Dynamic Gomory-Hu Tree

Soham Chakraborty
22CS02002

B.Tech and M.Tech (Dual Degree)
in
Computer Science and Engineering
Semester 7

Under the supervision of
Dr. Joy Chandra Mukherjee

School of Electrical and Computer Sciences
Indian Institute of Technology Bhubaneswar
November 2025

Abstract

Computing the maximum network flow for all pairs of nodes in an undirected graph and can be trivially done using a brute force approach with $\binom{n}{2}$ flow computations. Here we discuss about an algorithm proposed by Gomory-Hu, using which we can construct a Cut-Tree, which solves this problem using only $n - 1$ flow computations. Then, we discuss about an extremely fast and easy implementation of Gomory-Hu Cut Tree as constructed by Gusfield. We mention the construction and use of Dynamic Gomory-Hu trees as proposed by Tanja Hartmann and Dorothea Wagner, which handles edge weight updates, insertion and deletion in an efficient manner. Finally, we conclude with possible future-work ideas in Cut-based clustering algorithms and applications related to the real-life use cases of Dynamic Gomory-Hu Trees.

Acknowledgements

I have a very keen interest in the field of Data Structures and Algorithms. I am grateful to Joy Sir for giving me a BTP topic related to my interest in Algorithms and weekly interaction about insights into further developments and follow-ups of the project. Competitive programming being my favourite domain and C++ as the primary programming language it was a nice experience converting Pseudo-codes from various research papers in functioning C++ codes (with not so nice experience of debugging). Being a Dual Degree student, I have 3 more semesters before I graduate and in this time I wish to further develop this project and make some progress into other related topics with the help of Joy Sir to a full-fledged MTP project.

Contents

1 Introduction

In this project, we look at algorithms for solving graph-theoretic problems related to Multi-Terminal minimum cut - maximum flow in a network. It consists of 4 parts -

1. Gomory-Hu Cut Tree
2. Modified Cut-Tree algorithm by Gusfield
3. Dynamic Gomory-Hu Tree
4. Future Work

1.1 Overview

We here give a brief insight to the first 3 parts from above.

Part I: Gomory-Hu Tree

A Gomory-Hu tree is an undirected, weighted tree on the vertices of a given graph such that each edge in the tree represents a minimum separating cut in the underlying graph with respect to its incident vertices. The cost of the cut is given by the cost of the tree edge. From this property it follows directly that a minimum separating cut for two vertices that are not adjacent in the tree is given by a cheapeast tree edge on the unique path between both vertices. We first give a description of the very first algorithm proposed by Gomory and Hu for constructing a Cut-Tree on general graphs and explain the ideas and mechanisms it relies on. This lays the basics for newer algorithmic approaches to constructing Cut-Trees such as the one given by Gusfield in Part II.

Part II: Implementation of Cut Tree by Gusfield

Gusfield provides an algorithm that constructs the Gomory-Hu Tree in an extremely easy manner by extrapolating some relaxations in the inital construction for the maintainence of Gomory-Hu Tree formation. We start with describing the Equivalent Flow Tree Program, after which adding just a single *if statement* gives us a Cut Tree. Further, he shows that adding one more line of *for loop* into the code computes the complete All-Pair Max. Flow matrix.

Part III: Dynamic Gomory-Hu Tree

The Dynamic Gomory-Hu Tree as proposed by Tanja Hartmann and Dorothea Wagner aims at maintaining Gomory-Hu trees also in evolving graphs. We consider a dynamic scenario where two cases of the underlying graph differ due an atomic edge change[1]. In the incremental scenarios, we are interested in updating the Gomory-Hu tree each time as efficiently as possible, as compared to constructing the whole tree from scratch.

[1]Edge weight increase, Edge weight decrease, edge insertion and edge deletion

1.2 Notations and Definitions on Graphs

In this section, we present the notation used throughout the report and give definitions of most terms that occur. Some of the incidentally mentioned concepts however, are mentioned on that point only.

Undirected Graphs: Apart from maximum flows which are usually considered in *directed* graphs we only consider weighted *undirected* graphs here. An undirected,weighted graph is a graph $G = (V, E, c)$ with vertices V, edges E and a positive edge cost function $c : E \to {R_0}^+$, writing $c(u, v)$ as a shorthand for $c(u, v)$ with $\{u, v\} \in E$. We will also be using $n := |V|$ and $m := |E|$ as obvious notations which will be clear from the context.

