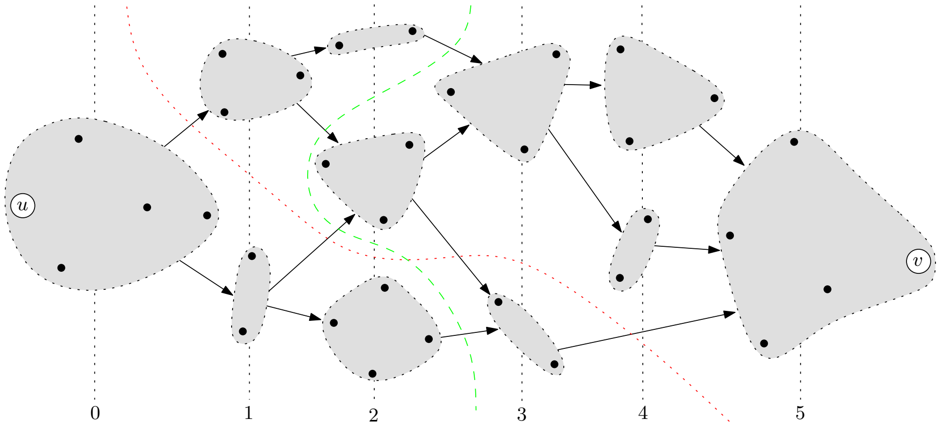
Definition: DAG-representation of all minimum u-v-cuts

A DAG-representation $DAG(u, v)$ for a maximum u-v-flow in G can be constructed in $O(n + m)$ time with the aid of a (slightly modified) breadth-first search from the source u. Calculating only the first minimum u-v-cut given by a maximum u-v-flow basically takes the same asymptotic time. With a second breadth-first search from the source, we can further assign a level to each node X in $DAG(u, v)$ that represents the maximum length of all paths in $DAG(u, v)$ from the node X_u containing the source to X. So the nodes become topologically ordered. Obviously it holds that there are no edges between two nodes X and X' on the same level in $DAG(u, v)$, and therefore, there are no edges between two sets of vertices in graph G that correspond to nodes X and X' on the same level in $DAG(u, v)$.

Figure 5.3a illustrates a DAG regarding the maximum u-v-flow f given in Figure5.1a. The mutually strongly connected vertices in the residual graph $G(f)$ (compare to Figure 5.1b) are not contracted yet, but bunched by dotted borders. The edges between the so bordered nodes are also not contracted yet. Note, that solid (black) edges have positive residual capacities in both directions, dotted (red) edges are saturated in the direction not shown. Figure 5.3b finally shows the DAG-representation $DAG(u, v)$ with nodes topologically ordered. The dotted (red) cut in this figure represents no minimum u-v-cut, as it is crossed by edges in both directions, and therefore, the side containing the source u does not constitutes a closed set. By contrast, the dashed (green) line correctly marks a minimum u-v-cut in G.



(a) Directed acyclic graph regarding the residual graph $G(f)$ in Figure 5.1b



(b) DAG-representation of all minimum u-v-cuts regarding the flow f in Figure 5.1a

Figure 5.3: Example of a directed acyclic graph and the reversed DAG-representation.

5.2.2 Updating the DAG-Representation

Updating DAG in case of edge addition

Concerning the updating steps for a DAG-representation $DAG(u, v)$, we distinguish the modifications of adding an edge $e_\oplus = \{b, d\}$ and deleting an edge $e_\ominus = \{b, d\}$ in graph G. In the case of edge addition and if there exists at least one minimum u-v-cut in G that does not separate the modified vertices b and d, the set $\Theta(\{u, v\})$ of all minimum u-v-cuts in G can get updated to $\Theta_\oplus(\{u, v\})$ by just deleting all cuts from $\Theta(\{u, v\})$ that separate b and d, by Lemma 8. We call this situation *updating edge addition regarding the pair* $\{u, v\}$. If otherwise all minimum u-v-cuts in G separate b and d, a new max-flow calculation and a new DAG construction becomes necessary. This situation is referred to as *recalculating edge addition regarding the pair* $\{u, v\}$.

Advantage of the DAG-representation

Fortunately with a DAG-representation it is very simple to decide whether a designated pair of vertices is separated by any minimum u-v-cut or even by all minimum u-v-cuts. For a pair $\{g, h\}$ of vertices there exists no separating minimum u-v-cut if and only if g and h are included in the same node $X_g = X_h$ in the DAG-representation $DAG(u, v)$. Vice versa, all minimum u-v-cuts separate a pair $\{g, h\}$ if and only if one vertex g shares its node X_g with the source u while the other vertex

h shares its node X_h with the target v. By implementing the DAG-representations as union-find structures (see for example La Poutré [Pou90]) such questions can get answered efficiently. Furthermore, union-find structures will turn out to also be very suitable for the updating operations, where nodes will be contracted in the DAG-representation.

In the case of an updating edge addition, i.e., if the modified vertices b and d do not both share their nodes with the source or the target in $DAG(u, v)$, the deletion of all cuts in the DAG-representation $DAG(u, v)$ that separate the modified vertices b and d can be simply achieved by contracting the nodes X_b and X_d containing b and d. However, as this operation may cause circles in the resulting graph, we actually need to contract all paths between X_b and X_d in $DAG(u, v)$ to get a feasible updated DAG-representation $DAG_\oplus(u, v)$. With the aid of a modified breadth-first search regarding the levels in $DAG(u, v)$ Algorithm 16 (UPDATEDAG) provides this functionality in $O(n + m)$ time, with n denoting the number of nodes in the DAG-representation $DAG(u, v)$ and m denoting the number of edges in $DAG(u, v)$. So this asymptotic bound in particular is valid for n and m denoting the numbers of vertices and edges in the underlying graph G. Note, that after updating $DAG(u, v)$ by a call of UPDATEDAG $(DAG(u, v), X_b, X_d)$ the topological order of the nodes in the resulting graph $DAG_\oplus(u, v)$ is not given anymore. So the levels need to be recalculated.

Algorithm 16: UPDATEDAG

```
Input: Directed acyclic graph $DAG()$ with nodes topologically ordered as
       described in Subsection 5.2.1, node $X$ and node $X'$ in $DAG()$
Output: Directed acyclic graph $DAG()$ with all paths between $X$ and $X'$
        contracted
Store $X$ and $X'$ in $X_1, X_2$ increasingly ordered by their level
if $X_1$ and $X_2$ are on the same level then
    %if there is no path between $X_1$ and $X_2$
    Contract $X_1$ and $X_2$ in $DAG()$
    return resulting directed acyclic graph
else
    %if there may exist paths from $X_1$ to $X_2$
    Mark all nodes reachable from $X_1$ (with a breadth-first search (BFS))
    Store all marked nodes reversely reachable from $X_2$ (with a BFS)
    %Note, that it suffices to consider levels between $X_1$ and $X_2$
    Contract all stored nodes together with $X_1$ and $X_2$ in $DAG()$
    return resulting directed acyclic graph
```

Updating DAG in case of edge deletion

Analogously in the case of edge deletion the DAG-representation can be updated if there exists at least one minimum u-v-cut in G that separates the modified vertices b and d, i.e., if the modified vertices are not included in the same node in $DAG(u, v)$. This situation describes an *updating edge deletion regarding the pair* $\{u, v\}$. The deletion of all cuts in $\Theta(\{u, v\})$ that do not separate b and d then yields the updated set $\Theta_\ominus(\{u, v\})$ of all minimum u-v-cuts in $G_\ominus$, by Lemma 9. This is equivalent to the contraction of all paths from X_u to X_b as well as all paths from X_d to X_v, with X_b lying on a lower level than X_d. So to update the DAG-representation $DAG(u, v)$ call UPDATEDAG $(DAG(u, v), X_u, X_b)$ and afterwards UPDATEDAG $(DAG(), X_d, X_v)$, with $DAG()$ the graph resulting from the former call. Note, that in the case of the deletion of an edge $e_\ominus = \{b, d\}$ the nodes X_b and X_d do never lie on the same level in $DAG(u, v)$, as there exist no edges in G between nodes on the same level in $DAG(u, v)$. The second situation, called *recalculating edge deletion* *regarding the pair* $\{u, v\}$, occurs if no minimum u-v-cut in G separates b and d. Then a new max-flow calculation and a new DAG construction becomes necessary.

Combining DAG-Representations with Adjusted Residual Graphs

Assume, beside a DAG-representation $DAG(u, v)$ of all minimum u-v-cuts, a related maximum u-v-flow f in graph G_a and a residual graph $G_a(f)$ to be given such that f in G_a represents all minimum u-v-cuts in graph G, i.e., calculating a DAG-representation of flow f regarding the residual graph $G_a(f)$ would yield again the given DAG-representation $DAG(u, v)$. Then it is feasible to regard the graph G_a with the flow f and the residual graph $G_a(f)$ as underlying graphs of $DAG(u, v)$. This allows to combine the approach of updating DAG-representations with the approach of adjusting residual graphs by Kohli and Torr [KT07].

In case of an updating edge addition

In the case of an updating edge addition regarding the pair $\{u, v\}$ the previous flow f in G_a also represents u-v-cuts in $G_\oplus$, as adding an edge never violates the capacity constraint of f. So this is a “gentle” modification, and it holds that $f = f'$. The underlying graph G_a and the residual graph $G_a(f)$ can also get adjusted easily by accordingly increasing the capacity and the residual capacity of the modified edge $e_\oplus = \{b, d\}$. The previous flow f in G_a even represents all minimum u-v-cuts in $G_\oplus$, as the maximum flow value in $G_\oplus$ remains the same. So there is no additional application of a max-flow algorithm necessary, we just update the DAG-representation $DAG(u, v)$ (compare to Table 5.2, Column 1). Note, that updating the DAG-representation $DAG_\oplus(u, v)$ is equivalent to calculating a new DAG-representation of the flow f' regarding the adjusted residual graph $G'_a(f')$, as increasing the residual capacity of the previously saturated edge $e_\oplus$ causes the opening of exactly such bottlenecks in $G'_a(f')$ crossed by the edge $e_\oplus$. Therefore, it is feasible to regard the adjusted graph G'_a with the flow f' and the adjusted residual graph $G'_a(f')$ as underlying graphs of the updated DAG-representation $DAG_\oplus(u, v)$.

In case of an recalculating edge addition

In the case of a recalculating edge addition regarding the pair $\{u, v\}$, it also holds that $f = f'$, as this is a “gentle” modification. So the underlying graph G_a and the residual graph $G_a(f)$ can also get adjusted easily by accordingly increasing the capacity and the residual capacity of the modified edge $e_\oplus = \{b, d\}$. However, the value of a maximum u-v-flow in $G_\oplus$ might have increased due to the edge addition. So instead of updating the DAG-representation $DAG(u, v)$, in this case we apply a max-flow algorithm to the adjusted residual graph $G'_a(f')$ (compare to Table 5.2, Column 2). The resulting flow $f^{\max}$ in the adjusted graph G'_a then again represents all minimum u-v-cuts in the modified graph $G_\oplus$. The new DAG-representation $DAG_\oplus(u, v)$ is constructed from scratch by a breadth-first search on the new residual graph $G'_a(f^{\max})$. Hence, we regard the adjusted graph G'_a with the new calculated flow $f^{\max}$ and the new residual graph $G'_a(f^{\max})$ as underlying graphs of the new DAG-representation $DAG_\oplus(u, v)$.

In case of an updating edge deletion

In the case of an updating edge deletion regarding the pair $\{u, v\}$, with weight $c(e_\ominus) = \Delta$, the previous flow f in G_a needs to get adjusted, as the deletion of a saturated edge violates the capacity constraint of f. Hence we adjust the graph G_a, the flow f and the residual graph $G_a(f)$ depending on whether the modified edge is incident with the source or the target or with neither. The adjusted flow f' in G'_a then is again valid and represents all minimum u-v-cuts in $G_\ominus$. So there is no additional application of a max-flow algorithm necessary, we just update the DAG-representation $DAG(u, v)$ (compare to Table 5.2, Column 3). Again updating the DAG-representation $DAG_\oplus(u, v)$ is equivalent to calculating a new DAG-representation of the flow f' regarding the adjusted residual graph $G'_a(f')$, as decreasing the residual capacity of the previously saturated edge $e_\ominus$ in a bottleneck causes a flow reduction and so indirectly causes the opening of exactly those bottlenecks in $G'_a(f')$ that are not crossed by edge $e_\ominus$. We regard the adjusted graph G'_a with the flow f' and the adjusted residual graph $G'_a(f')$ as underlying graphs of the updated DAG-representation $DAG_\ominus(u,v)$. Note, that we further need to store μ, as the value of flow f' differs from the weight of a minimum u-v-cut in $G_\ominus$ by either μ or even $2\,\mu$.

In case of an recalculating edge deletion

In the case of an recalculating edge deletion regarding the pair $\{u,v\}$, with weight $c(e_\ominus)=\Delta$, the previous flow f in G_a needs to get adjusted if the deletion of edge $e_\ominus$ violates the capacity constraint of f, i.e., if $f(e_\ominus)>0$. Otherwise (compare to Table 5.2, Column 5) it holds $f=f'=f^{\max}$ and adjusting G_a and $G_a(f)$ is as trivial as in the case of an recalculating edge addition. In case of a violation of the capacity constraint we adjust the graph G_a, the flow f and the residual graph $G_a(f)$ depending on whether the modified edge is incident with the source or the target or with neither. As the maximum flow value in $G_\ominus$ in this case might have decreased by an unknown amount lower than or equal to Δ, the adjusted flow f' in G'_a no longer represents minimum u-v-cuts in $G_\ominus$ for sure. So instead of updating the DAG-representation $DAG(u,v)$, in this case we apply a max-flow algorithm to the adjusted residual graph $G'_a(f')$ (compare to Table 5.2, Column 4). The resulting flow $f^{\max}$ in the adjusted graph G'_a then again represents all minimum u-v-cuts in the modified graph $G_\ominus$. The new DAG-representation $DAG_\ominus(u,v)$ then is constructed from scratch by a breadth-first search on the new residual graph $G'_a(f^{\max})$. We regard the adjusted graph G'_a with the new calculated flow $f^{\max}$ and the new residual graph $G'_a(f^{\max})$ as underlying graphs of the new DAG-representation $DAG_\ominus(u,v)$. Note, that we again need to store μ, as the value of flow $f^{\max}$ differs from the weight of a minimum u-v-cut in $G_\ominus$ by either μ or even $2\,\mu$.

5.2.3 Using DAG-Representations for Min-Cut Trees

Idea for solving the problem caused by hidden step pairs

Dynamically updating minimum u-v-cuts in min-cut trees with the aid of DAG-representations basically faces the same problems with hiding step pairs as the approach of adjusting residual graphs by Kohli and Torr [KT07] does. However, in a special case of hiding a previous step $\{x,y\}$, considered in the following Lemma 21, we can construct the set of all minimum y-v-cuts in G by the knowledge of all minimum x-y-cuts and all minimum u-v-cuts regarding the current step pair $\{u,v\}$. So apart from requiring a special condition for hiding step pairs, Lemma 21 basically expands the assertion of Lemma 12, which regards a single minimum u-v-cut, to the set $\Theta(\{u,v\})$ of all minimum u-v-cuts concerning a fixed pair $\{u,v\}$.

Lemma 21 considers an intermediate min-cut tree $T_\star(G)=(V,E_\star,c_\star())$ produced by the Gomory-Hu algorithm (Algorithm 3) with $e=\{S,\bar{S}\}$ an edge in $E_\star$. If we assume S to be split next, the edge e constitutes the connection edge of the subtree N of S containing $\bar{S}$ (see Figure 5.4). Let $\{x,y\}\subseteq V$ be the step pair that created the edge e, with $x\in S$ and $y\in\bar{S}$. Let u and v be any vertices in S (serving as next step pair), and let $\Theta(\{u,v\})$ denote the set of all minimum u-v-cuts in graph G such that each cut in $\Theta(\{u,v\})$ separates x and y. So $\{x,y\}$ definitely gets hidden, independently of the choice of a split cut in $\Theta(\{u,v\})$. Note, that this induces the cuts in $\Theta(\{u,v\})$ to be at least of the same weight as the cut induced by e. Now chose a minimum u-v-cut $(U(S),V(S)\setminus U(S))\in\Theta(\{u,v\})$ in $G(S)$ as split cut (compare to the thinly dashed line in Figure 5.4), with $\{y,v\}\subseteq U(S)$ (This is possible without loss of generality, as neither the sides of $(U(S),V(S)\setminus U(S))$ nor the vertices $\{u,v\}$ are fixed). Then Lemma 12 (second part) says that after the splitting of S also the pair $\{y, v\}$ is a cut pair for the cut induced by edge e, and Lemma 21 now states the following expanded assertion.

