Sensitivity Oracle for All-Pairs Mincuts

Abhyuday Pandey

BT/CSE/170039

Supervisor: Dr. Surender Baswana

May 10, 2021

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 insertion/deletion of any edge as query, can efficiently report the value of the mincut between s and t upon the update.

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. Recently, Baswana, Gupta, and Knollman [1] gave a data structure that can handle single edge insertion in $\mathcal{O}(n^2)$ space and $\mathcal{O}(1)$ query time. We present the following results.

1. We present a sensitivity oracle for all-pairs mincuts. Our data structure guarantees $\mathcal{O}(1)$ query time. The space occupied by this data structure is $\mathcal{O}(n^2)$ which matches the worst-case size of a graph on n vertices. A resulting (s, t)-mincut after edge insertion/deletion can be reported in $\mathcal{O}(n)$ time which is optimal. Our data structure also subsumes the results of Baswana, Gupta, and Knollman [1].

2. We give conditional lower bounds on data structures that can handle dual-deletions (or dual-faults) for a (s, t)-mincut. We also give a conditional lower bound on a data structure storing all static $(\{s, u\}, \{t, v\})$-mincut values with fixed $s, t \in V$. This implies a conditional lower bound for a generalized flow tree for 2×2 mincuts, i.e. a data structure that can report the value of static $(\{s, u\}, \{t, v\})$-mincut for given vertices $s, t, u, v \in V$.

Some parts of this work have been taken from a recent research of Baswana and Pandey [2], but presented here for sake of continuity.

Acknowledgments

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

Contents

Chapter 1

Introduction

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

While designing an algorithm for a graph problem, one usually assumes that the underlying graph is static. But, this assumption is unrealistic for most of the real-world graphs where vertices and/or edges do undergo change, though occasionally. Sensitivity Oracles can be used to efficiently make a query for a single update/delete operation. The data structure can be pre-processed offline.

1.1 Previous Results

There exists a classical $\mathcal{O}(n)$ size data structure that stores all-pairs mincuts [11] 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. A new (s,t)-mincut can itself be reported in $\mathcal{O}(n)$ time.

For handling edge insertions, Baswana, Gupta and Knollman [1] recently gave a data structure that can report the value of a (s,t)-mincut upon insertion of an edge. Moreover, a (s,t)-mincut incorporating the change can be reported in $\mathcal{O}(n)$ time.

Combining the above two results, a sensitivity oracle can be obtained that uses $\mathcal{O}(mn)$ space. However, the $\mathcal{O}(mn)$ space occupied by this data structure is far from the size of the graph. To the best of our knowledge, there exists no data structure for this problem which takes $o(mn)$ space and a non-trivial query time.

1.2 Our Contribution

We present a space efficient sensitivity oracle for the all-pairs mincuts problem in an undirected unweighted multigraph. The data structure occupies $\mathcal{O}(n^2)$ space while achieving the optimal $\mathcal{O}(1)$ query time to report the value of (s,t)-mincut for any $s,t \in V$ upon deletion/insertion of any given edge. A resulting (s,t)-mincut incorporating the change can be reported in $\mathcal{O}(n)$ time.

In order to design our data structure, we present an efficient solution for a related problem of independent interest, called edge-containment query on a mincut defined as follows.

EDGE-CONTAINED$(s,t,(x,y))$: Check if a given edge $(x,y) \in E$ belong to some (s,t)-mincut.

Using a data structure for this problem, we can answer a deletion query upon failure of edge (x,y) by performing the corresponding edge-containment query. The value of (s,t)-mincut will reduce by unity if the edge-containment query evaluates to true. We can keep a Gomory-Hu tree of $\mathcal{O}(n)$ size as an auxiliary data structure to lookup the old value of $c_{s,t}$. The following fact

allows us to do so.

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

We give conditional lower bounds for the following problems.