Special Cases: We define a *simple path* of n vertices as a sequence $v_1, v_2, ..., v_n$ of vertices such that $v_i \neq v_j$ for $i \neq j$, $\{i, j\} \subseteq 1, 2, ..., n$ and $\{v_i, v_j\}$ forms an edge for $i = 1, 2, ..., n-1$. A *tree* is a connected acyclic graph. The path between two vertices of a tree is unique, and we write $\pi(u, v)$ for the path between u and v.

Dynamic Graphs: We call a graph *dynamic* if its edges vary over time. Assume, change in graph G either involves an edge $\{b, d\}$ with $c(b, d) = \Delta > 0$. If the cost of $\{b, d\}$ in G decreases by $c(b, d) = \Delta > 0$, edge is deleted and resulting graph is termed $G^{\ominus}$. Similarly, inserting $\{b, d\}$ or increasing the cost yields $G^{\oplus}$. We denote the cost function after a change by $c^{\ominus}$ and $c^{\oplus}$ respectively. If we consider the graph after an arbitrary change, without any further information about the type of change, we denote it by $G^{\textcircled{U}}$

1.3 Cuts and Flows

A *cut* in an undirected weighted graph $G = (V, E, c)$ is a partition of V into two non-empty *cut sides* S and $V \backslash S$. The cost $c(S, V \backslash S)$ of a cut in G is the sum of the costs of all edges *crossing* the cut, that is, edges $\{u, v\}$ with $u \in S, v \in V \backslash S$ also denoted as $c(S)$. A *cut set* is the set of edges crossing the cut. Two cuts are said to be *nested* if their cut sides are nested and *non-crossing* if their cut sides are nested or the two cut sides are disjoint. A set of cuts is called *non-crossing* if all cuts are pairwise non-crossing. A *minimum u-v-cut* is a cut that separates u and v and is the cheapest cut among all cuts separating these vertices, denoted by $\lambda(u, v)$.

From equivalence of $\lambda(u, v)$ and the *value of maximum u-v-flow* in an undirected, weighted graph as stated by Ford-Fulkerson, computing minimum u-v-cut in an undirected, weighted graph is basically same as computing maximum u-v-flow in this graph. The best known complexity for solving the *max flow - min cut* problem is $O(nm + n^2 \log n$ by Nagamochi and Ibaraki.

Flow in G between two vertices s and t is a function $f : E \to R_0^+$ that satisfies:

1. $f(e) \leq c(e), \forall e \in E$
2. $\sum_{x \in N_i(v)} f(x, v) = \sum_{y \in N_o(v)} f(v, y), \forall v \in V \backslash \{s, t\}$

2 Gomory-Hu Tree

2.1 Definition

Gomory-Hu Tree is a weighted tree $T(G) = (V, E_T, c_T)$ on the vertices of an undirected (weighted) graph $G = (V, E, c)$ (with edges not necessarily in G) such that each $\{u, v\} \in E_T$ induces a minimum u-v-cut in G *by decomposing $T(G)$ into two connected components) and such that $c_T(\{u, v\})$ is equal to the cost of the induced cut. The cuts induced by $T(G)$ are non-crossing and for each $\{s, t\} \subseteq V$ each cheapest edge on the path $\pi(u, v)$ between s and t in $T(G)$ corresponds to a minimum s-t-cut in G. A Gomory-Hu tree thus represents $n - 1$ non-crossing minimum *s-t-cuts*.

Reconnecting an edge in $T(G)$ means replacing the edge by another edge with the same attributes without creating a cycle. *Step pairs* refers to the cut pairs that are considered for the computation of minimum separating cuts during the construction.

Flow-Equivalent Tree is a weighted tree $T(G) = (V, E_T, c_T)$ on the vertices of G having cost function $c_T : E_T \to R_0^+$ such that the cost $c_T(\{s, t\})$ of each edge $\{s, t\}$ corresponds to $\lambda_G(s, t)$. Hence, it also holds that edge connectivity $\lambda_G(s, t)$ for an arbitrary vertex pair $\{s, t\} \subseteq V$ is given by the cost of the cheapest edge on teh path between s and t in $T(G)$ i.e. $\lambda_{T(G)}(s, t) = \lambda_G(s, t)$. Hence, Gomory-Hu trees are a subclass of flow-equivalent trees.

