Sensitivity Oracles for All-Pairs Mincuts

Surender Baswana[*] Abhyuday Pandey[†]

Abstract

Let $G = (V, E)$ be an undirected unweighted graph on n vertices and m edges. We address the problem of sensitivity oracle 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 failure (or insertion) of any edge as query, can efficiently report the mincut between s and t after the failure (or the insertion).

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 the following results.

1. Our first data structure occupies $\mathcal{O}(n^2)$ space and guarantees $\mathcal{O}(1)$ query time to report the value of resulting (s, t)-mincut upon failure (or insertion) of any edge. Moreover, the set of vertices defining a resulting (s, t)-mincut after the update can be reported in $\mathcal{O}(n)$ time which is worst-case optimal.
2. Our second data structure optimizes space at the expense of increased query time. It takes $\mathcal{O}(m)$ space – which is also the space taken by G. 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. This query time is faster by a factor of $\Omega(\min(m^{1/3}, \sqrt{n}))$ compared to the best known deterministic algorithm [20, 26, 28] to compute a (s, t)-mincut from scratch.

[*]Department of Computer Science & Engineering, IIT Kanpur, Kanpur – 208016, India, sbaswana@cse.iitk.ac.in

[†]Department of Computer Science & Engineering, IIT Kanpur, Kanpur – 208016, India, pandey.abhyuday07@gmail.com

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.

Typically a graph algorithmic problem is solved assuming that the underlying graph is static. However, changes are inevitable in most real world graphs. For example, real world graphs are prone to failures of edges. These failures are transient in nature. As a result, the set of failed edges at any time, though small in size, may keep changing as time goes by. Therefore, a data structure for a problem on a real world graph has to consider a given set of failed edges as well to answer a query correctly. It is also important to efficiently know the *change* in the solution of the problem for a given set of failed edges or a given set of new edges. This knowledge can help to efficiently determine the *impact* of the failure/insertion of an edge on the solution of the problem. Solving a graph problem in this model requires building a compact data structure such that given any set of at most k changes, i.e. failures (or insertions) of edges (or vertices), the solution of the problem can be reported efficiently. It is important to note that this model subsumes the fault-tolerant model, in which we are only concerned about failure of edges (or vertices).

In the past, many elegant fault-tolerant data structures have been designed for various classical problems, namely, connectivity [19, 8, 17], shortest-paths [6, 12, 9], graph spanners [10, 7], SCC [2] , DFS tree [1], and BFS structure [29, 30]. Recently, a data structure was designed which could report the mincut between a pair of vertices amidst insertion of an edge [3]. However, little is known about fault-tolerant data structures for various types of mincuts.

The problem of sensitivity oracle for all-pairs mincuts aims at preprocessing a given graph to build a compact data structure so that the following queries can be answered efficiently for any $s, t \in V$ and $(x, y) \in V \times V$.

- FT-MINCUT(s, t, x, y): Report a (s, t)-mincut in G after the failure of edge $(x, y) \in E$.
- IN-MINCUT(s, t, x, y): Report a (s, t)-mincut in G upon insertion of edge $(x, y) \in V \times V$.

Previous Results: There exists a classical $\mathcal{O}(n)$ size data structure that stores all-pairs mincuts [22] 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. Recently, for the case of single edge insertion only, Baswana, Gupta, and Knollman [3] designed an $\mathcal{O}(n^2)$ data structure that can efficiently report the value of a (s, t)-mincut upon insertion of an edge (x, y) for any given

$s, t, x, y \in V$. To the best of our knowledge, there exists no sensitivity data structure for all-pairs mincuts problem which takes $o(mn)$ space and has a non-trivial query time.

Our Contribution: We present two sensitivity data structures for the all-pairs mincuts problem in an undirected unweighted graph.

1. Our first data structure occupies $\mathcal{O}(n^2)$ space and guarantees $\mathcal{O}(1)$ query time to report the value of resulting (s,t)-mincut for any $s, t \in V$ upon failure/insertion of any edge. Moreover, the set of vertices defining a resulting (s,t)-mincut after the update can be reported in $\mathcal{O}(n)$ time which is worst-case optimal. Our data structure also subsumes the previous results of Baswana, Gupta, and Knollman [3] which only handles an edge insertion. An interesting byproduct of our data structure, which is of independent interest, is that it can output a compact representation [32] storing all (s,t)-mincuts for any $s, t \in V$ in $\mathcal{O}(m)$ time which is optimal (see Appendix G).

2. Our second data structure occupies $\mathcal{O}(m)$ space. The query time guaranteed by this data structure to report the value of (s,t)-mincut for any $s, t \in V$ upon failure/insertion of any edge is $\mathcal{O}(\min(m, nc_{s,t}))$ where $c_{s,t}$ is the value of (s,t)-mincut in graph G. This query time is faster by a factor of $\Omega(\min(m^{1/3}, \sqrt{n}))$ (details in Appendix A) compared to the best known deterministic algorithm [20, 26, 28] to compute a (s,t)-mincut from scratch.

Remark 1.1. There exists a Las-Vegas algorithm [26] to compute static (s,t)-mincut in $\mathcal{O}(\max(m, nc_{s,t}))$ time. However, no deterministic algorithm till date matches this bound even within the factor of $m^{o(1)}$ [20, 26, 28].

Our first data structure achieves optimal query time but uses $\mathcal{O}(n^2)$ space. The second data structure, on the other hand, takes linear space but has $\mathcal{O}(\min(m, nc_{s,t}))$ query time. It remains an open problem if there exists a data structure that can achieve a space-time trade-off. Interestingly, this problem also remains open for much studied static all-pairs directed reachability problem [31, 21].

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 Fact 1.2, any fault-tolerant query can be answered using old value of (s,t)-mincut and edge-containment query on (s,t)-mincut for $E_y = \{(x,y)\}$. Fact 1.3 helps us in answering the query of (s,t)-mincut upon the insertion of any edge.

Fact 1.2. The value of (s,t)-mincut decreases (by unity) on failure of an edge (x,y) if and only if (x,y) lies in *at least one* (s,t)-mincut.

Fact 1.3. The value of (s,t)-mincut increases (by unity) on insertion of an edge (x,y) if and only if x and y are separated in *all* (s,t)-mincuts.

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 [33] 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 [23] 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 [25] 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 [11] 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 the insights developed in our paper may be helpful for this problem.

Overview of our results: Dinitz and Vainshtein [13, 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 the Steiner mincut. Katz, Katz, Korman and Peleg [27] 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 $\mathcal{T}$ 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. However, this data structure will require $\mathcal{O}(m)$ time to report the mincut between the given pair of vertices upon failure of an edge. As mentioned in previous results, storing m copies of Gomory-Hu tree trivially is a better data structure for this problem.

We focus our attention on a fixed s, t pair. There exists a dag structure [32, 15] that stores all (s, t)-mincuts. Exploiting the acyclic structure of the dag and the transversality of each (s, t)-mincut in this DAG, we design a data structure that occupies just $\mathcal{O}(n)$ space. Not only it can report the value of (s, t)-mincut after failure of an edge in $\mathcal{O}(1)$ time but also a resulting (s, t)-mincut in $\mathcal{O}(n)$ time. We can trivially store this data structure for all possible (s, t) pairs to get a sensitivity oracle for all-pairs mincuts. However, the $\mathcal{O}(n^3)$ space taken by this data structure is worse than the trivial structure mentioned in previous results.

In their seminal work, Dinitz and Vainshtein [13, 15] also showed that there exists a multi-terminal dag-like structure that implicitly stores all Steiner mincuts. This structure is called *flesh* and takes the majority of the space in the connectivity carcass. Now, the question arises if we can exploit the acyclicity and transversality of Steiner mincuts just like the way we did it for (s, t)-mincuts, to get a space efficient data structure for the problem. We show that the concepts analogous to topological ordering can be extended to the flesh as well. Moreover, we observe that only a part of connectivity carcass, avoiding the flesh, is required for our query. This part can be suitably augmented to get a data structure that can report a (s, t)-mincut upon failure of an edge if s and t are Steiner vertices separated by some Steiner mincut. This data structure takes only $\mathcal{O}(n)$ space and can report the value of (s, t)-mincut and a resulting such cut in $\mathcal{O}(1)$ and $\mathcal{O}(n)$ time respectively. By augmenting each internal node of the hierarchy tree $\mathcal{T}$ with this data structure, we get our $\mathcal{O}(n^2)$ space sensitivity oracle.

