Mincut Sensitivity Data Structures for the Insertion of an Edge

Surender Baswana 🄳
Department of Computer Science & Engineering, IIT Kanpur, India
sbaswana@cse.iitk.ac.in

Shiv Gupta
Department of Computer Science & Engineering, IIT Kanpur, India
shivguptamails@gmail.com

Till Knollmann 🄳
Heinz Nixdorf Institute, Paderborn University, Germany
tillk@mail.upb.de

— Abstract —

Let $G = (V, E)$ be an undirected graph on n vertices with non-negative capacities on its edges. The *mincut sensitivity problem* for the insertion of an edge is defined as follows.

Build a compact data structure for G and a given set $S \subseteq V$ of vertices that, on receiving any edge $(x, y) \in S \times S$ of positive capacity as query input, can efficiently report the set of all pairs from $S \times S$ whose mincut value increases upon insertion of the edge (x, y) to G.

The only result that exists for this problem is for a single pair of vertices (Picard and Queyranne, Mathematical Programming Study, 13 (1980), 8-16). We present the following results for the single source and the all-pairs versions of this problem.

1. *Single source*: Given any designated source vertex s, there exists a data structure of size $\mathcal{O}(|S|)$[1] that can output all those vertices from S whose mincut value to s increases upon insertion of any given edge. The time taken by the data structure to answer any query is $\mathcal{O}(|S|)$.

2. *All-pairs*: There exists an $\mathcal{O}(|S|^2)$ size data structure that can output all those pairs of vertices from $S \times S$ whose mincut value gets increased upon insertion of any given edge. The time taken by the data structure to answer any query is $\mathcal{O}(k)$, where k is the number of pairs of vertices whose mincut increases.

For both these versions, we also address the problem of reporting the values of the mincuts upon insertion of any given edge. To derive our results, we use interesting insights into the *nearest* and the *farthest* mincuts for a pair of vertices. In addition, a crucial result, that we establish and use in our data structures, is that there exists a directed acyclic graph of $\mathcal{O}(n)$ size that compactly stores the farthest mincuts from all vertices of V to a designated vertex s in the graph. We believe that this result is of independent interest, especially, because it also complements a previously existing result by Hariharan et al. (STOC 2007) that the nearest mincuts from all vertices of V to s is a laminar family, and hence, can be stored compactly in a tree of $\mathcal{O}(n)$ size.

---

[1] Data structure sizes are in *words* unless specified otherwise, where a word occupies $\Theta(\log n)$ bits.

1 Introduction

Let $G = (V, E)$ be a graph on $n = |V|$ vertices and $m = |E|$ edges with a non-negative capacity on each edge. A *mincut* for a pair of vertices u and v is a set of edges with least capacity whose removal disconnects v from u. It is a fundamental concept in graph theory. Moreover, the area of designing algorithms for the mincut and its variants has been extensively researched ever since the seminal result on the maxflow-mincut duality by Ford and Fulkerson [5].

It is often the case that one is more interested in the mincuts between vertices belonging to a relatively small part of the input graph than the mincuts between all vertices. Hence, consider a subset of vertices $S \subseteq V$ and any subset of pairs of vertices $Q \subseteq S \times S$ whose mincut we are interested in. The objective is to have the knowledge about how sensitive the mincuts of pairs of vertices from Q are, with respect to any change in the subgraph induced by S. This change could be a change in the capacity of an existing edge in the subgraph induced by S or insertion of a new edge between any two vertices in S. This knowledge of the impact on various mincuts due to any change in the network can make the network administrators well prepared for such changes when they indeed occur in future.

An important measure of the impact of a change in the capacity of an edge is the number of pairs of vertices whose mincut value changes. The change in the capacity of an edge could be either an increase or a decrease. We focus on the case when the capacity of an edge is allowed to increase only. For this data structure problem, the query input is a new edge with positive capacity or an existing edge whose capacity is increased. It can easily be observed that as far as the mincut between any pair of vertices is concerned, increasing the capacity of an existing edge (x, y) by amount Δ is equivalent to adding one more edge between x and y with capacity Δ. So henceforth, we only consider the insertion of an edge. On receiving any such edge, the objective is to efficiently report the pairs of vertices whose mincut increases.

Based on the discussion above, we now formally define the problem of mincut sensitivity.

▶ **Problem 1.** *Preprocess $G = (V, E)$, a set $S \subseteq V$, and a set $Q \subseteq S \times S$ to build a compact data structure that, on receiving any edge $(x, y) \in S \times S$ of positive capacity as query input, can efficiently report all those pairs from Q whose mincut value increases upon insertion of the edge (x, y) to G.*

We expect the bounds for the data structure of Problem 1 to depend on only the size of S instead of V. For simultaneously achieving efficient query time and compact space, the only previous solution that exists for this problem is for a single pair of vertices only, i.e., when $|Q| = 1$. It consists of a data structure that occupies $\mathcal{O}(|S|)$ space and achieves $\mathcal{O}(1)$ query time. This solution follows from an observation made by Picard and Queyranne in their seminal paper [11].

1.1 Our Contribution

We address the single source and the all-pairs versions of Problem 1, along with the problem of reporting the new value of the mincut between any pair of vertices after the edge insertion.

Single source all destinations: In the single source case, we are interested in the mincuts between a designated vertex $s \in S$ and all other vertices from set S, i.e., the (s,t)-mincut, for all $t \in S \setminus \{s\}$.

▶ **Theorem 2.** *For an undirected graph and any subset S of vertices with a designated vertex $s \in S$, there exists an $\mathcal{O}(|S|)$ size data structure that can report all those vertices from S whose mincut value to s increases upon insertion of any given edge from $S \times S$. The time taken by this data structure to answer any such query is $\mathcal{O}(|S|)$.*

The $\mathcal{O}(|S|)$ space and $\mathcal{O}(|S|)$ query time of our data structure for undirected graphs can be much less than $\mathcal{O}(|V|)$ for smaller S. Also, it is in sharp contrast with the following lower bound result for directed graphs that we also prove.

