Fault-Tolerant All-Pairs Mincuts

Abhyuday Pandey

BT/CSE/170039

Supervisor: Dr. Surender Baswana

December 15, 2020

Abstract

Let $G = (V, E)$ be an undirected unweighted graph on n vertices and m edges. We address the problem of fault-tolerant data structure for all-pairs mincuts in G defined as follows.

Build a compact data structure that, on receiving a pair of vertices $s, t \in V$ and any edge (x, y) as query, can efficiently report the value of the mincut between s and t upon failure of the edge (x, y).

To the best of our knowledge, there exists no data structure for this problem which takes $o(mn)$ space and a non-trivial query time. We present two compact data structures for this problem.

1. Our first data structure guarantees $\mathcal{O}(1)$ query time. The space occupied by this data structure is $\mathcal{O}(n^2)$ which matches the worst-case size of a graph on n vertices.

2. Our second data structure takes $\mathcal{O}(m)$ space which matches the size of the graph. The query time is $\mathcal{O}(\min(m, nc_{s,t}))$ where $c_{s,t}$ is the value of the mincut between s and t in G. The query time guaranteed by our data structure is faster by a factor of $\Omega(\sqrt{n})$ compared to the best known algorithm [19, 24] to compute a (s, t)-mincut.

Both these data structures can also report the resulting (s, t)-mincut incorporating the failure in $\mathcal{O}(\min(m, nc_{s,t}))$ time.

Acknowledgments

I am deeply indebted to Prof Surender Baswana for allowing me to work with him. I am thankful to him for having regular meetings despite his busy schedule and helping me acquire the necessary perseverance for solving a research problem. I would like to thank Prof Yefim Dinitz and Prof Alek Vainshtein for their seminal work "The connectivity carcass of a vertex subset in a graph and its incremental maintenance" which appeared in STOC 1994 and subsequently in SIAM Journal of Computing 2000 edition. The paper truly captures the entire anatomy of mincuts and also forms the foundation of our results. Last but not the least, I would like to thank my parents and sister for their incredible and unconditional support in this pandemic.

Contents

Chapter 1

Introduction

Graph mincut is a fundamental structure in graph theory with numerous applications. Let $G = (V, E)$ be an undirected unweighted connected graph on $n = |V|$ vertices and $m = |E|$ edges. Two most common types of mincuts are global mincuts and pairwise mincuts. A set of edges with the least cardinality whose removal disconnects the graph is called a global mincut. For any pair of vertices $s, t \in V$, a set of edges with the least cardinality whose removal disconnects t from s is called a pairwise mincut for s, t or simply a (s, t)-mincut. A more general notion is that of Steiner mincuts. For any given set $S \subseteq V$ of vertices, a set of edges with the least cardinality whose removal disconnects S is called a Steiner mincut for S. It is easy to observe that the Steiner mincuts for $S = V$ are the global mincuts and for $S = \{s, t\}$ are (s, t)-mincuts.

While designing an algorithm for a graph problem, one usually assumes that the underlying graph is static. But, this assumption is unrealistic for most of the real-world graphs where vertices and/or edges do fail, though occasionally. While it is impractical to assume that a real-world graph is immune to such failures, it is also a fact that these failures are transient - an edge (or a vertex) once failed, becomes active after some time due to the simultaneous repair mechanism that is undertaken in most of the real-world graphs. So the set of failed edges (or vertices), though small in size, keeps changing with time. Therefore, a more realistic way to model a real-world graph is to assume that, at any time, there will be at most k failed vertices or edges for some k defined suitably. Solving a graph problem in this model requires building a compact data structure such that given any set of at most k failed vertices or edges, the solution of the problem can be reported efficiently.

In the past, many elegant fault-tolerant algorithms have been designed for various classical problems, namely, connectivity [6, 18, 7, 17], shortest-paths [4, 11, 8], graph spanners [9, 5], SCC [2] , DFS tree [1], and BFS structure [26, 27]. However, little is known about fault-tolerant data structures for various types of mincuts. The problem of fault-tolerant all-pairs mincuts aims at preprocessing a given graph to build a compact data structure so that the following query can be answered efficiently for any $s, t \in V$ and $(x, y) \in E$.

FT-MINCUT(s, t, x, y): Report a (s, t)-mincut in G after the failure of edge (x, y).

Upon failure of edge (x, y), the value of (s, t)-mincut for any $s, t \in V$ either decreases by unity or remains unchanged. We now state the necessary and sufficient condition for the value of (s, t)-mincut to decrease upon failure of edge (x, y).

Fact 1.1. The value of (s, t)-mincut decreases on failure of an edge (x, y) if and only if (x, y) lies in *some* (s, t)-mincut.

It follows from Fact 1.1 that any FT-MINCUT(s, t, x, y) query can be answered by determining whether (x, y) belongs to any (s, t)-mincut. Unfortunately, there can be exponential number of (s, t)-mincuts in a given graph [28]. Thus, designing a compact data structure to answer this query efficiently turns out to be a challenging task.

1.1 Previous Results

There exists a classical $\mathcal{O}(n)$ size data structure that stores all-pairs mincuts [20] known as Gomory-Hu tree. It is a tree on the vertex set V that compactly stores a mincut between each pair of vertices. However, we cannot determine using a Gomory-Hu tree whether the failure of an edge will affect the (s, t)-mincut unless this edge belongs to the (s, t)-mincut present in the tree. We can get a fault-tolerant data structure by storing m Gomory-Hu trees, one for each edge failure. The overall data structure occupies $\mathcal{O}(mn)$ space and takes $\mathcal{O}(1)$ time to report the value of (s, t)-mincut for any $s, t \in V$ upon failure of a given edge. However, the $\mathcal{O}(mn)$ space occupied by this data structure is far from the size of the graph. To the best of our knowledge, there exists no data structure for this problem which takes $o(mn)$ space and a non-trivial query time.

1.2 Our Contribution

We present two space efficient fault-tolerant data structures for the all-pairs mincuts problem in an undirected unweighted graph.

1. Our first data structure occupies $\mathcal{O}(n^2)$ space while achieving the optimal $\mathcal{O}(1)$ query time to report the value of (s,t)-mincut for any $s,t \in V$ upon failure of any given edge. However, the size of this data structure is still impractical for large real-world graphs which are usually sparse. So it is natural to ask whether we can have a more compact data structure at the expense of increased query time. Our second data structure answers this question in affirmative.

2. Our second data structure occupies $\mathcal{O}(m)$ space which matches the space required to store the given graph. The query time to report the value of (s,t)-mincut for any $s,t \in V$ upon failure of any given edge is $\mathcal{O}(\min(m, nc_{s,t}))$ where $c_{s,t}$ is the value of (s,t)-mincut in graph G. The query time guaranteed by our data structure is faster by a factor of $\Omega(\sqrt{n})$ compared to the best known algorithm [19, 24] to compute a (s,t)-mincut.

Both these data structures can also report the resulting (s,t)-mincut incorporating the failure in $\mathcal{O}(\min(m, nc_{s,t}))$ time. In order to design our data structures, we present an efficient solution for a related problem of independent interest, called edge-containment query on a mincut defined as follows.

EDGE-CONTAINED(s,t,E_y): Check if a given set of edges $E_y \subset E$ sharing a common endpoint y belong to some (s,t)-mincut.

Using a data structure for this problem, we can answer a fault-tolerant query upon failure of edge (x,y) as follows. We perform the corresponding edge-containment query for $E_y = \{(x,y)\}$. The value of (s,t)-mincut will reduce by unity if the edge-containment query evaluates to true. We can keep a Gomory-Hu tree of $\mathcal{O}(n)$ size as an auxiliary data structure to lookup the old value of $c_{s,t}$. Our first data structure can answer edge-containment queries for $|E_y| = 1$ in $\mathcal{O}(1)$ time. On the other hand, our second data structure works for any given set E_y but takes $\mathcal{O}(\min(m, nc_{s,t}))$ time.

Remark 1.2. It is worthwhile to note that edge-containment queries cannot be extended to handle more than one edge failure. This is because Fact 1.1 doesn't hold for multiple edges failures.

1.3 Related Work

A related problem is that of maintaining mincuts in a dynamic environment. Until recently, most of the work on this problem has been limited to global mincuts. Thorup [29] gave a Monte-Carlo algorithm for maintaining a global mincut of polylogarithmic size with $\tilde{\mathcal{O}}(\sqrt{n})$ update time. He also showed how to maintain a global mincut of arbitrary size with $1 + o(1)$-approximation within the same time-bound. Goranci, Henzinger and Thorup [21] gave a deterministic incremental algorithm for maintaining a global mincut with amortized $\tilde{\mathcal{O}}(1)$ update time and $\mathcal{O}(1)$ query time. Hartmann and Wagner [23] designed a fully dynamic algorithm for maintaining all-pairs mincuts which provided significant speedup in many real-world graphs, however, its worst-case asymptotic time complexity is not better than the best static algorithm for an all-pairs mincut tree. Recently, there is a fully-dynamic algorithm [10] that approximates all-pairs mincuts up to a nearly logarithmic factor in $\tilde{\mathcal{O}}(n^{2/3})$ amortized time against an oblivious adversary, and $\tilde{\mathcal{O}}(m^{3/4})$ time against an adaptive adversary. To the best of our knowledge, there exists no non-trivial dynamic algorithm for all-pairs *exact* mincut. We feel that our insights in this paper may be helpful in this problem.

1.4 Overview of our results

Dinitz and Vainshtein [15] presented a novel data structure called *connectivity carcass* that stores all Steiner mincuts for a given Steiner set $S \subseteq V$ in $\mathcal{O}(\min(m, nc_S))$ space, where c_S is the value of Steiner mincut. Katz, Katz, Korman and Peleg [25] presented a data structure of $\mathcal{O}(n)$ size for labeling scheme of all-pairs mincuts. This structure hierarchically partitions the vertices based on their connectivity in the form of a rooted tree. In this tree, each leaf node is a vertex in set V and each internal node ν stores the Steiner mincut value of the set $S(\nu)$ of leaf nodes in the subtree rooted at ν. We observe that if each internal node ν of the hierarchy tree is augmented with the connectivity carcass of $S(\nu)$, we get a data structure for the edge-containment query. This data structure occupies $\mathcal{O}(mn)$ space. An edge-containment query can be answered using the connectivity carcass at the Lowest Common Ancestor (LCA) of the given pair of vertices.

