Maximal k-Edge-Connected Subgraphs in Weighted Graphs via Local Random Contraction

Chaitanya Nalam[*]   Thatchaphol Saranurak[†]

February 7, 2023

Abstract

The *maximal k-edge-connected subgraphs* problem is a classical graph clustering problem studied since the 70's. Surprisingly, no non-trivial technique for this problem in weighted graphs is known: a very straightforward recursive-mincut algorithm with $\Omega(mn)$ time has remained the fastest algorithm until now. All previous progress gives a speed-up only when the graph is unweighted, and k is small enough (e.g. Henzinger et al. (ICALP'15), Chechik et al. (SODA'17), and Forster et al. (SODA'20)).

We give the first algorithm that breaks through the long-standing $\tilde{O}(mn)$-time barrier in *weighted undirected* graphs. More specifically, we show a maximal k-edge-connected subgraphs algorithm that takes only $\tilde{O}(m \cdot \min\{m^{3/4}, n^{4/5}\})$ time. As an immediate application, we can $(1 + \epsilon)$-approximate the *strength* of all edges in undirected graphs in the same running time.

Our key technique is the first local cut algorithm with *exact* cut-value guarantees whose running time depends only on the output size. All previous local cut algorithms either have running time depending on the cut value of the output, which can be arbitrarily slow in weighted graphs or have approximate cut guarantees.

---

[*]University of Michigan, Ann Arbor. Email: `nalamsai@umich.edu`
[†]University of Michigan, Ann Arbor. Email: `thsa@umich.edu`

Contents

1 Introduction

We revisit a natural graph clustering problem of efficiently computing *maximal k-edge-connected subgraphs* that has been studied since the 70's [Tar72]. Given a (directed) graph $G = (V, E)$, we say that G is *k-edge-connected* if all the edge cuts have value at least k and, similarly, G is *k-vertex-connected* if all vertex cuts have value at least k. The *maximal k-edge-connected subgraphs* of G contains all subsets $\{V_1, \ldots, V_z\}$ of V such that, for each i, the induced subgraph $G[V_i]$ is k-edge-connected and there is no strict superset $V_i' \supset V_i$ where $G[V_i']$ is k-edge-connected. The *maximal k-vertex-connected subgraphs* of G is defined analogously.

It turns out that all the variants of these problems (whether the graph is weighted/unweighted or directed/undirected) admit a very simple *recursive-mincut* algorithm: For the k-edge-connectivity version, compute an edge cut (A, B) of value less than k by calling a global minimum cut subroutine and return $\{V\}$ if no such cut exists. Otherwise, recurse on both $G[A]$ and $G[B]$ and return the union of answers of the two recursions. For the k-vertex-connectivity version, similarly compute a vertex cut (A, S, B) of value less than k (where S separates A from B, i.e., $E(A, B) = \emptyset$). If such cut exists, recurse on both $G[A \cup S]$ and $G[B \cup S]$. The worst-case running time of this algorithm on a graph with n vertices and m edges is $\Theta(n \cdot T_{\mathrm{mincut}}) = \Omega(mn)$ where $T_{\mathrm{mincut}} = \Omega(m)$ is the time for computing minimum cuts.[1][2]

Surprisingly, the above straightforward algorithm has remained the fastest algorithm! Although the problem has been studied since the 70's, all known algorithms give a speed-up only when the graph is unweighted and k is small enough. The classical depth-first-search algorithm [Tar72] solves both maximal 2-edge-connected and 2-vertex-connected subgraphs problems in undirected unweighted graphs in $O(m)$ time. There was an effort to generalize this result to directed graphs [ES80, Mak88, Jab16], but the algorithms still take $\Omega(mn)$ time. Henzinger et al. [HKL15] later obtained $\tilde{O}(n^2)$-time algorithms in directed graphs when $k = 2$. Then, Chechik et al. [CHI+17] gave the first algorithm that breaks the $O(mn)$ bound for any constant k. Their algorithm takes $\tilde{O}(m\sqrt{m}k^{O(k)})$ time for all variants of the problems. They also gave an improved bound of $\tilde{O}(m\sqrt{n}k^{O(k)})$ for computing maximal k-edge-connected subgraphs in undirected graphs. Both bounds were then improved by Forster et al. [FNY+20] to $\tilde{O}(m\sqrt{m}k^{3/2})$ and $\tilde{O}(mk + n^{3/2}k^3)$, respectively. Nonetheless, in weighted graphs where k can be arbitrarily large or even in unweighted graphs when $k \geq m^{1/3}/n^{1/6}$, the simple recursive-mincut algorithm remains the fastest algorithm until now.