▶ **Theorem 3.** *Any data structure for a directed graph to answer a mincut sensitivity query from a designated source vertex to any designated subset of q vertices must use $\Omega(q^2)$ bits of space for at least one directed graph.*

The proof of Theorem 3 basically establishes that any such data structure can be used to store any balanced bipartite graph on $2|S|$ vertices implicitly. Interestingly, the same proof also establishes an $\Omega(|S|^2)$ lower bound for the single source version of two other fundamental problems for directed graphs, namely, *reachability sensitivity* as well as *distance sensitivity* for the insertion of an edge. These facts add more significance to the result stated in Theorem 2 for the single source mincut sensitivity.

All-pairs: When considering the mincuts between all pairs of vertices in S, i.e., the (u,v)-mincut, for all $u,v \in S$ such that $u \neq v$, we obtain the following result.

▶ **Theorem 4.** *For an undirected graph and any subset S of vertices, there exists an $\mathcal{O}(|S|^2)$ size data structure that can report all those pairs of vertices from $S \times S$ whose mincut value increases upon insertion of any given edge from $S \times S$. The time taken by the data structure to answer any such query is $\mathcal{O}(k)$, where k is the number of pairs whose mincut increases.*

Note that the query time of the data structure in Theorem 4 is optimal. Moreover, if the objective is to report just the number of all-pairs from the set $S \times S$ whose mincut increases upon insertion of any given edge, our data structure can accomplish this objective in $\mathcal{O}(\min(k, |S| \log |S|))$ time, which is $\mathcal{O}(|S| \log |S|)$ always.

▶ Remark 5. Our results for mincut sensitivity directly extend to maxflow sensitivity as well due to the equivalence between maxflow and mincut [5].

To achieve all our results, we use interesting insights into the *nearest* and the *farthest* mincuts for a pair of vertices – two concepts that exist since the seminal work of Ford and Fulkerson on maximum flow [5]. Additionally, a crucial result about the farthest mincuts that we establish and use in one of our data structures is the following.

▶ **Theorem 6.** *For an undirected graph on n vertices and a designated source vertex s, there exists a directed acyclic graph (DAG) of size $\mathcal{O}(n)$ that compactly stores the farthest mincuts from all vertices $v \in V \setminus \{s\}$ to s. For any v, the set of vertices defining the farthest mincut from v to s can be reported in time that is of the order of the size of the set.*

The graph theoretic result of Theorem 6 is of independent interest in addition to its applications in the mincut sensitivity problem. This is because, not only it adds to our understanding of mincuts, but it also complements an earlier result of Hariharan et al. [9] that showed that the nearest mincuts from all vertices to s form a laminar family, and hence, can be stored in a tree data structure occupying only $\mathcal{O}(n)$ space.

1.1.1   On reporting the value of mincut

In addition to reporting the pairs of vertices whose mincut increases upon insertion of a given edge, it may be important to output the new values of their mincuts. Indeed, if the edge capacities in the graph are integral and the inserted edge has unit capacity, our data structures from Theorems 2 and 4 can also report the new values of the affected mincuts upon insertion of an edge, i.e., the value of the mincut will be increased by one for the reported pairs of vertices. However, if there is no restriction on the capacity of the inserted edge, we show that even for the single source case it is not possible to accomplish this objective with any data structure of subquadratic size.

▶ **Theorem 7.** *There exists a set $\mathcal{G}$ of undirected graphs on n vertices with integer edge capacities in the range $[1, n^{2+\varepsilon}]$ (for any $\varepsilon > 0$) for which the following claim holds true. Any data structure for an undirected graph that can report the value of the mincut between a designated source vertex and any other vertex upon insertion of any edge of integer capacity polynomial in n must require $\Omega(n^2 \varepsilon \log n)$ bits of space for at least one graph from $\mathcal{G}$.*

For the all-pairs case, it turns out that any such data structure also provides a generalization of the flow-tree [6, 8]. That is, the data structure will also be able to report the mincut value between a vertex u and a pair $\{x, y\}$ of vertices for any $u, x, y \in V$. Chitnis, Kamma, and Krauthgamer [4] showed that there will be total $\mathcal{O}(n^2)$ distinct mincut values separating any vertex from any pair of vertices in an undirected graph. However, to the best of our knowledge, designing an $\mathcal{O}(n^2)$ size data structure that returns the value of any such mincut in non-trivial query time is still an open problem.

1.2   Overview of our results

We begin with the result of Picard and Queyranne [11] for mincut sensitivity for a source-destination pair (s, t). For any maximum flow f from s to t, let G_f be the corresponding residual graph. Notice that there is no path from s to t in G_f. Let R be the set of vertices which are reachable from s in G_f, and let T be the set of vertices from which t is reachable in G_f. Picard and Queyranne [11] made the following crucial observation.

▶ **Lemma 8** (Picard and Queyranne [11]). *The maxflow from s to t increases upon insertion of an edge (x, y) if and only if x belongs to R and y belongs to T.*

Without loss of generality, we can assume that the vertices of set S are labeled from 1 to $|S|$. Based on Lemma 8, the data structure for mincut sensitivity for a (s, t)-pair where $s, t \in S$, stores each of $R \cap S$ and $T \cap S$ in Boolean arrays indexed by the vertices of set S.

The subsets R and $V \backslash T$ in Lemma 8 turn out to be the smallest, and the largest subsets of vertices defining a mincut from s to t, respectively. In the literature on mincuts, R and $V \backslash T$ are respectively called the *nearest* and the *farthest* mincut from s to t; we define these notions more formally in the next section. Therefore, in order to design a compact and efficient data structure for the single source and the all-pairs versions of the mincut sensitivity problem, it is natural to explore if we can have a compact way to store these two types of cuts

for multiple pairs of vertices. We now provide an overview of the compact data structures for the nearest and the farthest mincuts, and the way these data structures are used to solve the mincut sensitivity problem.