As we move down the hierarchy tree, the size of Steiner set associated with an 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. 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. We now provide the crucial insight that 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 hierarchy tree $\mathcal{T}$ as follows. For the root node r, $G_r = G$. For any other internal node μ, we use the above property to build the graph G_μ from $G_{\mu'}$, where μ' is the parent of μ. Our $\mathcal{O}(m)$ space data-structure is the hierarchy tree $\mathcal{T}$ where each internal node μ is augmented with the connectivity carcass for G_μ and the Steiner set $S(\mu)$.

Organization of the paper: In addition to the basic preliminaries, Section 2 presents compact representation for various mincuts. Section 3 gives the details of the $\mathcal{O}(n^2)$ space sensitivity oracle. Section 4 gives insights into 3-vertex mincuts that form the foundation for transforming an edge-containment query in original graph to a compact graph. Section 5 builds tools for handling edge-insertion in linear space data structure. We give the construction of compact graph for query transformation in Section 6. Using this compact graph as a building block, we present the linear space sensitivity data structure in Section 7.

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 a (s,t)-cut. An (s,t)-mincut is a (s,t)-cut of minimum cardinality.

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

Detailed Preliminaries for Section 4 and beyond can be found in Appendix F.

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, 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. 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 2.3 ([15])**.** *$A \subset V$ defines a (s,t)-mincut if and only if A is a transversal in $\mathcal{D}_{s,t}$.*

The following lemma can be viewed as a corollary of Lemma 2.3.

Lemma 2.4. *A (s,t)-mincut contains an edge (x,y) if and only it appears in strip $\mathcal{D}_{s,t}$.*

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. It follows from the definition that each transversal in the strip $\mathcal{D}_{s,t}$ is defined by union of reachability cones in the direction of $\mathbf{s}$ for a set of non-terminals.

2.2 Compact representation for all Global mincuts

Dinitz, Karzanov, and Lomonosov [18] showed that there exists a graph $\mathcal{H}_V$ of size $\mathcal{O}(n)$ that compactly stores all global mincuts of G. In order to maintain the distinction between G and $\mathcal{H}_V$, henceforth, we shall use nodes and structural edges for vertices and edges of $\mathcal{H}_V$ respectively. Each vertex in G is mapped to a unique node in $\mathcal{H}_V$. The graph $\mathcal{H}_V$ has a nice tree-like structure with the following properties.

1. Any two distinct simple cycles of $\mathcal{H}_V$ have at most one node in common. So each structural edge of $\mathcal{H}_V$ belongs to at most one simple cycle. As a result, each cut in $\mathcal{H}_V$ either corresponds to a tree edge or a pair of cycle edges in the same cycle.

2. Let c_V denote the value of the global mincut of the graph G. If a structural 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 $\mathcal{H}_V$.

Property 1 implies that there exists a unique path of cycles and tree edges between any two arbitrary nodes ν and μ. Any global mincut separating ν from μ corresponds to a cut in this path.

The cactus $\mathcal{H}_V$ can be stored in a tree-structure [16], denoted by $T(\mathcal{H}_V)$. 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 $T(\mathcal{H}_V)$. For a tree edge $(\nu_1, \nu_2) \in \mathcal{H}_V$ we add an edge between $v(\nu_1)$ and $v(\nu_2)$ as well. Let ν be any node of $\mathcal{H}_V$ and let there be j cycles - $\pi_1, \ldots, \pi_j$ that pass through it. We add an edge between $v(\nu)$ and $v(\pi_i)$ for each $1 \leq i \leq j$. Moreover, 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 $\mathcal{H}_V$ to ensure that cycle information is retained. This completes the description of $T(\mathcal{H}_V)$.

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. It is a simple exercise to show that the size of $T(\mathcal{H}_V)$ is of the order of the number of nodes of $\mathcal{H}_V$, which is $\mathcal{O}(n)$.

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 [5]). Henceforth, we use *skeleton tree* for this structure.

Extendability of Proper Paths A *proper path* in a cactus refers to a path which contains at most one structural edge from a cycle. It is easy to observe that there is at most one proper path between a pair of nodes in the cactus. We describe a transitive relation between proper paths on skeleton called *extendable in a direction*.

Definition 2.5 (Extendable in a direction)**.** Consider two proper paths $P_1 = P(\nu_1, \nu_2)$ and $P_2 = P(\nu_3, \nu_4)$. P_2 is said to be extendable from P_1 in direction ν_2 if proper paths P_1 and P_2 are extendable to a proper path $P(\nu, \nu')$ with P_1 as the initial part and P_2 as the final part.

Lemma 2.6. Given two paths $P_1 = P(\nu_1, \nu_2)$ and $P_2 = P(\nu_3, \nu_4)$ in cactus $\mathcal{H}_V$, the following queries can be answered using skeleton tree $T(\mathcal{H}_V)$ using $\mathcal{O}(1)$ LCA queries.
1. Determine if P_1 intersects P_2 at a tree edge or cycle, and report the intersection (if exists).
2. Determine if P_2 is extendable from P_1 in direction ν_2 (given that P_1, P_2 are proper paths), and report the extended path (if exists).

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 (or S-mincuts) for a Steiner set $S \subseteq V$ in the graph. This data structure generalizes − (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, $\mathcal{D}_{s,t}$ 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 of two strips intersect at even one vertex then these nodes along with the inherent partitions coincide.

The first component of $\mathfrak{C}_S$, *flesh graph* $\mathcal{F}_S$, is a generalization of the strip. It is a quotient graph of G. The vertices of $\mathcal{F}_S$, denoted by *units*, can be obtained by contracting each unit of G to a single vertex. 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. A unit u is said to be reachable from unit u' if 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 same 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 S-mincuts in G.

The second component of $\mathfrak{C}_S$, *skeleton* $\mathcal{H}_S$, is a cactus graph. Each terminal unit of $\mathcal{F}_S$ is mapped to a node in the $\mathcal{H}_S$ 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. All the bunches can be stored in $\mathcal{H}_S$ in the form of subbunches (disjoint subsets of a bunch). Each cut in $\mathcal{H}_S$ 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.

1. 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).
2. Otherwise, u is kept as a non-terminal unit.

Following lemma conveys the relation between reachability of a stretched unit u and $\pi_S(u)$.

Lemma 2.7 ([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 $P(\nu_3, \nu_4)$ is extendable from $P(\nu_1, \nu_2)$ in direction ν_2. (see Definition 2.5)

Lemma 2.8 ([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 notion of projection mapping can be extended for an edge $(x, y) \in E$ as follows. 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 2.7. Projection mapping of (x, y) can be computed in $\mathcal{O}(1)$ time (using Lemma 2.6). The following lemma establishes a relation between projection mapping of an edge and the subbunches in which it appears.

Lemma 2.9 ([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 $P(x, y)$.

2.4 Compact representation of all-pairs mincuts values

We describe a hierarchical tree structure of Katz, Katz, Korman and Peleg [27] that was used for compact labeling scheme for all-pairs mincuts, denoted by $\mathcal{T}_G$. The key insight on which this tree is built is an equivalence relation defined for a Steiner set $S \subseteq V$ as follows.

Definition 2.10 (Relation $\mathcal{R}_S$)**.** Any two vertices $u, v \in S$ are said to be related by $\mathcal{R}_S$, that is $(u, v) \in \mathcal{R}_S$, if $c_{u,v} > c_S$, 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 $\mathcal{O}(n)$ 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 2.11. 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 allows us to use $\mathcal{T}_G$ for looking up (s,t)-mincut values for any $s, t \in V$.

Observation 2.12. Suppose $s, t \in V$ are two vertices and μ is their LCA in $\mathcal{T}_G$ then $c_{s,t} = c_{S(\mu)}$.

3 $\mathcal{O}(n^2)$ space sensitivity oracle for all-pairs mincuts

In this section, we shall present data structures that can handle edge-containment query for – (i) fixed source and destination pair $s, t \in V$ (in Section 3.1) and (ii) for $s, t \in S$ and $c_{s,t} = c_S$ (in Section 3.2). In Section 3.3, we augment the tree structure of Katz, Katz, Korman and Peleg [27] to get a sensitivity oracle for all-pairs mincuts.

3.1 Edge-containment query for fixed $s, t \in V$

Consider any edge $(x, y) \in E$. It follows from the construction of strip $\mathcal{D}_{s,t}$ that edge (x, y) lies in a (s,t)-mincut if and only if x and y are mapped to different nodes in $\mathcal{D}_{s,t}$. This query can be answered in $\mathcal{O}(1)$ time if we store the mapping from V to nodes of $\mathcal{D}_{s,t}$. This requires only $\mathcal{O}(n)$ space.