As we move down the hierarchy tree, the size of Steiner set associated with the internal node reduces. So, to make the data structure more compact, a possible approach is to associate a smaller graph G_ν for each internal node ν that is *small enough* to improve the overall space-bound, yet *large enough* to retain the internal connectivity of set $S(\nu)$. However, such a compact graph cannot directly answer the edge-containment query as it does not even contain the information about all edges in G. A possible way to overcome this challenge is to transform any edge-containment query in graph G to an equivalent query in graph G_ν. In this paper, we show that not only such a transformation exists, but it can also be computed efficiently. We model the query transformation as a multi-step procedure. The following result captures a single step of this procedure.

Given an undirected graph $G = (V, E)$ and a Steiner set $S \subseteq V$, let $S' \subset S$ be any maximal set with connectivity strictly greater than that of S. We can build a quotient graph $G_{S'} = (V_{S'}, E_{S'})$ such that $S' \subset V_{S'}$ with the following property.

For any two vertices $s, t \in S'$ and any set of edges E_y incident on vertex y in G, there exists a set of edges $E_{y'}$ incident on a vertex y' in $G_{S'}$ such that E_y lies in a (s,t)-mincut in G if and only if $E_{y'}$ lies in a (s,t)-mincut in $G_{S'}$.

We build graph G_μ associated with each internal node μ of the hierarchy tree as follows. For the root node r, $G_r = G$. For any other internal node μ, we use the above result to build the graph G_μ from $G_{\mu'}$, where μ' is the parent of μ. Our data-structure is the hierarchy tree where each internal node μ is augmented with the connectivity carcass for G_μ and the Steiner set $S(\mu)$. A high-level description of our query algorithm is as follows. We traverse the path from the root node to the LCA of s and t. We keep transforming the edge-containment query for each edge in this path. At the LCA of s and t, we stop and perform the query using the connectivity carcass stored at this node. Following a rigorous analysis, we show that this data structure takes only $\mathcal{O}(m)$ space and can answer any edge-containment query in $\mathcal{O}(\min(m, nc_{s,t}))$ time.

Chapter 2

Preliminaries

Let $G = (V, E)$ be an undirected unweighted multigraph without self-loops. To contract (or compress) a set of vertices $U \subseteq V$ means to replace all vertices in U by a single vertex u, delete all edges with both endpoints in u and for every edge which has one endpoint in U, replace this endpoint by u. A graph obtained by performing a sequence of vertex contractions is called a *quotient* graph of G.

For any given $A, B \subset V$ such that $A \cap B = \emptyset$, we use $c(A, B)$ to denote the number of edges with one endpoint in A and another in B. Overloading the notation, we shall use $c(A)$ for $c(A, \bar{A})$.

Definition 2.1 ((s,t)-cut)**.** A subset of edges whose removal disconnects t from s is called an (s,t)-cut. An (s,t)-mincut is an (s,t)-cut of minimum cardinality.

Definition 2.2 (set of vertices defining a cut)**.** A subset $A \subset V$ is said to define an (s,t)-cut if $s \in A$ and $t \notin A$. The corresponding cut is denoted by $\mathrm{cut}(A, \bar{A})$ or more compactly $\mathrm{cut}(A)$.

The following lemma exploits the undirectedness of the graph.

Lemma 2.3. Let x, y, z be any three vertices in G. If $c_{x,y} > c$ and $c_{y,z} > c$, then $c_{x,z} > c$ as well.

When there is no scope of confusion, we do not distinguish between a mincut and the set of vertices defining the mincut. We now state a well-known property of cuts.

Lemma 2.4 (Submodularity of cuts). For any two subsets $A, B \subset V$, $c(A) + c(B) \geq c(A \cup B) + c(A \cap B)$.

Lemma 2.5. Let $S \subset V$ define an (s,t)-mincut with $s \in S$. For any subset $S' \subset V \setminus S$ with $v \notin S'$,

$$c(S, S') \leq c(S, V \setminus (S \cup S'))$$

2.1 Compact representation for all (s,t)-mincuts

Dinitz and Vainshtein [15] showed that there exists a quotient graph of G that compactly stores all (s,t)-mincuts. This graph is called strip $\mathcal{D}_{s,t}$. The 2 node to which s and t are mapped in $\mathcal{D}_{s,t}$ are called the terminal nodes, denoted by $\mathbf{s}$ and $\mathbf{t}$ respectively. Every other node is called a non-terminal node. We now elaborate some interesting properties of the strip $\mathcal{D}_{s,t}$.

Consider any non-terminal node v, and let E_v be the set of edges incident on it in $\mathcal{D}_{s,t}$. There exists a unique partition, called *inherent partition*, of E_v into 2 subsets of equal sizes. These subsets are called the 2 sides of the inherent partition of E_v. Interestingly, if we traverse $\mathcal{D}_{s,t}$ such that upon visiting any non-terminal node using an edge from one side of its inherent partition, the edge that we traverse while leaving it belong to the other side of the inherent partition, then no node will be visited again. Such a path is called a *coherent* path in $\mathcal{D}_{s,t}$. Furthermore, if we begin traversal from a non-terminal node u along one side of its inherent partition and keep following a coherent path we are bound to reach the terminal $\mathbf{s}$ or terminal $\mathbf{t}$. So the two sides of the inherent partitions can be called side-$\mathbf{s}$ and side-$\mathbf{t}$ respectively. It is because of these properties that the strip $\mathcal{D}_{s,t}$ can be viewed as an undirected analogue of a directed acyclic graph with a single source and a single sink.

A cut in the strip $\mathcal{D}_{s,t}$ is said to be a *transversal* if each coherent path in $\mathcal{D}_{s,t}$ intersects it at most once. The following lemma provides the key insight for representing all (s,t)-mincuts through the strip $\mathcal{D}_{s,t}$.

Lemma 2.6 ([15]). $A \subset V$ defines a (s,t)-mincut if and only if A is a transversal in $\mathcal{D}_{s,t}$.

We now state the following two lemmas that can be viewed as a corollary of Lemma 2.6.

Lemma 2.7. A (s,t)-mincut contains a set of edges E_y incident on vertex y if and only if all edges in E_y must belong to the same side of the inherent partition of the node containing y in strip $\mathcal{D}_{s,t}$.

Lemma 2.8. If $A \subset V$ defines a (s,t)-mincut with $s \in A$, then A can be merged with the terminal node $\mathbf{s}$ in $\mathcal{D}_{s,t}$ to get the strip $\mathcal{D}_{A,t}$ that stores all those (s,t)-mincuts that enclose A.

Consider any non-terminal node x. Let $\mathcal{R}_s(x)$ be the set of all the nodes y in $\mathcal{D}_{s,t}$ that are reachable from x through coherent paths that originate from the side-$\mathbf{s}$ of the inherent partition of x – notice that all these paths will terminate at $\mathbf{s}$. It follows from the construction that $\mathcal{R}_s(x)$ defines a transversal in $\mathcal{D}_{s,t}$. We call $\mathcal{R}_s(x)$ the *reachability cone* of x towards s. The (s,t)-mincut defined by $\mathcal{R}_s(x)$ is the nearest mincut from $\{s,x\}$ to t. Interestingly, each transversal in $\mathcal{D}_{s,t}$, and hence each (s,t)-mincut, is a union of the reachability cones of a subset of nodes of $\mathcal{D}_{s,t}$ in the direction of s. We now state the following Lemma that we shall crucially use.

Lemma 2.9 ([15]). If $x_1,\ldots,x_k$ are any non-terminal nodes in strip $\mathcal{D}_{s,t}$, the union of the reachability cones of x_i's in the direction of $\mathbf{s}$ defines the nearest mincut between $\{s,x_1,\ldots,x_k\}$ and t.

2.2 Compact representation for all global mincuts

Let c_V denote the value of the global mincut of the graph G. Dinitz, Karzanov, and Lomonosov [12] showed that there exists a graph $\mathcal{H}_V$ of size $O(n)$ that compactly stores all global mincuts of G. Henceforth, we shall use nodes and structural edges for vertices and edges of $\mathcal{H}_V$ respectively. There exists a projection mapping $\pi : V(G) \to V(\mathcal{H}_V)$ assigning a vertex of graph G to a node in graph $\mathcal{H}_V$. In this way, any cut $(A,\bar{A})$ in cactus $\mathcal{H}_V$ is associated to a cut $(\pi^{-1}(A),\pi^{-1}(\bar{A}))$ in the original graph G. The graph $\mathcal{H}_V$ has a nice tree-like structure with the following properties.

1. Any two distinct simple cycle of $\mathcal{H}_V$ have at most a node in common. This is equivalent to the property that each structural edge of $\mathcal{H}_V$ belongs to at most one simple cycle. Each cut in $\mathcal{H}_V$ either corresponds to a tree edge or a pair of cycle edges in the same cycle.

2. If a stuctural edge belongs to a simple cycle, it is called a *cycle edge* and its weight is $\frac{c_V}{2}$. Otherwise, the structural edge is called a *tree edge* and its weight is c_V.

3. For any cut in the cactus $\mathcal{H}_V$, the associated cut in graph G is a global mincut. Moreover, any global mincut in G must have at least one associated cut in $\mathcal{H}_V$.

Let ν and μ be any two nodes in the cactus $\mathcal{H}_V$. If they belong to the same cycle, say c, there are two paths between them on the cycle c itself - their union forms the cycle itself. Using the fact that any two cycles in $\mathcal{H}_V$ can have at most one common node, it can be seen that these are the only paths between ν and μ. Using the same fact, if ν and μ are two arbitrary nodes in the cactus, there exists a unique path of cycles and tree edges between these two nodes. Any global mincut that separates ν from μ must correspond to a cut in this path.