Compact data structures for the nearest and the farthest mincuts

Interestingly, Hariharan et al. [9] showed that the nearest mincuts from all vertices to a designated vertex s in an undirected graph form a laminar family – If the subsets of vertices defining the nearest mincuts from u to s, and v to s intersect, then one of them must be a subset of the other. As a result, the nearest mincuts from all vertices to s can be stored compactly in a tree data structure occupying $\mathcal{O}(n)$ space only. However, the farthest mincuts do not constitute a laminar family. Let F_u and F_v be the subsets of vertices that define the farthest mincuts to s from u and v respectively. It is quite possible that F_u and F_v intersect each other but none of them is a subset of the other. In other words, two intersecting farthest mincuts to s may *cross* each other. Moreover, there may be $\Theta(n)$ vertices whose farthest mincuts to s cross the farthest mincut of a single vertex to s. This poses a challenge for designing a compact data structure for storing all the farthest mincuts to s. However, we overcome this challenge using crucial insights into the farthest mincuts.

Using the submodularity of cuts, we first establish the existence of a DAG on $\mathcal{O}(n)$ nodes that stores the farthest mincuts from all vertices to any designated vertex s. However, this DAG could have $\mathcal{O}(n^2)$ edges, and establishing the sparsity of the DAG turns out to be the main hurdle. We overcome this by proving the following interesting property of the farthest mincuts to s:

For any three vertices, either the intersection of their farthest mincuts to s is empty or the farthest mincut from at least one of them is a subset of one of the other two.

Using this property, we are able to prune away all the unnecessary edges from the DAG structure storing farthest mincuts to s. As a result, each node in the DAG turns out to have at most 2 incoming edges, so the size of the DAG is $\mathcal{O}(n)$.

For the objective of solving mincut sensitivity for a subset $S \subseteq V$, we present data structures for the nearest mincuts (likewise the farthest mincuts) that consist of vertices of S only instead of V. Their size is $\mathcal{O}(|S|)$. See Theorem 18 and Theorem 6.

Solving the mincut sensitivity problem

For solving the single source mincut sensitivity problem, we use the tree data structure storing the nearest mincuts and the DAG data structure storing the farthest mincuts, from all vertices of the set S to s. The size of the data structure is $\mathcal{O}(|S|)$. Following Lemma 8, it takes $\mathcal{O}(|S|)$ time using this data structure to determine whether the insertion of a given edge increases the (s, v)-mincut value for any vertex $v \in S$. This leads to $\mathcal{O}(|S|^2)$ time to find all vertices from S whose mincut value from s increases due to the insertion of a given edge. In order to reduce the query time to $\mathcal{O}(|S|)$, we make use of the following insight:

A vertex $x \in S$ belongs to the farthest mincut from v to s if and only if x is reachable from v in the DAG structure storing the farthest mincuts to s.

Solving the all-pairs version of the mincut sensitivity problem with optimal query time turns out to be more challenging. As the underlying graph is undirected, the subset of vertices that defines the farthest mincut from u to v is the complement of the subset of the vertices that defines the nearest mincut from v to u. As a result, keeping the nearest-mincut tree data structure for each vertex of S suffices to solve this problem. The data structure takes $\mathcal{O}(1)$ time to determine for any pair (u, v), whether insertion of an edge, say (x, y), increases

(u, v)-mincut value. This observation implies an $\mathcal{O}(|S|^2)$ time algorithm for computing all pairs of vertices from set S whose mincut value increases upon insertion of edge (x, y). But it is quite wasteful if k, the number of pairs whose mincut value increases, is much smaller than $|S|^2$. To accomplish $\mathcal{O}(k)$ query time, we make use of multiple insights into the structure of the nearest mincuts. The most crucial insight is the following:

The vertices whose mincut value to s increases upon insertion of an edge (x, y) lie contiguously on the paths from x and y to their lowest common ancestor in the tree that stores the nearest mincuts to s.

This insight leads to an $\mathcal{O}(|S| + k)$ query time. To get rid of the additive factor of $|S|$, we use another insight that helps finding the *right* pool of vertices whose nearest-mincut tree we need to query.

1.3   Related work

Our research is related to the field of dynamic graph algorithms that emphasizes on efficient data structures to handle changes in a network. For dynamic graph algorithms, the objective is to maintain the solution of a problem for an online sequence of edge insertions or deletions with worst case time complexity better than that of the best static algorithm. There do exist efficient dynamic algorithms for maintaining a global mincut – an incremental algorithm by Goranci, Henzinger, and Thorup [7], and a fully dynamic algorithm by Thorup [12]. However, there does not exist any dynamic algorithm for all-pairs mincuts whose worst case time complexity is better than the best static algorithm. Hartmann and Wagner [10] presented a fully dynamic algorithm for maintaining an all-pairs mincut tree for undirected graphs. Although it achieves a significant speedup over the best static algorithm on many real world graphs, its worst case asymptotic time complexity is not better than the best static algorithm for an all-pairs mincut tree.

1.4   Organization of the paper

Equipped with notations, definitions, and well known lemmas introduced in Section 2, we present the compact data structures for nearest and farthest mincuts in Sections 3 and Section 4, respectively. The data structures for the single source and the all-pairs versions of the mincut sensitivity problem are presented in Section 5 and 6 respectively. Due to the space constraint, the proofs of some theorems and lemmas had to be omitted from this version. So we recommend the reader to refer to the full version of this paper [2].

2   Preliminaries

Our results consider an undirected graph $G = (V, E)$ on n vertices where each edge is assigned a non-negative capacity through a function $c : E \to R^+$.

▶ **Definition 9** ((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 smallest capacity.*

▶ **Definition 10** (set 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 $cut(A, \bar{A})$ or more compactly $cut(A)$.*

When there is no scope of confusion, we do not distinguish between a mincut and the set defining the mincut. We can extend the capacity function c on edges to any subset $A \subset V$ in a natural way as follows: $c(A)$ denotes the sum of the capacities of all those edges which have exactly one endpoint in A. With this generalization, we now state a well-known property of cuts, namely, the submodularity of cuts.