Our Results. We give the first algorithm that breaks through the long-standing $\tilde{O}(mn)$-time barrier of the recursive-mincut algorithm in *weighted* graphs, in the case of maximal k-edge-connected subgraphs in *undirected* graphs, which is the variant of the problem that receives attention in practice and several heuristics were developed [ZLY+12, AIY13]. Formally, our main result is as follows:

Theorem 1.1. *There is a randomized algorithm that, given a weighted undirected graph $G = (V, E, w)$ where $w \in \mathbb{Z}_{\geq 0}^E$ with n vertices and m edges and any parameter $k > 0$, computes the maximal k-edge-connected subgraphs of G w.h.p. in $O(m \cdot \min\{m^{3/4} \log^{3.75} n, n^{4/5} \log^{3.6} n\})$ time.*

---

[1]Karger showed that [Kar00] it takes $T_{\mathrm{mincut}} = \tilde{O}(m)$ to compute minimum edge cuts in undirected graphs, but the start-of-the-art for other variants are worse. For minimum edge cuts in directed graphs, $T_{\mathrm{mincut}} = \widehat{O}(m \cdot \min\{n/m^{1/3}, \sqrt{n}\})$ [CLN+22] and $T_{\mathrm{mincut}} = \tilde{O}(mk)$ if the graph is unweighted [Gab95]. For minimum vertex cuts, $T_{\mathrm{mincut}} = \tilde{O}(mn)$ [HRG00] in both undirected and directed graphs and, if the graph is unweighted, we have $T_{\mathrm{mincut}} = \widehat{O}(m)$ for undirected graphs [LNP+21] and $T_{\mathrm{mincut}} = \min\{\widehat{O}(n^2), \tilde{O}(mk^2)\}$ for directed graphs [LNP+21, FNY+20].

[2]Throughout the paper, we use $\tilde{O}(\cdot)$ to hide polylog(n) factor and use $\widehat{O}(\cdot)$ to hide $n^{o(1)}$ factor.

It is known that the solution $\{V_1, \ldots, V_z\}$, the maximal k-edge-connected subgraphs of G, is unique and forms a partition of V. Moreover, it precisely determines whether an edge has *strength* at least k: each edge $(u, v) \in E$ has strength at least k iff both u and v are inside the same subset $V_i \in \{V_1, \ldots, V_z\}$. The edge strength introduced by Benczur and Karger [BK02] is a central notion for graph sparsification and its generalization [FHHP19, CKN20, CCPS21]. [BK02] gave a near-linear algorithm that underestimates the edge strength. Although their estimates are sufficient for graph sparsification [BK02, CX18, FHHP19], these estimates do not give an approximation for edge-strength of *every* edge; the estimate of some edge can deviate from its strength by even a $\Omega(n^{1/3})$ factor. See Appendix D for the discussion.

The fastest known algorithm for computing exact edge strengths is again the recursive-mincut algorithm which requires $\tilde{O}(mn)$ time. No faster algorithm is known, even when a large approximation ratio is allowed. In contrast, Theorem 1.1 improves the state-of-the-art for approximating edge strength by giving an $(1 + \varepsilon)$-approximation.

Corollary 1.2. *There is a randomized algorithm that, given a weighted undirected graph, can w.h.p. $(1 + \epsilon)$-approximates the strength of all edges in $\tilde{O}(m \cdot \min\{m^{3/4}, n^{4/5}\}/\epsilon)$ time.*

Main Technical Contribution: Local Exact Algorithms in Weighted Graphs. The critical tool behind Theorem 1.1 is the first *local cut algorithm* that has an exact cut-value guarantee and yet is fast in weighted graphs.

