All-Pairs k^{th} Mincuts: Combinatorial and Structural Results

Surender Baswana
Department of Computer Science & Engineering, IIT Kanpur, India

Anupam Roy
Department of Computer Science & Engineering, IIT Kanpur, India

Abstract

Let G be an undirected graph on a set V of n vertices. For any non-empty subset $A \subsetneq V$, cut defined by A is the ordered pair $(A, \overline{A})$. Suppose each cut is assigned a value, which is any arbitrary real number. Let $u, v \in V$ be any pair of vertices. A cut is said to be a (u, v)-cut if it separates u and v. A (u, v)-cut of the minimum value is called a (u, v)-mincut. A 2^{nd} (u, v)-mincut is a (u, v)-cut of second minimum value. We can define k^{th}-mincut accordingly. We present the following results for the all-pairs k^{th} mincuts.

(1) Distinct Values of all-pairs k^{th} Mincuts: There exist k spanning trees on V such that for any pair (u, v), the value of k^{th} (u, v)-mincut is equal to the capacity of an edge on the (u, v)-path in one of the k trees. We also show a matching lower bound of $\Omega(\min\{kn, n^2\})$. Our result generalizes the well-known result by Gomory and Hu [19] stating that there are at most $n - 1$ distinct values of all-pairs mincuts.

(2) Ancestor Trees for all-pairs k^{th} Mincuts: In 1991, Cheng and Hu [15] invented a rooted binary tree, called *ancestor tree*, whose leaves are the vertices of the graph and each internal node stores a cut with the following property. For any pair (u, v), the cut stored at their lowest common ancestor (LCA) is a (u, v)-mincut. We introduce a tree called gen-ancestor tree, that generalizes the ancestor tree for k^{th} mincuts, and achieve the following result. There exists a set of $\mathcal{O}(k \log n)$ gen-ancestor trees such that, for any pair (u, v), a k^{th} (u, v)-mincut is stored at the LCA of u and v in at least one of these trees.

(3) Data Structures: We present the following data structures for all-pairs k^{th} mincuts.
(i) There exists an $\mathcal{O}(n \log n)$ space data structure that can report the value of 2^{nd} (u, v)-mincut in $\mathcal{O}(\log n)$ time for any given pair (u, v). We generalize this data structure for k^{th} mincuts with a factor of k^2 in the space and query time.
(ii) There exists an $\mathcal{O}(kn^2 \log n)$ space data structure that can report a k^{th} (u, v)-mincut $(A, \overline{A})$ in $\mathcal{O}(|A|)$ time for any given pair (u, v).
For any constant k, the bounds stated above match, up to a logarithmic factor, the best-known bounds guaranteed by the data structure for all-pairs mincuts (Cheng and Hu [15]).

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Network flows

Keywords and phrases mincut, second mincut, kth mincut, suboptimal cuts, compact structure, all pairs, multi terminal cuts, generalization of Gomory Hu tree, ancestor tree, generalization of all pairs mincuts

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.119

Funding This research work was partially supported by the Research-I Foundation of the CSE Department, IIT Kanpur, India.

Acknowledgements We thank the anonymous reviewers for improving the readability of this article.

1 Introduction

The study of cuts is one of the major fundamental topics in graph theory and has several applications [5]. Let $G = (V, E)$ be an undirected weighted graph on $n = |V|$ vertices and $m = |E|$ edges. Each edge is assigned an arbitrary non-negative real value $c(e)$, called the capacity of edge e. For any non-empty proper subset of vertices $C \subsetneq V$, the cut defined by C is an ordered pair $(C, \overline{C})$, where $\overline{C} = V \setminus C$. For the sake of compactness, we denote $(C, \overline{C})$ by C. Let $u, v \in V$ be any pair of vertices. A cut C is said to *separate* pair (u, v) if $u \in C$ and $v \in \overline{C}$. A cut is said to be a (u, v)-cut if it separates (u, v).

The notion of capacity can be seamlessly extended to cuts as follows. For any cut C, its capacity $c(C)$ is the sum of the capacities of all the edges whose endpoints are separated by C. A (u, v)-cut of the least capacity is called a (u, v)-mincut. In 1956, Ford and Fulkerson [17] designed the first combinatorial algorithm for computing an (s, t)-mincut for a designated pair of vertices s and t in directed graphs, by establishing the *Maxflow-Mincut Theorem*. With this major breakthrough, began the extensive research on mincuts. In a seminal work in 1961, Gomory and Hu [19] addressed the problem of *all-pairs mincuts* in undirected graphs. In particular, they designed an algorithm that computes mincuts for all pairs using only $n - 1$ (s, t)-mincut computations. Thereafter, for the last six decades, extensive research has been done to obtain faster algorithms for this problem. This research has now culminated due to the recent result of Abboud et al. [4] which shows that computing all-pairs mincuts requires the same time as computing mincut for a single pair, up to a sub-polynomial factor.

A lot of research has also been done on cuts beyond minimum capacity in the past [22, 24, 16, 10, 18, 23] and recently as well [8]. Quite expectedly, they play important roles in designing efficient algorithms for fundamental problems such as network unreliability [21], minimum k-way cuts [20]. In 1992, Vazirani and Yannakakis [24] addressed the problem of computing a second (s, t)-mincut for a designated pair of vertices s and t, defined as follows. For any pair of vertices (u, v), C is said to be a second (u, v)-mincut if C is a (u, v)-cut of minimum capacity among the (u, v)-cuts, whose capacity is strictly greater than that of (u, v)-mincut. In other words, a second (u, v)-mincut is a (u, v)-cut of second minimum capacity. Likewise, k^{th} (u, v)-mincut is a (u, v)-cut of k^{th} minimum capacity. Vazirani and Yannakakis [24] then generalized their results for any fixed k, by designing an efficient algorithm for computing a k^{th} (s, t)-mincut.

All the results stated above crucially exploit the fact that the value associated with a cut C is its capacity $c(C)$. However, there exist many problems where associating a value different from $c(C)$, with each cut C, helps in solving the problem efficiently [9, 6, 7, 12]. Let F be any arbitrary real valued function defined on all cuts of G such that $F(C) = F(V \setminus C)$ for every cut C. An (u, v)-mincut is redefined as the (u, v)-cut with the least value of F. In 1991, Cheng and Hu [15] designed an algorithm that computes all-pairs mincuts using only $n - 1$ (s, t)-mincut computations, for any arbitrary cut function F. This result, apart from being an elegant generalization of the results of Gomory and Hu [19], has found many applications, even in directed graphs [9, 7].

The combinatorial and structural results have formed the backbone for designing efficient algorithms for various graph problems. This holds for all-pairs mincuts as well. Gomory and Hu [19] had shown, more than 6 decades ago, that there is a spanning tree, called the *Gomory-Hu tree* that compactly stores all-pairs mincuts. It would not be an exaggeration to state that there would not have been such a phenomenal success in the design of fast algorithms for all-pairs mincuts [2, 3, 1, 4] if the Gomory-Hu tree had not been invented. In a similar way, the *ancestor tree*, a compact structure invented by Cheng and Hu [15] for

storing all-pairs mincuts for any arbitrary cut function F, laid the foundation of compact data structures and efficient algorithms for many problems [9, 6, 7, 14]. We now provide a brief overview of these combinatorial and structural results.

There can be $\Omega(n^2)$ pairs of vertices in a graph. So, there can be potentially $\Omega(n^2)$ different values of mincuts for all pairs. Interestingly, the following combinatorial result by Gomory and Hu [19] shows that the distinct values of all-pairs mincuts is at most $n-1$.

▶ **Theorem 1** (1 spanning tree for mincut values [19]). *Let G_1 denote the complete graph on the vertices in V where the capacity of each edge is equal to its mincut value. Let T_1 be any maximum spanning tree (MST) for G_1. For any pair of vertices (u,v), the value of (u,v)-mincut is equal to the capacity of the minimum capacity edge on the (u,v)-path in T_1.*

Although Gomory and Hu [19] show this for the case when $F = c$, Cheng and Hu [15] note that the result holds for any arbitrary cut function F as well. Having shown the redundancy among the mincut values of all pairs, it is natural to look for similar redundancies among the mincuts for all pairs. Note that Theorem 1 alone fails to reveal any redundancy among cuts since for two pairs of vertices having the same mincut value, there might not exist a single cut that serves as a mincut for both the pairs. Gomory and Hu [19] show that there exists a set of $n-1$ different cuts such that for any pair (u,v), a (u,v)-mincut belongs to the set. Moreover, they show that each of these cuts can be defined by an edge in a spanning tree, famously known as Gomory-Hu tree. Unfortunately, the existence of Gomory-Hu tree crucially exploits the fact that F is submodular and hence might fail to exist when F is any arbitrary cut function. Interestingly, Cheng and Hu [15] obtain the following structural result showing that even for arbitrary cut functions, there exists a set of $n-1$ cuts such that a mincut for each pair belongs to the set.

▶ **Theorem 2** (Ancestor Tree [15]). *There exists a rooted full binary tree whose leaf nodes are the vertices in G and each internal node stores a cut such that for any pair of vertices (u,v), the Lowest Common Ancestor (LCA) of u and v stores a (u,v)-mincut.*

We can redefine k^{th} (u,v)-mincut as the (u,v)-cut of k^{th} minimum value (instead of capacity). Note that the results stated above reveal huge redundancies (factor of $\Omega(n)$) among the all-pairs mincuts, in terms of values as well as cuts. In a graph, for any pair of vertices (u,v), there exists 2^{n-2} (u,v)-cuts and each (u,v)-cut is a k^{th} (u,v)-mincut for some k. It is now natural to ask whether the above redundancies exist for all-pairs k^{th} mincuts as well. In this paper, we present the first combinatorial and structural results for the problem of *all-pairs k^{th} mincuts*, and also address their optimality. Our results are for undirected graphs and for any arbitrary cut function F. Henceforth, for a pair (u,v), we use $\lambda_k(u,v)$ to denote the value of k^{th} (u,v)-mincut.

We obtain all our results first for $k=2$ and then generalize them for any $k \geq 1$. We start by obtaining the following upper bound on the distinct values of all-pairs k^{th} mincuts.

1.1 Distinct Values of All-Pairs k^{th} Mincuts

For all-pairs second mincuts, we present the following theorem that generalizes Theorem 1 of Gomory and Hu [19] for second mincuts.

▶ **Theorem 3** (2 spanning trees for second mincut values). *For any undirected graph, there exist two spanning trees T_2^1 and T_2 storing mincut and second mincut value for all pairs of vertices as follows. For any pair of vertices (u,v),*

1. *$\lambda_2(u,v)$ is equal to either the capacity of minimum capacity edge on the (u,v)-path in T_2, or the capacity of an edge on the (u,v)-path in T_2^1.*
2. *$\lambda_1(u,v)$ is equal to the capacity of an edge on the (u,v)-path in T_2^1.*

It follows from Theorem 3 that the distinct values of second mincuts for all pairs is at most $2(n-1)$. For any $k \geq 1$, we extend Theorem 3 as follows. There exists a set of k spanning trees such that for any pair (u, v), $\lambda_k(u, v)$ is equal to the capacity of an edge on the (u, v)-path in one of the trees (refer to Theorem 17). Using this, we improve the upper bound on the distinct values of all-pairs k^{th} mincuts from $\mathcal{O}(n^2)$ to $\mathcal{O}(\min\{kn, n^2\})$. Moreover, we show that the bound is asymptotically tight as well. This bound shows that a redundancy factor of $\Omega(n/k)$ continues to exist for distinct values of all-pairs k^{th} mincuts, even for $k > 1$.

The distinct values of all-pairs mincuts is $n-1$ in the worst case. This matches the size of the smallest set of cuts required to contain a mincut for each pair, as stated in Theorem 2. However, the same fails to hold even for the case of second mincuts as follows. We improve the upper bound on distinct values of all-pairs second mincuts from $2n-2$ to $2n-4$, and also establish its tightness by showing a matching lower bound. In contrast, we show that there exist graphs such that no set of at most $2n-4$ cuts exists that contain a second mincut for each pair of vertices (refer to Appendix B). This fact leads to newer challenges compared to the mincuts as stated below.