Reporting a (s,t)-mincut that contains edge (x, y) requires more insights. Without loss of generality, assume that (x, y) lies in side-$\mathbf{t}$ of $\mathbf{x}$. If $\mathbf{x}$ is the same as $\mathbf{s}$, the set of vertices mapped to node $\mathbf{s}$ define a (s,t)-mincut that contains (x, y). Thus, assume that $\mathbf{x}$ is a non-terminal in the strip $\mathcal{D}_{s,t}$. Observe that the set of vertices mapped to the nodes in the reachability cone of $\mathbf{x}$ towards $\mathbf{s}$, $\mathcal{R}_s(\mathbf{x})$, defines a (s,t)-mincut that contains edge (x, y). Unfortunately, reporting this mincut requires $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ space. However, exploiting the acyclic structure of the strip and the transversality of each (s,t)-mincut, we get an important insight stated in the following lemma. This lemma immediately leads to an $\mathcal{O}(n)$ size data structure that can report a (s,t)-mincut containing edge (x, y) in $\mathcal{O}(n)$ time. (Proof in Appendix B).

Lemma 3.1. Consider the strip $\mathcal{D}_{s,t}$ with $\mathbf{x}$ as a non-terminal and edge (x, y) lying on side-$\mathbf{t}$ of $\mathbf{x}$. Suppose τ is a topological order on the nodes in the strip. The set of vertices mapped to the nodes in set $X = \{u \mid \tau(u) \leq \tau(\mathbf{x})\}$ defines a (s,t)-mincut that contains edge (x, y).

3.2 Edge-containment query for Steiner set S

Suppose S is a designated Steiner set and $s, t \in S$ are Steiner vertices separated by some Steiner mincut. It follows from Lemma 3.1 that we can determine if an edge $(x, y) \in E$ belongs to some (s, t)-mincut using the strip $\mathcal{D}_{s,t}$. Though this strip can be built from the connectivity carcass $\mathfrak{C}_S$, the time to construction it will be $\mathcal{O}(\min(m, nc_S))$. Interestingly, we show that only the skeleton and the projection mapping of $\mathfrak{C}_S$ 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 $\mathfrak{C}_S$.

The data structure $\mathfrak{D}(S)$ for edge containment query consists of the following components.

1. The skeleton tree $T(\mathcal{H}_S)$.

2. The projection mapping π_S of all units in flesh.

Lemma 3.2. Given an undirected unweighted multigraph $G = (V, E)$ on $n = |V|$ vertices and a designated steiner set $S \subseteq V$, $\mathfrak{D}(S)$ takes $\mathcal{O}(n)$ space and can report if an edge (x, y) lies in some (s, t)-mincut in $\mathcal{O}(1)$ time for any given $s, t \in S$ separated by some Steiner mincut.

Proof. We shall first show that an 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 $\mathcal{H}_S$. We say that two paths *intersect* if they intersect at a tree edge or cycle.

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 a 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 $\mathcal{B}$ separating s from t. $\mathcal{B}$ must correspond to a cut $\mathcal{C}$ in the path from ν_1 to ν_2 in skeleton $\mathcal{H}_S$. Also, $P(x, y)$ must contain (one of) the structural edge(s) defining $\mathcal{C}$ (from Lemma 2.9). Thus, $P(x, y)$ 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 the 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 2.9 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.

We can check if paths $P(\nu_1, \nu_2)$ and $P(s, t)$ in the skeleton $\mathcal{H}_S$ intersect with $\mathcal{O}(1)$ LCA queries on skeleton tree $\mathcal{T}(\mathcal{H}_S)$ (from Lemma 2.6). Thus, we can determine if an edge (x, y) lies in an (s, t)-mincut in $\mathcal{O}(1)$ time. The data structure takes only $\mathcal{O}(n)$ space. □

Reporting a (s, t)-mincut that contains edge (x, y) again requires more insights. Assume that $P(s, t)$ and $P(\nu_1, \nu_2)$ intersect at some tree edge or cycle. Let e be a tree or cycle-edge in proper path $P(\nu_1, \nu_2)$ that lies in intersection of these two paths. Suppose $\mathcal{B}$ is a subbunch corresponding to a cut in the skeleton $\mathcal{H}_S$ that contains e and separates s from t and $\mathcal{D}_\mathcal{B}$ be the strip corresponding to this subbunch. Without loss in generality, assume that ν_1 lies in the side of source s in this strip (denoted by $\mathbf{s}$). Using Lemma 2.9, it is evident that edge (x, y) lies in strip $\mathcal{D}_\mathcal{B}$. Assume $\mathbf{x}$ is a stretched unit in this strip, otherwise the source $\mathbf{s}$ is the required (s, t)-mincut. Suppose (x, y) lies in side-$\mathbf{t}$ of $\mathbf{x}$. The set of vertices mapped to $\mathcal{R}_s(\mathbf{x})$, i.e. reachability cone of $\mathbf{x}$ towards $\mathbf{s}$ in this strip, defines a (s, t)-mincut that contains edge (x, y). However, reporting this mincut is a tedious task. We must have the flesh $\mathcal{F}_S$ to construct the strip $\mathcal{D}_{\mathcal{B}}$ and then report the set $\mathcal{R}_s(\mathbf{x})$. This would require $\mathcal{O}(m)$ space and $\mathcal{O}(m)$ time.

Using insights developed in Section 3.1, we strive to report another (s,t)-mincut that contains (x,y) and has a simpler structure. In particular, we aim to report a set of units $Y = \{u \mid \tau_{\mathcal{B}}(u) \leq \tau_{\mathcal{B}}(\mathbf{x})\}$ for some topological ordering $\tau_{\mathcal{B}}$ of nodes in strip $\mathcal{D}_{\mathcal{B}}$. Using Lemma 3.1, we know that set of vertices mapped to nodes in set Y defines a (s,t)-mincut that contains edge (x,y). Unfortunately, we cannot store topological order of each stretched unit for each possible bunch in which it appear as a non-terminal. This is because doing so will require $\mathcal{O}(n.|S|^2)$ space. We show that we can augment $\mathfrak{D}(S)$ with an additional mapping τ that takes only $\mathcal{O}(n)$ space and can still efficiently report the set Y. τ maps each stretched unit in $\mathfrak{D}(S)$ to a number as follows. For all stretched units mapped to path $P(\nu_1,\nu_2)$, τ assigns a topological order on these stretched units as they appear in the (ν_1,ν_2)-strip. This additional augmentation will help us determine all those units of Y which are mapped to same proper path as $\mathbf{x}$. The challenge now is to identify the units of Y that are not mapped to the same path as $\mathbf{x}$. We use the notion of extendability of proper paths (Definition 2.5) to find such units.

Suppose stretched unit $\mathbf{x}$ is mapped to path $P(\nu,\nu')$ (ν lies in source $\mathbf{s}$). Let X be the set of stretched units appearing as non-terminals in strip $\mathcal{D}_{\mathcal{B}}$ for which one of the following holds true – (i) the stretched unit (say v) is mapped to $P(\nu,\nu')$ and $\tau(v) \leq \tau(\mathbf{x})$, or (ii) the stretched unit is not mapped to $P(\nu,\nu')$ but $\pi_S(v)$ is extendable from $P(\nu,\nu')$ in direction ν. The following lemma shows that $\mathbf{s} \cup X$ defines a desired (s,t)-mincut.

Lemma 3.3. The vertices mapped to units in $\mathbf{s} \cup X$ define a (s,t)-mincut and contains (x,y).

Proof. Consider $u \in X$ to be a non-terminal unit in $\mathcal{D}_{\mathcal{B}}$. We shall show that $\mathcal{R}_{\mathbf{s}}(u) \setminus \mathbf{s} \subseteq X$, i.e. reachability cone of u towards source $\mathbf{s}$ in the strip $\mathcal{D}_{\mathcal{B}}$ avoiding $\mathbf{s}$ is a subset of X. Suppose $\pi_S(u) = P(\mu,\mu')$ where μ is in source $\mathbf{s}$. It follows from the construction that either $P(\mu,\mu') = P(\nu,\nu')$ or $P(\mu,\mu')$ is extendable in direction ν from $P(\nu,\nu')$. Consider any unit v in $\mathcal{R}_{\mathbf{s}}(u) \setminus \mathbf{s}$. Suppose v is projected to $P(\nu,\nu')$. In this case, clearly $P(\mu,\mu') = P(\nu,\nu')$ (using Lemma 2.7). Since, v is reachable from u in direction ν, it follows that $\tau(v) < \tau(u) < \tau(\mathbf{x})$. Thus, $v \in X$. Now, suppose v is not projected to $P(\nu,\nu')$. In this case, $\pi_S(v)$ is extendable from $P(\mu,\mu')$ in direction μ (from Lemma 2.7). It follows from the transitivity of Definition 2.5 that $\pi_S(v)$ is extendable from $P(\nu,\nu')$ in direction ν. Thus, $v \in X$. Therefore, $\mathbf{s} \cup X$ defines a (s,t)-mincut (from Lemma 2.3).