1. Data structures that can handle dual-deletions (or dual-faults) for a (s,t)-mincut. In particular, we show that a data structure that can handle deletion (or failure) of two edges for a (s,t)-mincut even for fixed pair of vertices $s,t \in V$ requires $\tilde{\Omega}(n^2)$ space for non-trivial query time.

2. Generalized flow tree for 2×2 mincuts, i.e. a data structure that can report the value of static $(\{s,u\},\{v,t\})$-mincut for given vertices $s,t,u,v \in V$. We show that even if we fix $s,t \in V$, the data structure must require $\tilde{\Omega}(n^2)$ space for non-trivial query time.

1.3 Related Work

A related problem is that of maintaining mincuts in a dynamic environment. Until recently, most of the work on this problem has been limited to global mincuts. Thorup [17] 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 [12] 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 [13] 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 [4] that approximates all-pairs mincuts up to a nearly logarithmic factor in $\tilde{\mathcal{O}}(n^{2/3})$ amortized time against an oblivious adversary, and $\tilde{\mathcal{O}}(m^{3/4})$ time against an adaptive adversary. To the best of our knowledge, there exists no non-trivial dynamic algorithm for all-pairs *exact* mincut. We feel that our insights in this paper may be helpful in this problem.

Chapter 2

Preliminaries

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

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

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

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

Definition 2.3 (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^N_t.

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

Lemma 2.4 ([16])**.** The insertion of an edge (x, y) can increase the value of (s,t)-mincut by unity if and only if $x \in s^N_t$ and $y \in t^N_s$ or vice versa.

The following lemma exploits the undirectedness of the graph.

Lemma 2.5. Let x, y, z be any three vertices in G. If $c_{x,y} > c$ and $c_{y,z} > c$, then $c_{x,z} > c$ as well.

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

Lemma 2.6 (Submodularity of cuts)**.** For any two subsets $A, B \subset V$, $c(A) + c(B) \geq c(A \cup B) + c(A \cap B)$.

Lemma 2.7. Let $S \subset V$ define an (s,t)-mincut with $s \in S$. For any subset $S' \subset V \setminus S$ with $v \notin S'$,

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

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

Dinitz and Vainshtein [8] showed that there exists a quotient graph of G that compactly stores all (s,t)-mincuts. This graph is called strip $\mathcal{D}_{s,t}$. The 2 node to which s and t are mapped in $\mathcal{D}_{s,t}$ are called the terminal nodes, denoted by **s** and **t** respectively. Every other node is called a non-terminal node. We now elaborate some interesting properties of the strip $\mathcal{D}_{s,t}$.

Consider any non-terminal node v, and let E_v be the set of edges incident on it in $\mathcal{D}_{s,t}$. There exists a unique partition, called *inherent partition*, of E_v into 2 subsets of equal sizes. These subsets are called the 2 sides of the inherent partition of E_v. Interestingly, if we traverse $\mathcal{D}_{s,t}$ such that upon visiting any non-terminal node using an edge from one side of its inherent partition, the edge that we traverse while leaving it belong to the other side of the inherent partition, then no node will be visited again. Such a path is called a *coherent* path in $\mathcal{D}_{s,t}$. Furthermore, if we begin traversal from a non-terminal node u along one side of its inherent partition and keep following a coherent path we are bound to reach the terminal **s** or terminal **t**. So the two sides of the inherent partitions can be called side-**s** and side-**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 2.8 ([8]). *$A \subset V$ defines a (s,t)-mincut if and only if A is a transversal in $\mathcal{D}_{s,t}$.*

We now state the following two lemmas that can be viewed as a corollary of Lemma 2.8.