Consider any pair of vertices (u, v). Observe that T_1 can be used as an $\mathcal{O}(n)$ space data structure to retrieve $\lambda_1(u, v)$ efficiently. The two spanning trees T_2 and T_2^1 in Theorem 3 also expose a neat structure underlying the distinct values of all-pairs second mincuts. If $\lambda_2(u, v)$ is stored in T_2, we can use T_2 to retrieve it efficiently as well. However, if $\lambda_2(u, v)$ is stored in T_2^1, observe that we cannot retrieve $\lambda_2(u, v)$ using T_2^1 alone. Moreover for the set of such pairs, the smallest set of cuts that stores a second mincut for each of them may have at least $2n-4$ cuts in the worst case (refer to Appendix B); although the number of distinct values stored in T_2^1 are at most $n-1$. Hence, T_2 and T_2^1 are neither adequate as a data structure to efficiently report second mincut values nor able to reveal any redundancy among the second mincuts for all pairs.

1.2 Compact Structure for All-Pairs k^{th} Mincuts

We design a compact structure that stores a k^{th} mincut (not just value) for each pair of vertices. This also acts as a foundation for the data structure that can efficiently report a k^{th} mincut and its value for any given pair of vertices. To design this structure, we first generalize the ancestor tree of Cheng and Hu [15] (Theorem 2) for second mincuts as follows.

▶ **Lemma 4** (Gen-ancestor Tree). *For any given set P of pairs of vertices, there exists a rooted full binary tree satisfying the following properties.*

1. *Each internal node μ in the tree is associated with at least one pair of vertices $(a, b) \in P$ such that the cut stored at μ is a second mincut for (a, b).*
2. *For any pair $(u, v) \in P$, u and v are mapped to different leaf nodes in the tree and the cut stored at LCA of the nodes containing u and v, is either a (u, v)-mincut or a second (u, v)-mincut.*

Observe that the ancestor tree alone acts as a compact structure for storing all-pairs mincuts. However, as evident from Lemma 4, building gen-ancestor on all pairs might store second mincut for only a subset of pairs (at their LCA), while storing a mincut for the remaining pairs. To address this issue, we use a *novel* partitioning of the set of all pairs into $\mathcal{O}(\log n)$ sets and build the gen-ancestor tree on each set of pairs, providing us the following compact structure for storing all-pairs second mincuts.

▶ **Theorem 5.** *For any undirected graph on n vertices, there exists a set of $\mathcal{O}(\log n)$ gen-ancestor trees such that for any pair of vertices (u, v), the LCA of u and v stores a second (u, v)-mincut in one of the $\mathcal{O}(\log n)$ trees.*

To design a compact structure for all-pairs k^{th} mincuts for any fixed $k \geq 1$, we first generalize the gen-ancestor tree for any $k \geq 1$ (Theorem 20). We then generalize Theorem 5 by showing that there exists a set of $\mathcal{O}(k \log n)$ gen-ancestor trees such that for any pair of vertices (u, v), the LCA of u and v stores a k^{th} (u, v)-mincut in one of the $\mathcal{O}(k \log n)$ trees. Given the lower bound of $\Omega(kn)$ on the distinct values of all-pairs k^{th} mincuts, the number of cuts stored by our compact structure is optimal up to a logarithmic factor.

1.3 Data Structures for Reporting All-Pairs k^{th} Mincuts

The ancestor tree (Theorem 2) can be used as an efficient data structure for reporting a mincut or its value for any given pair of vertices (u, v). However, note that without the mapping from all pairs of vertices to the $\mathcal{O}(\log n)$ gen-ancestor trees, the compact structure in Theorem 5 fails to report a second (u, v)-mincut or $\lambda_2(u, v)$. By further exploring the properties of the gen-ancestor tree, we show that storing our compact structure in Theorem 5 along with the ancestor tree leads us to the following data structure.

▶ **Theorem 6.** *For any undirected graph on n vertices, there exists an $\mathcal{O}(n \log n)$ space data structure that given a pair of vertices (u, v), can report the value $\lambda_2(u, v)$ in $\mathcal{O}(\log n)$ time. Moreover, there exists an $\mathcal{O}(n^2 \log n)$ space data structure that can report a second (u, v)-mincut C in $\mathcal{O}(|C|)$ time.*

▶ Remark 7. The ancestor tree of Cheng and Hu [15] occupies $\mathcal{O}(n^2)$ space. Although, this bound is inferior to the $\mathcal{O}(n)$ space bound for storing Gomory-Hu tree, the ancestor tree is for the more generic setting of arbitrary cut functions. Moreover, the bound of $\mathcal{O}(n^2)$ has remained unbroken for the last 35 years. Theorem 6 shows that the space required for storing all-pairs second mincuts exceeds that of all-pairs mincuts by a factor of $\mathcal{O}(\log n)$ only.

We extend the data structure in Theorem 6 for any $k \geq 1$ with a factor of $\mathcal{O}(k^2)$ in the space and query time. Moreover this data structure can report the value of j^{th} (u, v)-mincut, for any $j \leq k$ instead of a fixed k.

2 Preliminaries

- $C^k_{u,v}$ denotes a k^{th} (u, v)-mincut for a pair of vertices (u, v).
- For any set P of pairs of vertices, $V(P)$ denotes the endpoints of pairs in P.

▶ **Lemma 8** (Cut separates at least one pair in any sequence). *Let $u, v \in V$ be any pair of vertices and $(A, \overline{A})$ be any (u, v)-cut. For every sequence $\langle x_1(= u), x_2, \ldots, x_\ell(= v)\rangle$, there exists $1 \leq i < \ell$ such that $(A, \overline{A})$ is also a (x_i, x_{i+1})-cut.*

Proof. Observe that $x_1(= u)$, the first vertex of the sequence, belongs to A, and $x_\ell(= v)$, the last vertex of the sequence, belongs to $\overline{A}$. So when we traverse the sequence from x_1 to x_ℓ, we are bound to find a vertex $x_i, 1 \leq i < \ell$ such that $x_i \in A$ and $x_{i+1} \in \overline{A}$. So $(A, \overline{A})$ is also an (x_i, x_{i+1})-cut. ◀

▶ **Lemma 9** (MST Property). *The capacity of an edge (u, v) in a graph H is upper bounded by the capacity of minimum capacity edge on the (u, v)-path in a MST of H.*

Although a mincut exists for each pair of vertices, a k^{th} mincut might not exist for some pair (u, v) even for $k = 2$. For example, in an undirected unweighted graph with $n - 2$ vertex disjoint paths from u to v, each of length 2, observe that each of the 2^{n-2} (u, v)-cuts is a (u, v)-mincut. In such situations, there are two possible variants one might like to consider.

(1) Consider only the subset of pairs of vertices for which a k^{th} mincut exists. (2) If for a pair (u, v), its k^{th} mincut does not exist, we redefine k^{th} (u, v)-mincut as the highest value (u, v)-cut that exists. As the reader may observe, all our results work for both the variants. Henceforth, without loss of generality, assume a k^{th} mincut exists for all pairs of vertices.

3 Tight Bounds on Distinct Values of All-Pairs k^{th} Mincuts

We present tight bounds on the distinct values of all-pairs k^{th} mincuts. We first obtain our results for $k = 2$, and then generalize them for $k > 2$. To obtain the bound of $n - 1$ on the distinct values of all-pairs mincuts, stated in Theorem 1, Gomory and Hu [19] first focus on any three vertices x, y, z and establish the following property of mincuts for them.

▶ **Lemma 10** (3-pairs Mincuts [19]). *Let x, y, z be any three vertices. If (x, z) has the least mincut value among the three pairs $(x, y), (y, z)$ and (x, z), then $\lambda_1(x, z) = \min\{\lambda_1(x, y), \lambda_1(y, z)\}$.*

Theorem 1 is obtained by crucially extending Lemma 10 for any sequence of vertices and using a well-known MST property. However, Lemma 10 fails to hold when it comes to second mincuts, that is, each of the three pairs $(x, y), (y, z), (x, z)$ might have three distinct second mincut values. This fact is illustrated in Figure 1. There are two distinct values of mincuts for the three pairs as shown in Figure 1(i). But, the second mincut for each of the three pairs is distinct as shown in Figure 1(ii).

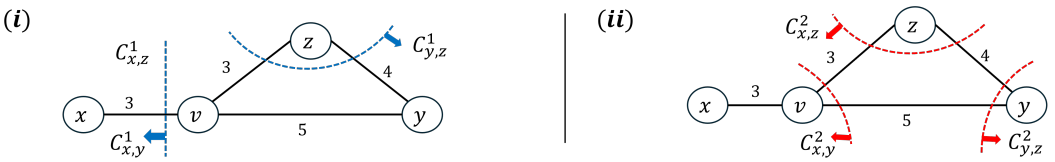

Figure 1 (i) $C^1_{x,y}, C^1_{y,z}, C^1_{x,z}$ are the mincuts and (ii) $C^2_{x,y}, C^2_{y,z}, C^2_{x,z}$ are the second mincuts for the pairs $(x, y), (y, z), (x, z)$ respectively.

Note that in Figure 1, $C^2_{x,z}$ coincides with $C^1_{y,z}$. However, this is not a coincidence. In fact, if there are three distinct values, we show that one of the second mincut values must coincide with mincut value of one of the three pairs, as shown in the following section.

3.1 3-Pairs Second Mincuts

The following classical observation about cuts acts as a basic ingredient for deriving several results on mincuts. If a cut C separates a pair of vertices (u, v), then $F(C)$ is an upper bound on $\lambda_1(u, v)$. We begin with the following lemma that can be viewed as a simple generalization of this observation.

▶ **Lemma 11** (Upper Bound on Second Mincut). *Let (u, v) be any pair of vertices and C be any cut that separates (u, v). Then $\lambda_1(u, v) = F(C)$ or $\lambda_2(u, v) \leq F(C)$.*

Proof. Since C is a (u, v)-cut, $\lambda_1(u, v) \leq F(C)$. So, either $\lambda_1(u, v) = F(C)$ or $\lambda_1(u, v) < F(C)$. Since C is a (u, v)-cut, $\lambda_1(u, v) < F(C)$ implies $\lambda_2(u, v) \leq F(C)$ by definition of second mincut. ◀

Let x, y, z be any three vertices. It follows from Lemma 10 that among the three pairs $(x, y), (y, z)$ and (x, z), the smallest mincut value is shared by at least two pairs. This is illustrated in Figure 2(i); the grey balls represent the mincut values of the three pairs and

their two possible arrangements. However, the smallest second mincut value might or might not be shared by two pairs. Hence, there are two major types of cases (overall four cases) based on the arrangements of second mincuts values of three pairs. This is illustrated in Figure 2(*ii*); the yellow balls represent the second mincut values of the three pairs in increasing order and their four possible arrangements. In Type 1 cases, at least two pairs share the smallest second mincut value, identical to the case of mincuts. Refer to Figure 2(*i*) and Type 1 in Figure 2(*ii*). On the other hand, Type 2 cases are those where the smallest second mincut value is unique (refer to Type 2 in Figure 2(*ii*)). We now handle both the Type 2 cases in the following lemma.