▶ **Lemma 11** (Submodularity of cuts). *Given an undirected graph $G = (V, E)$ with positive edge capacities, the following inequality holds true for any two subsets $A, B \subset V$.*
$$c(A) + c(B) \geq c(A \cup B) + c(A \cap B)$$

The following lemma states an important property of an (s, t)-mincut.

▶ **Lemma 12.** *Let $A \subset V$ define an (s, t)-mincut with $s \in A$. For any subset $A' \subset A$ with $s \notin A'$, if α is the number of edges incident on A' from $V \backslash A$, and β is the number of edges incident on A' from $A \backslash A'$, then $\alpha \leq \beta$.*

2.1   The nearest and the farthest mincuts

▶ **Definition 13** (Nearest and farthest mincuts from s to t). *The subset $A \subset V$ with $s \in A$ is said to define the nearest (likewise the farthest) mincut from s to t if (1) $cut(A, \bar{A})$ defines an (s, t)-mincut, and (2) For every other subset $A' \subset V$ that defines an (s, t)-mincut, $A \subset A'$ (likewise $A' \subset A$). We use s_t^N and s_t^F to denote the nearest and the farthest mincut from s to t, respectively.*

One can easily show using Lemma 11 that the nearest and the farthest mincut from s to t are unique. Additionally, t_s^N and s_t^F partition the set of vertices V as stated in the following lemma.

▶ **Lemma 14.** *For any pair of vertices $s, t \in V$, (i) $s_t^N \cap t_s^N = \emptyset$, and (ii) $s_t^F = V \backslash t_s^N$.*

In the light of Lemma 14, we can restate Lemma 8 for undirected graphs as follows.

▶ **Lemma 15** (Picard and Queyranne [11]). *The insertion of an edge (x, y) can increase the mincut between s and t if and only if $x \in s_t^N$ and $y \in t_s^N$ or vice versa.*

▶ Remark 16. In order to explore the relationship among the farthest mincuts from a set of vertices to a vertex s, we focus only on the connected component of s. This is because for each vertex outside this component, its farthest mincut to s is obvious. Therefore, without loss of generality we assume G to be connected in the rest of the paper.

3   A compact data structure for all nearest mincuts to vertex s

The following theorem plays the key role in compactly storing all nearest mincuts to s.

▶ **Theorem 17** (Hariharan et al. [9]). *For any two distinct vertices $u, v \in S$, either u_s^N and v_s^N are mutually disjoint or one of them is a subset of the other.*

For a given subset $S \subseteq V$, let $\mathcal{N} = \{x_s^N \cap S | x \in S \backslash \{s\}\}$. Using Theorem 17 we can arrange the sets of $\mathcal{N}$ in a forest of disjoint trees as follows. We refer to a vertex in this forest as *node*. For each set present in $\mathcal{N}$, we create a unique node in the forest. We assign each vertex $v \in S$ to the node ν corresponding to $v_s^N \cap S$. The parent of a node ν is defined as the unique node μ such that the set corresponding to μ is the smallest superset of the set corresponding to ν, if such a superset exists. If no such superset exists, ν will be the root of a tree. We create a dummy node and assign it as the parent of the root of every tree in this forest. Let us denote the resulting rooted tree by $\mathcal{T}(s)$. Figure 1 shows an example graph and the corresponding $\mathcal{T}(s)$ for the case $S = V$.

It can be observed that if a vertex $v \in S$ is assigned to node ν, then the subtree rooted at ν stores the set $v_s^N \cap S$. So it follows that a vertex, say x, belongs to $v_s^N \cap S$ if either x and v are assigned to the same node in $\mathcal{T}(s)$ or the node containing v is an ancestor of the
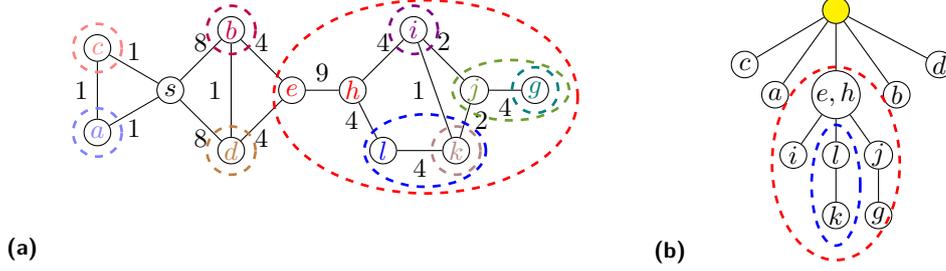
Figure 1 (a) The nearest mincut from a vertex to s is encircled with same color. (b) Tree $\mathcal{T}(s)$.

node containing x. This check can be easily done in $\mathcal{O}(1)$ time if we augment $\mathcal{T}(s)$ to answer lowest common ancestor (LCA) query for any pair of nodes (see [3]). We can thus state the following theorem.

▶ **Theorem 18.** *For an undirected graph $G = (V, E)$, a subset $S \subseteq V$, and any vertex $s \in S$, there exists an $\mathcal{O}(|S|)$ size data structure $\mathcal{T}(s)$ that can report in $\mathcal{O}(1)$ time whether $x \in v_s^N$ for any $x, v \in S$.*

4 A compact data structure for all farthest mincuts to vertex s

In this section, we present a novel data structure that compactly stores the farthest mincuts to vertex s from a subset of vertices. Our main result can be summarized as follows.

▶ **Theorem 19.** *For an undirected graph $G = (V, E)$, any subset $S \subseteq V$, and a designated vertex $s \in S$, there exists a directed acyclic graph $\mathcal{D}(s)$ having $\mathcal{O}(|S|)$ nodes and $\mathcal{O}(|S|)$ edges that can report $v_s^F \cap S$ in time of the order of the size of $v_s^F \cap S$ for any $v \in S\backslash\{s\}$.*

Lemma 14(ii) implies that $s_v^N = V\backslash v_s^F$. So we can compute $s_v^N \cap S$ in $\mathcal{O}(|S|)$ time once we have $v_s^F \cap S$. Therefore, we can state the following corollary of Theorem 19.