Lemma 2.9. *A (s,t)-mincut contains a set of edges E_y incident on vertex y if and only if all edges in E_y must belong to the same side of the inherent partition of the node containing y in strip $\mathcal{D}_{s,t}$.*

Lemma 2.10. *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 2.11. *The mincuts defined by $\mathbf{s}$ and $\mathbf{t}$ are the nearest mincut from s to t and t to s respectively.*

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

Lemma 2.12 ([8]). *If $x_1, \ldots, x_k$ are any non-terminal nodes in strip $\mathcal{D}_{s,t}$, the union of the reachability cones of x_i's in the direction of $\mathbf{s}$ defines the nearest mincut between $\{s, x_1, \ldots, x_k\}$ and t.*

2.2 Compact representation for all global mincuts

Let c_V denote the value of the global mincut of the graph G. Dinitz, Karzanov, and Lomonosov [5] showed that there exists a graph $\mathcal{H}_V$ of size $O(n)$ that compactly stores all global mincuts of G. Henceforth, we shall use nodes and structural edges for vertices and edges of $\mathcal{H}_V$ respectively. There exists a projection mapping $\pi : V(G) \to V(\mathcal{H}_V)$ assigning a vertex of graph G to a node in graph $\mathcal{H}_V$. In this way, any cut $(A, \bar{A})$ in cactus $\mathcal{H}_V$ is associated to a cut $(\pi^{-1}(A), \pi^{-1}(\bar{A}))$ in the original graph G. The graph $\mathcal{H}_V$ has a nice tree-like structure with the following properties.

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

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

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

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

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

Suppose $s, t \in V$ are two vertices such that $c_{s,t}$ is same as the global mincut value. So, each transversal of strip $\mathcal{D}_{s,t}$ corresponds to a global mincut that separates s and t. Recall that cactus $\mathcal{H}_V$ stores all global mincuts. So we

just need to contract it suitably so that only those cuts remain that separate s and t. For this purpose, we compute the path of cycles and tree edges between the nodes corresponding to s and t respectively. We compress each of the subcactus rooted to this path to a single vertex. The resultant graph we obtain will be the strip $\mathcal{D}_{s,t}$. The inherent partition of all the non-terminal units can be determined using the endpoints of the edges in the path.

2.2.2 Tree representation for cactus

We shall now show that $\mathcal{H}_V$ can be represented as a tree structure. This tree structure was also used by Dinitz and Westbrook in [9]. This representation will simplify our analysis on the cactus.

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

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

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

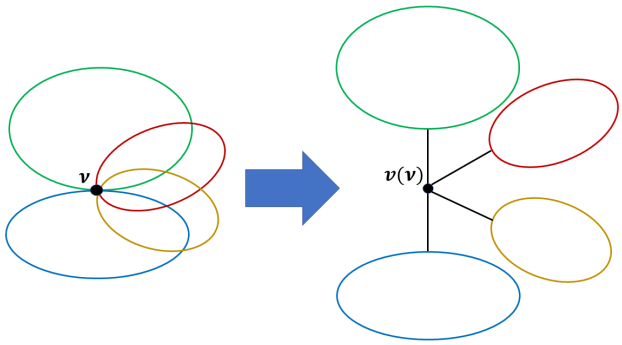

Figure 2.1: Transformation of cactus $\mathcal{H}_V$ to the tree $T(\mathcal{H}_V)$.

Lemma 2.13. Let ν, μ be any two arbitrary nodes in the cactus $\mathcal{H}_V$. The unique path between $v(\nu)$ and $v(\mu)$ in $T(\mathcal{H}_V)$ concisely captures all paths between ν and μ in $\mathcal{H}_S$.

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

2.3 Compact representation for all Steiner mincuts

Dinitz and Vainshtein [6] designed a data structure $\mathfrak{C}_S = (\mathcal{F}_S, \mathcal{H}_S, \pi_S)$ that stores all the Steiner mincuts for a Steiner set $S \subseteq V$ in the graph. We present a summary of this data structure.

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

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

Let $(S_B, \bar{S}_B)$ be the 2−partition of Steiner set induced by a bunch $\mathcal{B}$. If we compress all vertices in S_B to s and all vertices in $\bar{S}_B$ to t, the strip $\mathcal{D}_{s,t}$ will store all cuts in $\mathcal{B}$. We shall denote this strip by $\mathcal{D}_{\mathcal{B}}$. Any such strip has

the following property – if two non-terminals nodes of two strips intersect at even one vertex then these nodes coincide and the inherent partitions of these nodes in both strips coincide as well.