Consider edge (x,y). If y is in $\mathbf{t}$ then $y \notin X$ from the construction. Thus, assume y is a non-terminal unit in $\mathcal{D}_{\mathcal{B}}$. If y is projected to path $P(\nu,\nu')$ then $\tau(y) > \tau(u)$. Thus, $y \notin X$. Otherwise $\pi_S(y)$ is extendable from $P(\nu,\nu')$ in direction ν'. It follows from Definition 2.5 that $y \notin X$. Thus, the cut defined by $\mathbf{s} \cup X$ contains edge (x,y).

□

This data structure $\mathfrak{D}(S)$ now occupies $\mathcal{O}(n)$ space and can report a (s,t)-mincut containing edge (x,y) in $\mathcal{O}(n)$ time. Using this data structure, we build a sensitivity oracle for all-pairs mincuts.

3.3 Edge-containment Query for all-pairs Mincuts

The hierarchical tree structure $\mathcal{T}_G$ [27] can be suitably augmented to design a sensitivity oracle for all-pairs mincuts. We augment each internal node ν of the hierarchy tree $\mathcal{T}_G$ with $\mathfrak{D}(S(\nu))$. Determining if given edge (x,y) lies in some (s,t)-mincut for given pair of vertices $s,t \in V$ can be done using Algorithm 1 in constant time.

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

```
procedure EDGE-CONTAINED($s, t, x, y$)
    $\mu \leftarrow \text{LCA}(\mathcal{T}_G, s, t)$
    $\mathcal{P}_1 \leftarrow P(\pi_{S(\mu)}(s), \pi_{S(\mu)}(t))$
    $\mathcal{P}_2 \leftarrow P(x, y)$
    if $\mathcal{P}_1$ and $\mathcal{P}_2$ intersect at a tree edge or cycle then        // Using Lemma 2.6
        return True
    else
        return False
    end if
end procedure
```

We state the following theorem (for edge-insertion see Appendix H).

Theorem 3.4. Given an undirected unweighted multigraph $G = (V, E)$ on $n = |V|$ vertices, there exists an $\mathcal{O}(n^2)$ size sensitivity oracle that can report the value of (s, t)-mincut for any $s, t \in V$ upon failure (or insertion) of an edge in $\mathcal{O}(1)$ time. Moreover, a (s, t)-mincut after the failure (or insertion) can be reported in $\mathcal{O}(n)$ time.

4 Insights into 3-vertex mincuts

The main result of this section is the following theorem.

Theorem 4.1. 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 $t \in \overline{A}$. For any subset E_y of edges incident on any vertex $y \in \overline{A}$, 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 4.2 (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 $t \in \overline{A} \cap \overline{B}$, then the following assertions hold:

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

For a better illustration refer to Figure 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 1 (i) that illustrates these edges and the respective cuts.

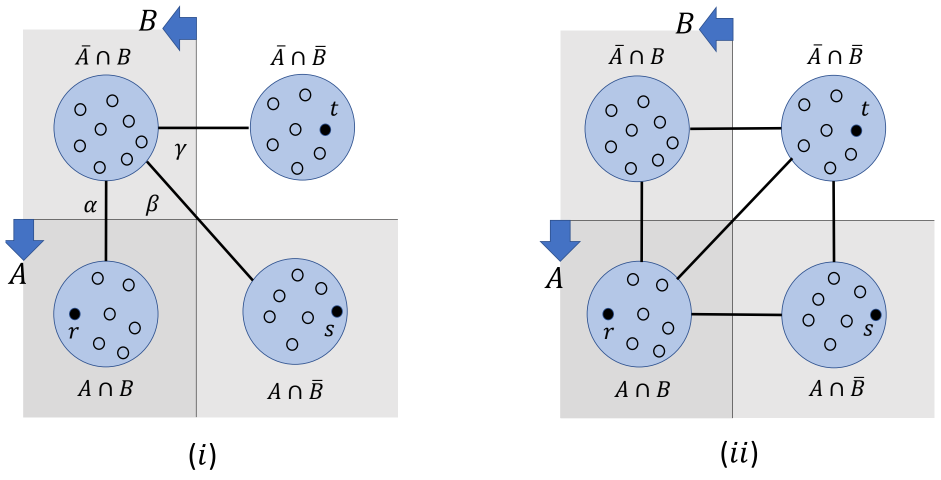

Figure 1: (i) α, β and γ denote the capacities of edges incident on $\bar{A} \cap B$ from $A \cap B$, $A \cap \bar{B}$, and $\bar{A} \cap \bar{B}$ respectively. (ii) There are no edges along the diagonal between $\bar{A} \cap B$ and $A \cap \bar{B}$.

Applying Lemma F.5 on (s,t)-mincut with $S = A$ and $S' = \bar{A} \cap B$, we get

$$\alpha + \beta \leq \gamma \tag{1}$$

Applying Lemma F.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 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 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 4.1. 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 $\mathbf{s}$ and $\mathbf{t}$ correspond to source A and sink t respectively.

Lemma 4.3. $A \cup B$ 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 4.2(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 1(ii). If $y \in \bar{A} \cap B$, then it follows from Lemma 4.2(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 4.3 and 2.4 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 4.1 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-$\mathbf{s}$, $R = \bigcup_{i=1}^{k} \mathcal{R}_s(x_i) \setminus \{\mathbf{s}\}$, that is, the union of the reachability cones of x_i's in the strip $\mathcal{D}_{A,t}$ towards $\mathbf{s}$ excluding the terminal $\mathbf{s}$. If all edges in E_y lie in side-$\mathbf{t}$, $R = \mathcal{R}_s(y) \setminus \{\mathbf{s}\}$. We define E_A to be the set of edges which are incident from R to terminal $\mathbf{s}$ as well as 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 4.1.

Lemma 4.4. 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 F.7), 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 $\mathbf{s}$ must remain intact in the cut defined by B. Therefore, if we replace the subgraph of G induced by $\overline{A}$ by the strip $\mathcal{D}_{A,t}$ lying above $\mathbf{s}$, 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-$\mathbf{s}$. 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 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 $\overline{A} \cap B$. Using this fact and Lemma 4.2 (1), all edges emanating from R on the side of $\mathbf{s}$ are incident only on $A \cap B$. But $A \cap B$ defines a (r, s)-mincut as follows from Lemma 4.2 (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.

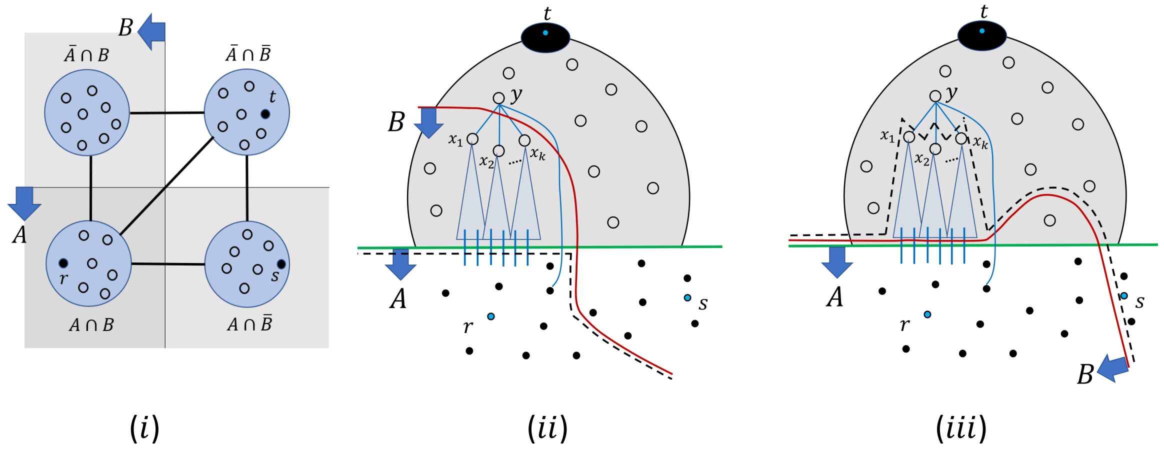

Figure 2: (i) There are no edges along the diagonal between $\overline{A} \cap B$ and $A \cap \overline{B}$. (ii) B cuts edges $\{(y, x_1), \ldots, (y, x_k)\}$. (iii) B cuts all outgoing edges of R.