▶ **Corollary 20.** *For an undirected graph $G = (V, E)$, any subset $S \subseteq V$, and a designated vertex $s \in S$, there exists a data structure of $\mathcal{O}(|S|)$ size that takes just $\mathcal{O}(|S|)$ time to compute $s_v^N \cap S$ for any $v \in S\backslash\{s\}$.*

Next, we show how to compute a DAG storing all farthest mincuts to s in space $\mathcal{O}(|S|^2)$. Thereafter, we reduce its space complexity to $\mathcal{O}(|S|)$ only.

4.1 A DAG of size $\mathcal{O}(|S|^2)$

Our data structure to store all farthest mincuts to a designated vertex s uses the observation captured in Lemma 21. In its core, it tells us that certain farthest mincuts are related by a subset relation which can be exploited to store them compactly.

▶ **Lemma 21.** *Let x and v be any two vertices in G. If $x \in v_s^F$, then $x_s^F \subseteq v_s^F$.*

Proof. We use Lemma 11 on the submodularity of cuts and provide a proof by contradiction. Let A and B refer to the sets v_s^F and x_s^F, respectively. It is given that $x \in v_s^F$. Assume that $x_s^F \nsubseteq v_s^F$. This would imply that $B\backslash A \neq \emptyset$, hence A must be a proper subset of $A \cup B$.

Observe that $A \cap B$ defines a valid (x, s)-cut since x is present in both A and B, whereas $s \notin x_s^F$. This observation implies that $c(A \cap B) \geq c(B)$ since B defines an (x, s)-mincut. This inequality and Lemma 11 imply the following inequality.

$$c(A \cup B) \leq c(A) \tag{1}$$

Now observe that $A \cup B$ defines a valid (v, s)-cut since v belongs to A, whereas s belongs neither to A nor to B. Since A defines a (v, s)-mincut, so Inequality 1 implies that $c(A \cup B)$ must be equal to the capacity of an (v, s)-mincut. But A is a proper subset of $A \cup B$. This would imply that the cut defined by A is not the farthest mincut from v to s − a contradiction. ◀

Let $\mathcal{F} = \{v_s^F \cap S \mid v \in S \backslash \{s\}\}$. We now use Lemma 21 to build a directed acyclic graph $D = (\mathcal{V}, \mathcal{E})$ that stores $\mathcal{F}$ as follows. We use *node* to refer to a vertex of this DAG.

For each set present in $\mathcal{F}$, we create a unique node in D. The set of nodes thus created constitutes $\mathcal{V}$. We denote by $\mathcal{F}(\nu)$ the set in $\mathcal{F}$ corresponding to node ν. The edge set $\mathcal{E}$ of D is defined as follows.

$$\mathcal{E} = \{(\nu, \mu) \mid \mathcal{F}(\mu) \subset \mathcal{F}(\nu)\}$$

It can be observed that if $X \subset Y$ for any two sets X and Y in $\mathcal{F}$, then $|X| < |Y|$. Hence D is acyclic. To efficiently retrieve $x_s^F \cap S$ for any given $x \in S$, we can augment D as follows.

- We create an array J_s indexed by vertices of set S such that for any $v \in S \setminus \{s\}$, $J_s[v]$ stores the pointer to node μ that corresponds to $v_s^F \cap S$, that is, $\mathcal{F}(\mu) = v_s^F \cap S$.
- Each node μ of D stores a list $L(\mu)$ of all those vertices $v \in S$ such that $J_s[v] = \mu$.
- We introduce a dummy node and add an edge from it to every other node which has no incoming edge.

Lemma 22 follows immediately from Lemma 21 and the construction of D described above.

▶ **Lemma 22.** *Let x and u be any two vertices of set S. x is present in $u_s^F \cap S$ if and only if either $J_s[u] = J_s[x]$ or there is an edge from $J_s[u]$ to $J_s[x]$ in D.*

Lemma 22 implies that for each vertex $v \in S \backslash \{s\}$, $v_s^F \cap S$ is the set of vertices stored in the list $L(J_s[v])$ and the lists of all the nodes with an incoming edge from $J_s[v]$ in D.

The subset relation $\subset$ is transitive. So, if there is a path from a node ν to another node μ in D, then (ν, μ) is also an edge in D. In other words, the transitive closure of D is D itself. This observation in conjunction with Lemma 22 leads us to the following lemma which will be crucial for our data structure for the single source mincut sensitivity problem.

▶ **Lemma 23.** *Let x and u be any two vertices in set S. x is present in $u_s^F \cap S$ if and only if $J_s[x]$ is reachable from $J_s[u]$ in D.*

Notice that D has $\mathcal{O}(|S|)$ nodes, but it could have $\Theta(|S|^2)$ edges. So the total space occupied by D could be $\Theta(|S|^2)$. A natural idea to overcome this hurdle is to remove as many edges as possible from D without affecting the reachability between any pair of its vertices so that Lemma 23 continues to hold. In other words, we compute another DAG D^τ which is the transitive reduction of D. Aho, Garey, and Ullman [1] showed that computing the transitive reduction of a DAG is as easy as computing its transitive closure. While in general a transitive reduction does not always lead to a reduced number of edges, it does so in the case of D. In the following subsection we present crucial insights into crossing farthest mincuts that ensure that each node of D^τ will have at most two incoming edges. The data structure $\mathcal{D}(s)$ in Theorem 19 for storing all farthest mincuts to s is D^τ only.

4.2 Bounding the in-degree of D^τ by 2

A set of vertices $\mathcal{I} \subset V$ is said to be a set of *incomparable* vertices with respect to the mincuts to s if for each $u, v \in \mathcal{I}$ with $u \neq v$ it holds that $u \notin v_s^F$ and $v \notin u_s^F$. The following lemma highlights an important property for a pair of incomparable vertices.

▶ **Lemma 24.** *For any two incomparable vertices u and v, there does not exist any edge between the set $u_s^F \cap v_s^F$ and the set $V \backslash (u_s^F \cup v_s^F)$.*

We shall now use Lemma 12 and Lemma 24 to derive the following lemma which will play a crucial role in establishing that D^τ has indegree 2 only.