2.2.1 Construction of (s, t)-strip from cactus

Suppose $s, t \in V$ are two vertices such that $c_{s,t}$ is same as the global mincut value. So, each transversal of strip $\mathcal{D}_{s,t}$ corresponds to a global mincut that separates s and t. Recall that cactus $\mathcal{H}_V$ stores all global mincuts. So we just need to contract it suitably so that only those cuts remain that separate s and t. For this purpose, we compute the path of cycles and tree edges between the nodes corresponding to s and t respectively. We compress each of the subcactus rooted to this path to a single vertex. The resultant graph we obtain will be the strip $\mathcal{D}_{s,t}$. The inherent partition of all the non-terminal units can be determined using the endpoints of the edges in the path.

2.2.2 Tree representation for cactus

We shall now show that $\mathcal{H}_V$ can be represented as a tree structure. This tree structure was also used by Dinitz and Westbrook in [16]. This representation will simplify our analysis on the cactus.

We now provide the details of the graph structure $T(\mathcal{H}_V)$ that represents $\mathcal{H}_V$. The vertex set of $T(\mathcal{H}_V)$ consists of all the cycles and the nodes of the cactus. For any node ν of the cactus $\mathcal{H}_V$, let $v(\nu)$ denote the corresponding vertex in $T(\mathcal{H}_V)$. Likewise, for any cycle π in the cactus, let $v(\pi)$ denote the corresponding vertex in $T(\mathcal{H}_V)$. We now describe the edges of $T(\mathcal{H}_V)$. Let ν

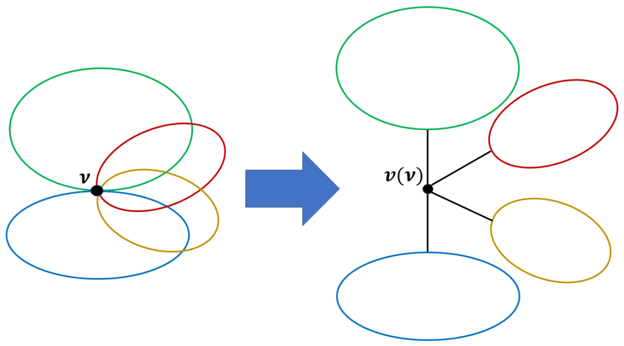

Figure 2.1: Transformation of cactus $\mathcal{H}_V$ to the tree $T(\mathcal{H}_V)$.

be any node of $\mathcal{H}_V$. Suppose there are j cycles - $\pi_1, \ldots, \pi_j$ that pass through it. We add an edge between $v(\nu)$ and $v(\pi_i)$ for each $1 \leq i \leq j$. Lastly, for each vertex $\nu(\pi)$ in $T(\mathcal{H}_V)$ we store all its neighbours in the order in which they appear in the cycle π in $\mathcal{H}_V$. This is done to ensure that information about the order of vertices in each cycle is retained. This complete the description of $T(\mathcal{H}_V)$. For a better understanding, the reader may refer to Figure 2.1 that succinctly depicts the transformation carried out at a node ν of the cactus graph to build the corresponding graph structure $T(\mathcal{H}_V)$.

The fact that the graph structure $T(\mathcal{H}_V)$ is a tree follows from the property that any two cycles in a cactus may have at most one vertex in common. Let us root $T(\mathcal{H}_V)$ at any arbitrary vertex, say $v(\nu)$, for some node ν of $\mathcal{H}_V$. Since each cycle in $\mathcal{H}_V$ has at least 3 vertices, so each vertex corresponding to a cycle of $\mathcal{H}_V$ will have at least 2 children each corresponding to distinct nodes of $\mathcal{H}_V$. This also shows that the number of cycles in $\mathcal{H}_V$ is at most half of the number of nodes in $\mathcal{H}_V$. Hence, the size of $T(\mathcal{H}_V)$ is of the order of the number of nodes of $\mathcal{H}_V$.

We know that if ν and μ are two nodes in the cactus, there exists a unique path of cycles and tree edges between them. It follows from the construction of $T(\mathcal{H}_V)$ that the unique path between the vertices $v(\nu)$ and $v(\mu)$ captures the same path. Thus we state the following lemma.

Lemma 2.10. Let ν, μ be any two arbitrary nodes in the cactus $\mathcal{H}_V$. The unique path between $v(\nu)$ and $v(\mu)$ in $T(\mathcal{H}_V)$ concisely captures all paths between ν and μ in $\mathcal{H}_S$.

We root the tree $T(\mathcal{H}_V)$ at any arbitrary vertex and augment it suitably so that it can answer any LCA query in $\mathcal{O}(1)$ time using [3]. Henceforth, we shall use skeleton tree $T(\mathcal{H}_S)$ to denote this data structure.

2.3 Compact representation for all Steiner mincuts

Dinitz and Vainshtein [13] designed a data structure $\mathfrak{C}_S = (\mathcal{F}_S, \mathcal{H}_S, \pi_S)$ that stores all the Steiner mincuts for a Steiner set $S \subseteq V$ in the graph. We present a summary of this data structure.

This data structure can be seen as a generalization of two already discussed data structures, (i) strip $\mathcal{D}_{s,t}$ storing all (s,t)-mincuts, and (ii) cactus graph $\mathcal{H}_V$ storing all global mincuts.

Two S-mincuts are said to be equivalent if they divide the Steiner set S in the same way. The equivalence classes thus formed are known as the *bunches*. Similarly, two vertices are said to be equivalent if they are not separated by any Steiner mincut. The equivalence classes thus formed are known as *units*. A unit is called a *Steiner unit* if it contains at least a Steiner vertex.

Let $(S_B, \bar{S}_B)$ be the $2-$partition of Steiner set induced by a bunch $\mathcal{B}$. If we compress all vertices in S_B to s and all vertices in $\bar{S}_B$ to t, the strip $\mathcal{D}_{s,t}$ will store all cuts in $\mathcal{B}$. We shall denote this strip by $\mathcal{D}_{\mathcal{B}}$. Any such strip has the following property – if two non-terminals nodes of two strips intersect at even one vertex then these nodes coincide and the inherent partitions of these nodes in both strips coincide as well.

The first component of the connectivity carcass is the *flesh graph* $\mathcal{F}_S$ which is a generalization of the strip. This graph is a quotient graph of graph G. The vertices of $\mathcal{F}_S$ can be obtained by contracting each unit of G to a single vertex. Thus, we denote the vertices of $\mathcal{F}_S$ simply by units. In addition to it, each unit of $\mathcal{F}_S$ is assigned a $2-$partition known as the *inherent partition* on the set of edges incident on it. Any unit that appears as a non-terminal in the strip corresponding to some bunch is called a *stretched unit*. Otherwise, it is called a *terminal unit*. The inherent partition assigned to a stretched unit consists of two sets of equal cardinality. On the other hand, inherent partition assigned to a terminal unit is a trivial partition (one of the set is empty). Note that all Steiner units are terminal units but the reverse is not true. The concept of reachability in $\mathcal{F}_S$ is similar to the strip. Whenever we say that a unit u is reachable from unit u', it means that there exists a coherent path between u and u'. The structure of $\mathcal{F}_S$ is such that a coherent path cannot start and finish at a single unit and hence, $\mathcal{F}_S$ is in a sense acyclic. There is a one-to-one correspondence between transversals in $\mathcal{F}_S$ and Steiner mincuts in G.

The second component of the connectivity carcass, skeleton $\mathcal{H}_S$, is a cactus graph. To avoid confusion with the original graph, the vertices and edges of the skeleton will be referred to as nodes and structural edges respectively. Each terminal unit of $\mathcal{F}_S$ is mapped to a node in the skeleton $\mathcal{H}_S$ by projection mapping π_S. A stretched unit on the other hand is mapped to a set of edges corresponding to a proper path in $\mathcal{H}_S$ by π_S. A *proper path* in the skeleton refers to an alternating sequence of nodes and structural edges $(\nu_1, \epsilon_1, \nu_2, ..., \nu_k)$ such that ϵ_i is incident on ν_{i-1} and ν_i and it intersects each cycle of the skeleton at at most one structural edge. All the bunches can be stored in a skeleton $\mathcal{H}_S$ in the form of subbunches (disjoint subsets of a bunch). Each cut in skeleton corresponds to a subbunch. The strip $\mathcal{D}_\mathcal{B}$ corresponding to this subbunch $\mathcal{B}$ can be obtained as follows. Let the cut in the skeleton separates it into two subcactuses $\mathcal{H}_S(\mathcal{B})$ and $\bar{\mathcal{H}}_S(\mathcal{B})$. If $P(\nu_1, \nu_2)$ be the path in the skeleton to which a unit u is mapped, it will be placed in $\mathcal{D}_\mathcal{B}$ as follows.

- If both ν_1 and ν_2 lie in $\mathcal{H}_S(\mathcal{B})$ (or $\bar{\mathcal{H}}_S(\mathcal{B})$) u is contracted in source (or sink).
- Otherwise, u is kept as a non-terminal unit.

Now we discuss an important property between the reachability of a stretched unit u and the proper path to which it is mapped in the skeleton $\mathcal{H}_S$.

Lemma 2.11 ([14]). Let u be a stretched unit and u' be any arbitrary unit in the flesh $\mathcal{F}_S$ and $\pi_S(u) = P(\nu_1, \nu_2)$, $\pi_S(u') = P(\nu_3, \nu_4)$. If u' is reachable from u in direction ν_2, then both these paths are extendable to a larger proper path with $P(\nu_1, \nu_2)$ as the initial part and $P(\nu_3, \nu_4)$ as the final part.

Lemma 2.12 ([13]). Let $s, t \in S$ such that $c_{s,t} = c_S$. Given the connectivity carcass $\mathfrak{C}_S$ storing all Steiner mincuts, the strip $\mathcal{D}_{s,t}$ can be constructed in time linear in the size of flesh graph.