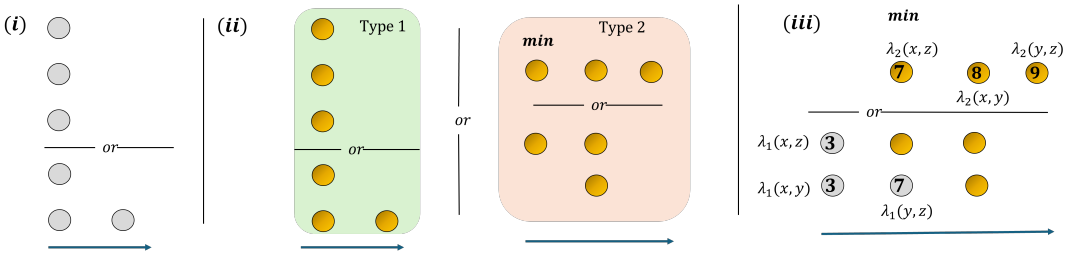

Figure 2 Arrangement of (*i*) mincut and (*ii*) second mincut values for three pairs in increasing order. (*iii*) Relation between mincut and second mincut values of the three pairs based on Figure 1.

▶ **Lemma 12.** *Let x, y, z be any three vertices. If $\lambda_2(x,z)$ is strictly less than $\lambda_2(x,y)$ as well as $\lambda_2(y,z)$, then $\lambda_2(x,z) = \max\{\lambda_1(x,y), \lambda_1(y,z)\}$.*

Proof. Let $C^2_{x,z}$ be any second (x,z)-mincut. We first establish that $\lambda_2(x,z)$ equals to $\lambda_1(x,y)$ or $\lambda_1(y,z)$. Surely, y lies on one of the sides of the cut $C^2_{x,z}$. Without loss of generality, assume that y lies on the side of x, that is, $y \in C^2_{x,z}$; otherwise swap z with x. Since $C^2_{x,z}$ is a cut for (y,z), it follows from Lemma 11 that $\lambda_1(y,z) = \lambda_2(x,z)$ or $\lambda_2(y,z) \leq \lambda_2(x,z)$. However, the latter case cannot be true since it follows from the premise that $\lambda_2(y,z) > \lambda_2(x,z)$. Therefore, $\lambda_2(x,z) = \lambda_1(y,z)$.

By Lemma 10, there can be at most two distinct mincut values for the three pairs, say α_1 and α_2, where $\alpha_1 \leq \alpha_2$. We first analyze the case $\alpha_1 < \alpha_2$. It follows from the above discussion that $\lambda_2(x,z) = \alpha_1$ or α_2. Assume $\lambda_2(x,z) = \alpha_1$. By definition of second mincut, $\lambda_1(x,z) < \alpha_1$. This implies that the least mincut value among the three pairs is strictly less than α_1 – a contradiction. We can show that the case $\alpha_1 = \alpha_2$ cannot arise by the same reasoning. Therefore, $\lambda_2(x,z) = \alpha_2 = \max\{\lambda_1(x,y), \lambda_1(y,z)\}$. This also implies that, for Type 2 cases, the distinct values of mincuts for the three pairs must be exactly two. Thus Figure 2 (*iii*) precisely depicts the Type 2 cases. ◀

As explained above, Type 1 cases satisfy the same property as 3-pairs mincuts (Lemma 10) while Type 2 cases satisfy Lemma 12. Hence, we combine the two types of cases (Type 1 and Type 2) in the following lemma.

▶ **Lemma 13** (3-pairs Second Mincut)**.** *Let x, y, z be any three vertices. If (x,z) has the least second mincut value among the three pairs of vertices $(x,y), (y,z)$ and (x,z), then exactly one of the following properties hold.*

1. $\lambda_2(x,z) = \min\{\lambda_2(x,y), \lambda_2(y,z)\}$ *or,*
2. $\lambda_2(x,z) = \max\{\lambda_1(x,y), \lambda_1(y,z)\}$.

3.2 All-Pairs k^{th} Mincuts

Lemma 13 generalizes 3-pairs mincuts (Lemma 10) for second mincuts. As a first step to generalize Lemma 13 for all pairs and for any $k \geq 1$, we extend Lemma 11 for k^{th} mincuts.

▶ **Lemma 14** (Upper Bound on k^{th} Mincut). *Let (x, y) be any pair of vertices and k be any positive integer. If C is a cut that separates (x, y), then either (1) $F(C) = \lambda_i(x, y)$ for some $i \leq k-1$ or (2) $F(C) \geq \lambda_k(x, y)$.*

Proof. We give a proof by induction on the value of k. Observe that for the base case when $k = 2$, it follows from Lemma 11 that the assertion is true. Assume the assertion is true for $k-1$. So, either (a) $\lambda_i(x, y) = F(C)$ for some $i \leq k-2$ or (b) $\lambda_{k-1}(x, y) \leq F(C)$. It follows from (b) that either $\lambda_{k-1}(x, y) = F(C)$ or $\lambda_{k-1}(x, y) < F(C)$. If (a) is true or $\lambda_{k-1}(x, y) = F(C)$, then (1) is true. Now, assume $\lambda_{k-1}(x, y) < F(C)$. Since C is a cut for (x, y) with value strictly greater than $\lambda_{k-1}(x, y)$, $\lambda_k(x, y) \leq F(C)$. Hence, either (1) or (2) is true. Therefore, the assertion is true for k. ◀

Let G_k denote the complete graph on V, where the capacity of an edge (u, v) is equal to $\lambda_k(u, v)$. The following lemma relates a MST of G_k with all-pairs k^{th} mincut values.

▶ **Lemma 15** (All-Pairs k^{th} Mincut Values). *Let T_k be any MST in G_k. For any pair of vertices (u, v), exactly one of the following properties hold.*

1. *$\lambda_k(u, v)$ is equal to the capacity of minimum capacity edge on the (u, v)-path in T_k.*
2. *$\lambda_k(u, v) = \lambda_j(x, y)$ for some edge (x, y) on the (u, v)-path in T_k, for some $j < k$.*

Proof. Let $C^k_{u,v}$ be any k^{th} (u, v)-mincut and $P_{u,v}$ denote the path between u and v in T_k. By Lemma 8, $C^k_{u,v}$ must separate at least one edge, say (x, y), in $P_{u,v}$. Since $C^k_{u,v}$ is a cut for (x, y), by Lemma 14, either (a) $\lambda_k(u, v) = \lambda_i(x, y)$ for some $i < k$ or (b) $\lambda_k(u, v) \geq \lambda_k(x, y)$. Observe that (a) completes the proof of Assertion (2). Let $\min\{P_{u,v}\}$ denote the capacity of minimum capacity edge on $P_{u,v}$. Observe that the capacity of edges $(u, v), (x, y)$ in T_k are their k^{th} mincut values. If (b) is true, then $\lambda_k(u, v) \geq \lambda_k(x, y) \geq \min\{P_{u,v}\}$. Since T_k is a MST for G_k, it follows from the MST Property (Lemma 9) that $\lambda_k(u, v) \leq \min\{P_{u,v}\}$. Hence, $\lambda_k(u, v) = \min\{P_{u,v}\}$. This completes the proof of Assertion (1). ◀

Observe that T_k does not store the k^{th} mincut value for the pairs that satisfy Assertion (2) in Lemma 15. To store the k^{th} mincut values of such pairs, we define a spanning tree T^j_k with edges same as T_k and the capacity of each edge (x, y) equal to $\lambda_j(x, y)$. By Lemma 15, the k spanning trees $T^1_k, T^2_k, \ldots, T^{k-1}_k, T_k$ store the k^{th} mincut value for all pairs of vertices leading to a bound of $k(n-1)$. However, since the number of distinct pairs is $\mathcal{O}(n^2)$, the bound becomes $\mathcal{O}(\min\{kn, n^2\})$. We now show that the $k-1$ spanning trees $T^1_k, T^2_k, \ldots, T^{k-1}_k$ also store the j^{th} mincut values for all pairs, for each $j < k$.

▶ **Lemma 16.** *For any pair of vertices (u, v), $\lambda_j(u, v)$, for any $j < k$, is equal to the capacity of some edge on the (u, v)-path in one of the trees $T^1_k, T^2_k, \ldots, T^{k-1}_k$.*

Proof. Let $C^j_{u,v}$ be any j^{th} (u, v)-mincut. By Lemma 8, $C^j_{u,v}$ must separate at least one edge, say (x, y), on the (u, v)-path. Since T_k is a MST for G_k, it follows from the MST Property (Lemma 9) and construction of T_k that $\lambda_k(u, v) \leq \lambda_k(x, y)$. So, $\lambda_j(u, v) < \lambda_k(u, v) \leq \lambda_k(x, y)$ for any $j < k$. Since $C^j_{u,v}$ is an (x, y)-cut, by Lemma 14, either (1) $\lambda_j(u, v) = \lambda_i(x, y)$ for some $i < k$ or (2) $\lambda_j(u, v) \geq \lambda_k(x, y)$. However, (2) contradicts that T_k is a MST for G_k. Therefore, (1) is true and completes the proof. ◀

Lemma 16 provides an upper bound of $k(n-1) = \mathcal{O}(kn)$ on the distinct values of all pairs j^{th} mincut, for all $j \leq k$. Lemma 15 and Lemma 16 together lead to the following theorem that generalizes Theorem 1.

▶ **Theorem 17** (k spanning trees for k^{th} mincuts values). *For any undirected graph, there exists a set of k spanning trees $T_k^1, \ldots, T_k^{k-1}, T_k$ storing the j^{th} mincut value for all pair of vertices, for all $j \leq k$, as follows. For any pair (u, v),*

1. *$\lambda_k(u, v)$ is equal to either the capacity of minimum capacity edge on the (u, v)-path in T_k or, the capacity of an edge on the (u, v)-path in one of the trees $T_k^1, T_k^2, \ldots, T_k^{k-1}$.*
2. *$\lambda_j(u, v)$, for any $j < k$, is equal to the capacity of an edge on the (u, v)-path in one of the trees $T_k^1, T_k^2, \ldots, T_k^{k-1}$.*

We show that upper bound on the distinct values of all-pairs k^{th} mincuts is asymptotically tight as well in Appendix C. Moreover, for the case $k = 2$, we establish a more interesting upper bound and a simpler lower bound which is tight upto additive factors. The details are provided in Appendix A and Appendix B.

4 Generalized Ancestor trees for All-Pairs k^{th} Mincuts

We present a compact structure that stores a k^{th} mincut for each pair of vertices. To design this structure, we first generalize the ancestor tree (Theorem 2) for any $k \geq 1$.

4.1 Generalized Ancestor Tree

We present below a recursive algorithm to build a gen-ancestor tree $T(P, k)$ for a given set P of pairs of vertices and an integer $k \geq 1$. Each recursive call receives as input, a subset of pairs and a subset of vertices. For the first recursive call, the input is P and $V(P)$.

1. Let $(a, b) \in P$ be the pair whose k^{th} mincut value is least among all pairs in P and let C be a k^{th} (a, b)-mincut. We create a node μ that stores the cut C and its value $F(C)$.
2. C partitions P into three sets: (i) P_C : pairs in P which are separated by C, (ii) P_L : pairs whose both endpoints belong to C, and (iii) P_R: pairs whose both endpoints belong to $\overline{C}$. Likewise, C partitions $V(P)$ into two sets $V(P) \cap C$ and $V(P) \cap \overline{C}$.
3. If $P_L = \emptyset$, $T(P_L, k)$ is a single node storing $V(P) \cap C$; otherwise we recursively compute $T(P_L, k)$ on $\{P_L, V(P) \cap C\}$.
4. If $P_R = \emptyset$, $T(P_R, k)$ is a single node storing $V(P) \cap \overline{C}$; otherwise we recursively compute $T(P_R, k)$ on $\{P_R, V(P) \cap \overline{C}\}$.
5. Attach $T(P_L, k)$ as the left child and $T(P_R, k)$ as the right child of μ.

It follows from the algorithm that the tree $T(P, k)$ is a rooted full binary tree. Observe that the leaf nodes define a partition of $V(P)$ such that for any pair $(u, v) \in P$, u and v are mapped to different leaf nodes. Hence, the tree has at most $|V(P)| - 1$ internal nodes and at most $|V(P)|$ leaf nodes. For any vertex $v \in V(P)$, let $L(v)$ denote the leaf node in $T(P, k)$ to which v is mapped. The following lemma is immediate from the algorithm.