We generally refer to local cut algorithms as the algorithms that, given a seed vertex x, find a cut S containing x with desirable properties, but only spend time close to the size of S and independent of the size of the whole graph. For example, an influential line of work on local cut algorithms for finding sparse cuts (e.g. [ST13, ACL06, OZ14, HRW20]) found wide-range applications, including solving linear systems [ST14] and dynamic graph algorithms [NSWN17, SW19]. Unfortunately, all these algorithms only have *approximate* guarantees.[3] There is another class of algorithms initiated in the area of property testing that checks whether there is a small cut S containing the seed x whose cut size is at most k. These algorithms have running time $\exp(k)$ [YI10, OR11, YI12] and can be improved to $\text{poly}(k)$ [GR97, PR02, FNY+20, NSY19]. Although these algorithms found interesting applications beyond property testing [FNY+20, CDK+21, JHSZ22], they are not useful in weighted graphs because k can be arbitrarily large.

To summarize, all local cut algorithms in the literature are either approximate or run in time proportional to the cut value k. Our key technique is an algorithm that bypasses both these drawbacks.

To state our key tool, we need some notations. From now, all graphs $G = (V, E, w)$ we consider are undirected. For any edge set $F \subseteq E$, $w(F) = \sum_{e \in F} w(e)$ is the total weight of F. For any set $X \subset V$, we write $\overline{X} = V \setminus X$.

Let $\delta(X) = E(X, \overline{X})$ denote the *cut set* of X and $w(\delta(X))$ is denote the *cut value* of X. The (unweighted) *volume* of X denoted by $\text{vol}(X) = |E(X, V)|$ counts all edges incident to X. We say that X is an *extreme set* if, for every strict subset $Y \subset X$, $w(\delta(Y)) > w(\delta(X))$. Equivalently, X is the unique minimum cut in $G/\overline{X}$, the graph after contracting $\overline{X}$ into a single vertex.

Definition 1.3. *For any vertex $x \in V$, we say X is a (x, ν, σ, k)-set if $x \in X$, $\text{vol}(X) < \nu$, $|X| < \sigma$, and $w(\delta(X)) < k$. If X is also an extreme set, then X is called an (x, ν, σ, k)-extreme set.*

Intuitively, an (x, ν, σ, k)-set is a "small local cut" containing the *seed* vertex x that we want to find. From now on we use ν to represent the volume of the small local cut and σ to represent

---

[3]If there is a cut S with conductance ϕ, these algorithms find a cut S' with conductance ϕ' where $\phi' > \phi$.

the number of vertices on the smaller side of the local cut. The key problem we consider is the following:

Problem 1.4 (LOCALKCUT). *Given a graph $G = (V, E, w)$, a vertex x, and parameters ν, σ and k, either find an (x, ν, σ, k)-set or return $\perp$ indicating that no (x, ν, σ, k)-extreme set exists.*

Note that when there is no (x, ν, σ, k)-extreme set but there exists an (x, ν, σ, k)-set, then we allow returning either $\perp$ or any (x, ν, σ, k)-set. Now, our key technical contribution is as follows:

Theorem 1.5. *There are randomized algorithms with the following guarantees with high probability:*

1. *Given access to the adjacency lists G, Problem 1.4 can be solved in time $O(\nu\sigma^2 \log^2 n)$.*

2. *After $O(m\sigma^2 \log^2 n)$ preprocessing time on G, Problem 1.4 can be solved in time $O(\sigma^4 \log n)$.*

The most important guarantee of Theorem 1.5 is that its running time is independent from k. Theorem 1.5 has an exact cut-size guarantee in the sense that there is no gap in k: it either returns a (x, ν, σ, k)-**set** or reports that no (x, ν, σ, k)-extreme set exists. It is possible to strengthen the guarantee of Theorem 1.5 to "either return an (x, ν, σ, k)-**extreme set** or report that no (x, ν, σ, k)-extreme set exists". We show such an algorithm in Appendix B. However, this algorithm is slower and so we will not use it for proving Theorem 1.1. Hence, we defer it to the appendix.