The size of flesh $\mathcal{F}_S$ is $\mathcal{O}(\min(m, \tilde{n}c_S))$ where $\tilde{n}$ is the number of units in $\mathcal{F}_S$. The size taken by skeleton is linear in the number of Steiner units. Thus, overall space taken by the connectivity carcass is $\mathcal{O}(\min(m, \tilde{n}c_S))$.

Chapter 3

Insights Into 3-vertex mincuts

The main result of this section is the following theorem.

Theorem 3.1. Let $s, r, t \in V$ be any 3 vertices such that $c_{s,r} \geq c_{s,t}$. Let $A \subset V$ define a (s,t)-mincut with $s, r \in A$ and $t \in \bar{A}$. For any subset E_y of edges incident on any vertex $y \in \bar{A}$, there exists a subset E_A of edges from the mincut defined by A such that the following assertion holds.

There is a (r,s)-mincut containing E_y if and only if there is a (r,s)-mincut containing E_A.

In order to prove the theorem stated above, we first prove the following lemma.

Lemma 3.2 (3-vertex Lemma)**.** Let $s, r, t \in V$ be any three vertices and $c_{r,s} \geq c_{s,t}$. Let $A \subset V$ define an (s,t)-mincut as well as an (r,t)-mincut with $r, s \in A$ and $t \notin A$. Let $B \subset V$ define a (r,s)-mincut with $r \in B$. Without loss of generality, assume $t \in \bar{A} \cap \bar{B}$, then the following assertions hold:

1. $c(\bar{A} \cap B, A \cap \bar{B}) = 0$
2. $A \cap B$ defines a (r,s)-mincut.
3. $\bar{A} \cap \bar{B}$ defines a (s,t)-mincut as well as a (r,t)-mincut.

For a better illustration refer to Figure 3.1(ii)

Proof. Let $\alpha = c(\bar{A} \cap B, A \cap B)$, $\beta = c(\bar{A} \cap B, A \cap \bar{B})$, $\gamma = c(\bar{A} \cap B, \bar{A} \cap \bar{B})$. Refer to Figure 3.1 (i) that illustrates these edges and the respective cuts.

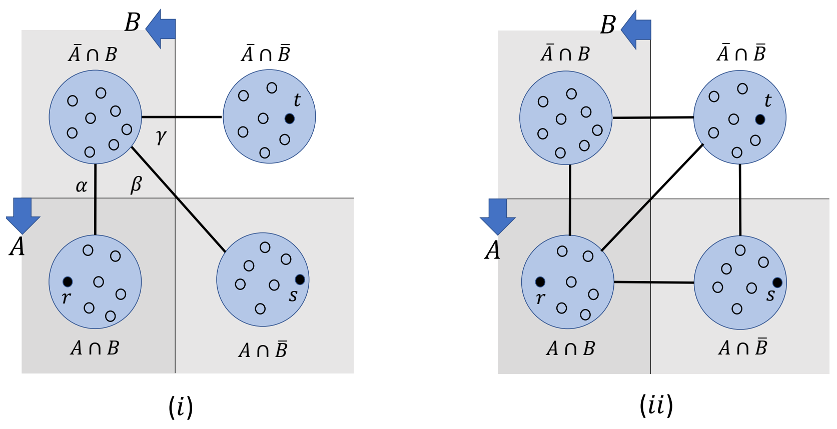

Figure 3.1: (i) α, β and γ denote the capacities of edges incident on $\bar{A} \cap B$ from $A \cap B$, $A \cap \bar{B}$, and $\bar{A} \cap \bar{B}$ respectively. (ii) There are no edges along the diagonal between $\bar{A} \cap B$ and $A \cap \bar{B}$.

Applying Lemma 2.5 on (s,t)-mincut with $S = A$ and $S' = \bar{A} \cap B$, we get

$$\alpha + \beta \leq \gamma \tag{3.1}$$

Applying Lemma 2.5 on (r,s)-mincut with $S = \bar{B}$ and $S' = \bar{A} \cap B$, we get $\gamma + \beta \leq \alpha$. This inequality combined with Inequality 3.1 implies that $\beta = 0$. That is, $c(\bar{A} \cap B, A \cap \bar{B}) = 0$. This completes the proof of Assertion (1). Refer to Figure 3.1 (ii) for an illustration. It follows from (1) that $\alpha = \gamma$. That is, $\bar{A} \cap B$ has equal number edges incident from $\bar{A} \cap \bar{B}$ as from $A \cap B$. This fact can be easily used to infer Assertions (2) and (3) by removing $\bar{A} \cap B$ from B and $\bar{A}$ respectively. $\square$

We shall now establish the proof of Theorem 3.1. Suppose B is any (r,s)-mincut containing edges E_y. Without loss of generality, assume that $r \in B$ and $s, t \notin B$. The following lemma states a crucial property of the cut defined by $A \cup B$ in strip $\mathcal{D}_{A,t}$, where **s** and **t** correspond to source A and sink t respectively.

Lemma 3.3. *$A \cup B$ will be a transversal in strip $\mathcal{D}_{A,t}$, and all edges in E_y are present in this cut.*

Proof. It follows from Lemma 3.2(3) that $A \cup B$ will be a (s,t)-mincut. Hence $A \cup B$ will be a transversal in strip $\mathcal{D}_{A,t}$ that stores all (s,t)-mincuts. From definition, y belongs to $\bar{A}$. Refer to Figure 3.1(ii). If $y \in \bar{A} \cap B$, then it follows from Lemma 3.2(1) that all neighbors of y corresponding to E_y will belong to $\bar{A} \cap \bar{B}$. So E_y belongs to the cut defined by $A \cup B$. The same holds for the case $y \in \bar{A} \cap \bar{B}$ as well since $B \subset A \cup B$. $\square$

It follows from Lemmas 3.3 and 2.7 that all edges in E_y must belong to the same side of the inherent partition of the node containing y in strip $\mathcal{D}_{A,t}$. Otherwise, there is no (r, s)-mincut which contains set of edges E_y. In this case, we can choose $E_A = E(A, \bar{A})$ and Theorem 3.1 trivially holds.

Let $x_1, \ldots, x_k$ be the neighbors of y defining E_y; that is, $E_y = \{(y, x_i) | \forall i \in [k]\}$. If all edges in E_y lie in side-**s**, $R = \bigcup_{i=1}^{k} \mathcal{R}_s(x_i) \setminus \{\mathbf{s}\}$, that is, the union of the reachability cones of x_i's in the strip $\mathcal{D}_{A,t}$ towards **s** excluding the terminal **s**. If all edges in E_y lie in side-**t**, $R = \mathcal{R}_s(y) \setminus \{\mathbf{s}\}$. We define E_A to be the set of edges which are incident from R to terminal **s** as well as the those edges in E_y having one endpoint in set A. Notice that all edges in set E_A belong to the cut defined by A. The following lemma suffices to establish Theorem 3.1.

Lemma 3.4. There is a (r, s)-mincut containing edges E_y if and only if there is a (r, s)-mincut containing all edges in set E_A.

Proof. $\mathcal{D}_{A,t}$ stores all (s, t)-mincuts that enclose the set A (see Lemma 2.8), and thus the mincut defined by $A \cup B$ as well. So all nodes of the strip $\mathcal{D}_{A,t}$, excluding the terminal node **s** must remain intact in the cut defined by B. Therefore, if we replace the subgraph of G induced by $\bar{A}$ by the strip $\mathcal{D}_{A,t}$ lying above **s**, the resulting graph, denoted by G_A, will preserve all (r, s)-mincuts of graph G. So in the remaining part of this proof, we focus on G_A instead of G. Let us consider the case when E_y is on side-**s**. The proof for the other case will follow likewise.

Let B be a (r, s)-mincut containing edges $\{(y, x_1), \ldots, (y, x_k)\}$. Refer to Figure 3.2($ii$) that demonstrates A and B in the graph G_A. Observe that y does not belong to A, so $\{x_1, \ldots, x_k\} \subset A \cup B$. Since $A \cup B$ is a transversal in $\mathcal{D}_{A,t}$, so $R \subset A \cup B$. It follows from the construction that R lies totally outside A, therefore, R must lie fully inside $\bar{A} \cap B$. Using this fact and Lemma 3.2 (1), all edges emanating from R on the side of **s** are incident only on $A \cap B$. But $A \cap B$ defines a (r, s)-mincut as follows from Lemma 3.2 (2). Furthermore, the cut defined by $A \cap B$ also contains all edges in E_y that have one endpoint in A. Thus, $A \cap B$ is the desired (r, s)-cut since it contains all edges in E_A.

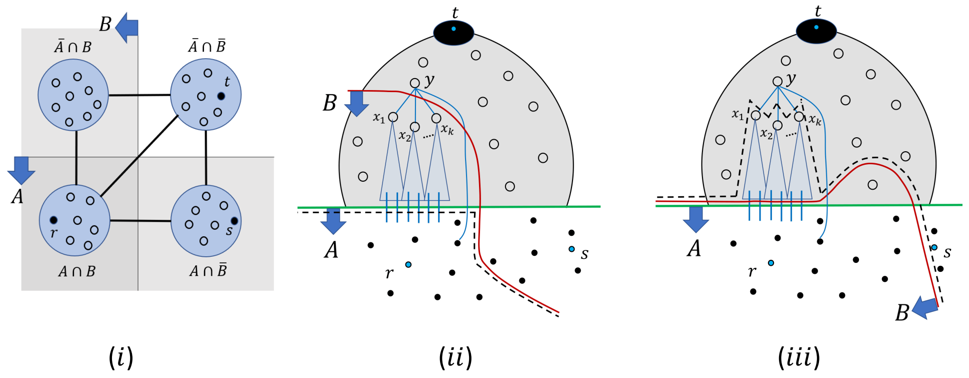

Figure 3.2: (i) There are no edges along the diagonal between $\bar{A} \cap B$ and $A \cap \bar{B}$. (ii) B cuts edges $\{(y, x_1), \ldots, (y, x_k)\}$. (iii) B cuts all outgoing edges of R.