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

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

- If both ν_1 and ν_2 lie in $\mathcal{H}_S(\mathcal{B})$ (or $\bar{\mathcal{H}}_S(\mathcal{B})$) u is contracted in source (or sink).

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

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

Lemma 2.14 ([7]). Let u be a stretched unit and u' be any arbitrary unit in the flesh $\mathcal{F}_S$ and $\pi_S(u) = P(\nu_1, \nu_2)$, $\pi_S(u') = P(\nu_3, \nu_4)$. If u' is reachable from u in direction ν_2, then both these paths are extendable to a larger proper path with $P(\nu_1, \nu_2)$ as the initial part and $P(\nu_3, \nu_4)$ as the final part.

Lemma 2.15 ([6]). 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.

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 2.16 ([6]). 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 $\mathcal{O}(\min(m, \tilde{n}c_S))$.

2.4 Compact representation of all-pairs mincuts values

We describe first a hierarchical data structure of Katz, Katz, Korman and Peleg [14] that was used for compact labeling scheme for all-pairs mincuts. This hierarchical data structure is actually a rooted tree, denoted by $\mathcal{T}_G$ henceforth. The key insight on which this tree is built is an equivalence relation defined for a Steiner set $S \subseteq V$ as follows.

Definition 2.17 (Relation $\mathcal{R}_S$). Any two vertices $a, b \in S$ are said to be related by $\mathcal{R}_S$, that is $(a, b) \in \mathcal{R}_S$, if $c_{a,b} > c_S$, where c_S is the value of a Steiner mincut of S.

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

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

Observation 2.18. Suppose $\nu \in \mathcal{T}_G$ and μ is its parent. $S(\nu)$ comprises of a maximal subset of vertices in $S(\mu)$ with connectivity strictly greater than that of $S(\mu)$.

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

Observation 2.19. Suppose $s, t \in V$ are two vertices and μ is their LCA in $\mathcal{T}_G$ then $c_{s,t} = c_{S(\mu)}$.

Chapter 3

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

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

Consider the problem of identifying if a given edge (x, y) lies in a (s, t)-mincut for a designated pair of vertices $s, t \in V$. It is quite evident that an edge (x, y) lies in a (s, t)-mincut if x and y are mapped to different nodes in the strip $\mathcal{D}_{s,t}$. This query can be reported in $\mathcal{O}(1)$ time by storing the node mapping of each vertex in $\mathcal{O}(n)$ space.

Reporting a (s, t)-mincut that contains edge (x, y) requires more insights. Without loss in generality, assume that edge (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 $\mathcal{D}_{s,t}$. It is important to observe that set of vertices mapped to 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). However, reporting this mincut requires $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ space.

To achieve better space and query time, we report another (s, t)-mincut that contains (x, y) and has a simpler structure. Suppose τ is a topological ordering on the node set of strip $\mathcal{D}_{s,t}$ with $\tau(\mathbf{s}) = 0$. We show that storing the node mapping of each vertex and topological number of each node τ of the strip $\mathcal{D}_{s,t}$ can be used to report a (s, t)-mincut efficiently. This data structure takes only $\mathcal{O}(n)$ space and can report a (s, t)-mincut containing edge (x, y) in $\mathcal{O}(n)$ time. We state the following lemma.

Lemma 3.1. Consider the strip $\mathcal{D}_{s,t}$ with $\mathbf{x}$ as a non-terminal, and edge (x, y) lies 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 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. We can determine if an edge $(x, y) \in E$ belongs to some (s, t)-mincut using the strip $\mathcal{D}_{s,t}$ that can be built from the connectivity carcass. However, the construction of strip requires $\mathcal{O}(\min(m, nc_S))$ time. Interestingly, we show that only the skeleton and the projection mapping of the connectivity carcass are sufficient for answering this query in constant time. Moreover, the skeleton and the projection mapping occupy only $\mathcal{O}(n)$ space compared to the $\mathcal{O}(\min(m, nc_S))$ space occupied by the entire connectivity carcass.