Lemma: Constructing set of cuts after hiding step pair

Lemma 21 *Instead of considering only one minimum x-y-cut induced by edge e, let $\Theta(\{x, y\})$ denote the set of all minimum x-y-cuts in G. If the potential split cuts in $\Theta(\{u, v\})$ are more expensive than the cuts in $\Theta(\{x, y\})$, then the deletion of all cuts in $\Theta(\{x, y\})$ that do not separate y and v yields the set $\Theta(\{y, v\})$ of all minimum y-v-cuts in G.*

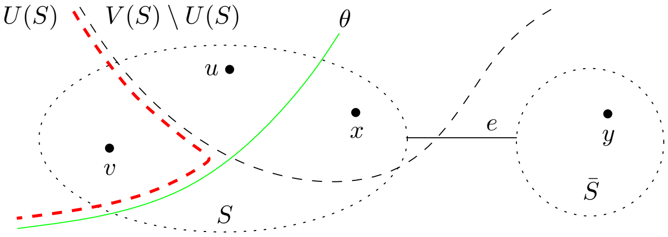


Figure 5.4: Illustration to Lemma 21.

Proof. As all minimum y-v-cuts in $\Theta(\{x, y\})$ have the same weight as the cut induced by edge e, and the cut induced by edge e is also a minimum y-v-cut in graph G, each cut in $\Theta(\{x, y\})$ that separates y and v constitutes a minimum y-v-cut in G. As all minimum u-v-cuts in $\Theta(\{u, v\})$ are supposed to be more expensive than the cut induced by edge e, there exists no minimum y-v-cut in G that separates u and v. We prove now that there further exists no minimum y-v-cut that does not separate x and y. If there was such a minimum y-v-cut θ (compare to the solid (green) line in Figure 5.4), this would cross the minimum u-v-cut $(U(S), V(S) \setminus U(S)) \in \Theta(\{u, v\})$ chosen as split cut (compare to the thinly dashed (black) line in Figure 5.4). However, this crossing, by Lemma 15, implies a minimum u-v-cut that does not separate x and y (see thickly dashed (red) line in Figure 5.4), which contradicts the assumption that all cuts in $\Theta(\{u, v\})$ separate x and y. □

Impossibility of general restriction for hiding step pairs

Unfortunately, we can not restrict in general the hiding of step pairs such that a pair gets hidden if and only if it is separated by all minimum u-v-cuts regarding a current step pair $\{u, v\}$ (as required by Lemma 21): Consider two arbitrary connection edges e_i and e_j of subtrees of a current split node S and a current step pair $\{u, v\}$. Let further $\{x_i, y_i\}$ denote the nearest cut pair of e_i (see Definition 9) and $\{x_j, y_j\}$ the nearest cut pair of e_j. Even if neither $\{x_i, y_i\}$ nor $\{x_j, y_j\}$ is separated by all minimum u-v-cuts in G, the existence of a single u-v-cut that does neither separate $\{x_i, y_i\}$ nor $\{x_j, y_j\}$ is still not guaranteed (compare to Figure 5.5a).

Restriction for hiding step pairs becomes possible in special situation

However, in the situation shown in Figure 5.5b, we can guarantee the existence of a minimum r-u-cut that separates exactly those nearest cut pairs that are separated by all minimum r-u-cuts in G, by the following Lemma 22. Always choosing such a cut as split cut guarantees the nearest cut pair of a connection edge (which we assume to be a step pair not hidden yet) to get hidden if and only if it is separated by all minimum separating cuts of the current step pair.

Lemma: First minimum cut makes restriction possible

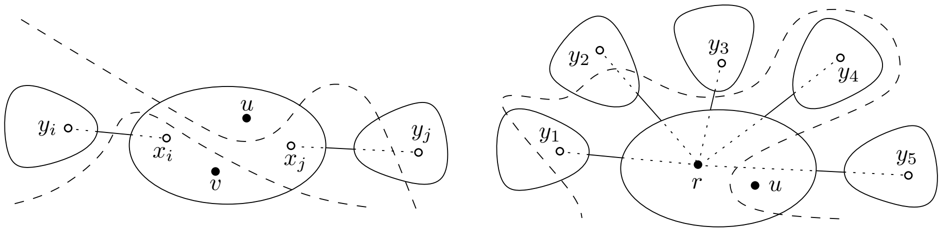
Lemma 22 *Consider a current split node S and a bunch of subtrees all connected by connection edges with the same vertex $r \in S$ included in their nearest cut pair. Furthermore, the center r is supposed to be part of the next step pair $\{u, r\}$ (see Figure 5.5b). Then the first minimum u-r-cut given by a maximum u-r-flow in* *graph G separates the nearest cut pair of such a connection edge if and only if all minimum u-r-cuts separate this nearest cut pair.*

Proof. ($\Rightarrow$): The first minimum u-r-cut given by a maximum u-r-flow in graph G separates a nearest cut pair $\{y, r\}$ if and only if vertex y lies on the same cut side as source u. If there was another minimum u-r-cut not separating y from r, this cut needed to separate the source u from y, and therefore, would cross the first minimum u-r-cut. This contradicts the fact that the first minimum u-r-cut does not cross any other minimum u-r-cut in G.

($\Leftarrow$): If a nearest cut pair $\{y, r\}$ is separated by all minimum u-r-cuts in graph G, it follows directly that it is particularly separated by the first minimum u-r-cut given by a maximum u-r-flow in G. □

Constructing new set of cuts after hiding cut pair

In the situation shown in Figure 5.5b, by Lemma 22 there always exists a valid split cut that only separates those nearest cut pairs that are separated by all minimum u-r-cuts. This allows us to restrict a previous step pair $\{y_i, r\}$ to get hidden if and only if the minimum u-r-cuts regarding the current step pair $\{u, r\}$ are more expensive than the split cut induced by $\{y_i, r\}$, and additionally, $\{y_i, r\}$ is separated by all minimum u-r-cuts. So by Lemma 21 we are then able to construct a DAG-representation $DAG(y_i, u)$ of all minimum y_i-u-cuts from the DAG-representation $DAG(y_i, r)$ of all minimum y_i-r-cuts and the current step pair $\{u, r\}$. This construction can be done in $O(n + m)$ time by a call of UPDATEDAG ($DAG(y_i, r), X_u, X_r$) (Algorithm 16), although this is no update operation.



(a) Each cut not separating $\{x_i, y_i\}$ separates $\{x_j, y_j\}$ and vice versa

(b) First minimum u-r-cut separates exactly those nearest cut pairs that are separated by all minimum u-r-cuts

Figure 5.5: Possibility of restricting the hiding of step pairs depends on situation.

Updating DAG-representation for a hidden pair

Note, that due to the open problem formulated in Subsection 5.1.3 hiding a step pair $\{y_i, r\}$ still causes the leakage of the corresponding maximum y_i-r-flow and the graphs to adjust. So combining the DAG-representations with the adjustment of residual graphs makes no sense for hidden pairs of vertices. In case of an updating edge addition or an updating edge deletion, therefore, only the DAG-representation is updated (compare to Table 5.1, Column 1 and 3). Updating the minimum y_i-u-cuts regarding a hidden pair $\{y_i, u\}$ in the case of a recalculating edge addition or a recalculating edge deletion requires the application of a max-flow algorithm to the complete modified graph. There is no chance to save effort by using an adjusted residual graph (compare to Table 5.1, Column 2 and 4). For the calculation of such a new maximum flow based on a complete, non-adjusted graph, we assume the max-flow algorithm by Goldberg and Tarjan [GT88], solving the problem in $O(nm \log \frac{n^2}{m})$ time, to be used. The steps for updating the DAG-representation $DAG(y_i, u)$ of all minimum y_i-u-cuts after a modification of the underlying graph

G are again outlined in Table 5.1. Table 5.2 lists the updating steps for a DAG-representation $DAG(u,v)$ in case of $\{u,v\}$ being visible, according to the description given in Subsection 5.2.2.

hidden pair $\{y_i, u\}$			
Updating Add	**Recalculating Add**	**Updating Del**	**Recalculating Del**
update DAG $[O(n+m)]$	GoldbergTarjan $[O(nm \log \frac{n^2}{m})]$ new DAG $[O(n+m)]$	update DAG $[O(n+m)]$	GoldbergTarjan $[O(nm \log \frac{n^2}{m})]$ new DAG $[O(n+m)]$

Table 5.1: An updating approaches for minimum y_i-u-cuts with $\{y_i, u\}$ a hidden pair.

visible pair $\{u, v\}$				
Updating Add	**Recalc. Add**	**Updating Del**	**Recalculating Del**	
gentle	**gentle**	**violating**	**gentle**	**violating**
adjust $[O(1)]$ update DAG $[O(n+m)]$	adjust $[O(1)]$ FlowAlgo ["good"] new DAG $[O(n+m)]$	adjust $[O(1)]$ update DAG $[O(n+m)]$	adjust $[O(1)]$ FlowAlgo ["good"] new DAG $[O(n+m)]$	adjust $[O(1)]$ FlowAlgo ["good"] new DAG $[O(n+m)]$

Table 5.2: Combining two updating approaches for minimum u-v-cuts if $\{u,v\}$ is visible.

Partial dynamic modifications allow DAG-representation method

Algorithm 11 repeatedly considers a situation as shown in Figure 5.5b (compare to the second aspect described in Subsection 4.3.2), and therefore, allows the restriction required by Lemma 21 to be made. So if we further provide the required situation in the first initial min-cut tree by choosing the step pairs iteratively as described in Definition 12, on condition that the underlying graph is only modified by edge deletions it is possible to continuously use the DAG-representation for updating the minimum u-v-cuts in the min-cut tree, even if step pairs get hidden.

Question of effort saving

However, the saving of effort and the additional effort caused by updating DAG-representations regarding edges in the min-cut tree that are already known to remain minimum cuts after a modification or by constructing new DAG-representations after hiding step pairs needs to be carefully calculated. A real effort saving is only possible if the structure of the underlying, dynamically changing graphs at all times provide a sufficiently large amount of minimum u-v-cuts for each of the considered pairs $\{u,v\}$.

Construction of min-cut tree representing sets of minimum cuts

Finally we remark that Lemma 21 together with the following Lemma 23 makes it possible to construct a min-cut tree $T_{\mathrm{DAG}}(G)$ that represents the sets $\Theta(\{u,v\})$ of all minimum separating cuts for all pairs $\{u,v\}$ in a graph G in the same asymptotic time as constructing a min-cut tree $T(G)$ according to Gomory and Hu [GH61]. By choosing the step pairs for the min-cut tree calculation iteratively as described in Definition 12 and restricting the hiding of step pairs as described above, we can construct a min-cut tree $T_{\mathrm{DAG}}(G)$ with DAG-representations associated to the edges. Thereby the max-flow calculation by Goldberg and Tarjan [GT88] takes $O(nm\log\frac{n^2}{m})$ time for each step pair $\{y_i, r\}$ and the breadth-first search to find the first minimum y_i-r-cut needs $O(n+m)$ time, which is asymptotically the same time as calculating a complete DAG-representation $DAG(y_i, r)$ of all minimum y_i-r-cuts. If the step pair $\{y_i, r\}$ gets hidden by a later step pair $\{u, r\}$, an additional effort of again $O(n+m)$ occurs for the construction of the new DAG-representation $DAG(y_i, u)$ according to Lemma 21. Note, that a step pair can only get hidden once, so this additional effort does not affect the asymptotic time bound. Lemma 23 now shows that the set $\Theta(\{u, v\})$ of all minimum separating cuts for an arbitrary pair $\{u, v\}$ in graph G is represented in the DAG-representations $DAG(g_i, h_i)$ for all minimum weighted edges $e_i = \{g_i, h_i\}$, $i = 1, \ldots, z$, on the path γ from u to v in $T_{\mathrm{DAG}}(G)$.

Lemma:
All minimum separating cuts for an arbitrary pair of vertices

Lemma 23 *Let $T_{DAG}(G)$ denote a min-cut tree constructed as described above. Let further u and v denote two arbitrary vertices in G and $e_i = \{g_i, h_i\}$ a minimum weighted edge on the path γ from u to v in $T_{DAG}(G)$. Then deleting all cuts in $\Theta(\{g_i, h_i\})$ that do not separate u and v yields a subset $\Theta(i) \subseteq \Theta(\{u, v\})$ of all minimum u-v-cuts in G. For the complete set $\Theta(\{u, v\})$ it holds that*

$$\Theta(\{u, v\}) = \bigcup_{i=1}^{z} \Theta(i).$$

Open problem of constructing merged DAG-representation

Proof. As the edge $e_i = \{g_i, h_i\}$ induces a minimum u-v-cut in G, all other minimum g_i-h_i-cuts that also separate u and v are minimum u-v-cuts, i.e., $\Theta(i) \subseteq \Theta(\{u, v\})$, and therefore, $\Theta(\{u, v\}) \supseteq \bigcup_{i=1}^{z} \Theta(i)$. Furthermore, it follows that all edges on path γ separated by a minimum u-v-cut in G are minimum weighted edges on γ. Vice versa, each minimum u-v-cut in graph G separates at least one edge on path γ, which turns out to be a minimum weighted edge on γ according to the former, i.e., $\Theta(\{u, v\}) \subseteq \bigcup_{i=1}^{z} \Theta(i)$. ◻

So by Lemma 23 calling UPDATEDAG $(DAG(g_i, h_i), X_u, X_v)$ for each DAG-representation $DAG(g_i, h_i)$, $i = 1, \ldots, z$, yields a set of directed acyclic graphs that all together represent the set $\Theta(\{u, v\})$ of all minimum u-v-cuts in G. The construction of a closed DAG-representation $DAG(u, v)$ from these graphs still constitutes an open problem.

Chapter 6

Updating Clusterings Based on Min-Cut Trees

In the former chapters we analyzed possibilities of efficiently updating complete min-cut trees. However, the aim of this work is to develop a correct and effort saving algorithm for updating clusterings. To this end we consider clusterings resulting from the cut-clustering method (see Algorithm 1) by Flake et al. [FTT04], which bases on the calculation of min-cut trees and guarantees an intra- and inter-clustering quality. Although the former chapters hence are motivated by this method, we will see that the knowledge of a complete min-cut tree is not necessary. In this chapter we propose four algorithms for dynamically updating clusterings without knowing a complete underlying min-cut tree structure. Our new algorithms are based on abbreviated versions of the updating algorithms developed for min-cut trees so far. They distinguish the four cases of deleting an edge between two different clusters, deleting an edge within a single cluster, adding an edge within a single cluster, and finally adding an edge between two different clusters. The two algorithms concerning edge deletions are described in Section 6.2, in Section 6.3 we introduce the algorithms for the edge addition cases. In the latter we further draw a bow to the approach of Saha and Mitra [SM06] discussed in Chapter 2. As already announced in Chapter 2, we now give a reasonable interpretation of the unaffect lemma regarding our new updating algorithm for an inter-cluster edge addition. We will see that our algorithm looks quite similar, but is different in details, to Algorithm 2 given by Saha and Mira [SM06]. Finally we will also catch up the proof of Lemma 5 omitted in Subsection 2.3.1.

6.1 Abbreviating the Cut-Clustering Method

The cut-clustering method (see Algorithm 1) given by Flake et al. [FTT04] calculates a clustering $\zeta(G)$ of an undirected weighted graph $G = (V, E, c())$. To this end it inserts an artificial sink t into G connected to each vertex by an edge with weight α. Then a min-cut tree $T(G_\alpha)$ of the so constructed graph $G_\alpha = (V_\alpha, E_\alpha, c_\alpha())$ is calculated (with the aid of the Gomory-Hu method, see Algorithm 3). The clustering $\zeta(G)$ of the original graph G finally is identified with the connected components resulting from the deletion of sink t in $T(G_\alpha)$. This is, the clusters in $\zeta(G)$ are given by the non-crossing cuts in graph G_α induced by those edges $\{y_i, t\}$, $i = 1, \dots, z$, that are incident with the sink t in the tree $T(G_\alpha)$ (see Figure 6.1). Note, that the vertex y_i together with the sink t forms the nearest cut pair of the edge $\{y_i, t\}$, and

Clustering is given by a set of non-crossing cuts

Abbreviating the Gomory-Hu algorithm

the edge-induced cut respectively. A set of such non-crossing minimum y_i-t-cuts, which partitions the vertices of graph G and isolates the sink t in graph G_α, is denoted in the following by $\Theta(G)$. The vertices $y_1, \dots, y_z$ are summarized in the set $W(G)$. To calculate a clustering $\zeta(G)$ instead of a complete min-cut tree $T(G_\alpha)$ hence a Gomory-Hu execution on the enlarged graph G_α can be stopped as soon as the node that contains t becomes a singleton. The clustering $\zeta(G)$ then is given by the set $\Theta(G)$ of the non-crossing minimum y_i-t-cuts cutting off the subtrees of t in the intermediate min-cut tree $T_\star(G_\alpha)$ after stopping the Gomory-Hu execution. Any further split cut calculated during the Gomory-Hu execution identifies an edge lying somewhere within a subtree. Those edges, or split cuts respectively, are ignored. Their calculation turns out to be vain.