2.2 Construction algorithm

Lemma 1: *A set $\mathcal{H}$ of $n - 1$ cuts is a Gomory-Hu set if and only if the cuts in $\mathcal{H}$ are pairwise non-crossing and for each cut exists a cut-pair that is exclusively separated by this cut, that is, not separated by any other cut in $\mathcal{H}$.*

Proof: Clearly, each Gomory-Hu tree satisfies the conditions of Lemma 1. Now let $\mathcal{H}$ denote a set of $n - 1$ pairwise non-crossing cuts and $\mathcal{P}$ a set of $|\mathcal{H}|$ exclusively separated cur pairs. Since cuts in $\mathcal{H}$ are non-crossing, their cuts are nested, and since there are $n - 1$ of these cuts, there exists exactly one vertex pair for each cut that is exclusively separated by this cut. Hence, these vertex pairs must correspond to the cut pairs in $\mathcal{P}$. Connecting cut pairs in $\mathcal{P}$ by edges then yields a Gomory-Hu tree that represents $\mathcal{H}$.

Lemma 2: (Non-Crossing Lemma) *Let $(X, V\backslash X)$ be a minimum x-y-cut in G, with $x \in X$. Let $(H, V\backslash H)$ be a minimum u-v-cut, with $u, v \in V\backslash X$ and $x \in H$. Then the cut $(H \cup X, (V\backslash H) \cap (V\backslash X))$ is also a minimum u-v-cut.*

The idea of the Algorithm proposed by Gomory and Hu is to iteratively construct a Gomory-Hu set by choosing in each step a vertex pair (*step pair*) that is not yet separated and computing a minimum separating cut for this pair. To ensure that each cut is

non-crossing (from Lemma 1) Gomory and Hu contract at least one cut side of each cut found so far and compute the minimum separating cut in the resulting graph. This is due to the result of Lemma 2 (Non-Crossing Lemma) that there always exists a minimum separating cut that does not cross any of the previous cuts.

The following pseudocode is for the Algorithm 1 by Gomory and Hu for the construction of Cut-Tree. The *intermediate tree* $T_* = (V_*, E_*, c_*)$ is initialized as an isolated, edgeless node containing all original vertices. Then, until each node of T_* is a singleton node, a node $S \in V_*$ is *split* and an edge representing a new cut ia added to E_*. Now, supernodes $S' \neq S$ which are related to cut sides of previously found cuts, are dealt with by contracting in G whole subtrees N_j of S in T_*, connected to S via edges $\{S, S_j\}$, to single nodes $[N_j]$ before cutting, whihc yields G_S. The split of S into S_u and S_v is then defined by a minimum u-v-cut (split cut) in G_S (with *step pair* $\{u, v\} \subseteq S$), which does not cross any of the previously used cuts due to the contraction technique. Reacll that this split cut in G_S also induces a minimum u-v-cut in G, due to the Non-Crossing Lemma (2). After splitting, the edge $\{S_u, S_v\}$ is added to E_*, and each N_j is reconnected, again by S_j, to either S_u or S_v depending on which side of the cut $[N_j]$ ended up. This reconnection ensures that the edge $\{S_u, S_v\}$ indeed represents the split cut.