Let B be a (r, s)-mincut that cuts all edges in set E_A. Refer to Figure 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 4.4.

Corollary 4.4.1. 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.

5 Insights into Nearest Mincuts

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 $t \in \overline{A}$. Suppose $y \in \overline{A}$ be another vertex. Suppose G_A is the graph obtained by compressing the set $\overline{A}$ to a single vertex. To keep the ideas simple, we shall denote this compressed vertex by $\overline{A}$. Using the notations from 2, sets s_r^N and r_s^N define the nearest (s, r)-mincut from s to r and r to s respectively in graph G. We state the following lemma. The proof follows from the definition of nearest mincuts.

Lemma 5.1. *If y lies in nearest s to r mincut in G, then $\overline{A}$ lies in nearest s to r mincut in G_A.*

Proof. We know that each (s, r)-mincut defined by $(B, \overline{B})$ in G_A is also a (s, r)-mincut in G. Suppose $\overline{A} \notin B$ and $s \in B$ where $(B, \overline{B})$ define nearest s to r mincut. Thus, $y \notin B$. Therefore, y does not lies in nearest s to r mincut in G. □

Suppose $\mathbf{s}$ and $\mathbf{t}$ correspond to source A and sink t respectively of strip $\mathcal{D}_{A,t}$. Suppose $R = \mathcal{R}_{\mathbf{s}}(y) \setminus \{\mathbf{s}\}$ denote the set of vertices that form the reachability cone of y towards source $\mathbf{s}$ in this strip (excluding $\mathbf{s}$). We define E_A to be the set of edges which are incident from R to terminal $\mathbf{s}$. We state the following lemma which concisely captures the condition for y to lie in nearest s to r mincut.

Lemma 5.2. *y lies in the nearest s to r mincut in G if and only if the following conditions hold:*

1. *$\overline{A}$ lies in nearest s to r mincut in G_A.*
2. *There is no (s, r)-mincut that contains all edges in E_A.*

Proof. The first proposition follows from Lemma 5.1. Thus, in the following analysis, we shall assume that $\overline{A}$ lies in nearest s to r mincut in G_A. Suppose $(\overline{B}, B)$ defines the nearest s to r mincut in G. It follows from 4.2 that $t \in \overline{B}$, otherwise $A \cap \overline{B}$ will define a s to r mincut which does not contain $\overline{A}$.

Suppose y does not lies in nearest s to r mincut i.e. $y \in \overline{A} \cap B$. In this case $R \subset \overline{A} \cap B$. Thus, $A \cap \overline{B}$ defines a (s, r)-mincut that contains all edges in E_A. Thus, if E_A does not lies in some (s, r)-mincut, y lies in nearest s to r mincut.

Now, consider the other direction of this proof. Suppose y lies in nearest s to r mincut. Suppose $y \in \mathbf{t}$, in this case trivially there is no (s, r)-mincut that contains all edges in E_A (note that $R = \overline{A}$ in this case). Thus, assume y is a non-terminal in strip $\mathcal{D}_{A,t}$. Suppose there exists a (s, r)-mincut defined by set of vertices B such that it cuts all edges in E_A and $s, t \in \overline{B}$. By construction R lies outside A and all edges in E_A are contained in the cut defined by $A \cup B$. Since $A \cup B$ is a transversal in $\mathcal{D}_{A,t}$, it follows that $(A \cup B) \cap R = \varnothing$; otherwise it would imply a coherent path in strip $\mathcal{D}_{A,t}$ that intersects $A \cup B$ twice – a contradiction. So $B \cap R = \varnothing$ too. That is, R lies entirely in the side of s 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 $\mathcal{D}_{A,t}$. Since R contributes all its outgoing edges in cut defined by B, it follows that $B \cup R$ also defines a (s,r)-mincut. Clearly, $y \notin \overline{B \cup R}$. Thus, $(\overline{B \cup R}, B \cup R)$ defines a (s,r)-mincut in which y is not on the side of s – a contradiction. Therefore, if y lies in nearest s to r mincut, no (s,r)-mincut contains all edges in E_A.

□

Now, the following lemma can be seen as a corollary of Lemma 4.4 and 5.2.

Corollary 5.2.1. *y lies in the nearest s to r mincut in G if and only if the following conditions hold:*

1. *$\overline{A}$ lies in nearest s to r mincut in G_A.*
2. *There is no (s,r)-mincut that contains all edges in side-**s** of y.*

6 A 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 $s, r \in S'$.

6.1 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 first 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 first 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 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 second 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 or 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 or tree-edge where the other endpoint lies.

The following lemma (Proof in Appendix C) states the property of the resulting graph.

Lemma 6.1. *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 $\mathcal{H}_S$.

6.2 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 6.2 (Gomory and Hu [22]). 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 6.1 and Lemma 6.2 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 $\textsc{Edge-Contained}(s,r,E_y)$ using $G_{S'}$ if all edges in E_y remain intact in graph $G_{S'}$. However, answering a query $\textsc{Edge-Contained}(s,r,E_y)$ 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 6.1 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 4.1. 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 6.3. 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 $\textsc{Edge-Contained}(s,r,E_y)$ to $\textsc{Edge-Contained}(s,r,E_{y'})$ using Theorem 6.3, 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 (Proof in Appendix D).

Lemma 6.4. Set of edges $E_{y'}$ in Theorem 6.3 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 6.5. 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 $\mathcal{F}_S$.

6.3 Nearest mincut queries in G

Let us see how to answer the following query – Given $y \in V$ check if y lies in nearest s to t mincut where $s, t \in S$. If s and t are mapped to different node in the skeleton $\mathcal{H}_S$, the procedure is fairly straightforward – construct the strip $\mathcal{D}_{s,t}$ and check if y lies in source vertex $\mathbf{s}$. So, we shall restrict our attention only to the case when s and t are mapped to same node ν of the skeleton $\mathcal{H}_S$. Algorithm 2 highlights an algorithm for same. The correctness of algorithm follows from Lemma 5.2.1.

Algorithm 2 Check if y lies in nearest s to t mincut in G

```
 1: procedure CHECK-NEAREST-MINCUT($G, s, t, y$)
 2:     if $s$ and $t$ are mapped to different nodes in $\mathcal{H}_S$ then
 3:         if $y \in s_t^N$ then return True else return False          // ref Appendix H
 4:     end if
 5:     $\nu \leftarrow$ node to which $s$ and $t$ are mapped in $\mathcal{H}_S$
 6:     if $y$ is mapped to $\nu$ then return CHECK-NEAREST-MINCUT($G_{S'}, s, t, y$)
 7:     $y' \leftarrow$ vertex to which $y$ is compressed          // Assume $y'$ corresponds to $\bar{A}$
 8:     if CHECK-NEAREST-MINCUT($G_{S'}, s, t, y'$) is False then
 9:         return False
10:     end if
11:     Pick a vertex $x \in S \cap \bar{A}$
12:     Construct the strip $\mathcal{D}_{A,x}$
13:     $E_y \leftarrow$ Edges in side-$\mathbf{s}$ of $y$ in the strip $\mathcal{D}_{A,x}$
14:     return $\neg$ EDGE-CONTAINED($s, t, E_y$)
15: end procedure
```

7 Compact Data Structure for Sensitivity Query

In the following section we present our $\mathcal{O}(m)$ space sensitivity data structure.

7.1 An $\mathcal{O}(m)$ size data structure

Consider any node μ in the hierarchy tree [27] $\mathcal{T}_G$. 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 2.11 we know that $S(\mu')$ is a maximal subset of $S(\mu)$ with connectivity strictly greater than that of $S(\mu)$. Using Theorem 6.3 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 6.1.

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.

Query algorithm for edge-containment query

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 6.3). This computation can be carried out in time linear in size of flesh at node μ using Lemma 6.4. We stop when $\mu = \nu$. Observe that $c_{s,t} = c_{S(\nu)}$ (using Observation 2.12) 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 2.4. 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 6.5. We stop at the root node and report the set of vertices defining the (s, t)-mincut in G.

A FT-MINCUT$(s, t, (x, y))$ can be accomplished using the query EDGE-CONTAINED$(s, t, \{(x, y)\})$ and the old value of (s, t)-mincut (using Fact 1.2).