▶ **Lemma 18.** *For any $u, v \in V(P)$ such that u and v are mapped to different leaf nodes in $T(P, k)$, the LCA of $L(u)$ and $L(v)$ stores a (u, v)-cut.*

Now, for any $(u, v) \in P$, we establish a stronger property than Lemma 18. Observe that for any internal node μ, the following properties hold. (1)μ stores a cut C and its value $F(C)$. (2) There is a subset of pairs $P_\mu \subseteq P$ associated with μ which are separated by C for the first time in the algorithm when μ is created. (3) C is the least valued k^{th} mincut among all pairs in P_μ. For any $(u, v) \in P_\mu$, the following lemma shows that C is not any arbitrary (u, v)-cut but a j^{th} (u, v)-mincut for some $j \leq k$.

▶ **Lemma 19.** *If C is a cut whose value is the least among the k^{th} mincuts for a given set of pairs, then for any pair (u, v) in the set, if C separates (u, v), C is a j^{th} (u, v)-mincut for some $j \leq k$.*

Proof. Since C is a cut for (u, v), it follows from Lemma 14 that either (1) $\lambda_i(u, v) = F(C)$ for some $i < k$ or (2) $\lambda_k(u, v) \leq F(C)$. Moreover, it follows from the premise that $\lambda_k(u, v) \geq F(C)$. Therefore, either (1) is true or $\lambda_k(u, v) = F(C)$. ◀

Property (2) stated above implies that for any $(u, v) \in P_\mu$, μ is the LCA of $L(u)$ and $L(v)$ in tree $T(P, k)$. So, the following theorem is immediate from the algorithm and Lemma 19.

▶ **Theorem 20** (Gen-ancestor Tree)**.** *Given an undirected graph on n vertices, for any given set P of pairs of vertices and an integer $k \geq 1$, there exists a rooted full binary tree $T(P, k)$ of size $\mathcal{O}(n|V(P)|)$ such that the following properties hold.*

1. *Each internal node μ in the tree is associated with at least one pair of vertices $(a, b) \in P$ such that the cut stored at μ is a k^{th} mincut for (a, b).*
2. *For any pair $(u, v) \in P$, u and v are mapped to different leaf nodes in $T(P, k)$ and the cut stored at $LCA(L(u), L(v))$ is a j^{th} (u, v)-mincut for some $j \leq k$.*

We say that the gen-ancestor $T(P, k)$ *stores* a j^{th} mincut for (u, v) if the cut stored at LCA of $L(u)$ and $L(v)$ is a j^{th} (u, v)-mincut. We can further augment the above algorithm to return the subset of pairs in P for which the k^{th} mincut is not stored in $T(P, k)$ as follows. Let Q_C be the pairs in P_C for which we have not stored k^{th} mincut. Q_L and Q_R are similarly defined for P_L and P_R, and are computed recursively. The union of Q_C, Q_L, Q_R, say $\mathcal{R}$, comprises of all pairs in P whose k^{th} mincut is not stored in the tree $T(P, k)$. We name this modified algorithm as BuildTree(P, k) that returns the tuple $\{T(P, k), \mathcal{R}\}$. We now use the gen-ancestor tree to arrive at our structure for storing a k^{th} mincut (first for $k = 2$) for each pair of vertices.

▶ Note 21. The construction of ancestor tree, stated in [15], is the same as the gen-ancestor tree for $k = 1$. However, unlike the ancestor tree which stores a mincut for all pairs, the gen-ancestor tree stores a k^{th} mincut for only a subset of pairs. Hence, the simple divide and conquer strategy, which works for mincuts fails for k^{th} mincuts, when $k \geq 2$. One might also consider the following greedy strategy. Build the gen-ancestor tree again on the remaining pairs, whose k^{th} mincut is not stored. Repeat this step till there are no such pairs left. However, with some effort, one can show that this strategy may require storing $\Omega(n)$ gen-ancestor trees on some graphs, e.g., the graph in Figure 3(i).

4.2 $\mathcal{O}(\log n)$ Gen-ancestor Trees for Storing All-Pairs Second Mincuts

Suppose we build the gen-ancestor tree $T(P, 2)$ on a set P of pairs of vertices. Let $(u, v) \in P$ be any pair of vertices. It is quite possible that $T(P, 2)$ fails to store a second (u, v)-mincut. Let us derive the necessary condition for this to happen. In this case, as evident from Theorem 20(2), the gen-ancestor tree must store a mincut, say C, for (u, v). So, by Theorem 20(1), there exists at least one other pair $(a, b) \in P$ such that C is a second mincut for (a, b). This leads to us to the notion of *adversary* as follows. If $\lambda_2(x, y) = \lambda_1(u, v)$, the pair (x, y) is defined to be an *adversary* for pair (u, v). Hence, the following lemma is immediate.

▶ **Lemma 22.** *If the gen-ancestor tree $T(P, 2)$ stores a mincut for a pair $(u, v) \in P$, then there exists at least one adversary for (u, v) in P.*

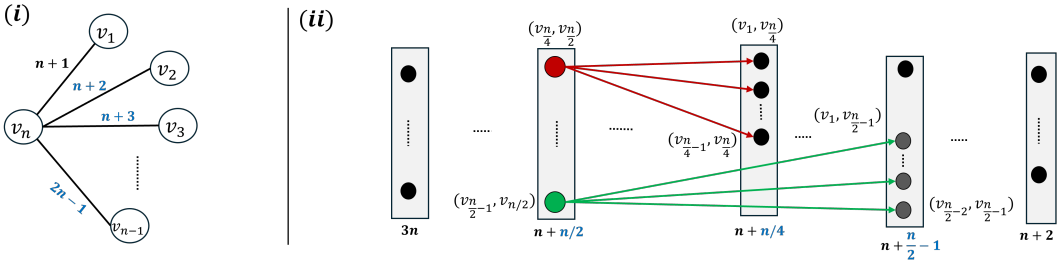

Figure 3 (i) A star graph H (ii) Bunches formed by grouping pairs based on second mincut value in H. The number below each bunch denotes the second mincut value.

Let us inspect all the adversaries for a pair of vertices (u, v). It is quite possible for (u, v) to have $\Omega(n)$ adversaries. In fact, there are graphs where there can be $\Omega(n^2)$ pairs with $\Omega(n)$ adversaries (refer to Note 24). However, by definition, each adversary, say (x, y) of (u, v) has $\lambda_2(x, y) = \lambda_1(u, v)$. Hence, all the adversaries for (u, v) have the same second mincut value. So, we partition the set of all the pairs based on their second mincut values as follows.

▶ **Definition 23** (Bunch). *Two pairs belong to the same bunch if the values of second mincuts for both pairs are same. $B(u, v)$ denotes the bunch to which a pair (u, v) belongs.*

▶ Note 24. We show that there can be $\Omega(n^2)$ pairs with $\Omega(n)$ adversaries. Refer to the graph shown in Figure 3(i). Let j, l be any pair of integers such that $1 \leq j < l \leq n-1$. Observe that for pair (v_j, v_l), its mincut value is $n + j$ and second mincut value is $n + l$. So, $\lambda_1(v_j, v_l) = \lambda_2(v_i, v_j)$ for each $i < j$. Hence, there are $j - 1$ pairs $\{(v_i, v_j) | 1 \leq i < j\}$ that are adversaries for pair (v_j, v_l). Moreover, for a fixed l, we can get $l - j$ pairs with j adversaries. Simply varying l from $n/2$ to $n - 1$ and j from $n/4$ to l, gives us $\Omega(n^2)$ pairs with $\Omega(n)$ adversaries.

Any two pairs belonging to the same bunch may have adversaries in different bunches (refer to Figure 3(iii) where pairs $(v_{\frac{n}{4}}, v_{\frac{n}{2}})$ and $(v_{\frac{n}{2}-1}, v_{\frac{n}{2}})$ have adversaries in different bunches). However, for a given pair (u, v), all its possibly $\Omega(n)$ adversaries now belong to a *single* bunch. We call this bunch the *adversarial bunch* for (u, v). Moreover, this bunch is different from $B(u, v)$. Hence, keeping $B(u, v)$ and the adversarial bunch for (u, v) in two separate sets ensures that (u, v) has no adversaries. This motivates us to use random partitioning on the bunches as follows. Partition all bunches into two sets randomly uniformly and independently. Observe that with probability $1/2$, all adversaries for a given pair are now mapped to a different set. Hence, it follows from Lemma 22 that building gen-ancestor tree on each set stores the second mincut for at least half the pairs on expectation. Using this insight, we now design our algorithm for storing all-pairs second mincuts.

4.2.1 Algorithm to build gen-ancestor trees for All-Pairs Second Mincuts

Our algorithm iteratively computes a set of gen-ancestor trees that would store a second mincut of each pair of vertices. Each iteration starts with a set P of pairs of vertices whose second mincut has not yet been computed. It computes a pair of gen-ancestor trees that store a second mincut for a subset of pairs in P. The remaining pairs of vertices in P are processed in the next iteration. The algorithm runs for $2 \log n$ iterations. The full details are provided in Algorithm 1.

Algorithm 1 Gen-ancestor trees storing all-pairs second mincuts.

procedure SECCOVER(G)
  $\mathcal{T} \leftarrow \emptyset$;
  Let P be the set of all pairs of vertices in G;
  repeat $2 \log n$ times
    Partition the set P of pairs into bunches; ▷ Based on Definition 23
    $S_1, S_2 \leftarrow \emptyset$;
    Add each bunch to S_1 or S_2 randomly uniformly and independently;
    Let Q_1 (likewise Q_2) be the set of pairs $(u, v) \in P$ s.t. $B(u, v) \in S_1$ (likewise S_2);
    $(T_j, R_j) \leftarrow$ BUILDTREE($Q_j, 2$) for each $j \in \{1, 2\}$; ▷ 2 gen-ancestor trees
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{T_1 \cup T_2\}$; $\mathcal{R} \leftarrow R_1 \cup R_2$;
    $P \leftarrow \mathcal{R}$; ▷ Executing on remaining pairs
  until
  return $\mathcal{T}$
end procedure

4.2.2 Analysis of Algorithm 1

We show that with constant probability, the $4 \log n$ trees returned by Algorithm 1 store a second mincut for all pairs of vertices as follows.

Let us analyze any iteration of the repeat loop in Algorithm 1. At the beginning of this iteration, let P be the set of pairs of vertices whose second mincut has not yet been stored. Let (u, v) be any pair of vertices in P. Without loss of generality, assume the bunch containing (u, v) in this iteration belongs to S_1. As discussed above, there can be at most one adversarial bunch for (u, v); and the probability this bunch belongs to S_1 is $1/2$ because each bunch is assigned to S_1 or S_2 uniformly and independently (refer to Statement 7 in Algorithm 1). Hence, no adversary of (u, v) is present in Q_1 with probability at least $1/2$. If so, it follows from Lemma 22 that the tree $T(Q_1, 2)$ returned by BUILDTREE($Q_1, 2$) stores a second mincut for (u, v). The algorithm thus fails to store a second mincut of (u, v) in the gen-ancestor trees during an iteration with probability at most $1/2$, independent of all the previous iterations. Hence, the probability that the algorithm fails to store a second mincut for (u, v) in first i iterations is at most $1/2^i$. In the following lemma, we use this bound to show that with probability at least $1/2$, the set P is empty at the end of $2 \log n$ iterations.

▶ **Lemma 25.** *SECCOVER(G) stores a second mincut for all pairs with probability* $\geq 1/2$.