Algorithm 1: Gomory–Hu

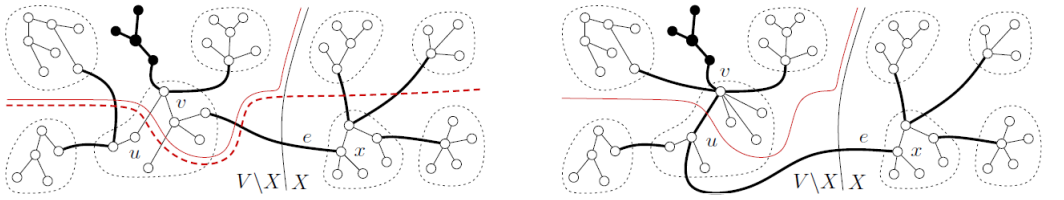
Input: Graph $G = (V, E, c)$
Output: Gomory–Hu tree of G
Initialize $T_* := (V_*, E_*, c_*)$ with $V_* \leftarrow \{V\}$, $E_* \leftarrow \varnothing$, c_* empty
while $\exists S \in V_*$ *with* $|S| > 1$ **do**
  $\{u, v\} \leftarrow$ arbitrary pair in S `// unfold node`
  foreach S_j *adjacent to* S *in* T_* **do**
    $N_j \leftarrow$ subtree of S in T_* with $S_j \in N_j$
  end
  $G_S = (V_S, E_S, c_S) \leftarrow$ contract each N_j to $[N_j]$ in G `// contraction`
  $(U, V \setminus U) \leftarrow$ min-u-v-cut in G_S, cost $\lambda_{G_S}(u, v)$
  $S_u \leftarrow S \cap U, \quad S_v \leftarrow S \cap (V_S \setminus U)$ `// split` $S = S_u \cup S_v$
  $V_* \leftarrow (V_* \setminus \{S\}) \cup \{S_u, S_v\}$
  $E_* \leftarrow E_* \cup \{\{S_u, S_v\}\}$
  $c_*(S_u, S_v) \leftarrow \lambda_{G_S}(u, v)$
  foreach *former edges* $e_j = \{S, S_j\}$ *in* E_* **do**
    if $[N_j] \in U$ **then**
      $e_j \leftarrow \{S_u, S_j\}$ `// reconnect to` S_u
    else
      $e_j \leftarrow \{S_v, S_j\}$ `// reconnect to` S_v
    end
  end
end
return T_*

3 Modified Cut-Tree algorithm by Gusfield

Gomory and Hu in their implementation of Cut-Tree use contractions in G to prevent crossings of the cuts. Gusfield provides a simplified version of the same by relaxing the above conditions and arbitrarily separating cuts directly in G without any contractions. Gusfield in his implementation resolves non-crossing cuts using the Non-Crossing Lemma (2). To maintain the reconnection of edges in the *Intermediate Tree* he uses a parent array and another array to represent the weight from a vertex to its parent.

3.1 Intermediate Trees

An intermediate tree is a partially formed Gomory-Hu Tree which has not yet completed all the $n-1$ iterations to reach the final state. In an Intermediate Tree represented by $T_* = (V_*, E_*, c_*)$ each node in V_*, which consists of original vertices in V, by an arbitrary tree of *thin* edges connecting the contained vertices in order to indicate their membership to the node. An edge connecting two nodes in V_* is represented by a *fat* edge in E_*, which we connect to an arbitrary vertex in each incident node. Fat edges represent minimum separating cuts in G, and are thus associated with costs. If a node contains only one vertex, we color this vertex black. Black vertices are only incident to at least one thin edge. In this way, $T_* = (V, E_t, E_f, c_f)$ can be also considered as a tree on V with two types of edges (thin edges in E_t, fat edges in E_f) and vertices (black and white), and a cost function c_f on the fat edges. See figure below:



(a) Original and reshaped split cut in an intermediate tree T_* before the reconnection. Nodes in T_* are indicated by dotted frames.

(b) Reconnecting edges according to Cut Tree ensures the reshaping of the split cut and the representation of the reshaped cut by the new fat edge $\{u, v\}$.

Figure 7.3: Reconnecting edges implements splitting of the current node and reshaping of the split cut. The original split cut $(U, V \setminus U)$ (dashed red line) with respect to step pair $\{u, v\}$ is bent along the cut $(X, V \setminus X)$ represented by edge e (solid black line) resulting in a non-crossing split cut (solid red line).

3.2 Description

From the above addressing of subtrees of a node using arrays in T_* we can easily address the vertices that are linked by a fat edge to a vertex in the node. As per the representation described above, Algorithm 2 considers the intermediate tree T_* as a tree on the vertices of G consisting of thin and fat edges and bypasses any contraction in G. Now, modify Algorithm 1 so that in every intermediate tree, every supernode S contains exactly one node called the *parent/representative* of S, denoted $r(S)$. Then any one node is arbitrarily chosen to be the representative of the first supernode of GH method (Algorithm 1). Next, each successive flow is computed between $r(S)$ and some other vertex $v \in S$. S is split into two supernodes $S_{r(S)}$ and S_v. Assign $r(S)$ as the representative of $S_{r(S)}$, and v to be the representative of S_v. If S' is a supernode neighbor of S in T before the split, and $r(S')$ falls on the v side of the cut, then replace the (S, S') edge with edge (S_v, S'). Now, after $n-1$ iterations each supernode decomposes into a node having only 1 vertex each.