Let B be a (r, s)-mincut that cuts all edges in set E_A. Refer to Figure 3.2(iii). By construction R lies outside A and all edges in E_y with one endpoint in A are contained in the cut defined by $A \cup B$. Therefore, the cut defined by $A \cup B$ cuts all edges in E_A. Since $A \cup B$ is a transversal in $\mathcal{D}_{A,t}$, it follows that $(A \cup B) \cap R = \emptyset$; otherwise it would imply a coherent path in strip $\mathcal{D}_{A,t}$ that intersects $A \cup B$ twice – a contradiction. So $B \cap R = \emptyset$ too. That is, R lies entirely on the side of t in the cut defined by B. Treating R as a single vertex, observe that its incoming edges are the same in number as its outgoing edges in $\mathcal{D}_{A,t}$. It is given that R currently contributes all its outgoing edges to the cut defined by B. So it follows that $B \cup R$, which also defines a (r, s)-cut, has the same capacity as B, but R now contributes all incoming edges to this cut. So $B \cup R$ is the desired (r, s)-mincut containing edges $(y, x_1), \ldots, (y, x_k)$. □

The following corollary follows from the construction in Proof of Lemma 3.4.

Corollary 3.4.1. Given a (r, s)-mincut that contains all edges in E_A another (r, s)-mincut can be constructed that contains all edges in E_y.

Chapter 4

Compact Graph for Query Transformation

Let $S \subseteq V$ be the Steiner set of vertices. Suppose $S' \subset S$ be any maximal subset of vertices with connectivity greater than that of S. Observe that the entire set S' will be mapped to a single node, say ν, in the skeleton $\mathcal{H}_S$. In this section, we present the construction of a compact graph $G_{S'}$ such that any query EDGE-CONTAINED(s, r, E_y) in graph G can be efficiently transformed to a query EDGE-CONTAINED$(s, r, E_{y'})$ in graph $G_{S'}$ for any $s, r \in S'$.

4.1 Construction of Compact Graph $G_{S'}$

The construction of $G_{S'}$ from the graph G, flesh $\mathcal{F}_S$ and skeleton $\mathcal{H}_S$ is a $2-$step process. In the 1st step, we contract the subcactuses neighbouring to node ν using the following procedure.

CONTRACT-SUBCACTUSES: For each tree-edge incident on ν (or cycle c passing through ν) in skeleton $\mathcal{H}_S$, remove the tree-edge (or the pair of edges from c incident on ν) to get 2 subcactuses. Compress all the terminal units of $\mathcal{F}_S$ that belong to the subcactus not containing ν into a single vertex. Moreover, compress all the stretched units with both endpoints within this subcactus into the same vertex.

In the quotient graph obtained after 1st step, each contracted subcactus defines a Steiner mincut. However, this graph may not necessarily be compact since there may be many stretched units that are not yet compressed. Let u be any such unit and suppose the path to which it is mapped in the

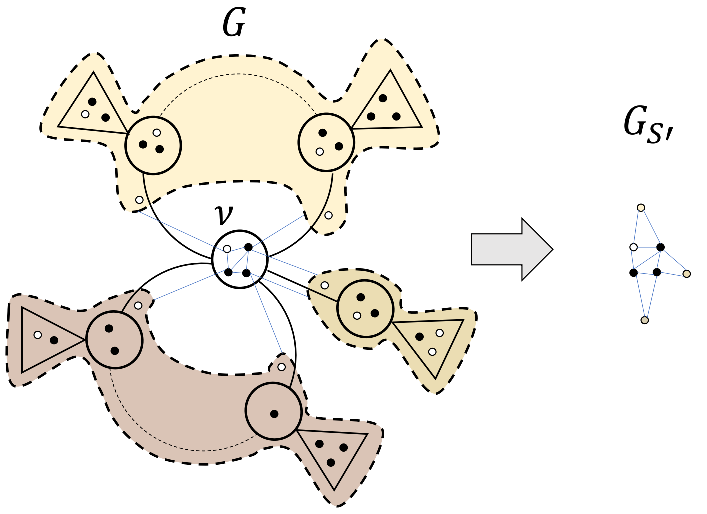

Figure 4.1: 2-step contraction procedure to construct $G_{S'}$. We only show the vertices and relevant edges of graph along with the skeleton $\mathcal{H}_S$. Solid vertices belong to Steiner set S and hollow vertices are non-Steiner vertices. All Steiner vertices inside node ν form the set S'.

skeleton is $P(\nu_1, \nu_2)$. If one of ν_1 or ν_2 is ν, we can compress the stretched unit to the contracted vertex corresponding to the other endpoint. To handle the case when the subcactuses containing ν_1 and ν_2 are compressed to different vertices, we define a total ordering on the set containing all tree-edges and cycles in the skeleton. The 2nd step uses this ordering to compress the stretched units as follows.

Contract-Stretched-Units: A stretched unit mapped to path $P(\nu_1, \nu_2)$, where $\nu_1 \neq \nu \neq \nu_2$, is compressed to the contracted subcactus corresponding to lesser ordered cycle/tree-edge in which endpoints lie. If one of ν_1 or ν_2 is ν, we compress it to the contracted subcactus corresponding to the cycle/tree-edge where other endpoint lies.

Figure 4.1 gives a nice illustration of the contraction procedure.

Let c be any cycle (or tree edge) passing through (incident on) ν in the skeleton $\mathcal{H}_S$. Let $\mathcal{D}_{s,t}$ be the strip corresponding to the sub-bunch defined by the structural edge(s) incident on ν by c. Let ν be on the side of the source $\mathbf{s}$ in this strip. Let $\mathcal{H}_S(c)$ be the subcactus formed by removing the structural edge(s) from c incident on ν and not containing ν. Recall that the subcactus $\mathcal{H}_S(c)$ was contracted into a vertex, say v_c, in the graph $G_{S'}$.

Lemma 4.1. Let u and u' be any two non-terminal units in $\mathcal{D}_{s,t}$ such that none of them is compressed to v_c in $G_{S'}$. If one of them is reachable from the other in the direction of $\mathbf{s}$, then both of them will be compressed to the same contracted vertex in $G_{S'}$.

Proof. Assume without loss of generality that u' is reachable from u in the direction of $\mathbf{s}$. Let the proper paths associated with each of u and u' in $\mathcal{H}_S$ be $P(\nu_1, \nu_2)$ and $P(\nu'_1, \nu'_2)$ respectively. It follows from the construction of $\mathcal{D}_{s,t}$ that $P(\nu_1, \nu_2)$ as well as $P(\nu'_1, \nu'_2)$ will pass through one of the structural edge(s) from c on ν. Without loss of generality, assume that $P(\nu_1, \nu_2)$ passes through e. Since $P(\nu_1, \nu_2)$ is a proper path, this implies that this is the only structural edge in this cut (of skeleton) through which this path passes. Since u' is reachable from u in flesh $\mathcal{F}_S$, so $P(\nu'_1, \nu'_2)$ will also have to pass through e (from Lemma 2.11). It again follows from Lemma 2.11, that $P(\nu_1, \nu_2)$ as well as $P(\nu'_1, \nu'_2)$ are subpaths of a path, say $P(\nu', \nu'')$, in skeleton $\mathcal{H}_S$. This combined with the above discussion establishes that $P(\nu', \nu'')$ has the structure shown in Figure 4.2.

Observe that any path in skeleton that passes through a node ν can intersect at most 2 cycles or tree-edges that are passing though ν. We know that suffix of $P(\nu', \nu'')$ after e lies in $\mathcal{H}_S(c)$, so the prefix upto e must have endpoint in subcactus $\mathcal{H}_S(c')$ where $c' \neq c$. This implies that u must be compressed to $v_{c'}$ because it is not compressed to v_c. Thus, c' precedes c in total order. It follows from the structure of path $P(\nu'_1, \nu'_2)$ that it will have an endpoint in $\mathcal{H}_S(c')$. Thus, u' will be compressed to the same compressed vertex $v_{c'}$ in $G_{S'}$. This completes the proof.

□

Consider the set of non-terminals in the strip $\mathcal{D}_{s,t}$ that are not compressed to contracted vertex v_c. Let this set be U. Observe that the set of units $\bigcup_{u \in U} \mathcal{R}_s(u)$ form a Steiner mincut (using Lemma 2.9). Moreover, it follows from Lemma 4.1 that each non-terminal unit in the set $\bigcup_{u \in U} \mathcal{R}_s(u)$ is not compressed to contracted vertex v_c. Thus, $U = \bigcup_{u \in U} \mathcal{R}_s(u) \setminus \{\mathbf{s}\}$. All the set of vertices compressed to v_c forms the complement of set $\bigcup_{u \in U} \mathcal{R}_s(u)$, and thus defines the same Steiner mincut. Therefore, the set of vertices corresponding to each contracted vertex defines a Steiner mincut.

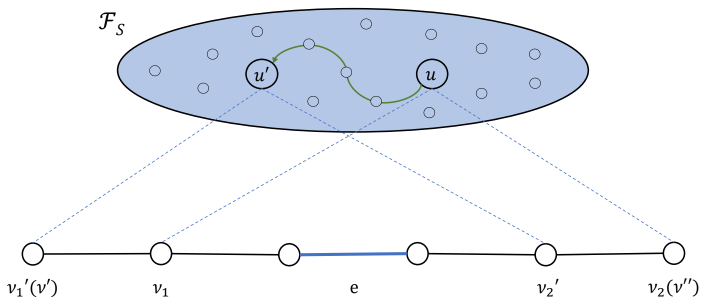

Figure 4.2: The structure of path $P(\nu', \nu'')$.

It follows from the construction that $G_{S'}$ is a quotient graph of G. Moreover, the number of contracted vertices equals the number of cycles and tree edges incident on node ν in the skeleton. We state the following lemma.

Lemma 4.2. Let $S' \subset S$ be a maximal subset of vertices such that $c_{S'} > c_S$ and ν be the node in $\mathcal{H}_S$ corresponding to S'. Let $G_{S'}$ be the graph obtained after 2-step contraction procedure.