Proof. Define $Y_{u,v}$ to be a Bernoulli random variable such that $Y_{u,v} = 1$ if no second mincut of (u, v) is stored after $2 \log n$ iterations. It follows from the discussion above that $Y_{u,v} = 1$ with probability at most $(1/2)^{2 \log n} = 1/n^2$. Define random variable Y as the number of pairs of vertices whose second mincut has not been stored after $2 \log n$ iterations. Observe that $Y = \sum_{(u,v) \in V} Y_{u,v}$. Hence, by linearity of expectation, $E[Y] \leq \frac{n(n-1)}{2} \times \frac{1}{n^2} \leq 1/2$. Therefore, using Markov Inequality, probability that $Y \geq 1$ is at most $1/2$. ◀

Observe that Algorithm 1 is Monte Carlo. However, we can easily convert it to a Las Vegas algorithm as follows. Repeat SECCOVER(G) till we get an instance where $\mathcal{R} = \emptyset$ and store the set of trees $\mathcal{T}$ for this instance. It follows from Lemma 25 that the algorithm terminates after 2 iterations on expectation. This completes the proof of Theorem 5.

4.3 $\mathcal{O}(k \log n)$ Gen-ancestor Trees for Storing All-Pairs k^{th} Mincuts

We now aim to extend our results for any $k \geq 1$. Consider any pair of vertices (u, v). It follows from Theorem 20(2) that for a set P of pairs, if gen-ancestor tree $T(P, k)$ fails to store a k^{th} mincut for $(u, v) \in P$, then the cut C stored at their LCA is a j^{th} (u, v)-mincut, for some $j < k$. Hence, by Theorem 20(1), there exists a pair $(a, b) \in P$ such that C is a k^{th} mincut for (a, b). To extend Algorithm 1 for $k > 2$, we define *adversary* for a pair as follows.

▶ **Definition 26** (Adversary). *A pair (x, y) is said to be an adversary for (u, v) if $\lambda_k(x, y) = \lambda_j(u, v)$ for some $j < k$.*

Based on Definition 26, Lemma 22 and Definition 23 can be seamlessly extended for any $k \geq 1$ as follows.

▶ **Lemma 27.** *If the gen-ancestor tree $T(P, k)$ fails to store a k^{th} mincut for a pair $(u, v) \in P$, then there exists at least one adversary for (u, v) that belongs to P.*

▶ **Definition 28** (Bunch). *Two pairs of vertices belong to the same bunch if the k^{th} mincut value of both the pairs is same. Let $B(u, v)$ denote the bunch to which a pair (u, v) belongs.*

Suppose we partition any given set P of pairs of vertices into a set of bunches based on Definition 28. Consider any pair $(u, v) \in P$. Observe that according to Definition 26 and Definition 28, the adversaries of (u, v) can belong to at most $k - 1$ different bunches.

▶ **Lemma 29.** *For any pair (u, v) of vertices, the number of adversarial bunches of (u, v) is at most $k - 1$.*

Suppose we add each of the bunches to a set S_1 or S_2 randomly uniformly and independently. Without loss of generality, assume $B(u, v) \in S_1$. We show in the following lemma that on expectation, only a constant fraction of the number of adversarial bunches of (u, v) are present in the set S_1.

▶ **Lemma 30.** *Let $\kappa(u, v)$ be the number of adversarial bunches for pair of vertices (u, v). If $B(u, v) \in S_1$, the expected number of adversarial bunches for (u, v) in S_1 is $\kappa(u, v)/2$.*

Proof. Let X be the random variable for the number of adversarial bunches of (u, v) in S_1. Recall that each bunch is added to S_1 with probability $1/2$. Therefore, using the linearity of expectation, the expected value of X is $\kappa(u, v)/2$. ◀

Lemma 30 motivates us to further partition the sets S_1, S_2 of bunches a sufficient number of times so that the expected number of adversarial bunches for a pair becomes zero. We now state the algorithm to obtain this partitioning.

4.3.1 Algorithm for Partitioning Bunches

Let P be a given set of pairs of vertices. Our algorithm computes a partitioning of the set P of pairs into $\mathcal{O}(k)$ sets.

Partition set P of pairs into bunches based on Definition 28. Let $h = \lceil \log_2(k - 1) \rceil$. Consider the numbers generated by the $h + 1$ bits: $\{0, 1, ..., 2^{h+1} - 1\}$. Assign each bunch a number from the set $\{0, 1, ..., 2^{h+1} - 1\}$ randomly uniformly and independently. This partitions the set of all bunches into $2^{h+1} = \mathcal{O}(k)$ sets: $S_1, S_2, \ldots, S_{2^{h+1}}$ such that a bunch belongs to set S_j if it is assigned the number $j - 1$. Likewise, we can partition the set of pairs P into 2^{h+1} sets $Q_1, Q_2, \ldots, Q_{2^{h+1}}$ as follows. Assign a pair $(u, v) \in P$ to Q_j if $B(u, v) \in S_j$. Refer to Algorithm 2 for the pseudocode.

Algorithm 2 Partitioning Pairs into Sets based on bunches.

procedure RandPartition(P)
  Assign each pair in P to a bunch, say B_j; ▷ Based on Definition 28
  $h \leftarrow \lceil \log_2(k-1) \rceil$;
  Assign each bunch B_j a number from $\{0, 1, \ldots, 2^{h+1}-1\}$ randomly uniformly and independently;
  Bunches are partitioned into 2^{h+1} sets: $S_1, S_2, \ldots, S_{2^{h+1}}$;
  Let Q_j be the set of pairs of vertices (u, v) such that $B(u, v) \in S_j, 1 \leq j \leq 2^{h+1}$;
  return $\{Q_1, Q_2, \ldots, Q_{2^{h+1}}\}$;
end procedure

▶ **Lemma 31.** *RandPartition(P) partitions P into sets $\{Q_1, Q_2, \ldots, Q_{2^{h+1}}\}$ such that for any pair $(u, v) \in P$, if $(u, v) \in Q_j$, with probability at least $1/2$, no adversary of $(u, v) \in Q_j$.*

Proof. It follows from the premise that bunch $B(u, v) \in S_j$. Let the number of adversarial bunches of (u, v) be $\kappa(u, v)$. Let $X_{u,v}$ be the random variable for the number of adversarial bunches of (u, v) in S_j. Since each bunch is added to S_j with probability $1/2^{h+1}$, using the linearity of expectation, $E[X_{u,v}] = \kappa(u, v) \times 1/2^{h+1}$. It follows from Lemma 29 that $\kappa(u, v) \leq k-1$, so we obtain $E[X_{u,v}] \leq \frac{k-1}{2.2^h}$. Since $h \geq \log_2(k-1)$, $E[X_{u,v}] \leq 1/2$. Thus, using Markov Inequality, probability that $X_{u,v} \geq 1$ is at most $1/2$. Since $X_{u,v}$ by definition is an integer random variable, no adversarial bunch of (u, v) belongs to S_j with probability at least $1/2$. Therefore, no adversary of (u, v) belongs to Q_j with probability at least $1/2$. ◀

4.3.2 Algorithm to build gen-ancestor trees for All-Pairs k^{th} Mincuts

We now describe our algorithm for computing a set of $\mathcal{O}(k \log n)$ gen-ancestor trees storing a k^{th} mincut for each pair. The algorithm differs from Algorithm 1 only in the partitioning of the formed bunches, stated as follows. In Algorithm 1, at each iteration, the sets of bunches are partitioned into two groups S_1, S_2. In contrast, the set of bunches are partitioned into $\mathcal{O}(k)$ sets: $\{S_1, S_2, \ldots, S_{2^{h+1}}\}$ using Algorithm 2. Refer to Algorithm 3 for the pseudocode.

Algorithm 3 Gen-ancestor trees storing all-pairs k^{th} mincuts.

procedure kthCover(G, k)
  $\mathcal{T} \leftarrow \emptyset$;
  Let P be the set of all pairs of vertices in G;
  repeat $2 \log n$ times
    $(Q_1, Q_2, \ldots, Q_{2^{h+1}}) \leftarrow$ RandPartition(P), where $h = \lceil \log_2(k-1) \rceil$;
    $\mathcal{R} \leftarrow \emptyset$;
    for each $1 \leq j \leq 2^{h+1}$ **do**
      $\{T(Q_j, k), R_j\} \leftarrow$ BuildTree(Q_j, k); ▷ $\mathcal{O}(k)$ gen-ancestor trees
      $\mathcal{T} \leftarrow \mathcal{T} \cup T(Q_j, k)$;
      $\mathcal{R} \leftarrow \mathcal{R} \cup R_j$; ▷ R_j = pairs whose k^{th} mincut is not stored in tree $T(Q_j, k)$
    end for
    $P \leftarrow \mathcal{R}$; ▷ Pairs whose k^{th} mincut is not stored in $\mathcal{T}$
  until
  return $\mathcal{T}$;
end procedure

4.3.3 Analysis of Algorithm 3

Let us analyze any iteration of the repeat loop in Algorithm 3. At the beginning of this iteration, let P be the set of pairs of vertices whose k^{th} mincut has not yet been stored. Let (u, v) be any pair of vertices in P. Without loss of generality, assume the bunch containing (u, v) in this iteration belongs to S_j for some $1 \leq j \leq 2^{h+1}$. Hence, it follows from Lemma 31 that no adversary of (u, v) is present in Q_j with probability at least $1/2$. If so, it follows from Lemma 27 that the tree $T(Q_j, k)$ returned by BuildTree(Q_j, k) stores a k^{th} mincut for (u, v). The algorithm thus fails to store a k^{th} mincut of (u, v) in the gen-ancestor trees during an iteration with probability at most $1/2$, independent of all the previous iterations. The rest of the analysis is exactly the same as the analysis of Algorithm 1 for second mincut in Section 4.2.2. Hence, with probability at least $1/2$, kthCover(G, k) returns a set of $\mathcal{O}(k \log n)$ gen-ancestor trees storing all-pairs k^{th} mincuts. Therefore, similar to Algorithm 1, Algorithm 3 can also be converted into a Las Vegas algorithm, leading to the following theorem.

▶ **Theorem 32.** *For any undirected graph on n vertices, there exists a set of $\mathcal{O}(k \log n)$ gen-ancestor trees such that for any pair of vertices (u, v), the LCA of u and v stores a k^{th} (u, v)-mincut in one of the trees.*

5 Data Structures for All-Pairs k^{th} Mincuts

In this section, we address the problem of designing an efficient data structure that, given any pair of vertices (u, v) as query, can report a k^{th} (u, v)-mincut and its value $\lambda_k(u, v)$. We design our data structures using the $\mathcal{O}(k \log n)$ gen-ancestor trees in Theorem 32 that store all-pairs k^{th} mincuts.

Observe that we can obtain a trivial data structure by explicitly storing a k^{th} mincut for each pair of vertices. This acts as a $\mathcal{O}(n^3)$ space data structure that given a pair (u, v), can report a k^{th} (u, v)-mincut C in $\mathcal{O}(|C|)$ time. For $k = \Omega(n)$, we simply store the trivial data structure. For $k = o(n)$, we now improve the space bound to $\mathcal{O}(kn^2 \log n)$ using Theorem 32. It follows from Theorem 32 that for any pair of vertices, at least one of the $\mathcal{O}(k \log n)$ gen-ancestor trees stores its k^{th} mincut. Hence, we associate each pair (x, y) with the first gen-ancestor tree that stores a k^{th} (x, y)-mincut at their LCA as follows.

Augmenting Trees and storing a Mapping. Let $T(Q_1, k), T(Q_2, k), \ldots, T(Q_\psi, k)$ be the set of $\psi = \mathcal{O}(k \log n)$ gen-ancestor trees in Theorem 32. Iterating i from 1 to ψ, if for any pair (x, y), tree $T(Q_i, k)$ stores a k^{th} mincut for (x, y), we map (x, y) to the tree $T(Q_i, k)$. Our data structure consists of the following two components. (1) The $\psi = \mathcal{O}(k \log n)$ gen-ancestor trees in Theorem 32, each augmented with a LCA data structure [11]. (2) The mapping from the set of all pairs of vertices to these ψ trees. It follows from Theorem 20 that the data structure occupies $\mathcal{O}(n^2 + kn^2 \log n)$ space (as $|V(Q_i)| = \mathcal{O}(n)$) and leads us to the following theorem.