▶ **Lemma 25.** *For any three incomparable vertices $u, v, w \in V$, $u_s^F \cap v_s^F \cap w_s^F = \emptyset$.*

Proof. We give a proof by contradiction. Let B, L, and R denote the sets u_s^F, v_s^F, and w_s^F, respectively. Figure 2(*i*) illustrates these sets. For a clear distinction, we have used different colors for these sets in this figure, and correspondingly assigned the labels B (for blue), L (for light green), and R (for red) to these sets.
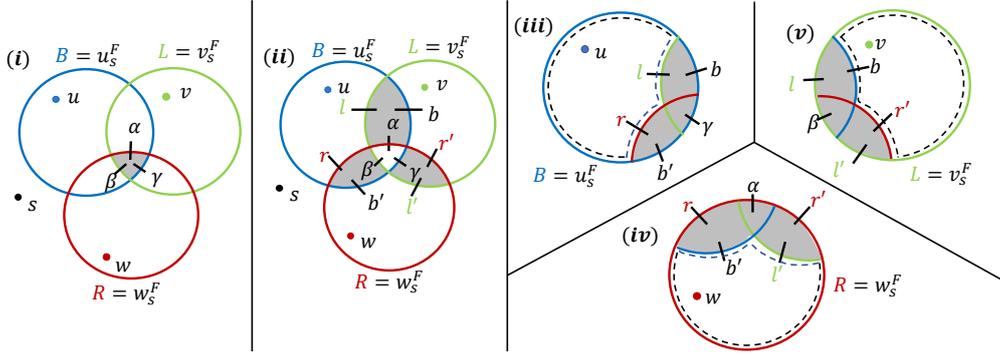
Figure 2 Intersection of the farthest mincuts to s.

Suppose the common intersection $B \cap L \cap R$ of these sets (shown shaded in Figure 2(*i*)) is not an empty set. By applying Lemma 24 for $B \cap L$, $L \cap R$, $R \cap B$, we can infer that each vertex in the common intersection will have edges incident only from the sets $B \cap L$, $L \cap R$, $R \cap B$. Considering the set $B \cap L \cap R$ as a single entity, let α, β, γ be the number of edges incident on it from $(B \cap L) \backslash R$, $(R \cap B) \backslash L$, $(L \cap R) \backslash B$, respectively. As the graph is connected (see Remark 16), we have:

$$\alpha + \beta + \gamma > 0. \tag{2}$$

Let us consider the set $(B \cap L) \backslash R$, that is, the set $B \cap L$ after removing the common intersection $B \cap L \cap R$. It follows from Lemma 24 that the edges incident on this set will be from $B \backslash L$ and $L \backslash B$ only, apart from the edges incident from $B \cap L \cap R$. Similar claims hold for the sets $(B \cap R) \backslash L$ and $(R \cap L) \backslash B$ as well. Figure 2(*ii*) shows these sets as shaded regions along with the edges incident on them. For example, l, b, α are the number of edges incident on $(B \cap L) \backslash R$ from $B \backslash L$, $L \backslash B$, and $B \cap R \cap L$, respectively.

The rest of the proof is as follows. Exploiting the fact that u, v, w are incomparable, we suitably apply Lemma 12 to arrive at inequalities that eventually leads to contradict Inequality 2.

B defines an (s, u)-mincut. Since u is incomparable with both v and w, it is not present in the set $B \cap (R \cup L)$ shown shaded in Figure 2(*iii*). Notice that the number of edges incident on this set from $V \backslash B$ is $b + b' + \gamma$ whereas the number of edges incident on this set from the rest of B, that is, the set $B \backslash (R \cup L)$ (enclosed by dotted boundary in Figure 2(*iii*)) is at most $l + r$. So we get the following inequality by substituting B and $B \cap (R \cup L)$ in place of A and A' respectively in Lemma 12:

$$b + b' + \gamma \leq l + r \tag{3}$$

In Figure 2, R defines an (s, w)-mincut and L defines an (s, v)-mincut. Hence, with similar arguments as above, analyzing the (s, w)-mincut in Figure 2(iv), and analyzing the (s, v)-mincut in Figure 2(v), we get the following inequalities, respectively:

$$r + r' + \alpha \leq b' + l', \qquad\qquad l + l' + \beta \leq b + r'$$

Adding the above inequalities with Inequality 3 and canceling identical terms on either sides we get $\alpha + \beta + \gamma \leq 0$. This contradicts Inequality 2 and completes the proof. ◀

The following is a simple corollary of Lemma 25.

▶ **Corollary 26.** *Let $A \in \mathcal{F}$. If U, V, W are any three distinct sets from $\mathcal{F}$ such that $A \subset U$, $A \subset V$, and $A \subset W$. Then at least one of the sets from $\{U, V, W\}$ must be a proper subset of one of the remaining two.*

We can use Corollary 26 to establish the following lemma.

▶ **Lemma 27.** *The indegree of any node in D^τ will be at most 2.*

Figure 3 shows farthest mincuts from a sample of vertices to s in our example graph. Notice that the farthest mincut b_s^F *crosses* the farthest mincut d_s^F. Also, the farthest mincuts from j and g to s are identical, so j and g are mapped to a single node in D^τ.

(a) **(b)**
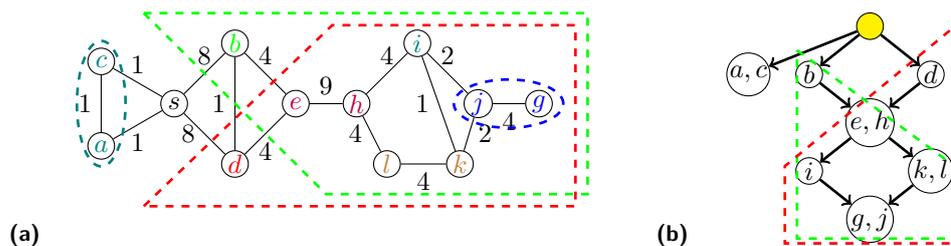
▪ **Figure 3 (a)** A dotted boundary defines a farthest mincut to s from a vertex of the same color. **(b)** The DAG D^τ.