1. The set of vertices compressed to a contracted vertex defines a Steiner mincut for set S.
2. The number of contracted vertices equals the number of cycles and tree edges incident on node ν in $\mathcal{H}_S$.

4.2 Query transformation in $G_{S'}$

We begin with a lemma that was used by Gomory and Hu to build a tree storing all-pairs mincuts.

Lemma 4.3 (Gomory and Hu [20]). Let A defines a (s,t)-mincut with $s \in A$. Let $r \in A$ be any vertex. For any (s,r)-mincut, say defined by B, there exists a (s,r)-mincut that keeps $\bar{A}$ intact and still contains all edges in cut defined by B that don't have both endpoints in $\bar{A}$.

Consider any two vertices $s, r \in S'$. Recall that S' is mapped to ν in the skeleton $\mathcal{H}_S$. Let A be the subset of vertices compressed to a contracted vertex in $G_{S'}$. Notice that all those (s,r)-mincuts in G that keep $\bar{A}$ intact remain preserved in $G_{S'}$. Moreover, it follows from Lemma 4.2 and Lemma

4.3 that there is at least one such (s,r)-mincut. So it suffices to work with graph $G_{S'}$ if one wishes to calculate the value of (s,r)-mincut in G or simply report a (s,r)-mincut in G for any $s,r \in S$. Moreover, we can answer a query EDGE-CONTAINED(s,r,E_y) using $G_{S'}$ if all edges in E_y remain intact in graph $G_{S'}$. However, answering a query EDGE-CONTAINED(s,r,E_y) for any arbitrary E_y using $G_{S'}$ is still challenging. This is because $G_{S'}$ may not even preserve all (s,r)-mincuts. In particular, all those (s,r)-mincuts that cut the set associated with a contracted vertex in $G_{S'}$ get lost during the transformation from G to $G_{S'}$. We shall now establish a mapping from the set of all such lost (s,r)-mincuts to the set of (s,r)-mincuts that are present in $G_{S'}$.

Let y belong to $\bar{A}$. It follows from Lemma 4.2 that the cut $(A,\bar{A})$ is a (s,t)-mincut for any $t \in S \cap \bar{A}$. Hence, A,s,t,r satisfy all conditions of Theorem 3.1. Now notice that entire $\bar{A}$ is compressed to a single vertex, say y', in $G_{S'}$. Hence we can state the following Theorem.

Theorem 4.4. Given an undirected graph $G=(V,E)$, a subset $S \subseteq V$, let $S' \subset S$ be a maximal subset of vertices such that $c_{S'} > c_S$. There exists a quotient graph $G_{S'}$ with the following property. For any two vertices $r,s \in S'$ and a set of edges E_y incident on vertex y in G, there exists a set of edges $E_{y'}$ incident on a vertex y' in $G_{S'}$ such that E_y lies in a (r,s)-mincut in G if and only if $E_{y'}$ lies in a (r,s)-mincut in $G_{S'}$.

We have already seen the construction of $G_{S'}$. In order to transform EDGE-CONTAINED(s,r,E_y) to EDGE-CONTAINED$(s,r,E_{y'})$ using Theorem 4.4, we give an efficient algorithm for computing $E_{y'}$. Moreover, once we find a (r,s)-mincut in $G_{S'}$ that contains $E_{y'}$ we can efficiently compute a (r,s)-mincut in G that contains all edges in E_y. Interestingly, we have algorithms that run in time linear in the size of flesh for both these tasks, stated in the following Lemma.

Lemma 4.5. Set of edges $E_{y'}$ in Theorem 4.4 can be obtained from E_y given flesh $\mathcal{F}_S$ and skeleton $\mathcal{H}_S$ in time linear in the size of flesh.

Lemma 4.6. Given a (r,s)-mincut in $G_{S'}$ that contains the all edges in set $E_{y'}$, we can construct a (r,s)-mincut in G that contains all edges in set E_y in time linear in the size of flesh $\mathcal{F}_S$.

Proof. Consider the case when y does not belong to any contracted vertex. In this case, all edges in E_y remain intact in $G_{S'}$ and thus $E_{y'} = E_y$.

Now, suppose y belong to contracted vertex y'. Let $\bar{A}$ be the set of vertices compressed to contracted vertex y'. We select a vertex $t \in \bar{A} \cap S$ and construct the $\mathcal{D}_{A,t}$ strip using the flesh $\mathcal{F}_S$ and skeleton $\mathcal{H}_S$ in time linear in the size of flesh (using Lemma 2.12). Using the construction outlined in Lemma 3.4 we can obtain the set of edges E_A by computing reachability cone(s) in strip $\mathcal{D}_{A,t}$. This takes time linear in size of $\mathcal{D}_{A,t}$. All edges in E_A share same endpoint y' in $G_{S'}$. Thus, we get the set of edges $E_{y'}$ which is simply all edges in $G_{S'}$ corresponding to set E_A. Clearly, this process can be accomplished in time linear in the size of flesh $\mathcal{F}_S$.

Suppose we have a (r, s)-mincut in $G_{S'}$, say B such that $s, t \notin B$ that contains all edges in $E_{y'}$. If y does not belong to any contracted vertex, this cut itself can be reported as $E_y = E_{y'}$. Suppose y belong to contracted vertex y'. We can construct another (r, s)-mincut $B \cup R$ (recall the definition of R in Proof of Lemma 3.4). This procedure also involves construction of $\mathcal{D}_{A,t}$ strip and computation of reachability cone(s) in this strip. This process can also be accomplished in time linear in the size of flesh $\mathcal{F}_S$. $\square$

Chapter 5

Compact Data Structures for Edge-Containment Queries

We describe first a hierarchical data structure of Katz, Katz, Korman and Peleg [25] that was used for compact labeling scheme for all-pairs mincuts. This hierarchical data structure is actually a rooted tree, denoted by $\mathcal{T}_G$ henceforth. The key insight on which this tree is built is an equivalence relation defined for a Steiner set $S \subseteq V$ as follows.

Definition 5.1 (Relation $\mathcal{R}_S$). Any two vertices $a, b \in S$ are said to be related by $\mathcal{R}_S$, that is $(a, b) \in \mathcal{R}_S$, if $c_{a,b} > c_S$, where c_S is the value of a Steiner mincut of S.

By using $\mathcal{R}_S$ for various carefully chosen instances of S, we can build the tree structure $\mathcal{T}_G$ in a top-down manner as follows. Each node ν of the tree will be associated with a Steiner set, denoted by $S(\nu)$, and the equivalence relation $\mathcal{R}_{S(\nu)}$. To begin with, for the root node r, we associate $S(r) = V$. Using $\mathcal{R}_{S(\nu)}$, we partition $S(\nu)$ into equivalence classes. For each equivalence class, we create a unique child node of ν; the Steiner set associated with this child will be the corresponding equivalence class. We process the children of ν recursively along the same lines. We stop when the corresponding Steiner set is a single vertex.

It follows from the construction described above that the tree $\mathcal{T}_G$ will have n leaves - each corresponding to a vertex of G. The size of $\mathcal{T}_G$ will be $O(n)$ since each internal node has at least 2 children. Notice that $S(\nu)$ is the set of vertices present at the leaf nodes of the subtree of $\mathcal{T}_G$ rooted at ν. The following observation captures the relationship between a parent and child node in $\mathcal{T}_G$.

Observation 5.2. Suppose $\nu \in \mathcal{T}_G$ and μ is its parent. $S(\nu)$ comprises of a maximal subset of vertices in $S(\mu)$ with connectivity strictly greater than that of $S(\mu)$.

The following observation conveys an important property about the value of (s,t)-mincut for any two vertices $s, t \in V$.

Observation 5.3. Suppose $s, t \in V$ are two vertices and μ is their LCA in $\mathcal{T}_G$ then $c_{s,t} = c_{S(\mu)}$.

$\mathcal{T}_G$ can be augmented to design a data structure for edge-containment query for any pair of vertices. For a single-edge-containment query, we get the following result.

5.1 An $\mathcal{O}(n^2)$ data structure for single edge-containment queries

Suppose S is a designated Steiner set and $s, t \in S$ are Steiner vertices separated by some Steiner mincut. We can determine if an edge $(x, y) \in E$ belongs to some (s,t)-mincut using the strip $\mathcal{D}_{s,t}$ that can be built from the connectivity carcass. However, the construction of strip requires $\mathcal{O}(\min(m, nc_S))$ time. Interestingly, we show that only the skeleton and the projection mapping of the connectivity carcass are sufficient for answering this query in constant time. Moreover, the skeleton and the projection mapping occupy only $\mathcal{O}(n)$ space compared to the $\mathcal{O}(\min(m, nc_S))$ space occupied by the entire connectivity carcass.

Similar to the projection mapping of the stretched units, Dinitz and Vainshtein [15] introduced the notion of projection mapping for edges as follows. Suppose $(x, y) \in E$. If x and y belong to the same unit, then $P(x, y) = \varnothing$. If x and y belong to distinct terminal units mapped to nodes, say ν_1 and ν_2, in the skeleton $\mathcal{H}_S$, then $P(x, y) = P(\nu_1, \nu_2)$. If at least one of them belongs to a stretched unit, $P(x, y)$ is the extended path defined in Lemma 2.11. This allows us to state the following lemma which follows from the construction of a strip corresponding to a subbunch given in Section 2.3.

Lemma 5.4 ([13]). Edge $(x, y) \in E$ appears in the strip corresponding to a subbunch if and only if one of the structural edge in the cut of $\mathcal{H}_S$ corresponding to this subbunch lies in $P(x, y)$.

We state the necessary and sufficient condition for an edge (x, y) to lie in an (s, t)-mincut. Note that two paths are said to intersect in the skeleton if the unique path of cycle and tree edges in both the paths intersect at some cycle or tree edge.

Lemma 5.5. Edge $(x, y) \in E$ belongs to a (s, t)-mincut if and only if the proper path $P(x, y)$ intersects a path between the nodes containing s and t in in $\mathcal{H}_S$.