▶ **Theorem 33.** *Given any positive integer k and an undirected graph on n vertices, there exists an $\mathcal{O}(kn^2 \log n)$ space data structure that given any pair of vertices (u, v) can report a k^{th} (u, v)-mincut C in $\mathcal{O}(|C|)$ time.*

The data structure described above (Theorem 33) can report the value of k^{th} (u, v)-mincut $\lambda_k(u, v)$ in $\mathcal{O}(1)$ time. However, the space occupied by the data structure is even worse than the trivial bound of $\mathcal{O}(n^2)$ space obtained by explicitly storing $\lambda_k(x, y)$ for each pair (x, y) of

vertices. Hence, we now aim to improve the bound on space for reporting the value $\lambda_k(u,v)$. To achieve our aim, we store only the value of the cut (instead of the cut) at each internal node in all the $\psi = \mathcal{O}(k \log n)$ gen-ancestor trees $T(Q_1,k), T(Q_2,k), \ldots, T(Q_\psi,k)$, stated in Theorem 32. We name this data structure as $\mathcal{S}_k$. It follows that this data structure requires $\mathcal{O}(kn \log n)$ space. Observe that $\mathcal{S}_k$ allows us to retrieve the potentially $\mathcal{O}(k \log n)$ values stored at the LCA of u and v in each tree (where u, v belong to different leaf nodes), in $\mathcal{O}(k \log n)$ time. It follows from Theorem 32 that one of these values is $\lambda_k(u,v)$. As explained below, the main challenge lies in retrieving $\lambda_k(u,v)$ from these $\mathcal{O}(k \log n)$ values without storing the mapping from all pairs to the set of ψ gen-ancestor trees.

Let us consider any tree $T(Q_i,k)$ in which u and v are mapped to different leaf nodes. It follows from Theorem 20(2) that if pair $(u,v) \in Q_i$, the tree $T(Q_i,k)$ stores $\lambda_j(u,v)$ for some $j \leq k$. However, if $(u,v) \notin Q_i$, it follows from Lemma 18 that the tree $T(Q_i,k)$ may store any arbitrary (u,v)-cut. Hence, the $\mathcal{O}(k \log n)$ values might not necessarily contain $\lambda_j(u,v)$ for each $j \leq k$. So, if one considers the k smallest distinct values among the $\mathcal{O}(k \log n)$ values, although one of these values is $\lambda_k(u,v)$, the rest of the values might be smaller or greater than $\lambda_k(u,v)$. Hence, simply taking the k^{th} minimum value from the $\mathcal{O}(k \log n)$ values might not give us $\lambda_k(u,v)$. Observe that given $\lambda_{k-1}(u,v)$, one can retrieve $\lambda_k(u,v)$ from the $\mathcal{O}(k \log n)$ values in $\mathcal{O}(k \log n)$ time as follows. Since $\lambda_k(u,v)$ belongs to the $\mathcal{O}(k \log n)$ values, report the smallest value that is strictly greater than $\lambda_{k-1}(u,v)$. However, $\lambda_{k-1}(u,v)$ might not belong to the $\mathcal{O}(k \log n)$ values. Hence, we build our data structure recursively as follows.

Data Structure for reporting $\lambda_k(u,v)$. Let $\mathcal{D}_j$ denote the data structure that can report $\lambda_j(u,v)$ for any given pair of vertices (u,v) and integer j. We initialize $\mathcal{D}_1$ as the ancestor tree (Theorem 2 or gen-ancestor tree for $k=1$) built on all pairs of vertices. $\mathcal{D}_2$ stores the structure $\mathcal{S}_2$ and $\mathcal{D}_1$. Similarly, $\mathcal{D}_k$ stores $\mathcal{S}_k$ and $\mathcal{D}_j$ for each $1 \leq j \leq k-1$. This leads to the following theorem and completes the proof for Theorem 6.

▶ **Theorem 34.** *Given any positive integer k and an undirected graph on n vertices,*

1. *there exists an $\mathcal{O}(k^2 n \log n)$ space data structure that, given any pair (u,v) and an integer $j \leq k$, can report the value $\lambda_j(u,v)$ in $\mathcal{O}(k^2 \log n)$ time.*
2. *there exists an $\mathcal{O}(k^2 n^2 \log n)$ space data structure that, given any pair (u,v) and an integer $j \leq k$, can report a j^{th} (u,v)-mincut C in $\mathcal{O}(|C|)$ time.*

Preprocessing Time. Consider the well-known case $F = c$. Vazirani and Yannakakis [24] show that for any designated pair of vertices s,t, a k^{th} (s,t)-mincut can be computed in $\mathcal{O}(n^{2k-2})$ (s,t)-maxflow computations. Hence, the data structure in Theorem 34 can be computed using $\mathcal{O}(n^2 \sum_{i=1}^{k} n^{2(i-1)}) = \mathcal{O}(n^2 \times n^{2(k-1)}) = \mathcal{O}(n^{2k})$ (s,t)-maxflow computations. Using the best-known algorithm [13] for computing (s,t)-maxflow, the preprocessing time is $n^{2k} m^{1+o(1)}$. Using the recent result of [8] (for $k=2$), the data structure in Theorem 6 can be built in $n^{2.5} m^{1+o(1)}$ time.

6 Conclusion

We initiate the research on all-pairs k^{th} mincuts in undirected graphs with arbitrary cut function F, and present optimal combinatorial bounds and near optimal structures. Similar to the ancestor tree [15], one can use our compact structures in directed graphs as well by defining the function F appropriately [9, 7]. Cheng and Hu [15] designed an algorithm for computing all-pairs mincuts that performs $\mathcal{O}(n)$ (s,t)-mincut computations. It is now interesting to see whether a non-trivial algorithm for computing all-pairs k^{th} mincuts can also be designed.

References

1 Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 884–895. IEEE, 2022. doi:10.1109/FOCS54457.2022.00088.

2 Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 48–61. SIAM, 2020. doi:10.1137/1.9781611975994.4.

3 Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster gomory-hu trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3630–3649. SIAM, 2022. doi:10.1137/1.9781611977073.143.

4 Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. All-pairs max-flow is no harder than single-pair max-flow: Gomory-hu trees in almost-linear time. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2204–2212. IEEE, 2023. doi:10.1109/FOCS57990.2023.00137.

5 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

6 Giorgio Ausiello, Paolo Giulio Franciosa, Isabella Lari, and Andrea Ribichini. Max flow vitality in general and st-planar graphs. *Networks*, 74(1):70–78, 2019. doi:10.1002/net.21878.

7 Surender Baswana and Koustav Bhanja. Vital edges for (s, t)-mincut: Efficient algorithms, compact structures, & optimal sensitivity oracles. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, pages 17–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.17.

8 Surender Baswana, Koustav Bhanja, and Anupam Roy. Faster Algorithm for Second (s,t)-Mincut and Breaking Quadratic Barrier for Dual Edge Sensitivity for (s,t)-Mincut. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *33rd Annual European Symposium on Algorithms (ESA 2025)*, volume 351 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2025.68.

9 András A. Benczúr. Counterexamples for directed and node capacitated cut-trees. *SIAM J. Comput.*, 24(3):505–510, 1995. doi:10.1137/S0097539792236730.

10 András A. Benczúr. A representation of cuts within 6/5 times the edge connectivity with applications. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 92–102. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492466.

11 Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005. doi:10.1016/J.JALGOR.2005.08.001.

12 Koustav Bhanja. Optimal Sensitivity Oracle for Steiner Mincut. In Julián Mestre and Anthony Wirth, editors, *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, volume 322 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2024.10.

13 Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514, 2023. doi:10.1109/FOCS57990.2023.00037.

14 Chung-Kuan Cheng, Ronald Graham, Ilgweon Kang, Dongwon Park, and Xinyuan Wang. Tree structures and algorithms for physical design. In *Proceedings of the 2018 International Symposium on Physical Design*, pages 120–125, 2018. doi:10.1145/3177540.3177564.

15 Chung-Kuan Cheng and T. C. Hu. Ancestor tree for arbitrary multi-terminal cut functions. *Ann. Oper. Res.*, 33(3):199–213, 1991. doi:10.1007/BF02115755.

16 Yefim Dinitz and Zeev Nutov. A 2-level cactus model for the system of minimum and minimum+ 1 edge-cuts in a graph and its incremental maintenance. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 509–518, 1995. doi:10.1145/225058.225268.

17 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.

18 Naveen Garg and Vijay V. Vazirani. A polyhedron with all s - t cuts as vertices, and adjacency of cuts. *Math. Program.*, 70:17–25, 1995. doi:10.1007/BF01585926.

19 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

20 Yoko Kamidoi, Noriyoshi Yoshida, and Hiroshi Nagamochi. A deterministic algorithm for finding all minimum k-way cuts. *SIAM Journal on Computing*, 36(5):1329–1341, 2007. doi:10.1137/050631616.

21 David R Karger. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In *Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing*, pages 11–17, 1995. doi:10.1145/225058.225069.

22 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, January 2000. doi:10.1145/331605.331608.

23 Hiroshi Nagamochi, Kazuhiro Nishimura, and Toshihide Ibaraki. Computing all small cuts in an undirected network. *SIAM Journal on Discrete Mathematics*, 10(3):469–481, 1997. doi:10.1137/S0895480194271323.

24 Vijay V Vazirani and Mihalis Yannakakis. Suboptimal cuts: Their enumeration, weight and number. In *International Colloquium on Automata, Languages, and Programming*, pages 366–377. Springer, 1992.

A Tight Bound on Distinct Values of All-Pairs Second Mincuts

As shown in Lemma 16, T_2^1 stores the mincut values for all pairs of vertices. This is interesting as T_2^1 might not be an MST for G_1 and hence not satisfy Theorem 1. This along with the fact that T_2 stores all-pairs second mincuts values (Theorem 3) implies that the distinct values of mincuts as well as second mincuts for all pairs of vertices can be at most $2(n-1)$. We now show that this upper bound can be improved to $2n-3$.

To show the redundancy among the $2n-2$ edge capacities in T_2 and T_2^1, we explore the redundancies among the mincuts and second mincuts of all the edges in T_2. We first reveal how a mincut and a second mincut of an edge in T_2 appear in the tree.

▶ **Lemma 35.** *Let (x,y) be any edge in T_2. Either a mincut or a second mincut of (x,y) separates the endpoints of at least one more edge in T_2.*

Proof. Let $C^1_{x,y}$ be any (x,y)-mincut. Observe that there can be two cases (1) $C^1_{x,y}$ separates the endpoints of only (x,y) in T_2 or (2) $C^1_{x,y}$ separates the endpoints of at least one more edge in T_2. However, if (1) is true, any second (x,y)-mincut $C^2_{x,y}$ must separate the endpoints of at least one more edge in T_2, since $C^2_{x,y} \neq C^1_{x,y}$. This completes the proof. ◀

Now, consider any edge (x,y) in T_2. Lemma 35 allows us to apply Lemma 11 and obtain an upper bound on the capacity of another edge, say (a,b), in T_2 or T_1^2. To match this upper bound, we simply consider (x,y) to be the least capacity edge in T_2 and arrive at the following interesting property.

▶ **Lemma 36.** *Among the $2n-2$ edges in T_2 and T_2^1, at least two edges have identical capacities.*

Proof. Let (x,y) be the edge in T_2 whose capacity is the least among all the edges in T_2. By Lemma 35, either a second mincut $C^2_{x,y}$ or a mincut $C^1_{x,y}$ for (x,y) separates another edge, say (a,b) in T_2. Suppose $C^2_{x,y}$ separates edge (a,b) in T_2. It follows from Lemma 11 that either (1) $C^2_{x,y}$ is a mincut for (a,b) or (2) $\lambda_2(a,b) \leq F(C^2_{x,y})$. Moreover, since (x,y) is the least capacity edge in T_2, it follows from the construction of T_2 that $\lambda_2(x,y) \leq \lambda_2(a,b)$. Therefore, $\lambda_2(x,y) = \lambda_1(a,b)$ or $\lambda_2(x,y) = \lambda_2(a,b)$.