We complement this algorithm with conditional lower bounds:

Theorem 1.6. *Given the parameters x, ν, σ, k, checking whether an (x, ν, σ, k)-set exists is W[1]-hard even when parameterized by σ. Moreover, assuming the OMv conjecture [HKNS15], there is no algorithm that, given access to the adjacency lists and matrix of G (with no further preprocessing), solves Problem 1.4 in $O(\sigma^{2-\Omega(1)})$ time.*

The first statement in Theorem 1.6 notes that the notion of extreme sets in Problem 1.4 is crucial for efficient algorithm; if we just want to check if an (x, ν, σ, k)-set exists, then it is unlikely that there exists an algorithm with $f(\sigma)n^{O(1)}$ time where f is an arbitrary function (see the definition of W[1]-hardness from Section 13.3 of [CFK+15]).

The second statement of Theorem 1.6 shows that the running time of Theorem 1.5 cannot be improved beyond $O(\sigma^{2-o(1)})$ if we only have access to the adjacency lists and matrix. Improving the running time of Theorem 1.5 to $\tilde{O}(\sigma^2)$, or even $\tilde{O}(\sigma)$ with non-trivial pre-processing, would speed up the running time of Theorem 1.1 to $\tilde{O}(mn^{2/3})$ and $\tilde{O}(mn^{1/2})$, respectively. We leave this as a very interesting open problem.

Key Technique: Local Random Contraction. The high-level message behind Theorem 1.5 is as follows: the famous *random contraction* technique by [Kar93] for computing global minimum cut can be localized into local cut algorithms for detecting unbalanced extreme sets in *weighted graphs*.

Applications of the random contraction technique to local cut algorithms appeared earlier in the property testing algorithms by Goldreich and Ron [GR97] and also in [PR02]. However, it is unclear a priori that their results extend to weighted graphs. This is because in setting of [GR97, PR02] they have that (1) the graph is unweighted and has bounded degree, (2) the target cut X they want to find is such that $|\delta(X)| < k$ and $|X| = O(k)$, and (3) their stated running time was poly(k). So the time bound inherently may depend on k. Some parts of the argument in [GR97] crucially exploit that the graph is unweighted and does not carry on to our weighted setting. Our observation is that the "local random contraction" technique still works in weighted graphs with the running time bound independent from k.

This shows an exciting aspect of the random contraction technique because, to our best knowledge, there is an inherent dependency on k in all other techniques in the current literature for local cut algorithms with exact cut-value guarantees. These techniques include local-search-based algorithms with deterministic branching strategies [YI10, OR11, CHI$^+$17], local-search-based algorithm with randomized stopping points [FNY$^+$20], and even the localized version of the Goldberg-Rao flow algorithm [OZ14, NSY19].

We view Theorem 1.5 as the first step toward stronger local cut algorithms in weighted graphs with exact cut-value guarantees. Because of the strong guarantee of these algorithms, we expect many more applications. For example, strengthening Theorem 1.5 to work in directed graphs would immediately generalize our main result, Theorem 1.1, to work in directed graphs.

Related Problems.   The related problem of computing k-*edge-connected components* of a graph $G = (V, E)$ is to find the unique partition $\{V_1', \ldots, V_{z'}'\}$ of V such that, for every vertex pair (s, t) in the same part V_i', the (s, t)-minimum cut in G (not in $G[V_i']$) is at least k. The maximal k-edge-connected subgraphs $\{V_1, \ldots, V_z\}$ can be different from the k-edge-connected components $\{V_1', \ldots, V_z'\}$. Consider for example an unweighted undirected graph G' where vertices s and t are connected through three parallel paths (s, u_1, t), (s, u_2, t), and (s, u_3, t). While s and t are in the same 3-edge-connected component of G', no two vertices are in the same maximal 3-edge-connected subgraphs of G'.

We note that computing k-edge-connected components in an undirected graph G can be reduced to computing all-pairs max flow in G, which can be done in $\tilde{O}(n^2)$ time using the recent algorithm of [AKL$^+$21]. This algorithm, however, does not solve nor imply anything to our problem of computing maximal k-edge-connected subgraphs.