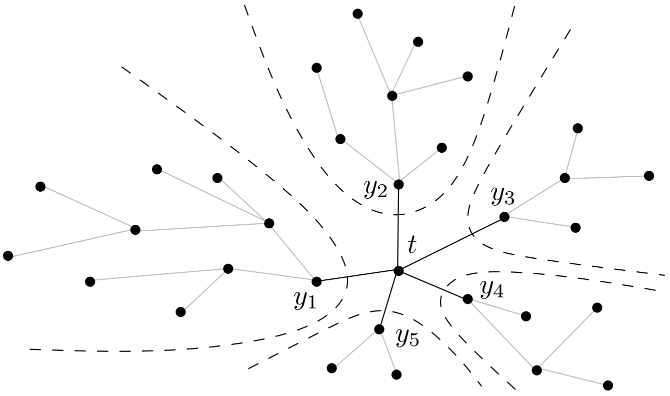


Figure 6.1: Clustering $\zeta(G)$ induced by non-crossing minimum y_i-t-cuts in graph G_α.

Heuristic by Flake et al.

Flake et al. propose a heuristic to reduce the number of such vain split cut calculations. Their idea is to choose the step pairs in a way that isolates the sink t quite soon during a Gomory-Hu execution. To this end they advise to define the sink t as representative or center of the initial node $S = V_\alpha$ at the beginning. Then they repeatedly pair the sink t with vertices decreasingly ordered by the total weight of their incident edges in graph G (compare also to Algorithm 23). If a step pair becomes hidden during a Gomory-Hu execution, its split cut, which is stored in the set $\Theta(G)$ under reserve, is replaced by the split cut that causes the hiding.

The algorithms for updating complete min-cut trees developed in Chapter 4 base on the Gomory-Hu method, and therefore, can be abbreviated in the same way. This is, to update a clustering instead of a complete min-cut tree we can stop the execution as soon as the sink t becomes isolated. The algorithms developed in this Chapter will base on such abbreviated versions of the former updating algorithms for min-cut trees.

Usable methods for updating individual cuts

As hidden step pairs are of no interest anymore, we can further use the method of Kohli and Torr [KT07] to adjust residual graphs for a faster recalculation of minimum y_i-t-cuts. Using DAG-representations instead, or even in combination as described in Section 5.2, might cause more additional effort than effort saving: Both methods require adjusting and updating operations also for those y_i-t-cuts that are already known not to change. However adjusting a residual graph according to Kohli and Torr takes constant time per cut, while updating DAG-representations needs $O(n + m)$ time per cut. Note, that for cuts which are already known not to change, a new max-flow calculation after adjusting the residual graph is not necessary. In some cases, i.e., if the underlying graph provides a sufficient amount of minimum y_i-t-cuts for each vertex y_i, it might be reasonable to also update DAG-representations. However the algorithms developed in this chapter just consider the method of Kohli and Torr.

6.2 Updating Algorithms for Edge Deletions

We start with the case of deleting an edge between two different clusters, as this case can be easily solved with the aid of Algorithm 11, the former algorithm for updating a complete min-cut tree after an edge deletion. The case of deleting an edge within a cluster can also be deduced from Algorithm 11, but in a less direct way. The algorithms for edge additions introduced in Section 6.3 will turn out to be quite simple, compared to those regarding edge deletions. In each case the modification in the original graph G constitutes an according modification in graph G_α enlarged by the artificial sink t.

6.2.1 Inter-Cluster Edge Deletion

Situation in case of inter-cluster edge deletion

In the case of deleting an inter-cluster edge $e_\ominus = \{b, d\}$ the modified vertices b and d lie in different clusters of graph G, which means, they lie in different subtrees of the sink t in a min-cut tree $T(G_\alpha)$. Remember that in the context of clusterings we do not know the tree structure of a min-cut tree $T(G_\alpha)$. We just know the clustering $\zeta(G)$ given by the set $\Theta(G)$ of non-crossing cuts and the set $W(G)$ of the vertices related to these cuts. Furthermore, we assume the according weights to be assigned to the cuts in $\Theta(G)$. Nevertheless, we know that there exists a min-cut tree $T(G_\alpha)$ that induces the given clustering $\zeta(G)$, and therefore, we know that the sink t needs to be a vertex on the unique path γ from b to d in this min-cut tree. Figure 6.2 exemplarily shows the sink t, the clusters bordered by dotted lines and the path γ concerning the min-cut tree introduced in Figure 6.1. The unknown edges on path γ are drawn as dashed lines.

Now assume Algorithm 11 to be applied to the unknown, but existing, min-cut tree $T(G_\alpha)$. As we consider a case of edge deletion the initial intermediate min-cut tree $T_\circ(G_\alpha)$ for this algorithm hence resulted from contracting all edges in $T(G_\alpha)$ that do not lie on path γ. In Figure 6.2 the nodes of $T_\circ(G_\alpha)$ are bordered by solid lines. In Line 10 in Algorithm 11 then the sink t became the representative $r(S)$ of a node S in the initial intermediate min-cut tree. Concerning this node S in $T_\circ(G_\alpha)$ the two subtrees N_b and N_d correspond to the clusters affected by the edge deletion and the node S consists of the remaining clusters in $\zeta(G)$ and the sink t (see Figure 6.2). In Line 19, Algorithm 11, a single call of the central Algorithm 12 regarding S would finally return an intermediate min-cut tree $T_\star(G_\alpha^\ominus)$ in which the sink t constitutes a singleton. So we could stop the execution of Algorithm 11 at this point and return the clustering $\zeta(G_\ominus)$ resulting from $T_\star(G_\alpha^\ominus)$ after deleting the sink t.

Input for new updating algorithm

The algorithm proposed now for updating a clustering $\zeta(G)$ in the case of an inter-cluster edge deletion basically abbreviates Algorithm 11 as just described. It is referred to as Algorithm 17. As input Algorithm 17 takes the set $\Theta(G)$ of non-crossing cuts in G_α, which isolate the sink t and define a clustering $\zeta(G)$ of G. To each cut in $\Theta(G)$ we assume the according weight to be assigned, and furthermore, a residual graph of a corresponding maximum flow according to the approach of Kohli and Torr. The set $W(G)$ stores the vertices that connect the clusters to the sink t. The two clusters containing the modified vertices b and d are denoted by C_b

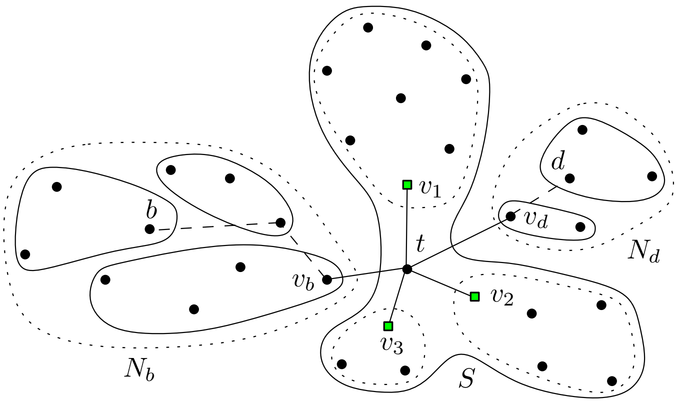


Figure 6.2: Initial min-cut tree $T_\circ(G_\alpha)$ in case of inter-cluster edge deletion.

and C_d, the related vertices in $W(G)$ are called v_b and v_d (compare to Figure 6.2). Analogously, θ_b and θ_d denote the related cuts in $\Theta(G)$. The remaining clusters, cuts and vertices in $\zeta(G)$, $\Theta(G)$ and $W(G)$, which are not affected by the modification, are just indexed by $1, \dots, z$.

Algorithm 17: Inter-Cluster Edge Deletion

```
   Input: $W(G)$, $\Theta(G)$ regarding $G = (V, E, c())$ with weights and residual
          graphs assigned, $G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c_\alpha^\ominus())$ with sink added and
          edge deleted, modified edge $\{b,d\}$ with weight $\Delta$
   Output: $W(G_\ominus)$, $\Theta(G_\ominus)$ regarding $G_\ominus$ with weights and residual graphs
           assigned
1  if $|\zeta(G)| = 2$ then
2      return $W(G)$, $\Theta(G)$ with weights and residual graphs assigned
3  else
       %---------- initialization ----------
4      $L(t) \leftarrow \emptyset$, $l(t) \leftarrow \emptyset$
5      for $i = 1, \dots, z$ do
6          Add $v_i$ to list $L(t)$    %decreasingly ordered
7          $D(v_i) \leftarrow \emptyset$
8      $\Theta(G_\ominus) \leftarrow \{\theta_b, \theta_d\}$
9      $W(G_\ominus) \leftarrow \{v_b, v_d\}$
       %-----------------------------------
10     $w(\min) \leftarrow \min\{c_\alpha(\theta_b), c_\alpha(\theta_d)\} - \Delta$
11     return Central-Clus $(W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, w(\min))$
```

In Line 11 Algorithm 17 reduces the loop of Algorithm 11, Line 17, to a single call of a modified central algorithm Central-Clus (see Algorithm 18) regarding the node S, which contains the sink t. According to the former algorithm Central-TreeDel (see Algorithm 12) this central algorithm considers new minimum v_i-t-cuts in the modified graph $G_\alpha^\ominus$ regarding the cut-vertices $v_1, \dots, v_z$ (see the (green) squares in Figure 6.2). These cut-vertices are related to the center t, and therefore, get stored in the list $L(t)$ in Line 6, Algorithm 17. According to the heuristic proposed by Flake et al. we further decreasingly order the vertices in $L(t)$ by the totaled weight of incident edges in graph G. The list $l(t)$ then stores the related cuts. Note, that the two vertices v_b and v_d are not added to the list $L(t)$. These vertices lie on the path γ, and hence, their new minimum separating cuts in $G_\alpha^\ominus$ are already known, as these are just the previous ones. Hence, if the input clustering $\zeta(G)$ consists of only two clusters, the input clustering $\zeta(G)$ is also a valid clustering for the modified graph $G_\ominus$, and the algorithm, therefore, returns $\zeta(G_\ominus) = \zeta(G)$ (see Line 1 and Line 2). Otherwise the known cuts θ_b and θ_d are added under reserve to the set $\Theta(G_\ominus)$ of non-crossing cuts finally defining the updated clustering $\zeta(G_\ominus)$ (see Line 8 and 9). The set $D(v_i)$ initialized in Line 7 for each cut-vertex v_i will store all those vertices included in $L(t)$ or $W(G_\ominus)$ that are separated from the sink t by the first minimum v_i-t-cut given by a maximum v_i-t-flow in the modified graph $G_\alpha^\ominus$ (see Line 18 and 21 in Algorithm 18).

Description of Inter-Cluster Edge Deletion algorithm

In the central algorithm CENTRAL-CLUS (see Algorithm 18) we accordingly assume the first minimum v_i-t-cut given by a maximum v_i-t-flow to be returned in Line 7. This minimum v_i-t-cut minimizes the side containing the cut-vertex v_i, and therefore, separates other vertices included in $L(t)$ or $W(G_\ominus)$ from the center t if and only if these vertices are separated from t by all minimum v_i-t-cuts in $G_\alpha^\ominus$, by Lemma 22. Furthermore, we apply the max-flow algorithm in Line 6 to a adjusted residual graph according to Kohli and Torr, instead of the complete modified graph $G_\alpha^\ominus$. Lemma 18 states a condition on which it is not necessary to calculate a new minimum v_i-t-cut. This condition is checked in Line 3 with the aid of $w(\min)$, which is accordingly initialized in Line 10 in Algorithm 17.

Description of Central-Clus algorithm

In Line 9 to Line 11 the central algorithm considers minimum v_i-t-cuts that turn out to remain of the same weight. Those cuts are not unfolded as in the previous central algorithm CENTRAL-TREEDEL, but instead added under reserve to the set $\Theta(G_\ominus)$ of non-crossing cuts defining the later updated clustering $\zeta(G_\ominus)$. If a nearest cut pair of such a remaining cut gets hidden by another minimum v_j-t-cut, the cut is removed again from $\Theta(G_\ominus)$ and just ignored in the following (compare to Line 19 to Line 21). Minimum v_i-t-cuts that get cheaper in the modified graph $G_\alpha^\ominus$ are stored in the list $l(t)$ and deleted if their nearest cut pair gets hidden by another minimum v_j-t-cut (see Line 13 to Line 18).

At the end, in Line 23 to Line 28, all cuts remaining in the list $l(t)$ are adjusted such that they neither cross their own treetops (which correspond to the clusters in $\zeta(G)$) nor each other, and again isolate the sink t. The so adjusted cuts are finally added to the set $\Theta(G_\ominus)$, which defines the updated clustering $\zeta(G_\ominus)$ for the modified graph $G_\ominus$.

Quality of the New Updating Algorithm

Our new algorithm for updating a clustering $\zeta(G)$ after an inter-cluster edge deletion guarantees that no previous cluster is split up in the new clustering $\zeta(G_\ominus)$. So Algorithm 17 finally returns a clustering $\zeta(G_\ominus)$ consisting of at most as many clusters as included in the input clustering $\zeta(G)$. A previous cluster is either merged with another cluster or again constitutes a cluster in $\zeta(G_\ominus)$. Thereby clusters are only merged if there is no other possibility concerning the cut-vertices defined by the input clustering. This holds, as the max-flow algorithm in Line 7 is supposed to return the first minimum v_i-t-cut given by a calculated maximum v_i-t-flow. By Lemma 22 hence a nearest cut pair $\{v_j, t\}$ gets hidden if and only if all minimum v_i-t-cuts in $G_\alpha^\ominus$ hide this pair.

Number of clusters

In this sense the updated clustering $\zeta(G_\ominus)$ returned by Algorithm 17 is as close as possible to the input clustering $\zeta(G)$. Moreover, if the previous clustering $\zeta(G)$ is also a valid clustering for the modified graph $G_\ominus$, our algorithm returns this clustering $\zeta(G_\ominus) = \zeta(G)$ under guarantee. The following Lemma 24 proves this assertion.

Similarity to previous clustering

Algorithm 18: CENTRAL-CLUS

```
Input: $W(G)$, $\Theta(G)$ regarding $G$ with weights and residual graphs assigned,
       initialized $W(G_\ominus)$, $\Theta(G_\ominus)$ regarding $G_\ominus$ with weights and residual
       graphs assigned, $G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b,d\}\}, c_\alpha^\ominus())$, minimum edge
       weight $w(\min)$
Output: Complete $W(G_\ominus)$, $\Theta(G_\ominus)$ regarding $G_\ominus$ with weights and residual
        graphs assigned
while $L(t)$ has next element $v_i$ do
    %---Calculate new minimum $v_i$-$t$-cuts
    Adjust residual graph related to $\theta_i$ according to Kohli and Torr
    if $c_\alpha(\theta_i) < w(min)$ then
        $\theta \leftarrow \theta_i$    %by Lemma 18
    else
        $f \leftarrow$ FLOWALGO $(v_i, t)$
        $\theta \leftarrow$ first minimum $v_i$-$t$-cut given by flow $f$
    Add $\theta$ to list $l(t)$    %in a valid relation to $v_i$
    if $c_\alpha^\ominus(\theta) = c_\alpha(\theta_i)$ then
        %---Consider cuts remaining of the same weight
        Move $v_i$ from list $L(t)$ to set $W(G_\ominus)$
        Move $\theta$ from list $l(t)$ to set $\Theta(G_\ominus)$    %in a valid relation to $v_i$
    else
        %---Consider cheaper cuts
        while $L(t)$ has next element $v_j \neq v_i$ do
            if $\theta$ separates $v_j$ and $t$ then
                Delete $v_j$ from list $L(t)$
                if $l(t)$ already contains a cut related to $v_j$ then
                    Delete cut related to $v_j$ from list $l(t)$
                    Add $v_j$ to set $D(v_i)$
        while $W(G_\ominus)$ has next element $v_j \neq v_i$ do
            if $\theta$ separates $v_j$ and $t$ then
                Move $v_j$ from set $W(G_\ominus)$ to set $D(v_i)$
                Delete cut related to $v_j$ from set $\Theta(G_\ominus)$
%---Adjust cheaper cuts to cluster-preserving cuts
%---$\theta = (R, V_\alpha \setminus R)$, $t \in R$, denotes cut in $l(t)$ related to $v_i$
while $L(t)$ has next element $v_i$ do
    $\theta \leftarrow (R \setminus C_i, (V_\alpha \setminus R) \cup C_i)$    %by partition-property
    forall vertices $v_j$ in $D(v_i)$ do
        $\theta \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$    %by case 1 and case 2
    forall vertices $v_j \neq v_i$ in $L(t)$ do
        $\theta \leftarrow (R \cup C_j, (V_\alpha \setminus R) \setminus C_j)$    %by case 3
Add all vertices in list $L(t)$ to set $W(G_\ominus)$
Add all related cuts in list $l(t)$ to set $\Theta(G_\ominus)$    %in a valid relation
return $W(G_\ominus)$, $\Theta(G_\ominus)$ with weights and residual graphs assigned
```

Number of max-flow calculations

Furthermore, the updating operation causes at most as many max-flow calculations as clusters are included in the input clustering $\zeta(G)$, minus the two clusters affected by the modification. The minimum separating cuts θ_b and θ_d of these clusters in G_α are already known to be also minimum separating cuts in the modified graph $G_\alpha^\ominus$. If the previous clustering remains the same, there are exact $|\zeta(G)| - 2$ max-flow calculations processed, with $|\zeta(G)|$ denoting the number of clusters included in $\zeta(G)$ (compare to Table 6.1, first row).