Proof. Observe that an edge (x, y) lies in a (s, t)-mincut if and only if it appears in the strip $\mathcal{D}_{s,t}$ (follows from Lemma 2.7). Infact, we can extend this notion for subbunch as well. The edge (x, y) lies in some (s, t)-mincut if and only if it appears in the strip corresponding to some subbunch that separates s from t.

Consider each subbunch that separates s from t. Let ν_1 and ν_2 be the nodes in $\mathcal{H}_S$ containing s and t respectively. A cut in $\mathcal{H}_S$ corresponding to any tree-edge (or pair of cycle edges in same cycle) in the path from ν_1 to ν_2 defines a subbunch separating s from t. Moreover, it follows from the structure of the skeleton that no other cut in the skeleton corresponds to a subbunch separating s from t.

Suppose (x, y) lies in some (s, t)-mincut. Thus, it must be in some subbunch separating s from t. From the above discussion, we know that this subbunch must correspond to a cut in the path from ν_1 to ν_2 in skeleton $\mathcal{H}_S$. Moreover, it follows from Lemma 5.4 that $P(x, y)$ contains one of the structural edge in this cut. This implies that $P(x, y)$ intersects the path from ν_1 to ν_2 in skeleton $\mathcal{H}_S$.

Now, consider the other direction of this proof. Suppose $P(x, y)$ and the path from ν_1 to ν_2 intersect at some cycle (or tree edge) c. Let e_1 and e_2 be structural edges belonging to the cycle c that are part of $P(x, y)$ and path from ν_1 to ν_2 respectively (in the case of tree edge $e_1 = e_2 = c$). Consider the cut in the skeleton corresponding to structural edges e_1 and e_2. It follows from Lemma 5.4 that (x, y) lies in the strip corresponding to this subbunch. Since this cut separates ν_1 from ν_2 in $\mathcal{H}_S$, therefore the subbunch separates s from t. $\square$

We build our data structure using the findings of Lemma 5.5. We augment each internal node μ of the hierarchy tree $\mathcal{T}_G$ with the skeleton tree $T(\mathcal{H}_{S(\mu)})$ and the projection mapping $\pi_{S(\mu)}$ corresponding to Steiner set $S(\mu)$. Since augmentation at each internal node takes $\mathcal{O}(n)$ space, therefore the total space occupied by the data structure is only $\mathcal{O}(n^2)$.

Determining whether a given edge belongs to a (s,t)-mincut can be done as follows. Let μ be the LCA of s and t in $\mathcal{T}_G$. It follows from Observation 5.3 that $c_{s,t} = c_{S(\mu)}$. Thus, s and t must be separated by some Steiner mincut for set $S(\mu)$. We check if paths $P(x,y)$ and $P(\pi_{S(\mu)}(s), \pi_{S(\mu)}(t))$ intersect in the skeleton $\mathcal{H}_{S(\mu)}$ (using Lemma 5.5). This can be done using $\mathcal{O}(1)$ LCA queries on the skeleton tree $T(\mathcal{H}_{S(\mu)})$. Since it takes $\mathcal{O}(1)$ time for answering one LCA query [3], so the query time will be $\mathcal{O}(1)$ only. Algorithm 1 presents a concise pseudocode of the query answering algorithm.

Algorithm 1 Single edge-containment queries in $\mathcal{O}(n^2)$ data structure

```
 1: procedure EDGE-COTAINED($s, t, x, y$)
 2:     $\mu \leftarrow$ LCA($\mathcal{T}_G, s, t$)
 3:     $\mathcal{P}_1 \leftarrow P(\pi_{S(\mu)}(s), \pi_{S(\mu)}(t))$
 4:     $\mathcal{P}_2 \leftarrow P(x, y)$
 5:     if $\mathcal{P}_1 \cap \mathcal{P}_2 = \varnothing$ then
 6:         return False
 7:     else
 8:         return True
 9:     end if
10: end procedure
```

We can thus state the following theorem.

Theorem 5.6. Given an undirected graph $G = (V, E)$ on $n = |V|$ vertices, there exists a data structure of $\mathcal{O}(n^2)$ size that takes $\mathcal{O}(1)$ time to determine whether an edge $(x,y) \in E$ belongs to a (s,t)-mincut for any $s, t \in V$ and $(x,y) \in E$.

In the following section we present our $\mathcal{O}(m)$ size data structure which is more compact and can even handle multiple-edge-containment query.

5.2 An $\mathcal{O}(m)$ size data structure for edge-containment queries

Consider any node μ in $\mathcal{T}_G$. We associate a compact graph with node μ, say $G_\mu = (V_\mu, E_\mu)$ with the following properties.

1. G_μ is a quotient graph of G with $S(\mu) \subseteq V_\mu$
2. For each $s, t \in S(\mu)$ and a set of edges E_y incident on vertex $y \in V$, there exists a set of edges $E_{y'}$ incident on vertex $y' \in V_\mu$ such that E_y lies in a (s,t)-mincut in G if and only if $E_{y'}$ lies in a (s,t)-mincut in G_μ.

For the root node r, $G_r = G$ and the two properties hold trivially. We traverse $\mathcal{T}_G$ in a top down fashion to construct G_μ for each node $\mu \in \mathcal{T}_G$ as follows. Let μ be the parent of μ' in $\mathcal{T}_G$. Assume we have graph G_μ already built with the properties mentioned above. Thus, $S(\mu) \subseteq V_\mu$. Using Observation 5.2 we know that $S(\mu')$ is a maximal subset of $S(\mu)$ with connectivity strictly greater than that of $S(\mu)$. Using Theorem 4.4 with $S = S(\mu)$ and $S' = S(\mu')$, it can be shown that there exists a graph $G_{S(\mu')}$ that satisfies both the properties above. We define $G_{\mu'}$ to be $G_{S(\mu')}$. The graph $G_{\mu'}$ itself can be obtained using the 2-step contraction procedure described in Section 4.1.

Our compact data structure will be $\mathcal{T}_G$ where each node μ will be augmented with the connectivity carcass of $S(\mu)$ in graph G_μ.

5.2.1 The query algorithm

A query EDGE-CONTAINED(s, t, E_y) can be answered by the data structure as follows. We start from the root node of $\mathcal{T}_G$ and traverse the path to the node ν which is the LCA of s and t. Consider any edge (μ, μ') on this path, where μ is parent of μ'. We modify the query EDGE-CONTAINED(s, t, E_y) in G_μ to an equivalent query EDGE-CONTAINED$(s, t, E_{y'})$ in $G_{\mu'}$ as we move to μ' (see Theorem 4.4). This computation can be carried out in time linear in size of flesh at node μ using Lemma 4.5. We stop when $\mu = \nu$. Observe that $c_{s,t} = c_{S(\nu)}$ (using Observation 5.3) and thus must be separated by some Steiner mincut for $S(\nu)$. Thus we compute the strip $\mathcal{D}_{s,t}$ at node ν and answer the edge-containment query using Lemma 2.7. If the query evaluates to true, we compute a (s,t)-mincut in G_ν using $\mathcal{D}_{s,t}$ that contains the required set of edges at this level. We retrace the path from ν to the root of $\mathcal{T}_G$. Consider any edge (μ, μ') on this path where μ is parent of μ'. We find a corresponding (s,t)-mincut in graph G_μ from the (s,t)-mincut of $G_{\mu'}$ using Lemma 4.6. We stop at the root node and report the set of vertices defining the (s,t)-mincut in G. We can thus state the following theorem.

Theorem 5.7. Given an undirected graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, there exists a data structure of $\mathcal{O}(m)$ size that can determine whether any given subset of edges E_y incident on vertex y belongs to a (s,t)-mincut for any $s, t, y \in V$ and report the same if exists. The time taken to answer this query is $\mathcal{O}(\min(m, nc_{s,t}))$.

5.2.2 Size and Time analysis of compact data structure

The data structure doesn't seem to be an $\mathcal{O}(m)$ size data structure at first sight. Observe that augmentation at any internal node can still take $\mathcal{O}(m)$ space individually. Interestingly, we show that collective space taken by augmentation at each internal node will still be $\mathcal{O}(m)$.

We begin with the following lemma which gives a tight bound on the sum of weights of edges in Gomory-Hu tree is $\Theta(m)$. We also give the proof for the same which was suggested in [22, 15].

Lemma 5.8 **([22, 15]).** The sum of weights of all edges in the Gomory-Hu tree is $\Theta(m)$.

Proof. Consider any edge $(u,v) \in E$. This edge must be present in every (u,v)-mincut. Thus, the sum of weights of all edges in Gomory-Hu tree is at least m. Now, root the Gomory-Hu tree at any arbitrary vertex r. Let f maps each edge in this tree to its lower end-point. It is easy to observe that f is a one-to-one mapping. Let e be an edge in the Gomory-Hu tree. Observe that $w(e) \leq deg(f(e))$ (where $deg()$ is the degree of vertex in G). Thus the sum of weights of all edges in Gomory-Hu tree is at most $2m$. This comes from the simple observation that the sum of the degree of all vertices in G equals $2m$. □

Let us assign each edge (μ, μ') in the hierarchy tree $\mathcal{T}_G$ weight equal to the Steiner mincut value for the Steiner set $S(\mu)$ (if μ is the parent of μ'). We shall show that sum of the weight of edges in hierarchy tree $\mathcal{T}_G$ is $\Theta(m)$.

To establish this bound refer to algorithm 2 that gives an algorithm to construct the hierarchy tree from Gomory-Hu tree. Observe that the variable *ctr* in this algorithm stores the sum of weights of all edges in $\mathcal{T}_G$. It is clear that for k edges removed from the Gomory-Hu tree, we add $k+1$ edges of equal weight in $\mathcal{T}_G$. Thus, the sum of the weight of all edges in $\mathcal{T}_G$ is at most $4m$ (since $k+1 \leq 2k$). Therefore, we state the following lemma.