5 Single source mincut sensitivity for insertion of an edge

We now present an $\mathcal{O}(|S|)$ space data structure that can report all those vertices from S whose mincut value to s increases upon insertion of any given edge $(x, y) \in S \times S$. The data structure will consist of the tree structure $\mathcal{T}(s)$ from Theorem 18 and DAG structure $\mathcal{D}(s)$ from Theorem 19.

Let $A_x = \{v \in S | x \in s_v^N\}$ and $A_y = \{v \in S | y \in s_v^N\}$. It follows from Lemma 15 that if (s, v)-mincut increases, then v must belong to A_x or A_y. Furthermore, for any $v \in A_x$, (s, v)-mincut increases if $y \in v_s^N$. Using Theorem 18, it takes just $\mathcal{O}(1)$ time to do this check for any given $v \in A_x$ (likewise A_y). So, in order to report all vertices from S whose mincut from s increases upon insertion of edge (x, y) in $\mathcal{O}(|S|)$ time, all we need is an $\mathcal{O}(|S|)$ time algorithm to compute A_x and A_y. We now provide $\mathcal{O}(|S|)$ time algorithm to compute A_x; we can compute A_y in $\mathcal{O}(|S|)$ time in a similar manner.

It follows from Lemma 14(ii) that computing A_x is equivalent to computing the set $\bar{A}_x = \{v \in S | x \in v_s^F\}$. Recall that $J_s[x]$ is the node containing x in $\mathcal{D}(s)$. It follows from Lemma 23 that $x \in v_s^F$ if and only if $J_s[x]$ is reachable from $J_s[v]$ in $\mathcal{D}(s)$. Therefore, we can compute $\bar{A}_x$ by first reversing the edges of $\mathcal{D}(s)$ and then traversing all the nodes reachable

from $J_s[x]$. For each node λ reachable from $J_s[x]$ in the reversed $\mathcal{D}(s)$, x is present in v_s^F for each vertex $v \in L(\lambda)$. Since $\mathcal{D}(s)$ has $\mathcal{O}(|S|)$ edges, it takes $\mathcal{O}(|S|)$ time to reverse it and traverse it to compute $\bar{A}_x$. This establishes the proof of Theorem 2 for the single source mincut sensitivity problem.

6 All-pairs mincut sensitivity data structure for insertion of an edge

Our data structure consists of the nearest-mincut tree $\mathcal{T}(z)$ from Theorem 18 for each $z \in S$. Each of these trees occupies $\mathcal{O}(|S|)$ space, so the space occupied by the data structure is $\mathcal{O}(|S|^2)$. For the rest of this section, x, y, z are any arbitrary vertices from S. Upon insertion of edge (x, y), let k be the number of pairs of vertices from $S \times S$ whose mincut value increases. We present an $\mathcal{O}(k)$ time algorithm to output all these pairs using four insights into the nearest-mincut trees. Our first insight is stated in Lemma 28. It implies that for all vertices belonging to a node μ in $\mathcal{T}(z)$, it suffices to determine for any single vertex, say $u \in \mu$, whether the (z, u)-mincut value increases upon insertion of any given edge.

▶ **Lemma 28.** *Let $u, v \in S$ be any two vertices belonging to the same node in $\mathcal{T}(z)$. Upon insertion of any given edge, (u, z)-mincut value increases iff (v, z)-mincut value increases.*

Let μ and ν be the nodes in $\mathcal{T}(z)$ containing x and y respectively. Our second insight, stated in the following lemma, specifies the location of vertices in $\mathcal{T}(z)$ whose mincut value to s increases upon insertion of edge (x, y).

▶ **Lemma 29.** *Let $\omega = LCA(\mu, \nu)$ in $\mathcal{T}(z)$. Upon insertion of edge (x, y), the mincut value from z to only those vertices may increase that belong to the nodes of (1) the path from μ to ω but excluding ω, and (2) the path from ν to ω but excluding ω.*
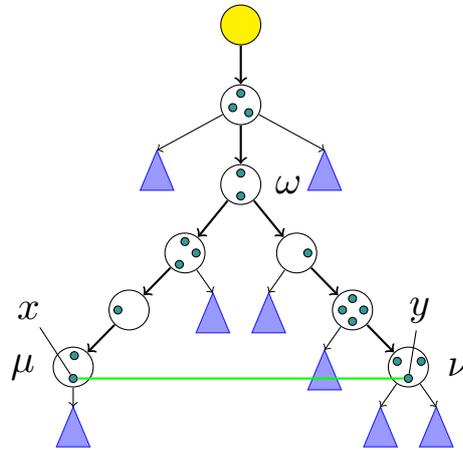
Figure 4 The tree $\mathcal{T}(z)$ from the perspective of μ and ν. If v is a vertex whose node in $\mathcal{T}(z)$ does not belong to either of ω-ν and ω-μ paths, then v must be present in one of the subtrees hanging from these paths (shown in blue). Now consider any node, say γ, lying on the path from ω to the root of $\mathcal{T}(z)$. Both x and y belong to the subtree rooted at γ.

Proof. Let us view $\mathcal{T}(z)$ from the perspective of the paths from μ and ν to the root of $\mathcal{T}(z)$. The reader is advised to refer to Figure 4 for a better understanding. If v is a vertex whose node in $\mathcal{T}(z)$ does not belong to these paths, then v must be present in one of the subtrees (shown in blue in Figure 4) hanging from these paths. Notice that neither x nor y belongs to

the subtree containing v. So it follows from Lemma 15 that the mincut from z to v is not affected by the insertion of edge (x, y). Now consider any node, say γ, lying on the path from ω to the root of $\mathcal{T}(z)$. Both x and y belong to the subtree rooted at γ. So using Lemma 15 again, the mincut from z to any vertex of γ remains unaffected by the insertion of edge (x, y). ◀

Our third and most crucial insight is that the vertices whose mincut value to z increases upon insertion of edge (x, y) belong to a *contiguous* sequence of nodes on the paths from the node containing y and the node containing x to their LCA in $\mathcal{T}(z)$. The following lemma states this insight for the node containing y.