Lemma: Guarantee for unchanging clusterings

Lemma 24 *In the case of an inter-cluster edge deletion, our new algorithm again returns the previous clustering $\zeta(G)$ if and only if the previous clustering $\zeta(G)$ is also a valid clustering for the modified graph $G_\ominus$.*

Proof. ($\Rightarrow$): This assertion is trivial, as our algorithm returns correct results.

($\Leftarrow$): Assume the previous clustering $\zeta(G)$ to be also a valid clustering for the modified graph $G_\ominus$. To return a new clustering $\zeta(G_\ominus)$ different from $\zeta(G)$ our algorithm needed to find a new cheaper minimum v_i-t-cut for at least one cut-vertex $v_i \in \{v_1, \ldots, v_z\}$. As the previous clustering is supposed to be also valid for $G_\ominus$, therefore, there must exist another vertex u in cluster C_i such that the cut θ_i also constitutes a minimum u-t-cut in the modified graph $G_\alpha^\ominus$. Then there exists a min-cut tree $T(G_\alpha^\ominus)$ such that the edge-induced minimum v_i-t-cut represented in this new min-cut tree $T(G_\alpha^\ominus)$ does not separate the modified vertices b and d. This contradicts Lemma 9, which says that each new minimum v_i-t-cut in $G_\alpha^\ominus$ which is cheaper than the previous one in graph G_α needs to separate the modified vertices b and d. □

6.2.2 Intra-Cluster Edge Deletion

Situation in case of intra-cluster edge deletion

In the case of deleting an intra-cluster edge $e_\ominus = \{b, d\}$ the modified vertices b and d lie in the same cluster of graph G, which means, they lie in the same subtree of the sink t in a min-cut tree $T(G_\alpha)$. Again we do not know the tree structure of a min-cut tree $T(G_\alpha)$. We only know that there exists a min-cut tree $T(G_\alpha)$ that induces the input clustering $\zeta(G)$, which is given by the set $\Theta(G)$ of non-crossing cuts and the set $W(G)$ of the vertices related to these cuts. Again we additionally assume the according weights to be assigned to the cuts in $\Theta(G)$. As the modified vertices b and d lie in the same cluster, in this case the sink t does not lie on the unique path γ from b to d in the unknown, but existing, min-cut tree. Figure 6.3 exemplarily shows the sink t, the clusters bordered by dotted lines and the unknown path γ (dashed lines) concerning the min-cut tree introduced in Figure 6.1.

As we consider a case of edge deletion the initial intermediate min-cut tree $T_\circ(G_\alpha)$ for any algorithm developed in Chapter 4 would again result from contracting all edges in the unknown min-cut tree $T(G_\alpha)$ that do not lie on path γ. So the sink t would not become the representative of a node S in the initial intermediate min-cut tree, but constitutes a vertex in S off the path γ (while $r(S)$ lies on γ). In Figure 6.3 the nodes of $T_\circ(G_\alpha)$ are bordered by solid lines. Analog to the case of inter-cluster edge deletion $C_{(b,d)}$ denotes the cluster affected by the modification. The vertex $v_{(b,d)}$, which connects this cluster to the sink t, lies on the path from t to $r(S)$ in the unknown min-cut tree $T(G_\alpha)$. Concerning the initial intermediate min-cut tree $T_\circ(G_\alpha)$ the affected cluster $C_{(b,d)}$ covers the two subtrees N_b and N_d of S as well as the wood $\sharp$ of the edge $\{v_{(b,d)}, t\}$ (compare to Definition 14). The remaining clusters in $\zeta(G)$ together with the sink t form the treetop $\Uparrow$ of the edge $\{v_{(b,d)}, t\}$. Hence we only know the subtrees to be included in the affected cluster and we know the node S in parts, namely the remaining clusters and the sink t.

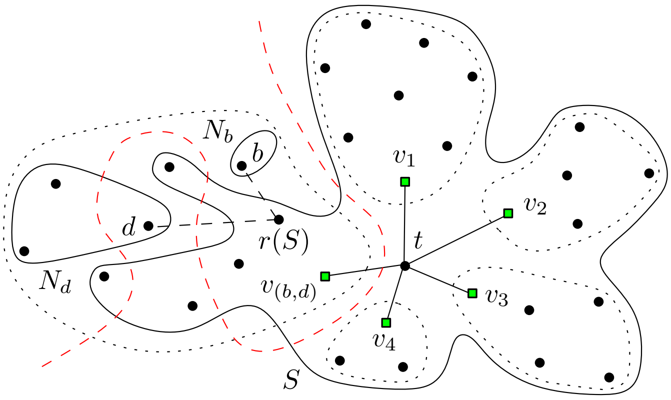


Figure 6.3: Initial min-cut tree $T_\circ(G_\alpha)$ in case of intra-cluster edge deletion.

Input for new updating algorithm

The algorithm proposed now for updating a clustering $\zeta(G)$ in the case of an intra-cluster edge deletion is referred to as Algorithm 19 and takes the same input as Algorithm 17, apart from the naming of the affected cluster $C_{(b,d)}$ and its related vertex $v_{(b,d)}$ and cut $\theta_{(b,d)}$.

Description of Intra-Cluster Edge Deletion algorithm

In contrast to an inter-cluster edge deletion in case of an intra-cluster edge deletion we do not know any edges on path γ, and therefore, none of the minimum separating cuts in the modified graph $G_\alpha^\ominus$ is known yet. So Algorithm 19 at first calculates a new minimum $v_{(b,d)}$-t-cut in $G_\alpha^\ominus$ (see Line 1 to Line 3) and checks in Line 4 whether this cut is of the same weight as the previous cut $\theta_{(b,d)}$ induced by edge $\{v_{(b,d)}, t\}$. If this is true, i.e., if $\theta_{(b,d)}$ is also a minimum $v_{(b,d)}$-t-cut in the modified graph $G_\alpha^\ominus$, it follows by Lemma 17 that the edges $\{v_i, t\}$, $i = 1, \dots, z$, connecting the remaining clusters to the sink t induce minimum v_i-t-cuts, $i = 1, \dots, z$, in $G_\alpha^\ominus$ as well. So in this case the clustering $\zeta(G)$ does not change, i.e., it holds that $\zeta(G) = \zeta(G_\ominus)$. The only thing still to be done is to adjust the residual graphs according to Kohli and Torr also for those cuts that are not newly calculated.

If the new minimum $v_{(b,d)}$-t-cut in $G_\alpha^\ominus$ is cheaper than the previous cut $\theta_{(b,d)}$, by Fact 3 we can adjust it such that it does not split the treetop $\Uparrow$ of the edge $\{v_{(b,d)}, t\}$ (see for example the dashed (red) cut in Figure 6.3). We further assume the first minimum $v_{(b,d)}$-t-cut given by a maximum t-$v_{(b,d)}$-flow to be returned in Line 3. So the originally calculated minimum $v_{(b,d)}$-t-cut minimizes the side containing the sink t, and therefore, the adjusted cut constitutes the best minimum $v_{(b,d)}$-t-cut in $G_\alpha^\ominus$ in the sense that it separates as many vertices from the sink t as possible while it preserves the treetop $\Uparrow$. Note, that the treetop then lies on the same side as the sink.

With this new, adjusted minimum $v_{(b,d)}$-t-cut we get the first potential cut in the set $\Theta(G_\ominus)$ of non-crossing cuts defining the later updated clustering $\zeta(G_\ominus)$ (compare to the cuts θ_b and θ_d in the case of inter-cluster edge deletion). So this cut is added under reserve to the set $\Theta(G_\ominus)$ in Line 15 and Line 16. As the treetop $\Uparrow$ is not split, by ignoring the vertices belonging to the wood $\sharp$ we then get a situation in node S which the central algorithm Central-Clus can be applied to (see Line 18). To this end in Line 13 the list $L(t)$ is initialized with the cut-vertices $v_1, \dots, v_z$ related to the center t (see (green) squares in Figure 6.3). According to the heuristic proposed by Flake et al. we further decreasingly order the vertices $v_1, \dots, v_z$ in $L(t)$ by the totaled weight of incident edges in graph G. Lemma 18, which states the condition for avoiding vain max-flow calculations with the aid of $w(\min)$, however, does not

Algorithm 19: Intra-Cluster Edge Deletion

```
   Input: $W(G)$, $\Theta(G)$ regarding $G = (V, E, c())$ with weights and residual
          graphs assigned, $G_\alpha^\ominus = (V_\alpha, E_\alpha \setminus \{\{b, d\}\}, c_\alpha^\ominus())$ with sink added and
          edge deleted, modified edge $\{b, d\}$ with weight $\Delta$
   Output: $W(G_\ominus)$, $\Theta(G_\ominus)$ regarding $G_\ominus$ with weights and residual graphs
           assigned
   %---Calculate new minimum $v_{(b,d)}$-$t$-cut $\theta$
1  Adjust residual graph related to $\theta_{(b,d)}$ according to Kohli and Torr
2  $f \leftarrow$ FlowAlgo $(t, v_{(b,d)})$
3  $\theta \leftarrow$ first minimum $v_{(b,d)}$-$t$-cut given by flow $f$
   %---Check if $\theta_{(b,d)}$ remains of the same weight
4  if $c_\alpha^\ominus(\theta) = c_\alpha(\theta_{(b,d)})$ then
5      for $i = 1, \dots, z$ do
6          Adjust residual graph related to $\theta_i$ according to Kohli and Torr
7      return $W(G)$, $\Theta(G)$ with weights and residual graphs assigned
8  else
       %---Adjust $\theta = (R, V_\alpha \setminus R)$, $t \in R$ not to split treetop
9      forall $C_i \neq C_{(b,d)}$ do
10         $\theta \leftarrow (R \cup C_i, (V_\alpha \setminus R) \setminus C_i)$   %by partition-property
       %---------- initialization ----------
11     $L(t) \leftarrow \emptyset$, $l(t) \leftarrow \emptyset$
12     for $i = 1, \dots, z$ do
13         Add $v_i$ to list $L(t)$   %decreasingly ordered
14         $D(v_i) \leftarrow \emptyset$
15     $\Theta(G_\ominus) \leftarrow \{\theta\}$
16     $W(G_\ominus) \leftarrow \{v_{(b,d)}\}$
       %-----------------------------------
17     $w(\min) \leftarrow 0$
18     $W(G_\ominus), \Theta(G_\ominus) \leftarrow$
         Central-Clus $(W(G), \Theta(G), W(G_\ominus), \Theta(G_\ominus), G_\alpha^\ominus, w(\min))$
19     Resolve all crossings in $\Theta(G_\ominus)$ by Lemma 15
20     Isolate the sink $t$
21     return $W(G_\ominus)$, $\Theta(G_\ominus)$ with weights and residual graphs assigned
```

hold anymore in this situation. So in Line 17 $w(\min)$ is initialized with zero as default value. The if-clause in Algorithm 18, Line 3, therefore, never becomes true.

After the execution of the central algorithm Central-Clus in Line 18, the set $\Theta(G_\ominus)$ consists of cuts which together separate the sink t from the rest of the treetop $\Uparrow$, which covers the clusters not affected by the modification, and from parts of the affected cluster $C_{(b,d)}$, but at least vertex $v_{(b,d)}$ (compare for example to the set of fat, dashed lines in Figure 6.4). So it might be necessary to finally isolate the sink t from some vertices (drawn as black bullets in Figure 6.4) belonging to the cluster $C_{(b,d)}$. Furthermore, at Line 19 the cuts included in $\Theta(G_\ominus)$ do neither cross within the treetop $\Uparrow$ nor do they split the unaffected clusters, by the cluster-preserving adjustment in Central-Clus. However they may cross somewhere out of the treetop $\Uparrow$, i.e., within the previous cluster $C_{(b,d)}$. Those crossings are resolved in Line 19, by Lemma 15.

Resolving crossings in the affected cluster

To finally isolate the sink t in Line 20 we need to choose new cut-vertices in $C_{(b,d)}$. At this point we again remember the heuristic advised by Flake et al. and

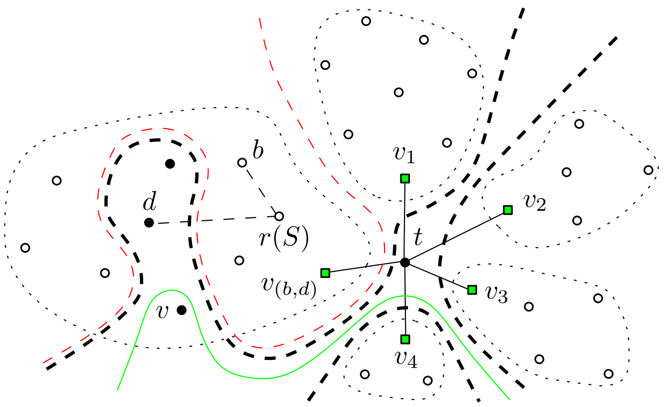


Figure 6.4: Example of a set $\Theta(G_\ominus)$ of non-crossing cuts at Line 20 in Algorithm 19.

```
Algorithm 20: Best Cut
Input: Set $W$, DAG-representation $DAG(v,t)$
Output: Best minimum $v$-$t$-cut in the sense described above
forall $u \in W$ do
    Mark node $X_u$ in $DAG(v,t)$ containing $u$
if $X_v$ is marked then
    return first minimum $v$-$t$-cut $(X_v, V_\alpha \setminus X_v)$
Start at node $X_v$
Merge all nodes that are not reachable from a marked node by searching per
  level and also marking all nodes that are reachable from a marked node
return minimum $v$-$t$-cut defined by the merged nodes
```

Finally isolating the sink t

accordingly order the vertices that are not separated from the sink t yet decreasingly by the totaled weight of their incident edges in G. Then we repeatedly calculate minimum v-t-cuts in the modified graph $G_\alpha^\ominus$, with vertex v being the next vertex in the decreasing order. Figure 6.4 exemplarily shows such a minimum v-t-cut as solid (green) line. To get *non-crossing* cuts in $\Theta(G_\ominus)$ in the end, each newly calculated minimum v-t-cut is adjusted such that it does not cross any cut already included in $\Theta(G_\ominus)$ before it is added to $\Theta(G_\ominus)$ under reserve. Such an adjustment is again feasible by Lemma 15. If the adjusted minimum v-t-cut separates the nearest cut pair of another cut already included in $\Theta(G_\ominus)$, this cut is replaced by the adjusted v-t-cut.

Calculating best minimum separating cuts

Concerning the new cut-vertices v in $C_{(b,d)}$ we can further calculate best minimum v-t-cuts in some sense. To this end consider a maximum v-t-flow and the related DAG-representation $DAG(v,t)$. Let X_v denote the node in $DAG(v,t)$ which contains the source v, and X_t contains the target t. Furthermore, imagine a set W which stores all those vertices in $\{v_1, \dots, v_z\}$ that are related to a new cut in the current set $\Theta(G_\ominus)$. Then the search given by Algorithm 20 starts at node X_v and returns in $O(n+m)$ time a minimum v-t-cut that maximizes the side containing v while it separates a vertex in W from the sink t if and only if all minimum v-t-cuts represented in the DAG-representation do so. So the returned cut is the best minimum v-t-cut in the sense that it separates as many vertices from the sink t as possible while it replaces as few cuts in $\Theta(G_\ominus)$ related to vertices of previously unaffected clusters as necessary.

The search given by Algorithm 20 is feasible, as each node on level i is adjacent to at least one node on the previous level $i-1$. If an unmarked node on level i is reachable from an initially marked node by a directed path, hence all nodes on lower levels on this path are already marked during the search at the time the node on level i is considered by the search. Figure 6.5 shows an example of a best minimum v-t-cut calculated by Algorithm 20 (see the dashed (green) line). The vertices in the set $W(G_\ominus)$ are drawn as (red) squares. The nodes in $DAG(v,t)$ that are initially marked in Line 2 are the dark-gray ones, the nodes that are marked during the search in Line 6 are bordered by dotted lines, but not filled. The order in which the nodes are considered during the search is given by the numbers assigned to the nodes.

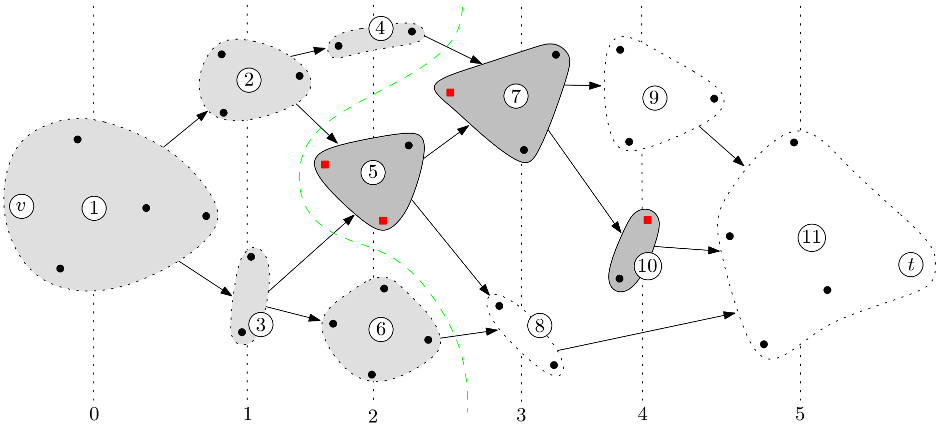


Figure 6.5: Example of a best minimum v-t-cut calculated by Algorithm 20.