Algorithm 2 Construct Hierarchy Tree $\mathcal{T}_G$ from Gomory-Hu Tree $\hat{\mathcal{T}}_G$

```
1:  $ctr \leftarrow 0$
2:  procedure CONSTRUCT-TREE($\hat{\mathcal{T}}_G$)
3:      if $\hat{\mathcal{T}}_G$ has single node then
4:          Create a node $\mu$
5:          $val(\mu) \leftarrow val(\hat{\mathcal{T}}_G)$
6:          return $\mu$
7:      end if
8:      $c_{\min} \leftarrow \min_{e \in \hat{\mathcal{T}}_G} w(e)$
9:      Let there be $k$ edges with weight $c_{\min}$
10:     Remove all edges of weight $c_{\min}$ in $\hat{\mathcal{T}}_G$ to get $(k+1)$ trees $T_1, .., T_{k+1}$
11:     Create a node $\mu$
12:     $ctr \leftarrow ctr + c_{\min} \times (k+1)$
13:     $children(\mu) \leftarrow \{$CONSTRUCT-TREE$(T_i) \mid \forall i \in [k+1]\}$
14:     return $\mu$
15: end procedure
```

Lemma 5.9. The sum of weights of all edges in the tree $\mathcal{T}_G$ is $\Theta(m)$.

We shall now give a bound on the size of connectivity carcass augmented at each internal node. The following lemma gives a bound on the size of flesh graph $\mathcal{F}_S$ for any Steiner set S.

Lemma 5.10. Let $\mathcal{V}_S$ and $\mathcal{W}_S$ denote the set of Steiner and non-Steiner units respectively in flesh graph $\mathcal{F}_S$ with Steiner set $S \subseteq V$. The size of $\mathcal{F}_S$ is bounded by $|\mathcal{V}_S|c_S + \sum_{u \in \mathcal{W}_S} deg(u)$.

Proof. Consider the Gomory-Hu tree of the flesh $\mathcal{F}_S$, say $\mathcal{T}$. It is evident that the value of mincut between any two units is at most c_S. This follows from the definition of a unit. Now, root this tree $\mathcal{T}$ at some Steiner unit. Let f maps each edge in this tree to its lower end-point. Any edge in this tree has weight at most c_S. However, for any non-Steiner unit u, $w(f^{-1}(u)) \leq deg(u)$ (where $deg()$ is the degree of vertex in G). Thus, the sum of weight of all edges in $\mathcal{T}$ is bounded by $|\mathcal{V}_S|c_S + \sum_{u \in \mathcal{W}_S} deg(u)$. Using Lemma 5.8, it follows that size of flesh $\mathcal{F}_S$ is bounded by $|\mathcal{V}_S|c_S + \sum_{u \in \mathcal{W}_S} deg(u)$. □

Consider the flesh graph $\mathcal{F}_{S(\mu)}$ stored at some internal node μ in the tree. Let $\mathcal{V}_{S(\mu)}$ and $\mathcal{W}_{S(\mu)}$ denote the set of Steiner and non-Steiner units respectively in $\mathcal{F}_{S(\mu)}$. Let u be some non-Steiner unit in $\mathcal{F}_{S(\mu)}$. It is evident that u consists of only contracted vertices (obtained after the contraction procedure at some ancestral node). This non-Steiner unit gets compressed to a new contracted vertex in all descendants, and in a sense, disappears. Thus, each contracted vertex appears in at most one non-Steiner unit.

Now, we shall count the total number of contracted vertices introduced at each internal node. We know that this count is an upper bound on the total number of non-Steiner units across all flesh graphs. It follows from Lemma 4.2 that the number of contracted vertices introduced by node μ to the graph $G_{\mu'}$ associated with its child μ' is equal to the number of cycles and tree edges incident on node corresponding to μ' in skeleton $\mathcal{H}_{S(\mu)}$. We sum this number for each child of μ. The total number of contracted vertices introduced by internal node μ to all its children is at most twice the number of tree and cycle edges. Since the skeleton is a cactus graph, thus the number of tree and cycle edges is $\mathcal{O}(|\mathcal{V}_{S(\mu)}|)$. Moreover, we know that the number of Steiner units in $\mathcal{F}_{S(\mu)}$ also equals the number of children of node μ in tree $\mathcal{T}_G$. Therefore, the number of Steiner units across all flesh graphs stored at each internal node is given by $\sum_{\mu \in \mathcal{T}_G} |\mathcal{V}_{S(\mu)}|$ which is $\mathcal{O}(n)$. Thus, the total number of non-Steiner units across all flesh graphs is also $\mathcal{O}(n)$.

Now, we shall bound the sum of the degree of all non-Steiner units across all flesh graphs. Since each contracted vertex appears in at most one non-Steiner unit, we can sum the degree of all contracted vertices to get an upper bound. It again follows from Lemma 4.2 that the degree of contracted vertex introduced by node μ is exactly $c_{S(\mu)}$. Thus, the sum of degree of all contracted vertices introduced by node μ is $\mathcal{O}(|\mathcal{V}_{S(\mu)}|c_{S(\mu)})$.

Combining the above observations, we can infer the following.

Inference 5.11. The total number of units across all flesh graphs is $\mathcal{O}(n)$.

Inference 5.12. The sum of degree of non-Steiner units across all flesh graphs stored at each internal node μ is $\mathcal{O}(\sum_{\mu \in \mathcal{T}_G} |\mathcal{V}_{S(\mu)}|c_{S(\mu)})$ i.e $\mathcal{O}(m)$ (follows from Lemma 5.9). In other words, $\sum_{\mu \in \mathcal{T}_G} \sum_{u \in \mathcal{W}_{S(\mu)}} deg(u)$ is $\mathcal{O}(m)$.

Size analysis of Data Structure

Combining Lemma 5.8, Lemma 5.9, Lemma 5.10 and Inference 5.12 we get the following result.

$$\begin{aligned}\sum_{\mu \in \mathcal{T}_G} |\mathcal{F}_{S(\mu)}| &\leq c_1 \times \sum_{\mu \in \mathcal{T}_G} (|\mathcal{V}_{S(\mu)}| c_{S(\mu)} + \sum_{u \in \mathcal{W}_{S(\mu)}} deg(u)) \\ &\leq c_2 \times m\end{aligned}$$

Time analysis of Data Structure

A trivial bound on the query time follows from the size analysis itself. Since the combined size of our data structure is $\mathcal{O}(m)$, it follows that the sum of sizes of all flesh graphs from the root node to $LCA(s,t)$ will also be $\mathcal{O}(m)$. Dinitz and Vainshtein [13] showed that size of flesh graph $\mathcal{F}_S$ is $\mathcal{O}(\tilde{n}c_S)$ for Steiner set S, where $\tilde{n}$ is the number of units in the flesh graph. Total number of units across all flesh graphs is only $\mathcal{O}(n)$ (Inference 5.11). The value of Steiner mincut increases as we traverse from the root towards a leaf. Thus, $c_{s,t}$ is the maximum Steiner mincut value in the path from root node to $LCA(s,t)$. Thus, the sum of sizes of all flesh graphs in this path is bounded by $\mathcal{O}(nc_{s,t})$. Thus, the query time we achieve is $\mathcal{O}(\min(m, nc_{s,t}))$.

Bibliography

[1] Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: Breaking the O(m) barrier. *SIAM J. Comput.*, 48(4):1335–1363, 2019.

[2] Surender Baswana, Keerti Choudhary, and Liam Roditty. An efficient strongly connected components algorithm in the fault tolerant model. *Algorithmica*, 81(3):967–985, 2019.

[3] Michael A. Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *J. Algorithms*, 57(2):75–94, 2005.

[4] Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 101–110. ACM, 2009.

[5] Gilad Braunschvig, Shiri Chechik, David Peleg, and Adam Sealfon. Fault tolerant additive and (μ, α)-spanners. *Theor. Comput. Sci.*, 580:94–100, 2015.

[6] Timothy M. Chan. Dynamic subgraph connectivity with geometric applications. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 7–13. ACM, 2002.

[7] Timothy M. Chan, Mihai Patrascu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011.

[8] Shiri Chechik and Sarel Cohen. Distance sensitivity oracles with subcubic preprocessing time and fast query time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1375–1388. ACM, 2020.

[9] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010.

[10] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. *CoRR*, abs/2005.02368, 2020.

[11] Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[12] E.A. Dinitz, A.V. Karzanov, and M.V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In *Studies in Discrete Optimizations*, 1976.

[13] Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 716–725. ACM, 1994.

[14] Yefim Dinitz and Alek Vainshtein. Locally orientable graphs, cell structures, and a new algorithm for the incremental maintenance of connectivity carcasses. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 302–311. ACM/SIAM, 1995.

[15] Yefim Dinitz and Alek Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000.

[16] Yefim Dinitz and Jeffery R. Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20(3):242–276, 1998.

[17] Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 490–509. SIAM, 2017.

[18] Daniele Frigioni and Giuseppe F. Italiano. Dynamically switching vertices in planar graphs. *Algorithmica*, 28(1):76–103, 2000.

[19] Andrew V. Goldberg and Satish Rao. Flows in undirected unit capacity networks. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 32–34. IEEE Computer Society, 1997.

[20] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[21] Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in polylogarithmic amortized update time. *ACM Trans. Algorithms*, 14(2):17:1–17:21, 2018.

[22] Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An õ(mn) gomory-hu tree construction algorithm for unweighted graphs. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 605–614. ACM, 2007.

[23] Tanja Hartmann and Dorothea Wagner. Fast and simple fully-dynamic cut tree construction. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, volume 7676 of *Lecture Notes in Computer Science*, pages 95–105. Springer, 2012.

[24] David R. Karger and Matthew S. Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. In Jeffrey Scott

Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 69–78. ACM, 1998.

[25] Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004.

[26] Merav Parter and David Peleg. Sparse fault-tolerant BFS structures. *ACM Trans. Algorithms*, 13(1):11:1–11:24, 2016.

[27] Merav Parter and David Peleg. Fault-tolerant approximate BFS structures. *ACM Trans. Algorithms*, 14(1):10:1–10:15, 2018.

[28] Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Math. Program.*, 22(1):121, 1982.

[29] Mikkel Thorup. Fully-dynamic min-cut. *Comb.*, 27(1):91–127, 2007.