Similar to the projection mapping of the stretched units, Dinitz and Vainshtein [8] introduced the notion of projection mapping for edges as follows. Suppose $(x, y) \in E$. If x and y belong to the same unit, then $P(x, y) = \varnothing$. If x and y belong to distinct terminal units mapped to nodes, say ν_1 and ν_2, in the skeleton $\mathcal{H}_S$, then $P(x, y) = P(\nu_1, \nu_2)$. If at least one of them belongs to a stretched unit, $P(x, y)$ is the extended path defined in Lemma 2.14. This allows us to state the following lemma which follows from the construction of a strip corresponding to a subbunch given in Section 2.3.

Lemma 3.2 ([6]). Edge $(x, y) \in E$ appears in the strip corresponding to a subbunch if and only if one of the structural edge in the cut of $\mathcal{H}_S$ corresponding to this subbunch lies in $P(x, y)$.

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

Lemma 3.3. Edge $(x, y) \in E$ belongs to a (s, t)-mincut if and only if the proper path $P(x, y)$ intersects a path between the nodes containing s and t in in $\mathcal{H}_S$.

Proof. Let ν_1 and ν_2 be the nodes in $\mathcal{H}_S$ containing s and t respectively. A cut in $\mathcal{H}_S$ corresponding to any tree-edge (or pair of cycle edges in same cycle) in the path from ν_1 to ν_2 defines a subbunch separating s from t. Moreover, it follows from the structure of the skeleton that no other cut in the skeleton corresponds to a subbunch separating s from t. Suppose (x, y) lies in some (s, t)-mincut. Thus, it must be in some subbunch separating s from t. From the above discussion, we know that this subbunch must correspond to a cut in the path from ν_1 to ν_2 in skeleton $\mathcal{H}_S$. Moreover, it follows from Lemma 3.2 that $P(x, y)$ contains one of the structural edge in this cut. This implies that $P(x, y)$ intersects the path from ν_1 to ν_2 in skeleton $\mathcal{H}_S$.

Now, consider the other direction of this proof. Suppose $P(x, y)$ and the path from ν_1 to ν_2 intersect at some cycle (or tree edge) c. Let e_1 and e_2 be structural edges belonging to the cycle c that are part of $P(x, y)$ and path from ν_1 to ν_2 respectively (in the case of tree edge $e_1 = e_2 = c$). Consider the cut in the skeleton corresponding to structural edges e_1 and e_2. It follows from Lemma 3.2 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 $\mathcal{T}(\mathcal{H}_S)$ (using Lemma **??**). Thus, we can store the skeleton tree $\mathcal{T}(\mathcal{H}_S)$ and projection mapping in $\mathcal{O}(n)$ space and determine if an edge (x, y) lies in an (s, t)-mincut for $c_{s,t} = c_S$ and $s, t \in S$ in $\mathcal{O}(1)$ time.

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 3.2, 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.

It is important to observe that the difficulty we face here is similar to the one highlighted in Section 3.1. To achieve better space and query time, we use similar ideas as used in Section 3.1. We aim 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 $X = \{u \mid \tau_{\mathcal{B}}(u) \leq \tau_{\mathcal{B}}(\mathbf{x})\}$ 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 X defines a (s,t)-mincut that contains edge (x,y). Clearly, storing topological order for each stretched unit for each possible bunch in which it appear as a non-terminal is not an option. This is because doing so will require a lot of space. Thus, we try to devise an algorithm to efficiently identify set X on without compromising $\mathcal{O}(n)$ size of the data structure.

Consider the data structure formed by the following components – (i) the skeleton tree $\mathcal{T}(\mathcal{H}_S)$, (ii) the projection mapping π_S of all units and (iii) a mapping τ from each stretched unit to a number. 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. We now show how this additional augmentation will help us report a (s,t)-mincut containing edge (x,y) efficiently.