Algorithm 2: Modified Cut-Tree

Input: Graph $G = (V, E, c)$
Output: Cut tree of G
for $s \leftarrow 2$ **to** n **do**
  Compute a minimum cut between nodes s and $t := p[s]$ in G.
  Let X be the set of nodes on the s-side of the cut.
  Output the maximum s, t flow value $f(s, t)$.
  $fl[s] \leftarrow f(s, t)$
  for $i \leftarrow 1$ **to** n **do**
    if $(i \neq s)$ *and* $i \in X$ *and* $p[i] = t$ **then**
      $p[i] \leftarrow s$
    end
  end
  if $p[t] \in X$ **then**
    $p[s] \leftarrow p[t]$
    $p[t] \leftarrow s$
    $fl[s] \leftarrow fl[t]$
    $fl[t] \leftarrow f(s, t)$
  end
end

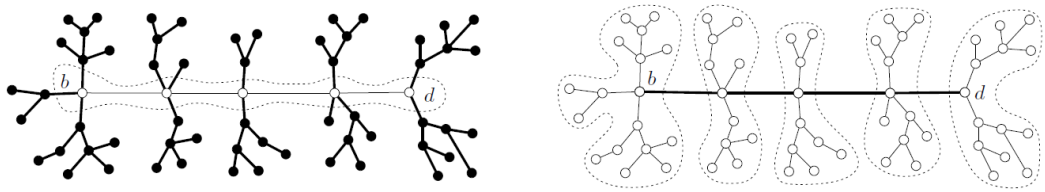
4 Dynamic Gomory-Hu Tree

Here we seek methods to efficiently maintain a Gomory-Hu Tree also in a dynamic scenario, where the underlying graph changes over time. We consider changes in the GH tree that occurs when a concrete change occurs on a concrete edge. Similar work has been done in context of *sensitivity analysis of multi-terminal flow networks* by Elmaghraby and *parametric maximum flows.* by Gallo et al. and Scutella.

4.1 Finding Reusable Cuts

The main idea of this approach is to determine cuts in a given Gomory-Hu tree $T(G)$ that are still minimum separating cuts in the graph $G^{Ⓤ}$ after a change, and thus, can be reused for the construction of a new Gomory-Hu tree $T(G^{Ⓤ})$. If an old cut is not reusable with respect to this some checked vertex, then using this method it is not guaranteed that the cut won't also be reusable with respect to some other vertex. Below are some **Lemmas** and **Corollaries** to detect reusable cuts.

Lemma 3: *If $c(b,d)$ changes by $\Delta > 0$, then each $\{u,v\} \in E_T$ remains a minimum u-v-cut (i) in $G^{\oplus}$ with cost $\lambda_G(u,v)$ if $\{u,v\} \notin \pi(b,d)$, (ii) in $G^{\ominus}$ with cost $\lambda_G(u,v) - \Delta$ if $\{u,v\} \in \pi(b,d)$.*



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

FIGURE 8.1: Intermediate trees representing reusable cuts detected by Lemma 8.1. Fat edges represent (reused) minimum cuts in $G^{Ⓤ}$, thin edges indicate compound nodes. Contracting thin edges yields nodes of white vertices (indicated by dotted lines). Black vertices correspond to singletons.

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

Corollary 1 directly allows us to use the whole Gomory-Hu tree $T(G)$ if $\{b,d\}$ is an existing bridge in G with increasing cost.

Corollary 2: *An edge $\{u, v\}$ is a bridge in G if and only if $c(u, v) = \lambda_G(u, v) > 0$. Then $\{u, v\}$ is also an edge in $T(G)$.*

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

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

Lemma 5 allows us to easily handle edge weight decrease in bridges with 0 new flow computations.