▶ **Lemma 30.** *Let ω be the LCA of the nodes containing x and y in $\mathcal{T}(z)$. Let u and v be any two vertices lying on the path from the node containing y to ω in $\mathcal{T}(z)$ such that the node containing u is an ancestor of the node containing v. If (z, u)-mincut value increases upon insertion of edge (x, y), then (z, v)-mincut value also increases upon insertion of (x, y).*

Proof. It follows from the construction of $\mathcal{T}(z)$ that $y \in v_z^N$ and $v_z^N \subseteq u_z^N$. It is given that the insertion of edge (x, y) increases (z, u)-mincut value and u is an ancestor of y in $\mathcal{T}(z)$. So Lemma 15 implies:

$$x \in z_u^N \tag{4}$$

It follows from Lemma 14(i) that $z_u^N \cap u_z^N = \emptyset$. So $v \notin z_u^N$ since $v \in u_z^N$. Applying Lemma 14(ii), we get $v \in u_z^F$. So it follows from Lemma 21 that $v_z^F \subseteq u_z^F$. Applying Lemma 14(ii) again, it follows that $z_u^N \subseteq z_v^N$. Using this fact and Equation 4, we can infer that $x \in z_v^N$. Since we have already established that $y \in v_z^N$, so using Lemma 15 we can conclude that (z, v)-mincut value will also increase upon insertion of edge (x, y). ◀

It is a simple corollary of Lemma 30 that if (y, z)-mincut value does not increase upon insertion of edge (x, y), then for any vertex v present in any ancestor of the node containing y in $\mathcal{T}(z)$, (v, z)-mincut value will also not increase. So, to compute all-pairs of vertices whose mincut increases, we need to explore the nearest-mincut tree of only those vertices z whose mincut value to y (and likewise to x) increases.

We now describe how to process $\mathcal{T}(z)$ for a vertex z given that (y, z)-mincut value increases upon insertion of (x, y). For each such z, first we enumerate all vertices present in the node, say ν, to which y belongs. We then begin an upward traversal of $\mathcal{T}(z)$ starting from the parent of ν. For any node, say λ, that we traverse, we pick any arbitrary vertex from it, say v, and determine whether $x \in z_v^N$ by querying $\mathcal{T}(v)$. It takes $\mathcal{O}(1)$ time to answer this query (see Theorem 18). If $x \in z_v^N$, it follows from Lemma 28 that each vertex present in λ has its mincut value to z increased. So we enumerate all vertices from λ, and continue processing the parent of λ in a similar manner. If $x \notin z_v^N$, we stop the traversal. It follows from Lemma 30 that the vertices enumerated in this way are precisely the vertices whose mincut value to z increases. To efficiently identify each vertex z, such that the (y, z)-mincut value increases upon insertion of edge (x, y), we exploit the fourth insight into the nearest-mincut trees which is stated in the following lemma. This lemma can be seen as a corollary of Lemma 15.

▶ **Lemma 31.** *(y, z)-mincut value increases upon insertion of edge (x, y) iff $x \in z_y^N$.*

It follows from Lemma 31 that the vertices present in the node containing x and its ancestors in $\mathcal{T}(y)$ are precisely the vertices whose mincut value to y increases. We can identify all these vertices in optimal time by traversing $\mathcal{T}(y)$ upward from the node containing x.

We have described above the processing of each z such that the (y, z)-mincut value increases due to the insertion of edge (x, y). A similar processing must be carried out for all vertices z, such that the (x, z)-mincut value increases due to the insertion of edge (x, y).

It follows from the description given above that we can compute all those pairs of vertices from $S \times S$ whose mincut value increases upon the insertion of any given edge in $\mathcal{O}(k)$ time, where k is the number of these pairs. If our goal is to just report the value of k, we can accomplish it in $\mathcal{O}(\min(k, |S| \log |S|))$ time by suitably augmenting the nearest-mincut trees. We can thus conclude with Theorem 32 which extends Theorem 4.

▶ **Theorem 32.** *For an undirected graph $G = (V, E)$, and a subset S of vertices, there exists an $\mathcal{O}(|S|^2)$ size data structure that can report all pairs of vertices whose mincut increases upon insertion of a query edge. The guaranteed query time is $\mathcal{O}(k)$, where k is the number of pairs whose mincut increases. We can also report k in $\mathcal{O}(\min(k, |S| \log |S|))$ time.*

References

1   Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972. `doi:10.1137/0201008`.

2   Surender Baswana, Shiv Gupta, and Till Knollmann. Mincut Sensitivity Data Structures for the Insertion of an Edge, 2020. URL: `http://www.cse.iitk.ac.in/users/sbaswana/Papers-published/esa-2020-fv.pdf`.

3   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`.

4   Rajesh Chitnis, Lior Kamma, and Robert Krauthgamer. Tight bounds for gomory-hu-like cut counting. In *Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers*, pages 133–144, 2016. `doi:10.1007/978-3-662-53536-3_12`.

5   L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. `doi:10.4153/CJM-1956-045-5`.

6   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`.

7   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`.

8   Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM J. Comput.*, 19(1):143–155, February 1990. `doi:10.1137/0219009`.

9   Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An $\tilde{O}(mn)$ gomory-hu tree construction algorithm for unweighted graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 605–614, 2007. See also the extended version at `http://hariharan-ramesh.com/papers/gohu.pdf`. `doi:10.1145/1250790.1250879`.

10   Tanja Hartmann and Dorothea Wagner. Fast and simple fully-dynamic cut tree construction. In *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, pages 95–105, 2012. `doi:10.1007/978-3-642-35261-4_13`.

11   Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *In Rayward-Smith V.J. (eds) Combinatorial Optimization II. Mathematical Programming Studies*, 13(1):8–16, 1980. `doi:10.1007/BFb0120902`.

12   Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. `doi:10.1007/s00493-007-0045-2`.