Query algorithm for nearest mincut queries

A query CHECK-NEAREST-MINCUT(s, t, y) can be answered using Algorithm 2 on the graph G with $S = V$. A naive implementation of the algorithm will however be inefficient. This is because there can be $\mathcal{O}(c_{s,t})$ internal nodes from the root node to $LCA(s, t)$ in hierarchy tree $\mathcal{T}_G$. Thus, the algorithm may invoke $\mathcal{O}(n)$ instances of edge-containment query which will give an $\mathcal{O}(\min(mc_{s,t}, nc_{s,t}^2))$ time algorithm for accomplishing this query. This query time is worse than the best known deterministic static algorithm to compute a (s, t)-mincut.

We crucially exploit the following fact from Algorithm 2. If EDGE-CONTAINED(s, t, E_y) (Line 14) in G_ν evaluates to True at some internal node, then we can conclude that $y \notin s_t^N$ in G. Thus, the query can possibly be true only if EDGE-CONTAINED(s, t, E_y) in G_ν is False for all possible levels. Algorithm 3 gives an efficient implementation of the CHECK-NEAREST-MINCUT query. The following lemma gives proof of correctness for the algorithm.

Lemma 7.1. Algorithm 3 determines if y lies in nearest s to t mincut in $\mathcal{O}(\min(m, nc_{s,t}))$ time.

Proof. The fact that the run-time of algorithm is $\mathcal{O}(\min(m, nc_S))$ follows from analysis for edge-containment query (see Appendix E). Thus, we shall focus on the correctness of algorithm.

Consider the nodes μ and its parent $\lambda = parent(\mu)$ on Line 8. Observe that E_y lies in a (s, t)-mincut in G_λ if and only if $E_{y'}$ lies in a (s, t)-mincut in G_μ (from Lemma 4.4). Suppose E'_μ denotes one-side of the inherent partition of y' in G_μ such that $E_{y'} \cap E'_\mu \neq \varnothing$. Observe that if $E_{y'}$ lie in a (s, t)-mincut then $E_{y'} \subseteq E'_\mu$ (from Lemma 4.3 and Lemma 2.4). In other words, if $E_{y'}$ lie in a (s, t)-mincut then $E_{y'}$ must be in one-side of the inherent partition of y'. Note that both sides of inherent partition of y' can be determined from G_μ, independent of the query procedure.

Suppose $\ell = LCA(s, t)$ is the node at which Line 3-4 is being executed.

Suppose $y \notin s_t^N$ in G. In this case, one of the following must happen – (i) $y \notin s_t^N$ in G_ℓ or (ii) EDGE-CONTAINED(s, t, E'_μ) evaluates to true for some node μ in the path from root to ℓ. Clearly if (i) is true, Algorithm 3 returns false (Line 3). Thus, assume (i) is false and (ii) is true. Suppose μ is one such node at which EDGE-CONTAINED(s, t, E'_μ) evaluates to true. Consider the execution of procedure CHECK-NEAREST-MINCUT at level μ. At Line 12, $E_{y'} \subseteq E'_\mu$. Thus, if E'_μ lies in some (s, t)-mincut then $E_{y'}$ must also lie in some (s, t)-mincut. Hence, EDGE-CONTAINED$(s, t, E_{y'})$ must also be true. Observe that in each subsequent recursive call of this procedure, that is from $child(\mu)$ to ℓ, Line 9 will never be true (from an earlier observation in this proof). Thus, the set of edges E_y we receive in the final call (level ℓ) of this procedure is simply the set of edges obtained from repeated query-transformation on $E_{y'}$ (at level μ). In other words, if $E_{y'}$ (at level μ) lie in a (s, t)-mincut in G_μ then E_y (at level ℓ) lie in a (s, t)-mincut in G_ℓ. Thus, EDGE-CONTAINED(s, t, E_y) (at Line 3) evaluates to true. As a result, our algorithm will return false.

Suppose $y \in s_t^N$ in G. In this case, $y \in s_t^N$ in G_ℓ (from Lemma 2). Moreover, EDGE-CONTAINED(G_μ, s, t, E'_μ) evaluates to false for all possible levels μ from root to $parent(\ell)$. Thus, EDGE-CONTAINED query at level ℓ on Line 3 will evaluate to false. Thus, our algorithm will return true.

□

Algorithm 3 Check if y lies in nearest s to t mincut (efficient)

```
1:  procedure CHECK-NEAREST-MINCUT($\mu, s, t, y, E_y$)
2:      if $\mu = LCA(s, t)$ then
3:          if $y \notin s_t^N$ in $G_\mu$ or EDGE-CONTAINED($s, t, E_y$) then return False
4:          return True
5:      end if
6:      $\nu \leftarrow$ Node to which $s, t$ are mapped in skeleton $\mathcal{H}_{S(\mu)}$
7:      if $y$ is mapped to $\nu$ then return CHECK-NEAREST-MINCUT($\nu, s, t, y, E_y$)
8:      $E_{y'} \leftarrow$ Query-Transformation on $E_y$ from $G_{parent(\mu)} \rightarrow G_\mu$
9:      if $E_{y'}$ does not lie in one-side of inherent partition of $y'$ then
10:         $E_{y'} \leftarrow$ one side of inherent partition of $y'$
11:     end if
12:     return CHECK-NEAREST-MINCUT($\nu, s, t, y', E_{y'}$)
13: end procedure
```

An IN-MINCUT$(s, t, (x, y))$ query can be reported using four instances of CHECK-NEAREST-MINCUT queries (from Lemma F.2) and the old value of (s, t)-mincut.

We can thus state the following theorem.

Theorem 7.2. Given an undirected unweighted multigraph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, there exists a data structure of $\mathcal{O}(m)$ size that can report the value of (s, t)-mincut for any $s, t \in V$ upon failure (or insertion) of any edge. The time taken to answer this query is $\mathcal{O}(\min(m, nc_{s,t}))$.

The detailed analysis of the query time of edge-containment query and space taken by the data structure stated in Theorem 7.2 is given in Appendix E.

References

[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. URL: https://doi.org/10.1137/17M114306X, doi:10.1137/17M114306X.

[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. doi:10.1007/s00453-018-0452-3.

[3] Surender Baswana, Shiv Kumar Gupta, and Till Knollmann. Mincut sensitivity data structures for the insertion of an edge. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.12.

[4] Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

[5] 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. doi:10.1016/j.jalgor.2005.08.001.

[6] 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. doi:10.1145/1536414.1536431.

[7] Gilad Braunschvig, Shiri Chechik, David Peleg, and Adam Sealfon. Fault tolerant additive and (μ, α)-spanners. *Theor. Comput. Sci.*, 580:94–100, 2015. doi:10.1016/j.tcs.2015.02.036.

[8] Timothy M. Chan, Mihai Patrascu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011. doi:10.1137/090751670.

[9] 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. doi:10.1145/3357713.3384253.

[10] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010. doi:10.1137/090758039.

[11] 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. URL: https://arxiv.org/abs/2005.02368, arXiv:2005.02368.

[12] 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. doi:10.1137/S0097539705429847.

[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. doi:10.1145/195058.195442.

[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. URL: http://dl.acm.org/citation.cfm?id=313651.313711.

[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. doi:10.1137/S0097539797330045.

[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. doi:10.1007/PL00009195.

[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. doi:10.1137/1.9781611974782.31.

[18] A.V. Karzanov E.A. Dinitz and M.V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In *Studies in Discrete Optimizations*, 1976. URL: http://alexander-karzanov.net/ScannedOld/76_cactus_transl.pdf.

[19] Daniele Frigioni and Giuseppe F. Italiano. Dynamically switching vertices in planar graphs. *Algorithmica*, 28(1):76–103, 2000. doi:10.1007/s004530010032.

[20] 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. doi:10.1109/SFCS.1997.646090.

[21] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2017. doi:10.1007/978-3-319-62127-2_36.

[22] 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. URL: http://www.jstor.org/stable/2098881.

[23] 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. doi:10.1145/3174803.

[24] 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. doi:10.1145/1250790.1250879.

[25] 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. doi:10.1007/978-3-642-35261-4_13.

[26] 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. doi:10.1145/276698.276714.

[27] Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004. doi:10.1137/S0097539703433912.

[28] Yang P. Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow. *CoRR*, abs/2003.08929, 2020. URL: https://arxiv.org/abs/2003.08929, arXiv:2003.08929.

[29] Merav Parter and David Peleg. Sparse fault-tolerant BFS structures. *ACM Trans. Algorithms*, 13(1):11:1–11:24, 2016. doi:10.1145/2976741.

[30] Merav Parter and David Peleg. Fault-tolerant approximate BFS structures. *ACM Trans. Algorithms*, 14(1):10:1–10:15, 2018. doi:10.1145/3022730.

[31] Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011. doi:10.1137/09075336X.

[32] Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Math. Program.*, 22(1):121, 1982. doi:10.1007/BF01581031.