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

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

Lemma 6 allows us to reuse all edges in E_T that lie on a common side of the cut.

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

4.2 Algorithm

Here we will show two update algorithms one for increase in edge costs or edge insertions and another for decrease in edge costs or edge deletion. Below in both cases it is assumed that the edge being updated is $\{b, d\} \in E$ with $\Delta > 0$.

4.2.1 Increasing Edge Costs

If $\{b, d\}$ is an existing bridge or newly inserted bridge in G, then it adapts $c_T(b, d)$ according to Corollary 1. Otherwise, if $\{b, d\}$ is a newly inserted bridge in G, it rebuilds $T(G)$ according to Lemma 4. In both cases they require no new cut-computations.

If $\{b, d\}$ is not a bridge we create the intermediate tree as in Figure 4.1(a) reusing all edges not in $\pi(b, d)$. Then as per Figure 4.1(a) we create an intermediate tree by compressing all vertices on the path $\pi(b, d)$ into a supernode and decomposing the supernode using a slightly modified version of Algorithm 2.

Algorithm 3: Increase or Insert

Input: $T(G)$, b, d, $c(b,d)$, $c^{\oplus}(b,d)$, $\Delta := c^{\oplus}(b,d) - c(b,d)$
Output: $T(G^{\oplus})$
$T_\star \leftarrow T(G)$
if $\{b,d\}$ *is a new bridge* **then**
  Remove an edge on $\pi(b,d)$ with cost $= 0$ `// Lemma 4`
  Add edge $\{b,d\}$ with cost $=c^{\oplus}(b,d)$
  return $T(G^{\oplus}) \leftarrow T_\star$
end
else if $\{b,d\}$ *is an existing bridge* **then**
  $c_*(b,d) \leftarrow \lambda(b,d) + \Delta$ `// Corollary 1`
  return $T(G^{\oplus}) \leftarrow T_\star$
end
Construct an intermediate tree according to Figure 4.1(a)
return $T(G^{\oplus}) \leftarrow CutTree(T_\star)$
`// Algorithm 1 could also have been used`

4.2.2 Decreasing Edge Costs

Algorithm 4 for Decrease or Deletion of edges starts by checking if $\{b,d\}$ is a bridge, and if so then reuses the whole Gomory-He tree $T(G)$ with adapted cost $c_T(b,d)$ (Lemma 5). Otherwise, it constructs the intermediate tree T_* as shown in Figure 4.1(b), reusing all edge on $\pi(b,d)$ with adapted costs. Then it proceeds with iterative steps where the cut pairs are chosen starting with those which have an incident vertex on $\pi(b,d)$. In this way each edge $\{u,v\}$ is found to remain valid to retain the maximal subtree (Lemma 6).

Lemma 8: *Let $(X, V\backslash X)$ denote a minimum x-y-cut in G with $x \in X, y \in V\backslash X$ and $\{b,d\} \subseteq V\backslash X$. Let $(U, V\backslash U)$ denote a further cut. If (i) $(U, V\backslash U)$ separates x, y with $x \in U$, then $c^{\ominus}(U \cup X, V\backslash(U \cup X))$. If (ii) $(U, V\backslash U)$ does not separate x, y with $x \in V\backslash U$, then $c^{\ominus}(U\backslash X, V\backslash(U\backslash X)) \leq c^{\ominus}(U, V\backslash U)$.*

We assume that G and $G^{\ominus}$ are global variables. $\pi(b,d)$ is assumed to be implicitly updated at each step of the iteration. Thin edges are weighted by the old connectivity, fat edges by the new connectivity of their incident vertices. Whenever an edge is reconnected (by reconnecting one of its incident vertices), the newly occurring edge inherits the cost and the thickness from the disappearing edge. Below is the implementation of Algorithm 4:

Algorithm 4: Decrease or Delete

Input: $T(G)$, b, d, $c(b,d)$, $c^{\ominus}(b,d)$, $\Delta := c(b,d) - c^{\ominus}(b,d)$
Output: $T(G^{\ominus})$
$T_\star \leftarrow T(G)$
if $\{b,d\}$ *is a bridge* **then**
  return $T(G^{\ominus}) \leftarrow T_\star$ `// Lemma 5`
end
Construct intermediate tree according to Figure 4.1(b)
$Q \leftarrow$ thin edges non-increasingly ordered by their costs
while $Q \neq \varnothing$ **do**
  $\{u,v\} \leftarrow$ most expensive thin edge with v on $\pi(b,d)$
  $N_x \leftarrow$ neighbors of v on $\pi(b,d)$
  $L \leftarrow \min_{x \in N_x}\{c_\star(x,v)\}$
  if $L \geq \lambda_G(u,v)$ ***or*** $\{u,v\} \in E$ *with* $\lambda_G(u,v) = c(u,v)$ **then**
    `// Lemma 7 or Corollary 3`
    draw $\{u,v\}$ as a fat edge
    consider the subtree U rooted at u with $v \notin U$ `// Lemma 6`
    draw all edges in U fat; remove fat edges from Q
    continue
  end
  $(U, V \setminus U) \leftarrow$ minimum u–v cut in $G^{\ominus}$ with $u \in U$
  draw $\{u,v\}$ as a fat edge, remove $\{u,v\}$ from Q
  if $\lambda_G(u,v) = c^{\ominus}(U, V \setminus U)$ **then**
    `// old cut still valid`
    goto line 10
  end
  $c_\star(u,v) \leftarrow c^{\ominus}(U, V \setminus U)$ `// otherwise`
  $N \leftarrow$ neighbors of v
  foreach $x \in N$ **do**
    `// bend split cut by Lemma 8 and Lemma 2`
    if $x \in U$ **then**
      reconnect x to u
    end
  end
end
return $T(G^{\ominus}) \leftarrow T_\star$

4.3 Running Time

The running time of this Dynamic Gomory-Hu update procedures are clearly dominated by the computations of the new minimum separating cuts in $G^{\textcircled{U}}$. The saving in the number of flow computations for increasing edge cost and edge insertions simply depends on the length of the $|\pi(b,d)|$. Thus, in most cases saving will be quite high unless the cut structure of the underlying graph is that much degenerated that the shape of the Gomory-Hu tree is close to a linear path. In most cases, Gomory-Hu trees tend to a star-like shape, particularly for very regular graph structures. But even though star contributes for the most optimal savings in case of edge insertion,

it constitutes the worst-case with respect to an edge deletion or cost decrease.

Figure 4.1 shows an example based on a regular (unweighted) grid. In this example each vertex has degree 4, and we assum ethat the (global) edge connectivity is also 4. Then, the edge connectivity $\lambda_G(u,v)$ is also 4 for each vertex pair $\{u,v\} \subseteq V$, and $(\{u\}, V\backslash\{u\})$ is a minimum u-v-cut. Hence, any star with an arbitrary vertex c as center and edges of cost 4 is a Gomory-Hu tree of G. Now assume the edge $\{b,d\}$ depicted in Figure 4.1 is deleted. For each vertex pair that contains at least b or d, this decreases the connectivity to 3, however, the deletion has no effect on the connectivity of the remaining vertex pairs. Thus, decreasing the costs at the edges $\{c,b\}$ and $\{c,d\}$ in $T(G)$ would suffice to update the Gomory-Hu tree. Instead, Algorithm 4 checks each edge, apart from the two edges on $\pi(b,d)$, by computing a new minimum separating cut in $G^{\ominus}$. Consequently, it needs $n-3$ cut computations, which is the worst possible running time, as according to Lemma 5, maintaining the whole tree is possible if $\pi(b,d) = \{b,b\}$, or $\{b,d\}$ is a bridge.

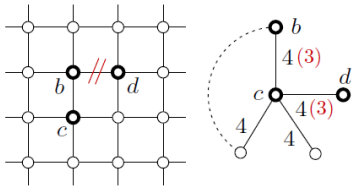


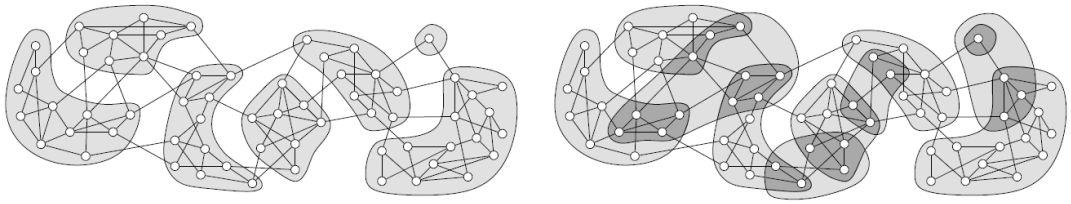
Figure 4.1: Grid graph G with star-shaped tree $T(G)$

.