Quality of the New Updating Algorithm

Similarity to previous clustering

Our new algorithm for updating a clustering $\zeta(G)$ after an intra-cluster edge deletion guarantees that no previous cluster, apart from the one affected by the deletion, is split up in the new clustering $\zeta(G_\ominus)$. Again this follows from the cluster-preserving adjustment of the cuts in $\Theta(G_\ominus)$. An unaffected cluster may be merged with other clusters while the affected cluster $C_{(b,d)}$ may be divided in different parts that are either also merged with previous clusters or become a new cluster in $\zeta(G_\ominus)$. Again clusters are only merged if there is no other possibility concerning the cut-vertices defined by the input clustering $\zeta(G)$. So the updated clustering $\zeta(G_\ominus)$ returned by Algorithm 19 is still quite close to the input clustering $\zeta(G)$.

Number of clusters

Furthermore, our updating algorithm tries to calculate as few maximum flows as necessary. In the case that the clustering does not change due to the edge deletion, our algorithm has the chance to detect this situation after only one max-flow calculation. Note, that it is, however, not ensured that our algorithm always detects this situation. We only can guarantee that the unaffected clusters $C_1,\dots,C_z$ also occur in the new clustering $\zeta(G_\ominus)$, which is proven by the following Lemma 25. So in the case that the previous clustering $\zeta(G)$ also constitutes a valid clustering for the modified graph $G_\ominus$ our algorithm returns a clustering $\zeta(G_\ominus)$ which has at least as many clusters as the previous one (compare to Table 6.1, second row). In the worst case there may occur as many max-flow calculations as vertices are included in the affected cluster $C_{(b,d)}$ plus the number of clusters in $\zeta(G)$. However, in this

Number of max-flow calculations

case it is very likely that a calculation of $\zeta(G_\ominus)$ with the aid of the cut-clustering method by Flake et al. needs about the same number of max-flow calculations, as both algorithms choose the selectable cut-vertices according to the same heuristic advised by Flake at al. Nevertheless, in general our algorithm tries to reduce the number of max-flow calculations by choosing best minimum v-t-cuts to finally isolate the sink t.

Lemma: Guarantee for unchanging clusterings

Lemma 25 *In the case that the previous clustering $\zeta(G)$ also constitutes a valid clustering for the modified graph $G_\ominus$ our algorithm returns a clustering $\zeta(G_\ominus)$ which contains the previous unaffected clusters $C_1, \dots, C_z$.*

Proof. Assume the previous clustering $\zeta(G)$ to be also a valid clustering for the modified graph $G_\ominus$. To return a new clustering $\zeta(G_\ominus)$ which does not contain all previous unaffected clusters $C_1, \dots, C_z$, our algorithm needed to find a new cheaper minimum v_i-t-cut for at least one cut-vertex $v_i \in \{v_1, \dots, v_z\}$. As the previous clustering is supposed to be also valid for $G_\ominus$, therefore, there must exist another vertex u in cluster C_i such that the cut θ_i also constitutes a minimum u-t-cut in the modified graph $G_\alpha^\ominus$. Then there exists a min-cut tree $T(G_\alpha^\ominus)$ such that the edge-induced minimum v_i-t-cut represented in this new min-cut tree $T(G_\alpha^\ominus)$ does not separate the modified vertices b and d. This contradicts Lemma 9, which says that each new minimum v_i-t-cut in $G_\alpha^\ominus$ which is cheaper than the previous one in graph G_α needs to separate the modified vertices b and d. □

6.3 Updating Algorithms for Edge Additions

The following two updating algorithms for the cases of edge additions are quite simple, compared to those regarding edge deletions. Both can be easily solved with the aid of Algorithm 14, the former algorithm for updating a complete min-cut tree after an edge addition. The case of intra-cluster edge addition is even trivial, as we will see in the following.

6.3.1 Intra-Cluster Edge Addition

Situation in case of intra-cluster edge addition

In the case of adding an intra-cluster edge $e_\oplus = \{b, d\}$ the modified vertices b and d lie in the same cluster of graph G, which means, they lie in the same subtree of the sink t in a min-cut tree $T(G_\alpha)$. As in the cases before we still do not know the tree structure of a min-cut tree $T(G_\alpha)$. We just know the clustering $\zeta(G)$ given by the set $\Theta(G)$ of non-crossing cuts and the set $W(G)$ of the vertices related to these cuts. The according weights are additionally assigned to the cuts in $\Theta(G)$. Nevertheless,

Algorithm 21: INTRA-CLUSTER EDGE ADDITION

```
Input: $W(G)$, $\Theta(G)$ regarding $G = (V, E, c())$ with weights and residual
       graphs assigned, $G_\alpha^\oplus = (V_\alpha, E_\alpha \setminus \{\{b, d\}\}, c_\alpha^\oplus())$ with sink and edge
       added, modified edge $\{b, d\}$ with weight $\Delta$
Output: $W(G_\oplus)$, $\Theta(G_\oplus)$ regarding $G_\oplus$ with weights and residual graphs
        assigned
forall cuts in $\Theta(G)$ do
    Adjust related residual graph according to Kohli and Torr
return $W(G)$, $\Theta(G)$ with weights and residual graphs assigned
```

we know that there exists a min-cut tree $T(G_\alpha)$ that induces the given clustering $\zeta(G)$. As the modified vertices b and d lie in the same cluster, sink t does again not lie on the unique path γ from b to d in the unknown, but existing, min-cut tree. Figure 6.6 exemplarily shows the sink t, the clusters bordered by dotted lines and the unknown path γ (dashed lines) concerning the min-cut tree introduced in Figure 6.1.

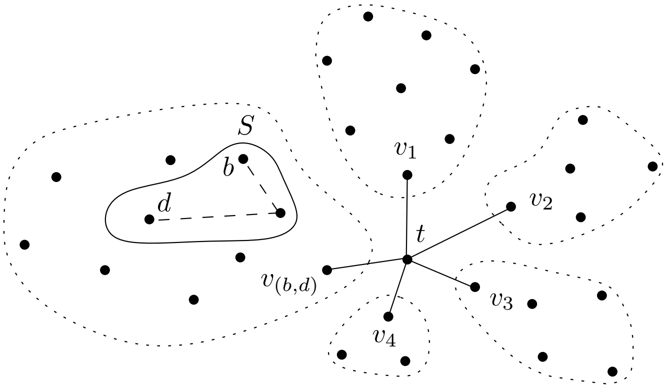


Figure 6.6: Initial min-cut tree $T_\circ(G_\alpha)$ in case of intra-cluster edge addition.

Now assume Algorithm 14 to be applied to the unknown, but existing, min-cut tree $T(G_\alpha)$. As we consider a case of edge addition the initial intermediate min-cut tree $T_\circ(G_\alpha)$ for this algorithm would hence result from contracting all edges in $T(G_\alpha)$ that lie on path γ. In Figure 6.6 the only node S containing more than one vertex in $T_\circ(G_\alpha)$ is bordered by a solid line. The remaining nodes are the singletons corresponding to the vertices off path γ.

Description of Intra-Cluster Edge Addition algorithm

As the sink t does not lie on γ, it is already isolated in the initial min-cut tree $T_\circ(G_\alpha)$. So we could stop the execution of Algorithm 14 without calling the central operation Central-Tree in Line 12. The updated clustering $\zeta(G_\oplus)$ resulted directly from the deletion of t in the intermediate min-cut tree $T_\circ(G_\alpha)$. Therefore, it is equivalent to the input clustering $\zeta(G)$. As our former algorithms for the cases of edge deletions use the method of Kohli and Torr to update individual minimum separating cuts, the only thing still to do is to accordingly adjust the residual graphs related to the cuts in $\Theta(G) = \Theta(G_\oplus)$. The new (trivial) algorithm is referred to as Algorithm 21.

6.3.2 Inter-Cluster Edge Addition

Situation in case of inter-cluster edge addition

In the case of adding an inter-cluster edge $e_\oplus = \{b, d\}$ the modified vertices b and d lie in different clusters of graph G, which means, they lie in different subtrees of the sink t in an unknown, but existing min-cut tree $T(G_\alpha)$. So the sink t lies on the path γ from b to d in $T(G_\alpha)$. The input clustering $\zeta(G)$ is given again by the set $\Theta(G)$, the set $W(G)$ and the according weights assigned to the cuts in $\Theta(G)$. Figure 6.7 exemplarily shows the sink t, the clusters bordered by dotted lines and the path γ concerning the min-cut tree introduced in Figure 6.1. The unknown edges on path γ are drawn as dashed lines.

As we again consider a case of edge addition the initial intermediate min-cut tree $T_\circ(G_\alpha)$ for Algorithm 14 would again result from contracting all edges in $T(G_\alpha)$ that lie on path γ. In Figure 6.7 the only node S containing more than one vertex

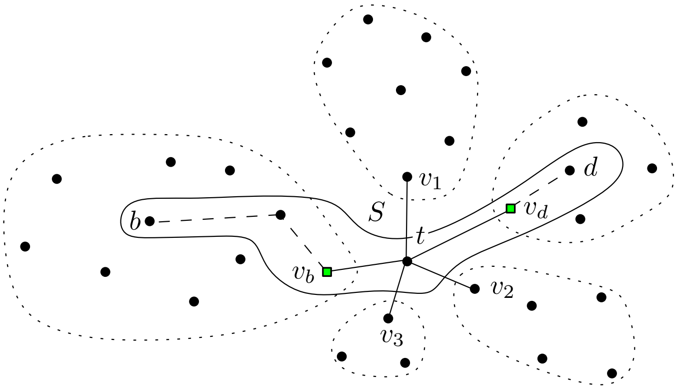


Figure 6.7: Initial min-cut tree $T_\circ(G_\alpha)$ in case of inter-cluster edge addition.

in $T_\circ(G_\alpha)$ is bordered by a solid line. The singletons corresponding to the vertices off the path γ constitute the remaining nodes in the initial min-cut tree.

Description of Inter-Cluster Edge Addition algorithm

Our new updating algorithm is now referred to as Algorithm 22. In contrast to the case of inter-cluster edge deletion, in the case of inter-cluster edge addition we know those cuts in $\Theta(G)$ to remain minimum separating cuts in the modified graph $G_\alpha^\oplus$ that induce the unaffected clusters $C_1, \ldots, C_z$. Therefore, in Line 4 Algorithm 22 initializes the set $W(G_\oplus)$ with the related vertices $v_1, \ldots, v_z$ and the set $\Theta(G_\oplus)$ with the cuts $\theta_1, \ldots, \theta_z$ (see Figure 6.7). Furthermore, as our former algorithms for the cases of edge deletions use the method of Kohli and Torr to update individual minimum separating cuts, Algorithm 22 accordingly adjusts the residual graphs related to these cuts. After the initialization the non-crossing cuts in $\Theta(G_\oplus)$ hence separate the sink t from all unaffected clusters (compare to the set of fat, dashed lines in Figure 6.8). So the sink t still needs to be isolated from the two affected clusters, which cover the path γ. To this end Algorithm 22 calculates new minimum separating cuts regarding the two cut-vertices v_b and v_d (see the (green) squares in Figure 6.8) and checks whether these new cuts are of the same weight as the previous cuts after the addition of Δ. According to the heuristic advised by Flake et al. [FTT04], in Line 1 the vertices v_b and v_d are ordered decreasingly by the total weight of their incident edges in G. In Line 10 we then assume the best minimum separating cut to be returned (by Algorithm 20) regarding the vertices $v_1, \ldots, v_z$ together with the other vertex remaining in $\{v_b, v_d\}$ as being blocked. If one of the cut-vertices in $\{v_b, v_d\}$ gets separated from the sink t by the new cut related to the other cut-vertex, the separated cut-vertex is ignored in the remainder of the algorithm (see Line 16 to Line 20).

If the new minimum separating cuts are of the same weight as the previous ones after the addition of Δ, they are replaced by the previous cuts and added to the set $\Theta(G_\oplus)$ under reserve (see Line 12 to Line 14). Otherwise, the algorithm stores the cut-vertices that are separated from the sink t by these new cuts in the set $D(v_b)$ and the set $D(v_d)$ (see Line 24) and then adjusts the new cuts in Line 25 to Line 29 such that they do not split the previously unaffected clusters, by Theorem 3. Figure 6.8 shows a new minimum v_d-t-cut, drawn as solid (red) line, which separates the cut-vertex v_1 from the sink t.

At Line 30 the new separating cuts in the list $l(t)$ may still cross somewhere within the affected clusters C_b and C_d (compare to the solid (red) lines in Figure 6.8). However, this can be easily resolved by Lemma 15. The so adjusted non-crossing cuts are then added to the set $\Theta(G_\oplus)$. In Line 33 it finally might be necessary to

Algorithm 22: INTER-CLUSTER EDGE ADDITION

```
Input: $W(G)$, $\Theta(G)$ regarding $G = (V, E, c())$ with weights and residual
       graphs assigned, $G_\alpha^\oplus = (V_\alpha, E_\alpha \setminus \{\{b, d\}\}, c_\alpha^\oplus())$ with sink and edge
       added, modified edge $\{b, d\}$ with weight $\Delta$
Output: $W(G_\oplus)$, $\Theta(G_\oplus)$ regarding $G_\oplus$ with weights and residual graphs
        assigned
Store $v_b$ and $v_d$ in $u_1, u_2$ decreasingly ordered by
the total weight of their incident edges
%---------- initialization ----------
$L(t) \leftarrow \{u_1, u_2\}$, $l(t) \leftarrow \emptyset$
$D(u_1) \leftarrow \emptyset$, $D(u_2) \leftarrow \emptyset$
$W(G_\oplus) \leftarrow \{v_1, \ldots, v_z\}$, $\Theta(G_\oplus) \leftarrow \{\theta_1, \ldots, \theta_z\}$
forall cuts in $\Theta(G_\oplus)$ do
    Adjust related residual graph according to Kohli and Torr
%------------------------------------
%---Calculate best new minimum separating cuts
while $L(t)$ has next element $u_i$ do
    Adjust residual graph related to $\theta_i^u$ according to Kohli and Torr
    $f \leftarrow$ FLOWALGO $(u_i, t)$
    $\theta \leftarrow$ best cut given by flow $f$   %regarding vertices in $W(G_\oplus)$
     and $u_j$ $(j \neq i)$ as being blocked
    Add $\theta$ to list $l(t)$   %in a valid relation to $u_i$
    if $c_\alpha^\oplus(\theta) = c_\alpha(\theta_i^u) + \Delta$ then
        Move $u_i$ from list $L(t)$ to set $W(G_\oplus)$
        Move $\theta$ from list $l(t)$ to set $\Theta(G_\oplus)$
    else
        while $L(t)$ has next element $u_j \neq u_i$ do
            if $\theta$ separates $u_j$ and $t$ then
                Delete $u_j$ from list $L(t)$
                if $l(t)$ already contains a cut related to $u_j$ then
                    Delete cut related to $u_j$ from list $l(t)$
        while $W(G_\oplus)$ has next element $v_i$ do
            if $\theta$ separates $v_i$ and $t$ then
                Delete cut related to $v_i$ from set $\Theta(G_\oplus)$
                Move $v_i$ from set $W(G_\oplus)$ to set $D(u_i)$
%-- Adjust best new cuts to cluster-preserving cuts
%-- $\theta = (R, V_\alpha \setminus R)$, $t \in R$, denotes cut in $l(t)$ related to $u_i$
while $L(t)$ has next element $u_i$ do
    forall vertices $v_j$ in $D(u_i)$ do
        $\theta \leftarrow (R \setminus C_j, (V_\alpha \setminus R) \cup C_j)$   %by Theorem 3
    forall vertices $v_j$ in $W(G_\oplus) \setminus D(u_i)$ do
        $\theta \leftarrow (R \cup C_j, (V_\alpha \setminus R) \setminus C_j)$   %by Theorem 3
Resolve all crossings in list $l(t)$   %by Lemma 15
Add all vertices in list $L(t)$ to set $W(G_\oplus)$
Add all (non-crossing) cuts in list $l(t)$ to set $\Theta(G_\oplus)$
%in a valid relation
Isolate the sink $t$
return $W(G_\oplus)$, $\Theta(G_\oplus)$ with weights and residual graphs assigned
```

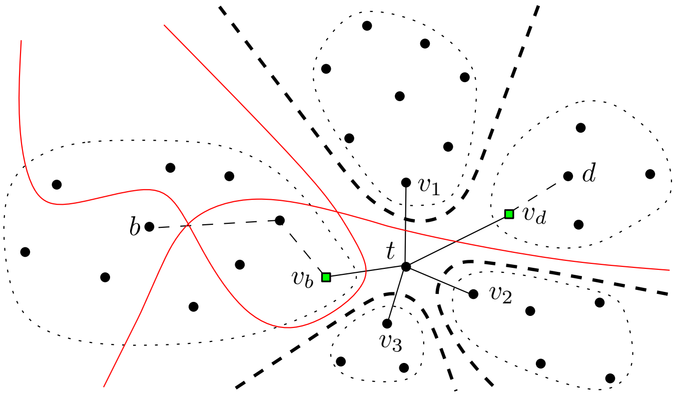


Figure 6.8: Example of a set $\Theta(G_\oplus)$ after initialization.

isolate the sink t from some remaining vertices. This is done as already described for the case of inter-cluster edge deletion in Subsection 6.2.2.