[33] Mikkel Thorup. Fully-dynamic min-cut. *Comb.*, 27(1):91–127, 2007. doi:10.1007/s00493-007-0045-2.

A Comparison of Query Time with Static Algorithms

Goldberg and Rao [20] gave a deterministic algorithm for computing (s,t)-mincut in an undirected unweighted graph that runs in $\mathcal{O}(n^{3/2}m^{1/2})$ time. Karger and Levine [26] gave an algorithm that runs in $\mathcal{O}(nm^{2/3}c_{s,t}^{1/6})$ time for the same problem.

We show that the query time of our data structure is $\Omega(\sqrt{n})$ times faster than both of them. The query time we achieve is $\mathcal{O}(\min(m, nc_{s,t}))$. It is clear that $\frac{n^{3/2}m^{1/2}}{m} \geq \sqrt{n}$ and thus, the query time is $\Omega(\sqrt{n})$ times faster than Goldberg and Rao's algorithm [20]. We can show the same for Karger and Levine's algorithm [26] by separately considering the cases $m < nc_{s,t}$ and $m \geq nc_{s,t}$. Recently, Liu and Sidford [28] gave a deterministic algorithm for computing (s,t)-mincut on unweighted graphs in $\mathcal{O}(m^{4/3+o(1)})$ time.

B Proof of Lemma 3.1

Proof. Consider $u \in X$ to be a non-terminal in strip $\mathcal{D}_{s,t}$. We shall show that $\mathcal{R}_s(u) \setminus \mathbf{s} \subseteq X$, i.e. reachability cone of u towards source $\mathbf{s}$ in the strip $\mathcal{D}_{s,t}$ avoiding $\mathbf{s}$ is a subset of X. Consider any non-terminal $v \in \mathcal{R}_s(u) \setminus \mathbf{s}$. Since u is reachable from v in direction $\mathbf{t}$, therefore $\tau(v) < \tau(u)$. Therefore, $v \in X$. Therefore, $\mathbf{s} \cup X$ defines a transversal in the strip $\mathcal{D}_{s,t}$ (from Lemma 2.3) and thus defines a (s,t)-mincut. The fact that (x,y) lies in this (s,t)-mincut follows from the fact that $\tau(y) > \tau(x)$ and thus, $y \notin X$. □

C Proof of Lemma 6.1

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 C.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 2.7). It again follows from Lemma 2.7, 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 3.

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

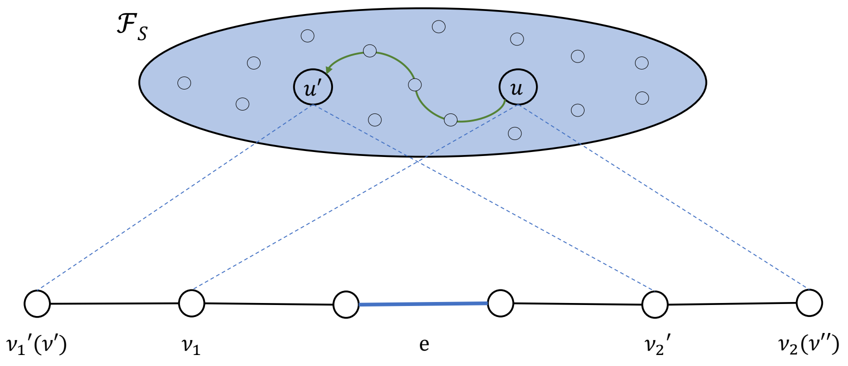

Figure 3: The structure of path $P(\nu', \nu'')$.

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 F.6). Moreover, it follows from Lemma C.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.

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. Figure 4 gives a nice illustration of the contraction procedure.

D Proof of Lemma 6.4 and 6.5

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 2.8). Using the construction outlined in Lemma 4.4 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 4.4). 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 $\mathcal{F}_S$. □

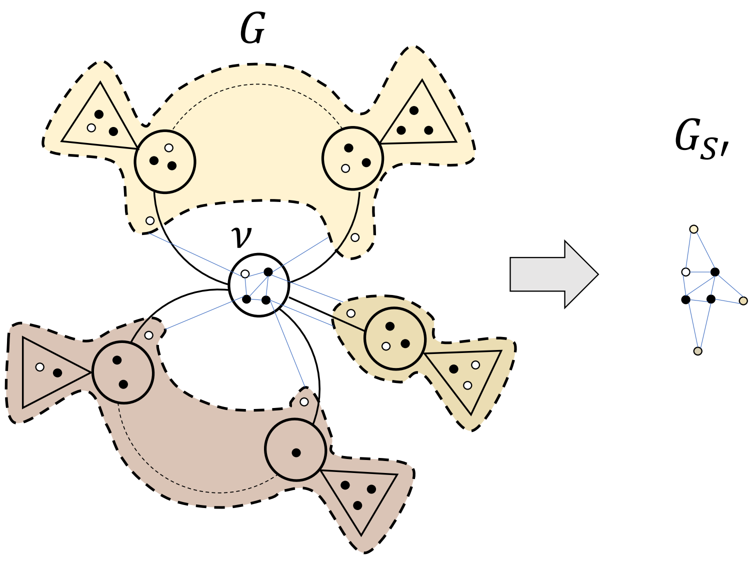

Figure 4: 2-step contraction procedure to construct $G_{S'}$. We only show the vertices and relevant edges of graph along with the skeleton $\mathcal{H}_S$. Solid vertices belong to Steiner set S and hollow vertices are non-Steiner vertices. All Steiner vertices inside node ν form the set S'.

E 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 [24, 15].

Lemma E.1 ([24, 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 4 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 4 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 E.2. 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 E.3. 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 E.1, 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 6.1 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 6.1 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 E.4. The total number of units across all flesh graphs is $\mathcal{O}(n)$.

Inference E.5. 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 E.2). 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 E.1, Lemma E.2, Lemma E.3 and Inference E.5 we get the following result.

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

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 E.4). 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 $\mathcal{O}(\min(m, nc_{s,t}))$.

Details of projection mapping

We have shown that combined size of flesh graphs stored at each internal node in hierarchical tree is $\mathcal{O}(m)$ and the total number of units across them is $\mathcal{O}(n)$.

In this subsection, we show how to store the exact projection mapping for each flesh unit in our data structure without affecting the $\mathcal{O}(m)$ size bound achieved in previous subsection. One natural starting point is to inherit mapping of flesh units from the parent. More specifically, if ν is a child of μ in the hierarchy tree, $\mathcal{F}_\nu$ inherits its mapping from $\mathcal{F}_\mu$. However, this simple approach may blow up the size of data-structure to $\mathcal{O}(n^2)$. The way we perform the mapping is as follow, whenever a new contracted vertex is introduced at internal node μ we map it directly to the stretched unit it appears in any subsequent levels. Note that it might be possible that such a contracted vertex never appears in any stretched units in the subsequent level in which case we map it to the leaf node of the hierarchy tree it finally ends up in.

Whenever we wish to check if a contracted vertex v is mapped to node ν in flesh $\mathcal{F}_\mu$, we go to the mapping of v, let it be $\pi(v)$. Now, $\pi(v)$ might be a stretched unit corresponding to some internal node or a leaf node itself. Assume the internal node corresponding to $\pi(v)$ is ν', we lookup the LCA(ν, ν'). If LCA(ν, ν')= ν, we can conclude that v is mapped to node ν in this level.

Let us look the query procedure more closely. The query procedure shall be of the form EDGE-CONTAINED(s, t, E_y) at any internal node μ where s and t are vertices of graph and E_y is a set of edges incident on a unit y in the flesh graph $\mathcal{F}_\mu$. The task at this stage is to understand how to transform this query to the form EDGE-CONTAINED$(s, t, E_{y'})$ for a child node ν of μ. One of these two cases are possible, (i) y is in the unit same as s, t which is mapped to node ν in the skeleton. In this case, $E_{y'} = E_y$ and $y = y'$, so we need not do anything. (ii) y is some other unit (not same as s, t) in the flesh $\mathcal{F}_\mu$. Observe that E_y must also be edges in $\mathcal{F}_\mu$ otherwise the query can be answered in negative. After doing the query transformation, we shall get a set of edges E_v where v is a contracted vertex. Thus, we need to identify the unit in $\mathcal{F}_\nu$ where v lies. How can one do the same? We know that contracted vertices are not mapped to units in flesh $\mathcal{F}_\nu$ but are directly mapped to the stretched unit (or the leaf node) it ever appears in. So, we can lookup the mapping $\pi(v)$ which will be a stretched unit in flesh $\mathcal{F}_{\mu'}$ or simply a leaf node μ'. If $\nu = \mu'$, $y' = \pi(v)$. Otherwise, we can't infer directly the steiner unit in which v lies. We perform this simple procedure – for each Steiner unit we check if v lies in that Steiner unit. We have already seen that the "check" operation only takes constant time as it can be accomplished using a LCA query on the hierarchy tree. Since, we already discount $|\mathcal{F}_\nu|$ time for the node ν in the query procedure, identifying y' does not poses a challenge to query time.