In order to make the ideas more simple, we describe a transitive relation between proper paths on skeleton called *extendable in a direction*.

Definition 3.4 (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.

Moreover, verifying if $P(\nu_3,\nu_4)$ is extendable from $P(\nu_1,\nu_2)$ in direction ν_2 can be done in $\mathcal{O}(1)$ LCA queries on the skeleton tree.

Suppose stretched unit $\mathbf{x}$ is mapped to path $P(\nu,\nu')$. Consider the set X formed by 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 $\tau(v) \leq \tau(\mathbf{x})$, and (ii) the stretched unit is not mapped to $P(\nu,\nu')$ but $\pi_S(v)$ is extendable from $P(\nu,\nu')$ in direction ν. We state the following lemma.

Lemma 3.5. The vertices mapped to units in $\mathbf{s} \cup X$ define a (s,t)-mincut and contains all the edges in side-$\mathbf{t}$ of the inherent partition of $\mathbf{x}$.

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. It follows from the construction that u is either projected to $P(\nu, \nu')$ or $\pi_S(u)$ is extendable in direction ν from $P(\nu, \nu')$. Consider any unit v in $\mathcal{R}_{\mathbf{s}}(u) \setminus \mathbf{s}$. Suppose u and v are both projected to $P(\nu, \nu')$. 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 $P(\nu, \nu')$ in direction ν (from Theorem 2.14). It follows from the transitivity of Definition 3.4 that $\pi_S(v)$ is extendable from $\pi_S(u)$ in direction ν. Thus, $v \in X$. Therefore, $\mathbf{s} \cup X$ defines a (s, t)-mincut (from Lemma 2.8).

Now, we prove the second part of the lemma. Consider any edge (x, z) in the side-$\mathbf{t}$. If z is in $\mathbf{t}$ then $z \notin X$ from the construction. Assume that z is a non-terminal unit in $\mathcal{D}_{\mathcal{B}}$. If z is projected to path $P(\nu, \nu')$ then $\tau(z) > \tau(u)$. Thus, $z \notin X$. Otherwise $\pi_S(z)$ is extendable from $P(\nu, \nu')$ in direction ν'. It follows from Definition 3.4 that $z \notin X$. Thus, the cut defined by $\mathbf{s} \cup X$ contains all edges in side-$\mathbf{t}$ of the inherent partition of $\mathbf{x}$.

□

This data structure occupies $\mathcal{O}(n)$ space and can report a (s, t)-mincut containing edge (x, y) in $\mathcal{O}(n)$ time. We shall use this data structure to build a sensitivity oracle for all-pairs mincuts.

3.3 Edge-containment query for all-pairs Mincuts

The hierarchical tree structure of Katz, Katz, Korman and Peleg [14] can be suitably augmented to design a sensitivity oracle for all-pairs mincuts. We augment each internal node ν of the hierarchy tree $\mathcal{T}_G$ by the data structure discussed in Section 3.2 for the Steiner set $S(\nu)$. 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

```
1: procedure EDGE-COTAINED(s, t, x, y)
2:     μ ← LCA(T_G, s, t)
3:     P_1 ← P(π_{S(μ)}(s), π_{S(μ)}(t))
4:     P_2 ← P(x, y)
5:     if P_1 ∩ P_2 = ∅ then                    //Check if paths intersect
6:         return False
7:     else
8:         return True
9:     end if
10: end procedure
```

3.4 Edge-insertion query on Data Structure

Determining whether insertion of an edge (x, y) increases the (s, t)-mincut is comparatively simpler. Again, let μ be the LCA of s and t in $\mathcal{T}_G$. For sake of simplicity, assume that x, y, s and t be units in flesh graphs corresponding to vertices x, y, s and t respectively. Using Lemma 2.4, we only need to determine if $x \in s_t^N$ and $y \in t_s^N$ or vice-versa. Using Lemma 2.16, we can perform this operation in $\mathcal{O}(1)$ time. Reporting (s, t)-mincut after insertion of an edge can be done in $\mathcal{O}(n)$ time (using Lemma 2.16) as s_t^N is one such mincut. Thus, we can summarize the results of previous two sections in the following theorem.