Quality of the New Updating Algorithm

Similarity to previous clustering

Our new algorithm for updating a clustering $\zeta(G)$ after an inter-cluster edge addition guarantees that no previous cluster, apart from the two clusters affected by the addition, is split up in the new clustering $\zeta(G_\oplus)$. This follows from the cluster-preserving adjustment of the new minimum separating cuts regarding the vertices v_b and v_d. The affected clusters C_b and C_d may be divided in different parts that are either merged with previous unaffected clusters or become a new cluster in $\zeta(G_\oplus)$. By calculating the best cut in Line 10 we try to merge as few previous clusters as necessary with parts of the affected clusters and at the same time to separate as many vertices covered by $C_b \cup C_d$ as possible from the sink t. So the updated clustering $\zeta(G_\oplus)$ returned by Algorithm 2 is again quite close to the input clustering $\zeta(G)$, while the algorithm tries to calculate as few maximum flows as necessary.

Number of max-flow calculations

In the case that the clustering does not change due to the edge addition, our algorithm has the chance to detect this situation after only two max-flow calculations regarding the vertices v_b and v_d. Note, that neither is there a guarantee that our algorithm always detects this situation nor can we guarantee any further, apart from the fact that the unaffected clusters do not get split up in the new clustering $\zeta(G_\oplus)$ (compare to Table 6.1, last row). In the worst case there may occur as many calculations as vertices are included in the affected clusters C_b and C_d. However, in this case it is again very likely that a calculation of $\zeta(G_\oplus)$ with the aid of the cut-clustering method by Flake et al. needs about the same number of max-flow calculations, as both algorithms choose the selectable cut-vertices according to the same heuristic advised by Flake at al [FTT04]. Nevertheless, in general our algorithm tries to reduce the number of max-flow calculations by choosing best minimum v-t-cuts to finally isolate the sink t.

Summary of quality analysis

The numbers of max-flow calculations occurring in the differnt cases of edge addition and edge deletion are again outlined in Table 6.1. To this end we list the number of max-flow calculations in the worst case regarding the different modifications in general (see first column). Furthermore, we consider the special situation that the previous clustering $\zeta(G)$ also constitutes a valid clustering after the modification. Concerning this situation we give a lower and upper bound of max-flow calculations and of clusters included in the new clustering $\zeta(G_{\oplus(\ominus)})$. The number of vertices in a cluster C is denoted by $|C|$, the number of clusters in a clustering $\zeta(G)$ is denoted by $|\zeta(G)|$.

	Worst Case [flows]	**Case of Unchanging Clustering**		
		Lower Bound	**Upper Bound**	**Guarantee**
Inter Del	$\lvert\zeta(G)\rvert - 2$	$\lvert\zeta(G)\rvert - 2$ [flows]	$\lvert\zeta(G)\rvert - 2$ [flows]	Yes
		$\lvert\zeta(G)\rvert$ [clusters]	$\lvert\zeta(G)\rvert$ [clusters]	
Intra Del	$\lvert\zeta(G)\rvert + \lvert C_{(b,d)}\rvert - 1$	1 [flow]	$\lvert\zeta(G)\rvert + \lvert C_{(b,d)}\rvert - 1$ [flows]	No Chance: 1 [flow]
		$\lvert\zeta(G)\rvert$ [cluster]	$\lvert\zeta(G)\rvert + \lvert C_{(b,d)}\rvert - 1$ [clusters]	
Intra Add	0	0 [flows]	0 [flows]	Yes
		$\lvert\zeta(G)\rvert$ [clusters]	$\lvert\zeta(G)\rvert$ [clusters]	
Inter Add	$\lvert C_b\rvert + \lvert C_d\rvert$	1 [flow]	$\lvert C_b\rvert + \lvert C_d\rvert$ [flows]	No Chance: 2 [flows]
		1 [cluster]	$\lvert\zeta(G)\rvert + \lvert C_b\rvert + \lvert C_d\rvert - 2$ [clusters]	

Table 6.1: Bounds of max-flow calculations and clusters.

6.3.3 Bow to the Approach of Saha and Mitra

At the beginning of this work, in Chapter 2, we discussed the approach of Saha and Mitra [SM06] for dynamically updating clusterings after the addition of an inter-cluster edge to the underlying graph G. Apart from a massive methodical error, we illustrated that Saha and Mitra apply one lemma, namely the merging lemma, on invalid conditions, while they use another lemma, namely the unaffect lemma, to prove the correctness of their algorithmic approach in a way that is not feasible. Therefore, we announced in Subsection 2.3.2 that we will give a reasonable interpretation of the unaffect lemma later. Now in this subsection we first review the unaffect lemma and then interpret it according to the knowledge we gained by developing our updating algorithm regarding an inter-cluster edge addition.

Lemma: Review of the unaffect lemma

Lemma 26 *(Unaffect lemma) Let C_b and C_d be two clusters in a clustering $\zeta(G)$ resulting from the cut-clustering method of Flake et al. [FTT04]. If there are some insertions and deletions of edges across and within the clusters C_b and C_d of the dynamic graph G, then all clusters in $\zeta(G) \setminus \{C_b, C_d\}$ "remain unaffected".*

Saha and Mitra do not prove this lemma properly, but use it to develop the algorithmic idea of their inter-edge-add algorithm (see Algorithm 2). This idea bases on the assumption that the expression "remain unaffected" in Lemma 26 means that each cluster C in $\zeta(G) \setminus \{C_b, C_d\}$ again constitutes a single cluster in a correctly updated clustering $\zeta(G_\oplus)$ after the addition of an edge $e_\oplus = \{b, d\}$. Therefore, Saha and Mitra contract all these "unaffected" clusters to one big node in an intermediate graph G'_α and then calculate a new min-cut tree of G'_α (compare to *CASE 3* in Algorithm 2). We disproved this approach in Subsection 2.3.2.

New interpretation of the unaffect lemma

Nevertheless, the assertion of the unaffect lemma becomes true if we interpret the expression "remain unaffected" in the sense that a cluster in $\zeta(G) \setminus \{C_b, C_d\}$ might be merged with other clusters, but never gets split up in the new clustering $\zeta(G_\oplus)$. This follows from the cluster-preserving adjustment as it is used for the non-crossing cuts that define the updated clustering $\zeta(G_\oplus)$ in our approach. Roughly spoken, our algorithm contracts the "unaffected" clusters individually in graph G_α, which already contains the artificial sink t, and therefore, allows these clusters to be reconnected as subtrees by the Gomory-Hu method finally applied. By the way, our trivial updating algorithm in case of an inter-cluster edge addition is basically the same as the one mentioned by Saha and Mitra [SM06]. So for this case Saha and Mitra actually developed a feasible solution.

Proof of the New Merging Lemma

In Subsection 2.3.1 we omitted the proof of the new merging lemma, as for this proof we use the following equivalence which we did not know by then in Chapter 2.

Observation: Equivalence of min-cut tree and clustering existence

Observation 3 *Consider a clustering $\zeta(G)$ of a graph G and a fixed parameter α. Then the existence of a min-cut tree $T(G_\alpha)$ inducing the clustering $\zeta(G)$ is equivalent to the existence of a vertex y in each cluster $C \in \zeta(G)$ such that $(C, V_\alpha \setminus C)$ constitutes a minimum y-t-cut in the enlarged graph G_α (compare to Figure 6.1).*

We now review and then prove the new merging lemma introduced as Lemma 5 in Subsection 2.3.1. The cuts mentioned in the new merging lemma are defined as follows: Cut $\theta_{\min}(b) := (C_b, V_\alpha \setminus C_b)$ is a minimum y_b-t-cut in the enlarged graph G_α for a vertex y_b in the cluster C_b affected by a later addition of an edge $e_\oplus = \{b, d\}$. It separates cluster C_b from the sink t. Cut $\theta_{\min}(d) := (C_d, V_\alpha \setminus C_d)$ is defined analogously for a vertex y_d in cluster C_d. The cut $\theta_{(b,d)} := (C_b \cup C_d, V_\alpha \setminus (C_b \cup C_d))$ separates both clusters $C_b \cup C_d$ from the sink t in G_α.

Lemma: Review of the new merging lemma

Lemma 27 *(New merging lemma) On the new condition that $c_\alpha(\theta_{(b,d)}) = c_\alpha(\theta_{min}(b)) = c_\alpha(\theta_{min}(d))$ merging cluster C_b and C_d of the input clustering $\zeta(G)$ yields a clustering $\zeta(G_\oplus)$ of the modified graph $G_\oplus$ that again results from the cut-clustering method, and therefore, also respects the clustering quality.*

Proof. According to Observation 3 we need to show that in each cluster C in a clustering $\zeta(G_\oplus)$ resulting from merging C_b and C_d in the previous clustering $\zeta(G)$ there exists a vertex y such that $(C, V_\alpha \setminus C)$ is a minimum y-t-cut in $G_\alpha^\oplus = (V_\alpha, E_\alpha \cup \{b, d\}, c_\alpha^\oplus())$. As the input clustering $\zeta(G)$ is assumed to result from the cut-clustering method, each cluster $C \in \zeta(G)$ meets this condition. Furthermore, each unaffected cluster $C \in \zeta(G)$ also constitutes a cluster in the new clustering $\zeta(G_\oplus)$, and each minimum y-t-cut in G_α that does not separate the modified vertices b and d is also a minimum y-t-cut in the modified graph $G_\alpha^\oplus$, by Lemma 8. This is, in each unaffected cluster $C \in \zeta(G)$, which also constitutes a cluster in $\zeta(G_\oplus)$, there exists a vertex y such that $(C, V_\alpha \setminus C)$ is a minimum y-t-cut in $G_\alpha^\oplus$. So we just need to find a vertex $\bar{y}$ in $\bar{C} = C_b \cup C_d$ such that the cut $\theta_{(b,d)} = (\bar{C}, V_\alpha \setminus \bar{C})$ is a minimum $\bar{y}$-t-cut in $G_\alpha^\oplus$.

Let $y_b \in C_b$ and $y_d \in C_d$ denote the vertices concerning the minimum y_b-t-cut $(C_b, V_\alpha \setminus C_b)$ and the minimum y_d-t-cut $(C_d, V_\alpha \setminus C_d)$ in G_α. As the cut $\theta_{(b,d)}$ does not separate the modified vertices b and d, the weight of this cut is also not affected by the addition of $e_\oplus = \{b, d\}$. So $\theta_{(b,d)}$ has the same weight in both graphs G_α and $G_\alpha^\oplus$. As we assume that $c_\alpha(\theta_{(b,d)}) = c_\alpha(\theta_{\min}(b))$ the cut $\theta_{(b,d)}$ is a minimum y_b-t-cut in G_α as well as in $G_\alpha^\oplus$, by Lemma 8. So $\theta_{(b,d)}$ meets the required condition for the vertex $\bar{y} := y_b$. $\square$

Chapter 7

Experimental Analysis

This chapter considers two experiments to affirm the theoretically predicted behavior of the updating approach developed in this work. The first experiment bases on the friendship network of Zachary [Zac77] and serves to illustrate some special situations which may occur while updating clusterings with our new algorithms. The second experiment concentrates on a dynamic real-wold e-mail graph. In this experiment we compare the performance of our approach to a simple repeated clustering calculation from scratch using the heuristic recommended by Flake et al. [FTT04] (compare to Section 6.1). Note, that this method is a heuristic regarding the number of calculated maximum flows, i.e., this number, and thus the performed effort, might not be minimal. Nevertheless, it always returns a correct result. We call this heuristic *cut-clustering heuristic* and shortly review it in Algorithm 23.

Algorithm 23: Cut-Clustering Heuristic

```
Input: Graph $G = (V, E, c())$, parameter $\alpha$ of clustering quality
Output: Clustering $\zeta(G)$ with clustering quality depending on $\alpha$
$V_\alpha \leftarrow V \cup \{t\}$ %add artifical sink $t$
$E_\alpha \leftarrow E \cup \{\{t, v\} | v \in V\}$ %connect $t$ to each vertex $v$ in $V$
$\forall e \in E : c_\alpha(e) \leftarrow c(e)$
$\forall v \in V : c_\alpha(\{t, v\}) \leftarrow \alpha$ %each edge incident with $t$ gets weight $\alpha$
$L \leftarrow V$ decreasingly ordered by the totaled weight of incident edges
$T_\star(G_\alpha) \leftarrow (\{V_\alpha\}, \emptyset, c_\alpha())$ %initialize intermediate min-cut tree
$S \leftarrow V_\alpha$
$r(S) \leftarrow t$ %choose representative
while $t$ is not isolated yet in $T_\star(G_\alpha)$ do
    $S \leftarrow$ node containing the sink $t$
    $u \leftarrow$ next vertex in $L$ included in $S$
    $\theta \leftarrow$ minimum $u$-$t$-cut in $G_\alpha(S)$
    $T_\star(G_\alpha) \leftarrow$ SplitAndReconnect $(T_\star(G_\alpha), S, u, \theta)$
    %see Algorithm 13
Remove $t$ from $T_\star(G_\alpha)$
$\zeta(G) \leftarrow$ set of connected components resulting from the removal of $t$
return $\zeta(G)$
```

Also clusterings of an unconnected graph G can easily be updated, as the subsequently added artificial sink connects all components at any time. The insertion of a vertex in graph G causes the addition of a new cluster in the current clustering

$\zeta(G)$. In the enlarged graph G_α the inserted vertex hence gets connected to the sink t by an edge of weight α. The removal of a vertex only is allowed if the vertex is unconnected in the underlying graph G, i.e., if it is only adjacent to the sink t in G_α. In this situation the considered vertex constitutes a single cluster in clustering $\zeta(G)$, and therefore, can easily be deleted. In both cases this procedure yields a correctly updated clustering, as shown in Chapter 4, Subsection 4.1.2.

7.1 Zachary's Friendship Network

Fixed underlying graph

In this experiment we consider the graph G illustrated in Figure 7.1, which constitutes the friendship network of Zachary. The vertices in this network represent individuals which cultivate several friendships to others, so friends are accordingly connected by edges. As this experiment is not meant to provide a systematic evaluation, but a proof of concept, and because clustering is more interesting regarding graphs with differently weighted edges, we further modify the network by assigning weights regarding the following cost function: $c(\{u,v\}) := deg(u) + deg(v)$, with $deg()$ denoting the degree of a vertex in the original network. This cost function bases on the assumption that an individual which has many friends serves as a kind of leader, and therefore, a friendship to such a leader has a higher importance than a friendship to an individual which has less friends. For a better readability, however, we omit showing the weights in Figure 7.1.

Fixed initial clustering

Due to the additionally assigned weights in the network, the cut-clustering heuristic of Flake et al. identifies, apart from a few singletons, two main groups in graph G. The resulting clustering $\zeta(G)$ regarding the parameter $\alpha = 10.5$ consists of 17 clusters and is also shown in Figure 7.1. Each trapezoid vertex constitutes a single cluster (for those singletons no color is defined), the round (green) vertices form a group to the left, while the square (red) vertices present another group to the right. The smaller, right group seems to depend on character 33 as a kind of leader. By contrast, the bigger, left group is dominated by two leaders, namely character 0 and character 1, which are both highly connected to the members of this group.

Choice of parameter α

For a discussion about how to chose a reasonable parameter α see [FTT04], Section 3.3, page 393. If α goes to zero, the minimum separating cut between the sink t and any other vertex in the underlying graph G is trivial, as it just isolates the sink t from all vertices in G. In this case the resulting clustering consists of only one big cluster. Vice versa, for α going to infinity, the min-cut tree $T(G_\alpha)$ of the enlarged graph G_α becomes a star with the sink t at its center. Then the resulting clustering contains each vertex of graph G as a single cluster. As a reasonable value between these two trivial cases depends on the structure of graph G, Flake et al. advise a binary search-like approach to determine the parameter α. For many graphs, and in particular for the graphs considered in our experiments, this technique works quite well. Furthermore, Flake et al. recommend another approach to chose α in order not to calculate a whole clustering, but to find one cluster with special properties.

In the following we assume different, independent events to occur in the network shown in Figure 7.1 and analyze how our new updating algorithms behave in the resulting situation compared to the cut-clustering heuristic. Thereby, we consider the graph G with its edges weighted as described above and the related clustering $\zeta(G)$ shown in Figure 7.1 as fixed initial instance for each event. For the calculation of the initial clustering $\zeta(G)$ the cut-clustering heuristic needs 17 max-flow calculations, i.e., it calculates as many minimum separating cuts as the number of clusters that are finally included in $\zeta(G)$.