5 Future Work

After going through multiple research papers I observed that almost all, if not all of further follow-ups of Static and Dynamic Gomory-Hu trees are on various Clustering Algorithms and Connectivity.

Graph Clustering problems aims at decomposing a given graph along sparse cuts into somehow dense subgraphs called *clusters*. It focuses on *intra-cluster density* and *inter-cluster sparsity*. Further clustering problems are used to get good heuristics in NP-hard problems that rely in finding good clusters.



(a) Disjoint subgraphs. (b) Overlapping subgraphs.

Figure 5.1: Decomposition of a graph into subgraphs

Some of the popular quality measure to measure clustering in a graph are -

1. Modularity of clustering in an undirected unweighted graph is based on the total edge cost in G that is covered by clusters with values ranging from -0.5 to 1, with 1 being the best value. Modularity $\mathcal{M}$ of a clustering Ω is defined as -

$$\mathcal{M}(\Omega) = \sum_{C \in \Omega} c(E_C)/c(E) - \sum_{C \in \Omega} (\sum_{v \in C} deg(v))^2/4c(E)$$

2. Expansion is a quality measure of cuts in weighted graphs. Higher the expansion the better. Expansion $\psi(U, C\backslash U)$ of a clustering $(U, C\backslash U)$ is defined as -

$$\psi(U, C\backslash U) = \frac{c(U, C\backslash U)}{min(|U|, |C\backslash U|)}$$

3. Conductance is a quality measure of cuts in weighted graphs. Lower the conductance the better. Condutance $\phi(U, C\backslash U)$ of a clustering $(U, C\backslash U)$ is defined as -

$$\phi(U, C\backslash U) = \frac{c(U, C\backslash U)}{min(deg(U), deg(C\backslash U)}$$

Among the Cut-based Clustering techniques commonly used are -

1. Static Hierarchies of Cut Clustering - Aim is to compute and organize all relevant minimum cuts of a graph into a nested hierarchical structure to form a hierarchy of clusters, so that the global cut structure of the graph can be further examined/analyzed. One algorithm for this was presented by Flake et al. that exploits the propoerties of minimum separating cut together with an input parameter α to find hierarchial clustering C.

2. Maximum Source-Community Clustering - identifies *communities* in a graph by repeatedly finding min-Cuts that best separate a chosen vertex from the rest of the graph. It can be used for Community Detection, Hierarchial Clustering Tree, etc.

3. Unrestricted Cut-Clustering Algorithm Similar to previous one this also aims at finding communities in a graph but without any restrictions on the end vertices $s - t$.

4. Fully Dynamic Hierarchies of Unrestricted Cut-Clustering Algorithm Instead of a Static unrestricted Cut-Clustering, this one accommodates for mobility in the clusters do dynamically update and detect or erase newer clusters. Similar to Dynamic Gomory-Hu trees this one also supports concrete changes on a concrete edge and finding changes at each step.

Bibliography

[BBDF06] D. Barth, Pascal Berthomé, Madiagne Diallo, and Afonso Ferreira. Revisiting parametric multi-terminal problems: Maximum flows, minimum cuts and cut-tree computations. discrete optimization. page 3(3):195–205, 2006.

[Elm64] Salah E. Elmaghraby. Sensivity analysis of multiterminal flow networks. operations research. page 12(5):680–688, 1964.

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

[GH61] R.E. GOMORY and Z. C. HU. Multiterminal network flows. *SIAM J. Appl. Math*, pages 551–570, 1961.

[GT89] Giorgio Gallo Michail D. Grigoriadis and Robert E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, page 18(1):30–55, 1989.

[Gus90] Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, pages 19(1):143–155, 1990.

[Har14] Tanja Hartmann. Algorithms for graph connectivity and cut problems. Master's thesis, 2014.

[HT13] Wagner D Hartmann T. Dynamic gomory–hu tree construction—fast and simple. 2013.