F Extended Preliminaries

Definition F.1 (Nearest mincut from s to t)**.** An (s, t)-mincut $(A, \bar{A})$ where $s \in A$ is called the nearest mincut from s to t if and only if for any (s, t)-mincut $(A', \bar{A'})$ where $s \in A'$, $A \subseteq A'$. The set of vertices A is denoted by s_t^N.

The following lemma gives necessary and sufficient condition for an edge (x, y) to increases the value of (s, t)-mincut.

Lemma F.2 ([32])**.** The insertion of an edge (x, y) can increase the value of (s, t)-mincut by unity if and only if $x \in s_t^N$ and $y \in t_s^N$ or vice versa.

The following lemma exploits the undirectedness of the graph.

Lemma F.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 F.4 (Submodularity of cuts)**.** For any two subsets $A, B \subset V$, $\quad c(A) + c(B) \geq c(A \cup B) + c(A \cap B)$.

Lemma F.5. Let $S \subset V$ define an (s,t)-mincut with $s \in S$. For any subset $S' \subset V \setminus S$ with $v \notin S'$,

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

Compact Representation for all (s,t)-mincuts

Suppose $\mathcal{D}_{s,t}$ is a strip containing all (s,t)-mincuts and x is a non-terminal in this strip. 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 F.6 ([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.

We also use the following Lemma.

Lemma F.7. 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.

Another simple observation helps us describe the nearest mincuts in the strip.

Lemma F.8. The mincuts defined by $\mathbf{s}$ and $\mathbf{t}$ are the nearest mincut from s to t and t to s respectively.

Compact Representation for all Steiner Mincuts

It is important to note that nearest s to t and t to s mincuts are easier to identify in the connectivity carcass. The following lemma conveys the fact.

Lemma F.9 ([13])**.** Let $s, t \in S$ such that $c_{s,t} = c_S$. Determining if a unit u lies in nearest s to t mincut (or vice-versa) can be done using skeleton $\mathcal{H}_S$ and projection mapping π_S using $\mathcal{O}(1)$ LCA queries on skeleton tree.

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 $\mathcal{O}(\min(m, \tilde{n}c_S))$.

G Reporting (s,t)-strip for steiner mincuts

Let s and t be any two arbitrary vertices. We shall now provide an algorithm to report (s,t)-strip using our data structure presented in Section 3.

We first compute the LCA of s and t in the hierarchy tree $\mathcal{T}_G$. Let this node be ν. Let $S(\nu)$ be the Steiner set associated with ν, and $\mathcal{F}_{S(\nu)}$ be the corresponding flesh graph . Recall that we augment ν with the skeleton $\mathcal{H}_{S(\nu)}$ and the projection mapping of all the units present in $\mathcal{F}_{S(\nu)}$.

Each (s,t)-mincut is a Steiner mincut for $S(\nu)$ that separates s and t as well, and the (s,t)-strip will precisely store these cuts. The location of these cuts follows from the structure of the skeleton $\mathcal{H}_{S(\nu)}$ – each of these cuts is defined by a tree edge or a (suitable) pair of edges from the cycle that appear on the path between the unit containing s and the unit containing t in $\mathcal{H}_{S(\nu)}$. Using this observation, the (s,t)-strip will be a quotient graph of the flesh graph $\mathcal{F}_{S(\nu)}$ that preserves only these cuts. We shall compute it using the skeleton tree for $\mathcal{H}_{S(\nu)}$ and the projection mapping of various units (Without loss of generality we assume that the skeleton tree is rooted at the unit containing s). We process the units of the flesh graph $\mathcal{F}_{S(\nu)}$ as follows.

1. Processing the Steiner units:
   Let μ be any Steiner unit in the flesh graph $\mathcal{F}_{S(\nu)}$. Let ω be the LCA of μ and the unit containing t in the skeleton tree. If ω is a tree node in the skeleton tree, we merge μ with the unit corresponding to ω in the flesh graph. If ω is a cycle node, we merge μ with the child of ω that is ancestor of μ. It requires a single LCA query and a level ancestor query [4] on the skeleton tree. Both of them can be accomplished in $\mathcal{O}(1)$ time to accomplish this task.

2. Processing the non-Steiner units:
   Let μ be any non-Steiner unit in the flesh graph $\mathcal{F}_{S(\nu)}$. Let $\pi(\mu)$ be the proper path in the skeleton to which μ is mapped. μ will appear as nonterminal in (s,t)-strip if and only if $\pi(\mu)$ intersects the (s,t)-path in the skeleton $\mathcal{H}_{S(\nu)}$. This requires $O(1)$ LCA queries only on the skeleton tree to determine it. If μ turns out to intersect the (s,t)-path, we retain it as it is in the quotient graph. Otherwise, we merge μ with the same node in the quotient graph to which both the two endpoints of $\pi(\mu)$ are merged.

Once we have the quotient graph corresponding to (s,t)-strip, we need to determine the direction of each edge in the quotient graph. The resulting graph will be a DAG such that each (s,t)-mincut appears as transversal in this DAG. We now describe the procedure of assigning the direction to any edge (a,b) in the quotient graph. Let (u,v) be the corresponding edge in the flesh graph $\mathcal{F}_{S(\nu)}$; if there are multiple such edges pick any one of them arbitrarily. Let u be compressed to a and v be compressed to b while forming the quotient graph.

Let P be the proper path to which the edge (u,v) is mapped in the skeleton. P is obtained by extending path $\pi(u)$ to $\pi(v)$ in a specific direction. Direct the path P in arbitrary direction. Without loss of generality, assume that $\pi(u)$ is prefix of P and $\pi(v)$ is suffix of the directed P. Note that P must be intersecting the path joining the unit containing s and the unit containing t in the skeleton. Let one such edge be (x,y) with x lying on the side of the unit containing s and y lying on the side of unit containing t. While traversing P, if edge (x,y) is traversed along $x \to y$, we assign the edge (a,b) direction $a \to b$; otherwise we assign the direction $b \to a$.

H Edge insertion on $\mathcal{O}(n^2)$ space data structure

We shall now present an algorithm to determine using our $\mathcal{O}(n^2)$ size data structure whether insertion of any edge increases the mincut between any pair of vertices. It follows from Lemma F.2 that in order to accomplish this objective in $\mathcal{O}(1)$ time, it suffices if we can determine whether $x \in s_t^N$ in $\mathcal{O}(1)$ time for any $s,x,t \in V$.

Let ν be the LCA of s and t in the hierarchy tree $\mathcal{T}$. Let $\mathbf{s}$, $\mathbf{t}$, and $\mathbf{x}$ be the units in the flesh graph $\mathcal{F}_{S(\nu)}$ for s,t, and x respectively. Recall that we do not store $\mathcal{F}_{S(\nu)}$ at ν. Rather, we just store only the units of the flesh, their projection mapping, and the skeleton associated with $S(\nu)$ at ν.

Observe that x belongs to s_t^N if and only if $\mathbf{x}$ gets mapped to the source node in the (s,t)-strip. Refer to the algorithm given in the previous section for the construction of the (s,t)-strip. As a result, the following conditions in terms of the skeleton tree at ν capture the necessary and sufficient condition for $x \in s_t^N$.

- If $\mathbf{x}$ is a Steiner unit:
  $x \in s_t^N$ if and only if LCA of $\mathbf{x}$ and $\mathbf{s}$ in the skeleton tree is $\mathbf{s}$.
- If $\mathbf{x}$ is a non-Steiner unit:
  Let $\mathbf{x}$ be mapped to path (ω_1, ω_2) in skeleton $\mathcal{H}_{S(\nu)}$. $x \in s_t^N$ if and only if LCA of ω_1 and $\mathbf{s}$ as well as LCA of ω_2 and $\mathbf{s}$ in the skeleton tree is $\mathbf{s}$.

So it just requires a couple of LCA queries on the skeleton tree to determine if $x \in s_t^N$. We can thus state the following theorem.

Theorem H.1. For any undirected and unweighted graph on n vertices, there exists an $\mathcal{O}(n^2)$ size data structure that can answer any sensitivity query for all-pairs mincuts in $\mathcal{O}(1)$ time.