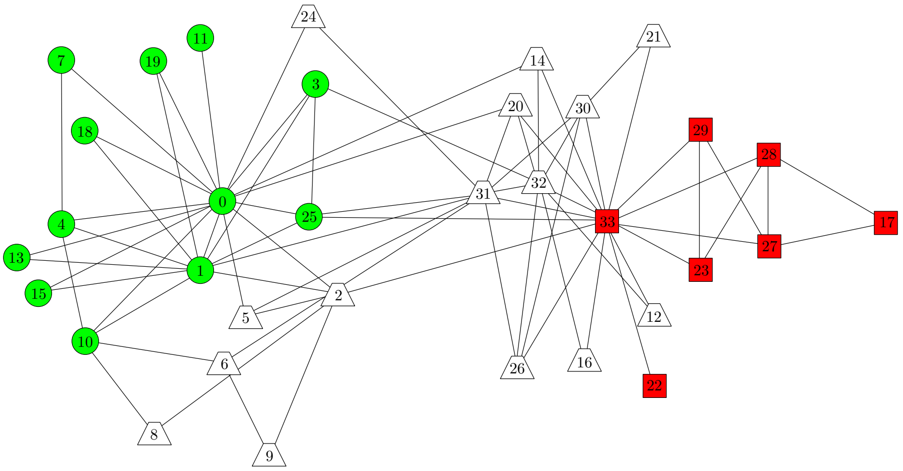


Figure 7.1: Friendship network with a clustering regarding $\alpha = 10.5$. The trapezoid, void vertices represent singletons in the clustering.

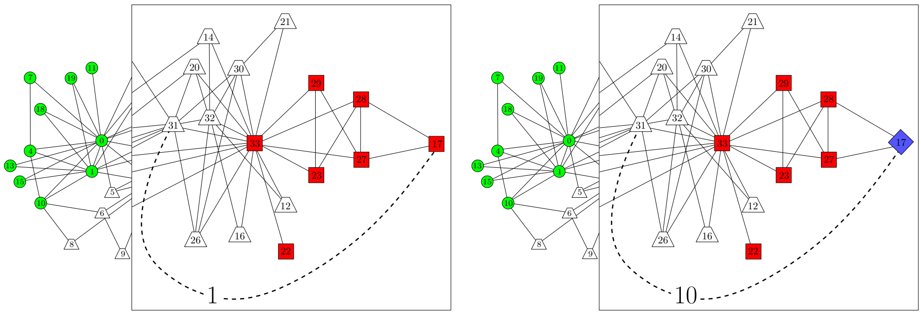
7.1.1 Exemplary Inter-Cluster Edge Additions

Unchanging clustering

At first we consider events which yield an inter-cluster edge addition in the underlying graph G. So assume for example the addition of an edge of weight one between character 17 in the right group and the single character 31 (see Figure 7.2a). The cut-clustering heuristic applied to the so modified graph $G_\alpha^\oplus$ returns the same clustering $\zeta(G_\oplus) = \zeta(G)$ as before and thereby again calculates 17 minimum separating cuts. In contrast, our Inter-Cluster Edge Addition algorithm (see Algorithm 22) at first checks whether the two cuts previously defining the affected clusters still constitute minimum separating cuts in the modified graph $G_\alpha^\oplus$. By doing so, it realizes that the previous minimum 17-t-cut as well as the previous minimum 31-t-cut effectively remains a minimum separating cut in $G_\alpha^\oplus$, and therefore, the previous clustering $\zeta(G)$ also constitutes a valid clustering for the modified graph $G_\oplus$. So this is an example for the Inter-Cluster Edge Addition algorithm using the chance listed in the fourth row of Table 6.1. Note, that both characters 17 and 31 accidentally constitute the cut-vertices connecting the according clusters to the sink t. Hence, in this example both algorithms finally return the same clustering $\zeta(G_\oplus)$, but our new algorithm only needs two max-flow calculations, while the cut-clustering heuristic again calculates 17 minimum separating cuts.

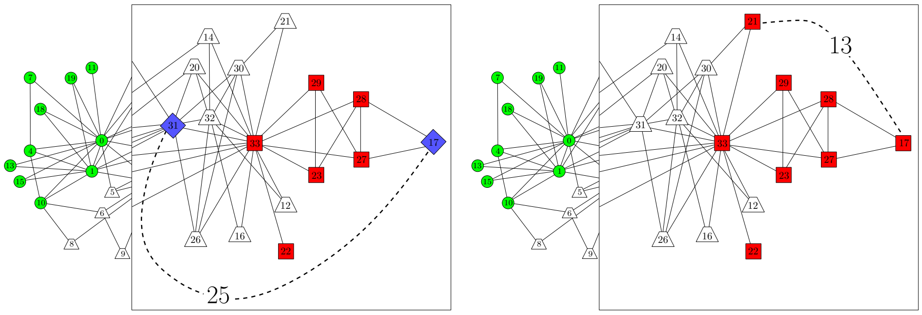
Increasing number of clusters

As another event imagine the addition of an edge with weight 10 between character 17 and character 31. In this case the cut-clustering heuristic returns a clustering $\zeta(G_\oplus)$ of now 18 clusters, in which character 17 has left its previous group due to the slightly stronger connection to character 31 (vertex 17 has become a (blue) rhombus in Figure 7.2b). For the calculation of this clustering the heuristic needs 18 max-flow calculations. Our algorithm returns the same updated clustering $\zeta(G_\oplus)$, but already finishes after 3 max-flow calculations. The sequence of cut-vertices considered by our algorithm is $17, 31, 22$. The new minimum 17-t-cut just separates 17 from the remaining vertices, and the minimum 31-t-cut turns out to be the same as before. Vertex 22 then is the next cut-vertex in the decreasing order by the totaled weight of incident edges. Remember, that the unaffected clusters are not considered, as their related cuts are already known not to change. The new minimum 22-t-cut finally isolates the sink t and the algorithm stops.



(a) Edge $\{17, 31\}$ with weight one causes 17 clusters and 2 max-flow calculations

(b) Edge $\{17, 31\}$ with weight 10 causes 18 clusters and 3 max-flow calculations



(c) Edge $\{17, 31\}$ with weight 25 causes 17 clusters and 2 max-flow calculations

(d) Edge $\{17, 21\}$ with weight 13 causes 16 clusters and 1 max-flow calculation

Figure 7.2: Four special inter-cluster edge additions.

Cluster foundation

Assume a new edge with weight 25 to be added between character 17 and character 31 (see Figure 7.2c). Both clustering algorithms in this case return a new clustering $\zeta(G_\oplus)$ of again 17 clusters, but with character 17 founding a new cluster together with character 31 (see the rhombic (blue) vertices in Figure 7.2c). The cut-clustering heuristic again needs 17 max-flow calculations, while updating the previous clustering $\zeta(G)$ only costs two max-flow calculations.

Decreasing number of clusters

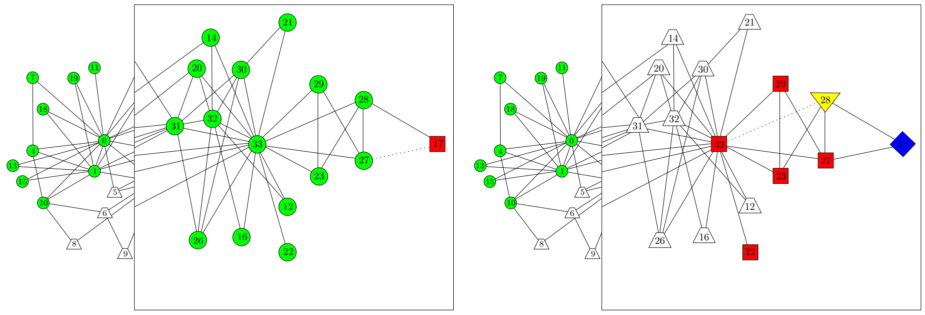
If character 17 becomes connected to the single character 21 by a new edge of weight 13, this edge addition would yield a new clustering $\zeta(G_\oplus)$ with 16 clusters, as the new edge commits the single vertex 21 to the group already including vertex 17 (see the square (red) vertices in Figure 7.2d). The cut-clustering heuristic in this example uses 16 max-flow calculations. The new updating algorithm only calculates one minimum 17-t-cut which already separates character 21 together with the previous cluster including character 17 from the sink t.

7.1.2 Exemplary Intra-Cluster Edge Deletions

Events in a friendship network that yield an intra-cluster edge deletion are any events that interrupt the amicable relationship between two individuals. So in the first example we assume an interruption between character 17 and character 27 both being members of the smaller, right group marked by square (red) vertices in Figure 7.1. This edge deletion yields a special situation, as the clustering $\zeta(G_\ominus)$ returned by the cut-clustering heuristic is almost trivial. The only vertex not included in the big (blue) cluster marked by round vertices is vertex 17 (see Figure 7.3a).

The heuristic again needs as many max-flow calculations as clusters are included in $\zeta(G_\ominus)$, which means it needs two max-flow calculations. However, concerning the almost trivial new clustering $\zeta(G_\ominus)$ it might be reasonable to chose a new parameter α greater than $\alpha = 10.5$ to get a clustering consisting of more than two clusters. Unfortunately none of our new updating algorithms is able to deal with a changing parameter α. So in this case a new initial clustering calculation becomes necessary. Developing an approach for also dynamically updating the parameter α might be future work. In this example a new parameter $\alpha = 10.8 > 10.5$ would again yield the clustering shown in Figure 7.2a, in which vertex 17 has left its previous group. Without a dynamic parameter update our Intra-Cluster Edge Deletion algorithm (Algorithm 19) returns the same almost trivial clustering $\zeta(G_\ominus)$ as the cut-clustering heuristic (see Figure 7.3a), but needs one max-flow calculation *more* than the heuristic. However, in our experiments such a behavior occurred if and only if also the cut-clustering heuristic returned an almost trivial new clustering.

Almost trivial clustering



(a) Deleting edge $\{17, 27\}$ causes 2 clusters and 3 max-flow calculations regarding the new algorithm

(b) Deleting edge $\{28, 33\}$ causes 19 clusters and 19 max-flow calculations regarding the heuristic

Figure 7.3: Two special intra-cluster edge deletions.

In contrast to the previous example an interrupted relationship between character 28 and character 33 does not necessarily change the clustering. Our new algorithm finds the updated clustering $\zeta(G_\ominus) = \zeta(G)$ by calculating only one maximum flow, i.e. in this example it uses the chance listed in the second row of Table 6.1. This example shows that Algorithm 19 is able to find an updated clustering which is closer to the previous one than the new clustering calculated from scratch by the cut-clustering heuristic. The heuristic needs 19 max-flow calculations and returns the clustering shown in Figure 7.3b, which contains 19 clusters as the vertices 17 and 28 become single clusters shown as a (blue) square and a (yellow) triangle.

Closer clustering

7.1.3 Exemplary Inter-Cluster Edge Deletion

Finally we give an example of an inter-cluster edge deletion. So assume the friendship between character 0, which belongs to the left group marked by round (green) vertices, and the single character 2 to run dry (see Figure 7.1). This modification again does not necessarily change the previous clustering $\zeta(G)$. Our new Inter-Cluster Edge Deletion algorithm realizes this fact by calculating two max-flow calculations less than the number of clusters included in $\zeta(G)$, which is the best possible performance in such a case of an inter-cluster edge deletion (compare to Table 6.1, first row). This best possible behavior is even guaranteed for the case that the previous clustering $\zeta(G)$ also constitutes a valid clustering for the modified

Guaranteed best performance

graph $G_{\ominus}$. Remember that Algorithm 17 in this situation recalculates the minimum separating cuts related to the cut-vertices of the unaffected clusters. The only cuts being known not to change for sure are the two cuts which define the two clusters affected by the edge deletion. The heuristic also returns the previous clustering, but needs again 17 max-flow calculations.

7.2 Real World E-Mail Graph

Initial underlying graph

Our second experiment serves to affirm the theoretically expected good performance of the updating approach developed in this work. To this end we process a queue of 12 560 elementary modifications on the initial graph G shown in Figure 7.4. This initial graph has 310 vertices and 450 edges. During the experiment the number of vertices varies from 236 to 3 439. The vertices represent members of the Fakultät für Informatik, Universität Karlsruhe (TH), the edges correspond to the e-mail correspondence between those members. Each edge is weighted by the total number of e-mails sent between two individuals in the last 72 hours. This means, each e-mail has a fixed time to live. After that time the contribution of the e-mail to the weight of the edge expires and the weight of the edge decreases. An edge is deleted if its weight becomes zero. A vertex is removed if it becomes unconnected in the underlying graph G. Vice versa, the weight of an edge increases if more e-mails are sent, and a vertex is inserted if a member which has been inactive for more than 72 hours again starts to correspond via e-mail. Accordingly an edge is added to graph G as soon as the first e-mail is sent between two unconnected members.

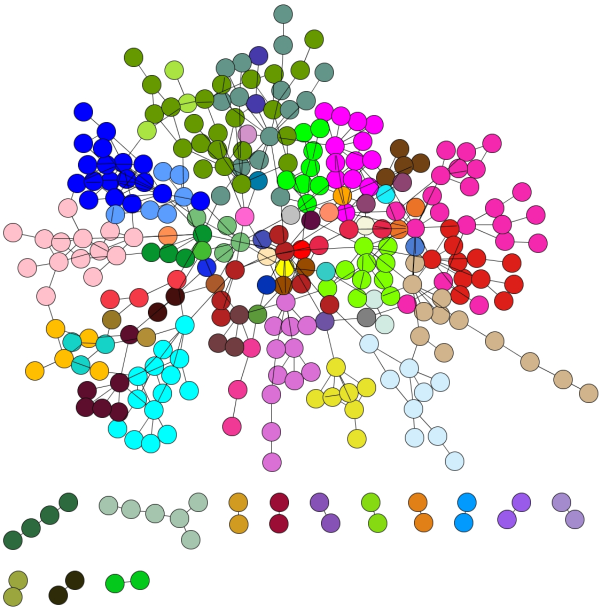


Figure 7.4: Initial real world e-mail graph.

The following statistic compares the performance of our new updating algorithms to the performance of the cut-clustering heuristic, which repeatedly calculates the new clusterings from scratch. As the performance of both methods mainly depends on the number of calculated maximum flows, or minimum u-v-cuts respectively, we concentrate on this indicator. Note, that for both methods the insertion or removal of a vertex only causes a modification of the underlying graph and an adjustment of the current clustering in constant time. Therefore, the vertex insertions and removals are ignored in the statistic. For both clustering methods we choose the parameter $\alpha = 0.15$ concerning the guaranteed clustering quality, which yields a clustering consisting of 45 clusters for the initial graph shown in Figure 7.4.

Amount of effort saving steps

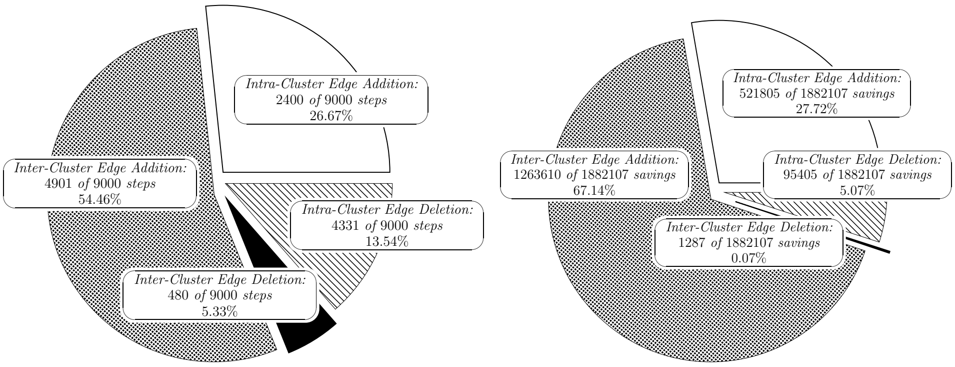
By ignoring the 3 344 vertex insertions and the 216 vertex removals the original queue of 12 560 modifying steps in this experiment shrinks to 9 000 steps of edge modifications. In 8 612 of these 9 000 steps of edge modifications updating the previous clustering causes less max-flow calculations than a new calculation from scratch. This is, in 95.7% of all considered steps our new updating algorithms perform better than the cut-clustering heuristic.

Note further, that updating the previous clustering after increasing the weight of an edge can be done by one of the new algorithms regarding edge additions. The addition of an edge is considered as a special case of increasing the weight of an edge. Analogously, after decreasing the weight of an edge the previous clustering is updated by one of our algorithms regarding edge deletions. So in the following we just talk about intra-cluster and inter-cluster edge additions and edge deletions as the four elementary modifications.