Theorem 3.6. 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 insertion/deletion of an edge. A (s, t)-mincut incorporating the insertion/deletion can be reported in $\mathcal{O}(n)$ time.

Chapter 4

Conditional Lower Bounds on Mincut Data Structures

4.1 Strip representation of reachability in directed graphs

The problem of reachability in directed graph is as follows – Given a directed graph $\vec{G}$, preprocess it to form a data structure which can efficiently report if a given vertex v is reachable from another vertex u. The problem becomes interesting if we allow the underlying graph to be sparse. Is there any data structure that takes $o(n^2)$ space and still offers efficient query time? Patrascu [15] stated it as a difficult open problem and also gave a partial answer to it. In particular, if constant query time is required, the space needs to be $n^{1+\Omega(1)}$. Goldstein et al [10] stated a conjecture that concisely conveys the belief.

Conjecture 4.1 (Directed Reachability Hypothesis [10]). Any data structure for the problem of reachability in directed graphs must either use $\tilde{\Omega}(n^2)$ space, or linear query time.

The reachability in $\vec{G}$ is same as reachability in G_{SCC}, a directed acyclic graph which can be obtained by contracting each of the Strongly Connected Components (SCCs) to a single vertex. Henceforth, we shall assume that $\vec{G}$ is a directed acyclic graph.

We can transform a directed acyclic graph $\vec{G}$ into a (s,t)-strip $\mathcal{D}_{s,t}$ as follows. Create two additional vertices, namely s and t. Suppose Δ_v denotes

the difference in indegree and outdegree of any vertex v in $V(\vec{G})$. For each vertex $v \in V(\vec{G})$, if $\Delta_v > 0$ we add Δ_v edges from v to t. Likewise, if $\Delta_v < 0$ we add $|\Delta_v|$ edges from s to v. Lastly, add two additional edge(s) from s to v and v to t for all $v \in V(\vec{G})$. Note that the resultant graph is a strip by making all the edges undirected and taking the inedges and outedges as the inherent partition of each non-terminal unit. Moreover, the number of edges in this graph is only $\mathcal{O}(m)$. A non-terminal unit v is reachable from another non-terminal unit u if and only if there exists a coherent path from u to v in $\mathcal{D}_{s,t}$. Thus, we state the following lemma.

Lemma 4.2. For a directed graph $\vec{G}$ with n vertices and m edges, there exists a strip $\mathcal{D}_{s,t}$ with $\mathcal{O}(m)$ edges and $n+2$ vertices such that $\textsc{Reachable}(\vec{G}, u, v)$ is true if and only if there exists a coherent path from u to v in the strip $\mathcal{D}_{s,t}$.

4.2 Conditional Lower Bound for Generalized Flow Tree for 2×2 mincuts

We shall show how to transform a $\textsc{Reachable}$ query to a 2×2 static mincut query on the strip $\mathcal{D}_{s,t}$. Although we show a reduction of query to strip $\mathcal{D}_{s,t}$, the reduction applies to the underlying undirected unweighted graph G composed of nodes and edges from $\mathcal{D}_{s,t}$. Without loss in generality, assume v exceeds u in some topological ordering. We state the following lemma.

Lemma 4.3. Suppose u and v are two vertices of a directed graph $\vec{G}$ and $\mathcal{D}_{s,t}$ be the corresponding strip representation. Moreover, v exceeds u in some topological ordering of $\mathcal{D}_{s,t}$. $\textsc{Reachable}(\vec{G}, u, v)$ evaluates to true if and only if $\textsc{Mincut}(\{s, v\}, \{u, t\}) > c_{s,t}$.