Suppose $C^1_{x,y}$ separates edge (a,b) in T_2. It follows from Lemma 11 that $\lambda_1(a,b) = \lambda_1(x,y)$ or $\lambda_2(a,b) \leq \lambda_1(x,y)$. Since (x,y) is the edge with least capacity in T_2, $\lambda_1(x,y) < \lambda_2(x,y) \leq \lambda_2(a,b)$. Therefore, $\lambda_1(a,b) = \lambda_1(x,y)$ and the two edges (x,y) and (a,b) have the same capacity in T_1. ◀

It follows from the Lemma 36 that there are at most $2n-3$ distinct values in T_2 and T_2^1, for all-pairs mincuts and second mincuts. We now show that the bound is tight as well. Refer to the line graph in Figure 4. Observe that (1) the capacity of each edge is distinct and (2) the sum of capacities of any two edges is never equal to the capacity of a single edge. The mincut for each pair (v_i, v_{i+1}), where $1 \leq i \leq n-1$, is the cut defined by the edge (v_i, v_{i+1}). The second mincut for each pair (v_i, v_{i+1}), where $2 \leq i \leq n-1$, is defined by the pair of edges (v_i, v_{i+1}) and (v_1, v_2). Therefore, we get at least $2n-3$ distinct values for all-pairs mincuts and second mincuts.

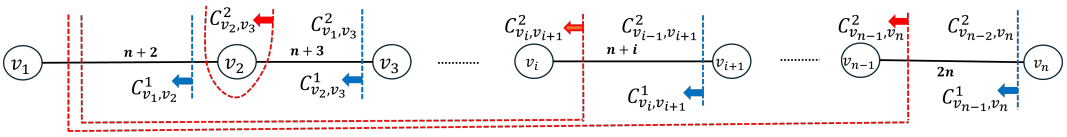

Figure 4 A blue dotted line denotes a (v_i, v_{i+1})-mincut for $1 \leq i \leq n-1$, and a second mincut for (v_i, v_{i+2}) for $1 \leq i \leq n-2$. A red dotted line denotes a second mincut for (v_i, v_{i+1}), for $2 \leq i \leq n-1$.

Now, we focus on providing a tight bound on distinct values of all-pairs second mincuts. By Theorem 3(2), there exists an edge (x,y) in T_2^1 whose mincut value is equal to the global mincut value. Hence, one of the $2n-3$ values is the global mincut value. However, any global mincut cannot be a second mincut for any pair by definition. Therefore, there are at most $2n-4$ distinct values of second mincuts for all pairs. Figure 4 provides a matching lower bound for the same. As shown above, there are at least $n-2$ pairs with distinct second mincut values which are not equal to any mincut value. Now, observe that for any pair (v_i, v_{i+2}), where $1 \leq i \leq n-2$, its second mincut is equal to the mincut for (v_{i+1}, v_{i+2}). Hence, there are precisely $2(n-2) = 2n-4$ distinct values of all-pairs second mincuts. This leads to the following theorem.

▶ **Theorem 37.** *For each $n \geq 3$, there exist graphs on n vertices such that the distinct values of second mincuts for all pairs is at least $2n-4$.*

B At Least $2n-3$ Second Mincuts Required for All Pairs

Consider the graph in Figure 5. The vertices in the graph are $v_1, v_2, \ldots, v_n$. There are a set of $2(n-1)$ edges in the graph: edges (v_i, v_{i+1}) of capacity $n+i+1$ for $1 \leq i \leq n-1$, edges (v_i, v_n) of capacity 1 for $2 \leq i \leq n-1$, and edge (v_1, v_n) of capacity $n+2$. Observe that the following facts hold for each i, where $2 \leq i \leq n-1$. Any (v_i, v_{i+1})-cut must separate

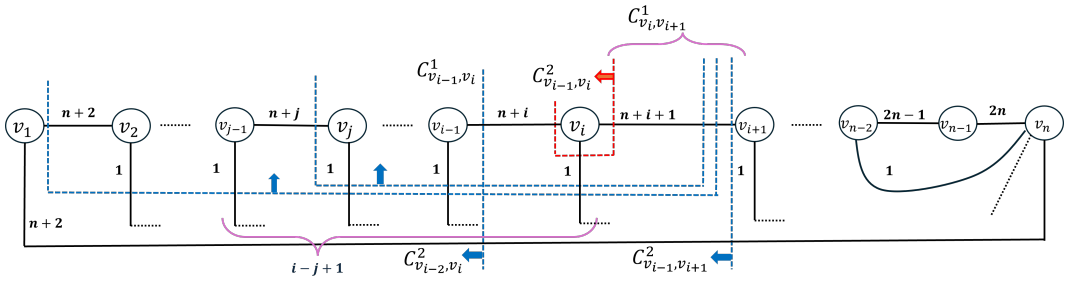

Figure 5 A graph in which every second mincut cover has size $2n-3$.

the edge (v_i, v_{i+1}) and at least one other edge in the cycle $O = \langle v_1, v_2, \ldots, v_n, v_1 \rangle$. The cut $\{v_i\}$, shown in red color in Figure 5, is a (v_i, v_{i+1})-cut. Observe that $F(\{v_i\}) = 2n + 2i + 2$. For each $j < i$, the cut $\{v_j, \ldots, v_i\}$, shown in blue color in Figure 5, is also a (v_i, v_{i+1})-cut, and $F(\{v_j, \ldots, v_i\}) = F(\{v_i\})$. Let us now consider any other (v_i, v_{i+1})-cut. Observe that such a cut must separate (v_i, v_{i+1}) and (1) exactly one other edge (v_j, v_{j+1}) in O, where $j > i$ or (2) at least three other edges in O. Cuts of type (1) have value at least $c(v_i, v_{i+1}) + c(c_j, v_{j+1}) > 2n + 2i + 2$ since $c(c_j, v_{j+1}) > c(v_i, v_{i+1})$ for each $j > i$. The cuts of type (2) have value strictly greater than $4n$, whereas $2n + 2i + 2 \leq 4n$ for any value of i. Hence, the set of cuts: $\{v_i\}$ and $\{v_j, \ldots, v_i\}$, for each $j < i$, are the only mincuts for pair (v_i, v_{i+1}).

Now, consider the pairs (v_{i-1}, v_i), for $2 \leq i \leq n-1$. Observe that $\lambda_1(v_i, v_{i+1}) = \lambda_1(v_{i-1}, v_i) + 2$. Moreover, $\{v_i\}$, which is a (v_i, v_{i+1})-mincut, is also a (v_{i-1}, v_i)-cut. As the reader may verify, there is no (v_{i-1}, v_i)-cut of capacity $\lambda_1(v_{i-1}, v_i) + 1$. Hence, $\lambda_2(v_{i-1}, v_i) = \lambda_1(v_i, v_{i+1})$. However, note that the set of cuts $\{v_j, \ldots, v_i\}, \forall j < i$ are not (v_{i-1}, v_i)-cuts. Hence, $\{v_i\}$ is the only second (v_{i-1}, v_i)-mincut. Now, consider the pairs (v_{i-1}, v_{i+1}) for $2 \leq i \leq n-1$. Every (v_{i-1}, v_i)-mincut is a (v_{i-1}, v_{i+1})-mincut, and every (v_i, v_{i+1})-mincut except $\{v_i\}$ is a (v_{i-1}, v_{i+1})-cut. Hence, similar to above lines, it can be shown that $\lambda_2(v_{i-1}, v_{i+1}) = \lambda_1(v_i, v_{i+1})$. So, we can conclude with the following lemma.

▶ **Lemma 38.** *For $2 \leq i \leq n-1$, the following assertions hold for the graph in Figure 5.*

1. $\lambda_2(v_{i-1}, v_i) = \lambda_2(v_{i-1}, v_{i+1}) = \lambda_1(v_i, v_{i+1}) = 2n + 2i + 2$.
2. *$\{v_i\}$ is the second mincut for (v_{i-1}, v_i).*
3. *For each $j < i$, $\{v_j, \ldots, v_i\}$ is a second mincut for (v_{i-1}, v_{i+1}).*

We define *second mincut cover* for a set P of pairs of vertices as a set of cuts, such that for any pair $(u, v) \in P$, there exists a second (u, v)-mincut that belongs to the set.

▶ **Lemma 39.** *For any $n \geq 4$, there exists a graph such that the following lower bounds hold.*

1. *Let P_1 be a set of pairs whose second mincut is a mincut for some pair. The size of second mincut cover for P_1 is at least $2n-4$.*
2. *The size of second mincut cover for all pairs is at least $2n-3$.*

Proof. For each i, where $2 \leq i \leq n-1$, it follows from Lemma 38(1) that there exist $2(n-2)$ pairs whose second mincut is mincut for some pair. It follows from Lemma 38(2) that any second mincut cover for the pairs (v_{i-1}, v_i) must contain $\{v_i\}$. While it follows from Lemma 38(3) that any second mincut cover for the pairs (v_{i-1}, v_{i+1}) must contain a cut $\{v_j, \ldots, v_i\}$, for least one value of j where $j < i$. Note that the sets $\{v_j, \ldots, v_i\}$ are unique for each value of i. This completes the proof of assertion (1).

Now, observe that for the pair (v_{n-1}, v_n), the mincut value is the largest. So, any second mincut for pair (v_{n-1}, v_n) cannot be a mincut for any pair and hence is different from the $2(n-2)$ cuts mentioned above. Therefore, at least $2n-3$ cuts are required to store a second mincut for all pairs. This completes the proof of (2). ◀

C Lower Bound on Distinct Values of All-Pairs k^{th} Mincuts

In this section, we give two lower bounds. We first show that there exists a graph H on n vertices such that the distinct values of all pairs j^{th} mincuts, for each $j \leq k$, is $\Omega(kn)$. Using the graph H itself, we show that the distinct values of all pairs k^{th} mincuts is $\Omega(\min\{kn, n^2\})$.

To obtain a lower bound of $\Omega(kn)$ on the distinct values of all-pairs j^{th} mincuts, for all $j \leq k$, we ensure the following two properties of H for each i, where $1 \leq i \leq n'$ and $n' = \Omega(n)$. (1) A j^{th} mincut exists for pair (v_i, v_{i+1}). (2) The value $\lambda_j(v_i, v_{i+1})$ is unique for each pair of integers (i, j), where $j \leq k$. However, as stated in Lemma 15(1), the distinct values of all-pairs k^{th} mincuts can still be $\mathcal{O}(n)$ if they do not coincide with j^{th} mincut value of any pair. Hence, to show a lower bound of $\Omega(kn)$ on all-pairs k^{th} mincuts, we ensure that k^{th} mincut of n' pairs is equal to j^{th} mincut of some pair, for each $j < k$.

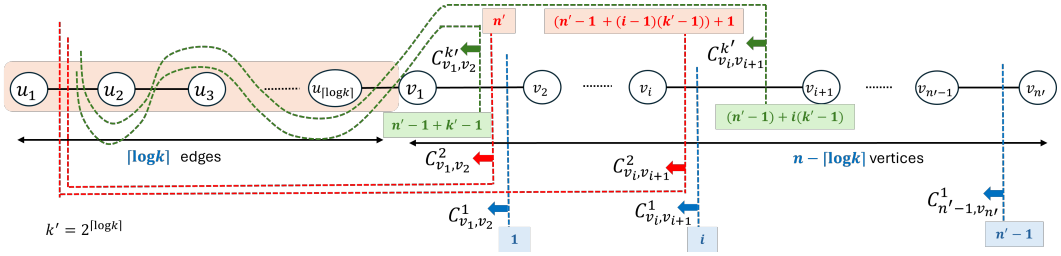

Figure 6 A blue dotted line denotes a mincut, red denotes a second mincut, green denotes a k'^{th} mincut for a pair (v_i, v_{i+1}), where $1 \leq i \leq n'-1$.