Organization.   After giving a high-level technical overview in Section 2, we start by formally describing the recursive mincut algorithm for finding the maximal k-edge connected subgraphs in Section 3 as a warm-up for similar proofs. In Section 4, we show a local cut algorithm for Theorem 1.5(1). We then use this local cut algorithm to obtain the maximal k-edge-connected subgraphs in $\tilde{O}(mm^{3/4})$ time in Section 5. This proves the first running time claimed in Theorem 1.1. In Section 6, we give another local cut algorithm stated in Theorem 1.5(2) which is faster than the one in Section 4 when the volume of the cut is high. Similar to Section 5, in Section 7, we apply this local algorithm to obtain the maximal k-edge-connected subgraphs in $\tilde{O}(mn^{4/5})$ time, completing the proof of Theorem 1.1.

In Section 8 we show hardness for solving Problem 1.4, proving the Theorem 1.6. In Section 9, we show an application of our main result to approximating edge strength, i.e., Corollary 1.2.

2   Overview of techniques

We say that $X \subset V$ is a (x, ν, k)-set if X is a (x, ν, σ, k)-set except that we do not require that $|X| < \sigma$. Similarly, X is a (x, σ, k)-set if we do not require that $\text{vol}(X) < \nu$.

The Framework.   Our algorithm is based on the framework of [CHI$^+$17], so we start by explaining their ideas. Their first crucial observation is that the recursive-mincut algorithm can take $\Omega(mn)$ time only when the recursion depth is $\Omega(n)$. This, in turn, happens when many cuts S of value less than k found are highly unbalanced, say $|S| = O(1)$. The main technical message of [CHI$^+$17] is that, given a sub-linear time algorithm for finding such unbalanced cuts S, one can improve the $O(mn)$ running time.

More precisely, their reduction shows that, given a vertex x, if we can check whether there exists a (x, ν, k)-set in $O(\nu)$ time, then the maximal k-edge-connected subgraphs can be computed in $O(m\sqrt{m})$ time. In fact, their reduction works even if we slightly relax the algorithm guarantee to "either report that (x, ν, k)-sets do not exists or return a $(x, O(\nu k), k)$-set". Chechik et al. [CHI$^+$17] solves this relaxed problem in $O(\nu k^{O(k)})$ time and Forster et al. [FNY$^+$20] improved it to $\tilde{O}(\nu k^2)$ time, which implies their final algorithms with running time of $\tilde{O}(m^{3/2}k^{O(k)})$ and $\tilde{O}(m^{3/2}k^{3/2})$, respectively.[4] Unfortunately, these algorithms are too slow when k is large.

We first observe that the reduction framework [CHI$^+$17] can be adjusted so that it is compatible with our local algorithm of Theorem 1.5 whose running time is independent from k. More precisely, although the relaxed guarantee of "either report that (x, ν, k)-sets do not exists or return a $(x, O(\nu k), k)$-set" is not compatible with Problem 1.4, we show that local algorithms for Problem 1.4 can still be applied to break the $O(mn)$ barrier. The basic idea why algorithms for Problem 1.4 is sufficient follows from the fact that if a graph is not k-edge-connected, then either there must exist an (x, ν, k)-extreme set (not just an (x, ν, k)-set) for some vertex x, otherwise all cuts of size less than k has volume at least ν, i.e., it is quite balanced, which is a good case for us (see Lemma 5.2 for the formal proof).

Since the framework of [CHI$^+$17] can be made compatible with local algorithms for Problem 1.4, we now simply apply our local algorithms with running time $\tilde{O}(\nu\sigma^2) = \tilde{O}(\nu^3)$ from Theorem 1.5(1) or $\tilde{O}(\sigma^4)$ from Theorem 1.5(2) and obtain our final algorithm with running time $\tilde{O}(mm^{3/4})$ or $\tilde{O}(mn^{4/5})$, respectively. This concludes Theorem 1.1.

Local Cut Algorithms. First, we recall the equivalent view of the random contraction technique based on minimum spanning trees (as observed in Karger's original paper [Kar93]). Assume for a moment that the input graph $G = (V, E)$ is unweighted.