Savings of max-flow calculations

Figure 7.5a shows the proportions of the elementary modifications regarding the total number of 9 000 modifying steps. The case occurring most often is, with 54.46%, the addition of an edge between two different clusters. The inter-cluster edge deletion, by contrast, only occurs 480 times which corresponds to 5.33%. During the whole experiment the cut-clustering heuristic of Flake et al. [FTT04] calculates 2 080 897 maximum flows. Our updating algorithms, however, only need 198 790 max-flow calculations. This yields a saving of 1 882 107 max-flow calculations which constitutes 90.45% of effort saving. Figure 7.5b shows the proportions of the elementary modifications regarding the total number of 1 882 107 savings. We see that the ratio of the percentaged savings provided by edge additions to the proportion of the edge additions regarding the number of total steps is greater than one, while the edge deleting proportion in Figure7.5a provides a smaller proportion of the total savings in Figure 7.5b. More precisely, the inter-cluster edge additions for example are the most efficient modifications, as 54.46% of the total number of steps provide 67.14% of the savings. So each unit of the inter-cluster edge addition proportion on average causes 1.23% of all savings. The least efficient modifications are the inter-cluster edge deletions with 5.33% of all steps gaining only 0.07% of

Efficiency of different modifications



(a) Total number of steps decomposed into elementary modifications

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

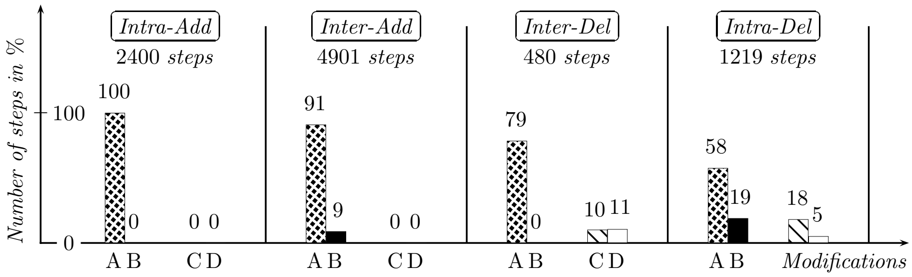
Figure 7.5: Total number of steps and savings of max-flow calculations.

all savings. This corresponds to 0.01% of all savings on average per unit of the inter-cluster deletion proportion.

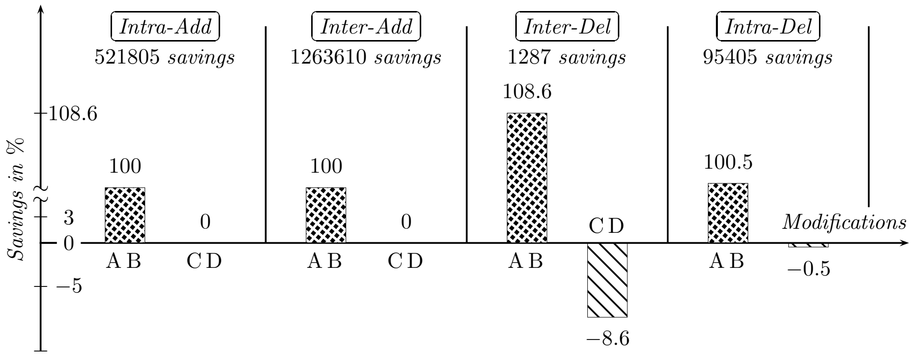
One might have expected the intra-cluster edge addition to be the most efficient modification, as updating the clustering in this case never causes any max-flow calculation, and therefore, saves 100% compared to the cut-clustering heuristic calculating the new clusterings from scratch. However, this only holds from an independent point of view ignoring the remaining modifications. Regarding the total savings over all modifications, nevertheless, the inter-cluster edge additions together save more max-flow calculations than the intra-cluster edge additions.

Independent view of different modifications

An independent view of the four different modifications is given by Figure 7.6. The chart in Figure 7.6a again concentrates on the number of modifying steps, while Figure 7.6b regards the savings of max-flow calculations as a basis. The bar denoted by A in Figure 7.6a for each kind of modification represents the proportion of steps in which our new approach needs less max-flow calculations than the cut-clustering heuristic and needs even less or the same number of max-flow calculations as given by the lower bound or the chance in the case of an unchanging clustering in Table 6.1. This is, for the intra-cluster edge addition the limit is zero, for the inter-cluster edge addition we consider the chance of 2 max-flow calculations, the inter-cluster edge deletion has a lower bound of $|\zeta(G)|-2$, while the intra-cluster edge deletion has the chance to finish after only one max-flow calculation. The second bar denoted by B stands for the remaining steps in which updating causes less max-flow calculations than the heuristic. Accordingly, the bar C represents those cases in which the new algorithm calculates more flows than the heuristic, and bar D finally shows how often both methods need the same number of max-flow calculations. In Figure 7.6b



(a) Number of steps per elementary modification



(b) Savings of max-flow calculations per elementary modification

Figure 7.6: Number of steps and savings of max-flow calculations per modification.

the bars filled with black squares show the proportions of positive savings caused by the steps in which updating is cheaper than calculating from scratch. By contrast, the striped bars illustrate how much saving is again destroyed by the steps in which the new algorithm calculates more maximum flows than the heuristic. For example, 91% of all 4 901 inter-cluster edge additions are updated with less than or equal to 2 max-flow calculations, and hence, save some effort compared to the cut-clustering heuristic. Further 9% of the updated inter-cluster edge additions also save some effort, but cause more than 2 max-flow calculations. By contrast, no updating step for an inter-cluster edge addition needs more or the same number of max-flow calculations as the heuristic. By considering the 480 inter-cluster edge deletions we see that 79% of the updatings save some effort by calculating less than or equal to 2 maximum flows, while for 10% of all inter-cluster edge deletions updating is more expensive than using the cut-clustering heuristic. Another 11% of updatings do neither save nor cause any effort. However, the 49 inter-cluster edge deletions for which updating causes more effort than a new calculation from scratch destroy only 8.6% of the total saving gained by all updating steps regarding inter-cluster edge deletions. Remember, that the inter-cluster edge deletions together only contribute 0.07% to the total savings in this experiment. For the intra-cluster edge deletions the impact of the 222 effort increasing steps on the total savings is with only 0.5% even weaker.

Summary

Altogether in this experiment each updating step after the addition of an edge saves some effort compared to a new calculation from scratch. After the deletion of an edge the new updating algorithms in some cases calculate more maximum flows than the cut-clustering heuristic, but these few cases only have a weak impact on the savings gained by the remaining edge deletion updates. So together also the edge deletion modifications contribute some effort saving regarding the whole experiment.

Future work

To measure how close the updated clusterings are to the previous ones in a real-world experiment is future work. For more information about comparators for clusterings see [DGGW08]. Regarding the comparison we expect a good and helpful preservation of the mental map, as our new algorithms never split any unaffected cluster and try to reuse as big parts as possible of the affected clusters before calculating new clusters.

Chapter 8

Conclusion

This work concentrated on a specific static clustering algorithm, namely the cut-clustering method introduced by Flake et al. [FTT04]. We chose this method to extend it to cluster fully dynamic graphs, as it is one of the very few that guarantees a clustering quality. This clustering quality results from properties of min-cut trees. The special structure of min-cut trees constitutes the key to the algorithmic idea of Flake et al. Saha and Mitra [SM06] gave a first proposal how a dynamic extension of the cut-clustering algorithm might look like, but unfortunately, their approach turned out not to be feasible. We illustrated that Saha and Mitra made a methodical error, as they indirectly assumed an invariant to be met, which is provably violated by the updated clusterings returned by their algorithms. Furthermore, we detected some inconsistencies concerning the application and the proof of a lemma called merging lemma and the formulation of the so called unaffect lemma. We completed the proof of the merging lemma, as Saha and Mitra missed to consider a certain type of cuts. We developed a new merging lemma by replacing the condition the previous merging lemma was based on by an equivalent, but different formulated condition which evinces the previous condition to be very strict. Therefore, the case in the inter-edge-add algorithm of Saha and Mitra which was induced by the merging lemma, turned out to be very unlikely. The new merging lemma and its new condition further allowed a much shorter proof compared to the proof of the previous version. For the unaffect lemma we gave a reasonable interpretation, while Saha and Mitra seemed to misinterpret it. They deduced an invalid procedure from this lemma. Altogether, in the first part of this work we in some parts completed the approach of Saha and Mitra [SM06] and in other parts disproved and corrected it.

The second part we began with some basic lemmas and an introduction of the Gomory-Hu method [GH61], which constructs minimum-cut trees of undirected, weighted graphs. This method constituted the key to many arguments in our work. As the guaranteed clustering quality of the cut-clustering method results from properties of min-cut trees, hence, we first explored, independently from the context of clustering, how min-cut trees can be updated for dynamic graphs. This is, we developed some ideas for a fully dynamic version of the Gomory-Hu method. Thereby we gained several updating algorithms distinguished by the elementary modifications of edge addition and deletion as well as vertex insertion and removal. The formulation of these algorithms covered several levels of detail.

To construct a minimum-cut tree the Gomory-Hu method uses minimum u-v-cuts in the underlying graph, which correspond to maximum flows. To achieve non-crossing minimum separating cuts, as required by the min-cut tree structure, it further uses the technique of contracting vertices. However, these contracting operations make the implementation of the Gomory-Hu method very involved. Gusfield [Gus90] hence modified the Gomory-Hu method such that arbitrary minimum u-v-cut can be used to construct the tree, and therefore, omitted the contraction of vertices. In Section 4.2 we showed that Gusfield's ideas are also adaptable to our dynamic algorithms for updating min-cut trees, which is by far not not obvious. The modified algorithm developed by Gusfield is of a "closed form", which means, it does not allow to choose the step pairs arbitrarily. However, our updating algorithms use initial intermediate min-cut trees which are not guaranteed to constitute a valid intermediate tree in such a closed process. Therefore, we proved the correctness of Gusfield's ideas in a more general situation, which allows to use it for our algorithms, too.

In the case of an edge addition our newly developed algorithm saves as many calculations of maximum flows as edges are excluded from the path between the two vertices defined by the modified edge in the previous min-cut tree. The contrary is true for an edge deletion, where the new algorithm saves as many max-flow calculations as edges lie on this path in the previous min-cut tree. We further stated some facts which allow to modify this edge-deletion algorithm such that at least a chance of saving more effort occurs.

Additionally, as the performance of the Gomory-Hu method is highly affected by the calculation of minimum u-v-cuts, we analyzed the possibilities of dynamically updating these cuts individually. It turned out that those approaches that base on updating corresponding maximum flows, like the approach of Kohli and Torr [KT07], are not able to deal with the hiding of step pairs during a Gomory-Hu execution. Furthermore, we introduced another approach which considers a DAG-representation of all minimum u-v-cuts concerning two fixed vertices u and v and updates the whole set of minimum u-v-cuts. However, the performance of this approach strongly depends on the structure of the underlying graph. If there exist only few minimum separating cuts per fixed pair of vertices, many new max-flow calculations become necessary anyway. Nevertheless, this procedure is at least able to deal with the hiding of step pairs on certain conditions. Theoretically it can hence be used for updating min-cut trees in a partially dynamic graph where only edge deletions occur. Furthermore, it became possible to calculate a min-cut tree which represents not only one minimum separating cut per pair of vertices, but the set of all minimum separating cuts.

Chapter 6 then returned to the subject of graph clustering. At this point we were able to deduce new updating algorithms to cluster fully dynamic graphs from the algorithms developed so far. We illustrated that for the calculation of a reasonable clustering according to the cut-clustering method of Flake et al. it is not necessary to construct a complete min-cut tree. Instead, it suffices to isolate the artificial sink t. So abbreviating the previous algorithms for updating complete min-cut trees yielded new updating algorithms regarding the following modifications: The addition of an edge within a single cluster, the addition of an edge between two different clusters, the deletion of an edge within a single cluster and finally the deletion of an edge between two different clusters. The insertion or the removal of a vertex simply corresponds to the addition or deletion of a singleton in the current clustering. Our INTRA-CLUSTER EDGE ADDITION algorithm thereby turned out to be almost trivial. This result basically meets the assertion Saha and Mitra stated for this case. We further proved that the INTER-CLUSTER EDGE DELETION algorithm always returns the previous clustering by calculating a guaranteed number of maximum flows if we assume the previous clustering to also constitute a valid clustering for the modified graph. Each of our newly developed updating algorithms further tries to return a clustering as close as possible to the previous one.

Finally we affirmed the theoretically predicted behavior of the new updating algorithms with the aid of a small experiment. In this experiment we compared the performance of the four new algorithms to a repeated calculation of new clusterings from scratch with the aid of the cut-clustering heuristic given by Flake at al. [FTT04]. In 95.7% of all modifications our algorithms needed less max-flow calculations than the heuristic, while together they achieved a saving of about 90.45% of max-flow calculations.

Open Problems

In the context of updating complete min-cut trees by individually updating minimum u-v-cuts it turned out that those approaches that base on updating corresponding maximum flows are not able to deal with the hiding of step pairs during a Gomory-Hu execution. This is caused by the difficulty of rerouting a given flow to a different target without a complete recomputation. So efficiently solving such reroutings constitutes an open problem. Another open problem related to the former, is the construction of a compact, merged DAG-representation resulting from several DAG-representations all containing minimum u-v-cuts for a pair $\{u, v\}$, which is not adjacent in a min-cut tree $T_{\text{DAG}}(G)$ representing all minimum separating cuts for all pairs of vertices (compare to Chapter 5, Subsection 5.2.3).

A more extensive experimental analysis of the performance of our new updating algorithms may also be future work. An interesting questions in this context is whether the new algorithms are able to achieve even more savings by storing and updating DAG-representations of minimum separating cuts. Furthermore, we did not yet measure or evaluate the similarity between the previous and the updated clusterings our new algorithms try to achieve. The new updating algorithms never split an unaffected cluster and further try to reuse as big parts as possible of the affected clusters before calculating new clusters. So we expect a high similarity between the updated and the previous clustering. Regarding the offline problem the cluster preserving behavior of our new algorithms allows to reorder those elementary modifications in the queue that affect clusters which are not affected by other modifications. Assume for example a sequence of intra-cluster edge additions regarding one fixed cluster not affected by other modifications. Such a sequence can simply be ignored, as the addition of an edge within a cluster turned out not to change the clustering. A more extensive analysis of the offline problem is also future work.

The last and probably most important open problem mentioned here constitutes the parameter α, which the clustering quality of our algorithms depends on. The algorithms developed in this work are not yet able to simultaneously update this parameter dynamically, as well. However, this might become necessary if the repeatedly modified, underlying graph changes that much that the originally defined parameter does not make sense anymore. The returned clusterings then become trivial.

Bibliography

[BE05] U. Brandes and T. Erlebach, editors. *Network Analysis*. Springer, 2005. 1, 2

[BK04] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI '04)*, 26(9):1124 – 1137, 2004. 74

[DGGW08] D. Delling, M. Gaertler, R. Görke, and D. Wagner. Engineering comparators for graph clusterings. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM '08)*, volume 5034 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 2008. 117

[Din70] E.A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Doklady*, 11:1277–1280, 1970. 74

[DKL76] Y. Dinitz, A. V. Karzanov, and M. Lomonosov. On the structure of a family of minimal weighted cuts in a graph. In A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306, 1976. (in Russian). 79

[DN95] Y. Dinitz and Z. Nutov. A 2-level cactus model for the system of minimum and minimum+1 edge-cuts in a graph and its incremental maintenance. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC '95)*, pages 509–518, 1995. 79

[FF56] Jr. L. R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. 71

[Fle99] L. Fleischer. Building chain and cactus representations of all minimum cuts from hao-orlin in the same asymptotic run time. *Journal of Algorithms*, 33:51–72, 1999. 2, 79

[FTT04] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004. 5, 7, 2, 6, 8, 9, 11, 12, 16, 18, 19, 21, 22, 25, 39, 89, 102, 104, 105, 109, 110, 115, 119, 121

[GGW07] M. Gaertler, R. Görke, and D. Wagner. Significance-driven graph clustering. In *Proceedings of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM '07)*, Lecture Notes in Computer Science, pages 11–26. Springer, June 2007. 2

[GH61] R. E. Gomory and T.C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9:551–570, December 1961. 5, 7, 2, 4, 25, 31, 32, 35, 39, 86, 119

[Gol08] A. Goldberg. The partial augment-relabel algorithm for the maximum flow problem. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA '08)*. Springer, 2008. 74

[GT88] A. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988. 67, 74, 79, 85, 87

[Gus90] D. Gusfield. Very simple method for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, February 1990. 7, 32, 35, 47, 48, 49, 51, 52, 120

[HO94] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17:424 – 446, 1994. 79

[KS08] H. Kaplan and N. Shafir. Finding path minima in incremental unrooted trees. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA '08)*. Springer, 2008. 71

[KT07] P. Kohli and P. H. S. Torr. Dynamic graph cuts for efficient inference in markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI '07)*, 29(12):2079–2088, 2007. 2, 71, 73, 74, 82, 83, 90, 120

[Nag06] H. Nagamochi. Computing a minimum cut in a graph with dynamic edges incident to a designated vertex. *IEICE Transactions on Info and Systems*, Volume E90-D(2):428–431, 2006. 71

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004. 2

[Pou90] J.A. La Poutre. New techniques for the union-find problem. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 54–63. Society for Industrial and Applied Mathematics, 1990. 81

[PQ80] J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. *Mathematical Programming Study*, 13:8 – 16, 1980. 71, 72, 78, 79

[SM06] B. Saha and P. Mitra. Dynamic algorithm for graph clustering using minimum cut tree. In *Proceeding of the 6th IEEE International Conference on Data Mining - Workshops (ICDMW '06)*, pages 667–671, 2006. 5, 7, 2, 6, 8, 9, 10, 11, 12, 13, 15, 16, 18, 20, 21, 22, 25, 89, 105, 106, 119

[Tar74] R. E. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974. 44

[Zac77] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977. 109