Proof. Suppose $\textsc{Reachable}(\vec{G}, u, v)$ is true. Using Lemma 4.2, there exists a coherent path from u to v. Thus, $u \in \mathcal{R}_s(v)$. It follows from 2.12, that u lies in nearest $\{s, v\}$ to t mincut. Therefore, there is no $\{s, v\}$ to t mincut that keeps u on the side of t. Therefore, $\textsc{Mincut}(\{s, v\}, \{u, t\}) > \textsc{Mincut}(\{s, v\}, t) = c_{s,t}$. If $\textsc{Reachable}(\vec{G}, u, v)$ is false, $u \notin \mathcal{R}_s(v)$. Thus, $\mathcal{R}_s(v)$ is a (s, t)-mincut as well as a $(\{s, v\}, \{u, t\})$-mincut. Thus, $\textsc{Mincut}(\{s, v\}, \{u, t\}) = c_{s,t}$. □

Using Conjecture 4.1 and Lemma 4.3 we state the following conditional lower bound.

Theorem 4.4. Assuming Directed Reachability Hypothesis holds, any data structure that can report the value of $(\{s, v\}, \{u, t\})$-mincut for given $u, v \in V$ and designated $s, t \in V$ in an undirected unweighted multigraph G must either use $\tilde{\Omega}(n^2)$ space, or linear query time.

4.3 Conditional Lower Bound for dual-fault-tolerant (s, t)-mincut

We show that REACHABLE$(\vec{G}, u, v)$ can be determined using the fault-tolerant query FT-MINCUT$(s, t, \{(s, u), (v, t)\})$ on the strip $\mathcal{D}_{s,t}$. v is reachable from u if and only if the fault-tolerant query returns value equal to the original mincut between s and t less unity. In other words, v is reachable from u, if and only if FT-MINCUT$(s, t, \{(s, u), (v, t)\}) = c_{s,t} - 1$. We state the following lemma.

Lemma 4.5. Suppose u and v are two vertices of a directed graph $\vec{G}$ and $\mathcal{D}_{s,t}$ be the corresponding strip representation. Moreover, v exceeds u in some topological ordering of $\mathcal{D}_{s,t}$. REACHABLE$(\vec{G}, u, v)$ evaluates to true if and only if FT-MINCUT$(s, t, \{(s, u), (v, t)\}) = c_{s,t} - 1$.

Proof. It is easy to observe that FT-MINCUT$(s, t, \{(v, t)\}) = c_{s,t} - 1$. Thus, FT-MINCUT$(s, t, \{(s, u), (v, t)\}) \leq c_{s,t} - 1$. We give a simple proof by failing one edge at a time.

Suppose edge (v, t) is removed. As a result, new (s, t)-mincuts are those which contain edge (v, t), i.e. keep v on side of s. These mincuts correspond to all $(\{s, v\}, t)$-mincuts in strip $\mathcal{D}_{s,t}$. Now, removal of edge (s, u) decreases the value further if and only if there exists an $(\{s, v\}, t)$-mincut in $\mathcal{D}_{s,t}$ that keeps u on side of t. Thus, FT-MINCUT$(s, t, \{(s, v), (u, t)\}) = c_{s,t} - 1$ if and only if MINCUT$(\{s, v\}, \{u, t\}) > c_{s,t}$. Using Lemma 4.3, FT-MINCUT$(s, t, \{(s, v), (u, t)\}) = c_{s,t} - 1$ if and only if v is reachable from u in $\vec{G}$. □

Using Conjecture 4.1 and Lemma 4.5 we state the following conditional lower bound.

Theorem 4.6. Assuming Directed Reachability Hypothesis holds, any data structure that can report the value of (s, t)-mincut for designated $s, t \in V$

upon failure of 2 edges in an undirected unweighted multigraph G must either use $\tilde{\Omega}(n^2)$ space, or linear query time.

Bibliography

[1] 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.

[2] Surender Baswana and Abhyuday Pandey. Fault-tolerant all-pairs mincuts. *CoRR*, abs/2011.03291, 2020.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

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