Construction of graph H. We now give the construction of a line graph H on n vertices. Refer to Figure 6. Let $n' = n - \lceil \log k \rceil$. The vertices of H are $u_1, u_2, \ldots, u_{\lceil \log k \rceil}, v_1, v_2, \ldots, v_{n'}$. We now define the cut function F on all cuts of H as follows. A cut C is said to be defined by a set of edges E' if C separates only the edges in E',e.g., $C^1_{v_1,v_2}$ is the cut defined by edge (v_1, v_2) in Figure 6. Consider a cut defined by the edge (v_i, v_{i+1}), where $1 \leq i \leq n'-1$, and an unique combination of the first $\lceil \log k \rceil$ edges: $(u_1, u_2), (u_2, u_3), \ldots, (u_{\lceil \log k \rceil}, v_1)$. Let $k' = 2^{\lceil \log k \rceil}$. There can be $k' = \Omega(k)$ such cuts defined for the edge (v_i, v_{i+1}) and hence, $\Omega(kn')$ total cuts. Assign a distinct value to these $k'(n'-1)$ cuts as stated in the following lines. For the remaining $(2^{n-1}-1) - k'(n'-1)$ cuts, simply assign a value larger than these $k'(n'-1)$ values.

We assign values to the cuts in two stages. *First Stage:* Assign values to the $n'-1$ cuts defined by each edge (v_i, v_{i+1}) in increasing order of i, from $i = 1$ to $n'-1$. Refer to Figure 6. The cuts are denoted by blue dotted lines. Assign value 1 to the cut $C^1_{v_1,v_2}$ defined by edge (v_1, v_2), value i to the cut $C^1_{v_i,v_{i+1}}$ defined by edge (v_i, v_{i+1}) and likewise value $n'-1$ to the cut $C^1_{v_{n'-1},v_{n'}}$ defined by edge $(v_{n'-1}, v_{n'})$. In the next stage, we assign greater values starting from n' to the remaining $(k'-1)(n'-1)$ cuts. Observe that this immediately ensures the following two properties in H for each i, where $1 \leq i \leq n'-1$.

▶ **Property 1.** *The unique (v_i, v_{i+1})-mincut is the cut defined by the edge (v_i, v_{i+1}).*

▶ **Property 2.** *For each $l \leq j \leq \min\{k, n'-i\}$, the l^{th} (v_i, v_{i+j})-mincut is the (v_{i+l-1}, v_{i+l})-mincut, that is, $\lambda_l(v_i, v_{i+j}) = \lambda_1(v_{i+l-1}, v_{i+l})$.*

Second Stage: There are exactly $k'-1$ non-empty subsets of the $\lceil \log k \rceil$ edges. Let us enumerate these sets of edges, say $U_1, U_2, \ldots, U_{k'-1}$, such as $U_1 = (u_1, u_2)$, $U_{k'-1} = \{(u_1, u_2), (u_2, u_3), \ldots, (u_{\lceil \log k \rceil}, v_1)\}$. We proceed as follows for each value of i in increasing order, from $i = 1$ to $n'-1$. For any i, assign values to all the $k'-1$ (v_i, v_{i+1})-cuts defined by the edge (v_i, v_{i+1}) and U_l, in increasing order from $l = 1$ to $k'-1$. However, unlike the first stage, we assign values to these $k'-1$ (v_i, v_{i+1})-cuts, only after assigning values to all the $k'-1$ (v_{i-1}, v_i)-cuts. Refer to the following lines for better understanding.

For $i = 1$, proceed as follows. Assign value $(n'-1)+1$ to the (v_1, v_2)-cut defined by edges (v_1, v_2) and $U_1 = \{(u_1, u_2)\}$ (the red (v_1, v_2)-cut $C^2_{v_1,v_2}$ in Figure 6). Then, assign value $(n'-1)+l$ to the (v_1, v_2)-cut defined by edges (v_1, v_2) and U_l for $1 \leq l \leq k'-1$. Then proceed for $i = 2$ and in a similar way for $1 \leq i \leq n'-1$ as follows. Refer to Figure 6. The cut $C^{l+1}_{v_i,v_{i+1}}$ denotes the (v_i, v_{i+1})-cut defined by edge (v_i, v_{i+1}) and set of edges U_l. Assign value $(n'-1+(i-1)(k'-1))+l$ to the (v_i, v_{i+1})-cut $C^{l+1}_{v_i,v_{i+1}}$ for $1 \leq l \leq k'-1$. In this way, we have assigned distinct values to all the $k'(n'-1)$ cuts. This second stage of the construction of H along with Property 1 ensure the following.

▶ **Property 3.** *For each value of i, $1 \leq i \leq n'-1$, the j^{th} (v_i, v_{i+1})-mincut is the cut defined by the edge (v_i, v_{i+1}) and a unique subset of the first $\lceil \log k \rceil$ edges, for each $j \leq k$.*

▶ **Property 4.** *For any $i' < i$, $\lambda_j(v_{i'}, v_{i'+1}) < \lambda_l(v_i, v_{i+1})$ for each value of j, l, where $1 \leq j, l \leq k$.*

We now establish some assertions for H which will help in achieving our aim. Since we assign distinct values to each of the $k'n' = \Omega(k'n)$ cuts, the following lemma is immediate from Property 3 and Property 4.

▶ **Lemma 40.** *For each pair of vertices (v_i, v_{i+1}), $\lambda_l(v_i, v_{i+1})$ is unique for each value of i and l, where $l \leq k$ and $1 \leq i \leq n'-1$.*

▶ **Lemma 41.** *The following properties hold for any vertex v_i, where $1 \leq i \leq n'-1$.*
1. *If $j = \min\{k, n'-i\}$, $\lambda_k(v_i, v_{i+j}) = \lambda_1(v_{i+j-1}, v_{i+j})$.*
2. *If $j < \min\{k, n'-i\}$, $\lambda_k(v_i, v_{i+j}) = \lambda_{k-j+1}(v_i, v_{i+1})$.*

Proof. Let us locate the k^{th} mincut of pair (v_i, v_{i+j}) for a fixed i and all $j \leq \min\{k, n'-i\}$. It follows from the construction of H that each (v_i, v_{i+j})-cut of finite value is a l^{th} mincut for consecutive pair of vertices $(v_{i'}, v_{i'+1})$, for some i', l where $i \leq i' \leq j-1$ and $l \leq k'$. It follows from Property 2 that the first j distinct valued (v_i, v_{i+j})-cuts are mincuts for each of the consecutive pair of vertices $(v_{i'}, v_{i'+1})$, where $i \leq i' \leq j-1$. Hence, if $j = \min\{k, n'-i\}$, then $\lambda_k(v_i, v_{i+j}) = \lambda_1(v_{i+k-1}, v_{i+k})$. This completes the proof of (1).

Now, we provide the proof of assertion (2). Let us consider the case when $j < \min\{k, n'-i\}$. It follows from the above discussion that we now have to locate only the remaining $(k-j)$ distinct valued cuts for pair (v_i, v_{i+j}), that is, $\lambda_l(v_i, v_{i+j})$, for $j+1 \leq l \leq k$. It follows from Property 4 that these $(k-j)$ (v_i, v_{i+j})-cuts are among the first $k-j+1$ cuts for pair (v_i, v_{i+1}). Since, we have already established above using Property 2 that $\lambda_1(v_i, v_{i+j}) = \lambda_1(v_i, v_{i+1})$, we have $\lambda_l(v_i, v_{i+j}) = \lambda_{l'}(v_i, v_{i+1})$, for $j+1 \leq l \leq k$ and $2 \leq l' \leq k-j+1$ in the following way. $\lambda_{j+1}(v_i, v_{i+j}) = \lambda_2(v_i, v_{i+1})$, $\lambda_{j+2}(v_i, v_{i+j}) = \lambda_3(v_i, v_{i+1}), \ldots, \lambda_{j+(k-j)}(v_i, v_{i+j}) = \lambda_{(k-j)+1}(v_i, v_{i+1})$. This completes the proof of assertion (2). ◀

We now show a lower bound on distinct values of all-pairs j^{th} mincuts in H, for all $j \leq k$.

▶ **Lemma 42.** *The distinct values of all-pairs j^{th} mincuts, for each $j \leq k$ in H is $\Omega(kn)$.*

Proof. It follows from Lemma 40 that the set of k cuts for each consecutive pair (v_i, v_{i+1}) have unique values for all $1 \leq i \leq n' - 1$. So, at least $n' - 1$ pairs have unique j^{th} mincut values, for each $j \leq k$. This provides us a lower bound of $(n'-1)k = (n-1)k - k\lceil \log k \rceil \geq (n-1)k - (n-1)k/2 = \Omega(kn)$. ◀

This matches the upper bound of $\mathcal{O}(kn)$ provided by Theorem 15 on the distinct values of all-pairs j^{th} mincuts, for all $j \leq k$. Note that to establish Lemma 42, we only consider the consecutive pairs (v_i, v_{i+1}), for each $1 \leq i \leq n'$. However, if we consider the k^{th} mincut values for only these pairs, then there are only $n' - 1$ distinct values. So, the main challenge now lies in considering all the $\binom{n'}{2}$ pairs carefully and establishing the following result.

▶ **Lemma 43.** *The distinct values of all-pairs k^{th} mincuts in graph H is $\Omega(kn)$ for $1 \leq k < n$, and $\Omega(n^2)$ for $n \leq k \leq 2^{nc}$, where $0 \leq c < 1$ is any constant.*

Proof. Let us first consider the case when $k < n'$. Consider a pair (v_i, v_{i+j}) for any $j \leq k$ and a fixed i. It follows from Lemma 40 and Lemma 41 that $\lambda_k(v_i, v_{i+j})$ is unique for each value of i and j. Hence, if $i \leq n' - k$, we vary j from 1 to k to get k distinct values for each i. Whereas if $i > n' - k$, we vary j from 1 to $n' - i$ to get $n' - i$ distinct values. Therefore, the total number of distinct values for all-pairs k^{th} mincuts in H is at least

$$\sum_{i=1}^{n'-k} k + \sum_{i=n'-k+1}^{n'-1} (n'-i) = k(n'-k) + \sum_{i=k-1}^{1} i = k(n'-k) + (k-1)k/2$$
$$= kn' - k(k+1)/2 \geq kn'/2 = \Omega(kn') \quad \text{since } k < n' \tag{1}$$

Now, consider the case when $k > n'$. Observe that here the subcase $i \leq n' - k$ (stated above) does not arise since $i \geq 1$. Hence, consider only the subcase $i > n' - k$. Therefore for all $i \geq 1$, the number of distinct values of all-pairs k^{th} mincuts is simply $\sum_{i=1}^{n'-1}(n'-i) = \sum_{i=n'-1}^{1} i = n'(n'-1)/2 = \Omega(n'^2)$.

For $k < n$, $n' = n - \lceil \log k \rceil \geq n - \lceil \log n \rceil \geq n/2$. Putting $n' = \Omega(n)$ in Equation 1 provides a lower bound of $\Omega(kn)$. Moreover, for $k \leq 2^{nc}$ and $0 \leq c < 1$, $n' = n - \lceil \log k \rceil$, $n' \geq n(1-c) = \Omega(n)$. Therefore, by putting the value of n', we get a lower bound of $\Omega(n^2)$. ◀

Lemma 42 and Lemma 43 together lead to the following theorem.

▶ **Theorem 44.** *For any $k \geq 1$ and $n \geq 2\lceil \log k \rceil$, there exist an undirected graph on n vertices such that the distinct values of all-pairs j^{th} mincuts, for each $j \leq k$, is $\Omega(kn)$, and the distinct values of all-pairs k^{th} mincuts is $\Omega(\min\{kn, n^2\})$.*