Define a random rank function $r : E \to [0, 1]$ where the rank $r(e)$ of each edge e is an independent uniform random number in $[0, 1]$. Let MST_r be the minimum spanning tree w.r.t. rank function r.

Lemma 2.1 ([Kar93]). *For any minimum cut $(S, \overline{S})$ in G, with probability $\Omega(1/n^2)$,*

$$r(e') > r(e)$$

for all $e' \in E(S, \overline{S})$ and $e \in \mathrm{MST}_r \setminus E(S, \overline{S})$.[5]

Next, we sketch how to prove Theorem 1.5. Suppose that there is a (x, ν, σ, k)-extreme set S in G. Our goal is to return some (x, ν, σ, k)-set. Note that S is the unique minimum cut of the graph $G/\overline{S}$ by definition of an extreme set. By simply applying Lemma 2.1 to the graph $G/\overline{S}$, which contains at most σ vertices, we have the following:

Lemma 2.2. *With probability $\Omega(1/\sigma^2)$,*

$$r(e') > r(e) \tag{1}$$

for all $e' \in E(S, \overline{S})$ and $e \in G[S] \cap \mathrm{MST}_r \setminus E(S, \overline{S})$.

---

[4]Their improved running time in undirected graphs used techniques and arguments that are specific to unweighted graphs. So these techniques do not work in our setting.

[5]For readers familiar with Karger's analysis, the ranks of edges correspond to the ordering of edge contraction. This condition is equivalent to saying that after contracting until there are only two vertices left, the unique remaining cut in the contracted graph is exactly the minimum cut $(S, \overline{S})$.

We say that the rank function r is *nice* with respect to a set S or S *respects* r when this condition holds. We will generate $\tilde{O}(\sigma^2)$ many independent rank functions so that one of them is nice with high probability by Lemma 2.2. Now, suppose that r is nice and suppose we run Prim's minimum spanning tree algorithm starting from the vertex x to grow a "partial" MST $T_r \subseteq \text{MST}_r$. We know from Eq (1) that T_r must span S first before crossing the cut $(S, \overline{S})$. At that point $V(T_r) = S$ is a (x, ν, σ, k)-set and we can return $V(T_r)$.

Let $X = V(T_r)$. Our algorithm is to simply return X whenever X is a (x, ν, σ, k)-set. We can check this by maintaining $\text{vol}(X)$, $|X|$ and $\delta(X)$. If r is not nice, then X might not ever be a (x, ν, σ, k)-set but we will just terminate whenever X is too big, i.e., $\text{vol}(X) \geq \nu$ or $|X| \geq \sigma$.

It is simple to maintain $\text{vol}(X)$, $|X|$ and $\delta(X)$ in $\tilde{O}(\nu)$ time given the access to the adjacency lists of G. We can also maintain this information in $\tilde{O}(\sigma^2)$ time by using some simple pre-processing on G (see Section 6 for details). Since there are $\tilde{O}(\sigma^2)$ many rank functions, the total running time becomes $\tilde{O}(\nu\sigma^2)$ and $\tilde{O}(\sigma^4)$, respectively. This is how we obtain Theorem 1.5 when the graph is unweighted.

Lastly, we remove the assumption that the graph is unweighted. We observe that only the correctness of Lemma 2.1 and Lemma 2.2 is based on the fact that the graph is unweighted. In weighted graphs, the rank function needed to be defined differently. Given a weighted graph $G = (V, E, w)$, if we treated G as an unweighted multigraph and generated an independent random rank in $[0, 1]$ for each edge, then Lemma 2.1 would work, but this simulation is too slow. What we need is as follows: if an edge e has weight $w(e)$, then we should define the rank $r(e)$ to be precisely the minimum of $w(e)$ independent random numbers in $[0, 1]$. We derive this formula explicitly (see Claim 4.6) and show that it can be computed efficiently and also numerically stable (see Appendix A), which is the only place we need that the edge weights are integral. This allows us to work in weighted graphs efficiently. This concludes the sketch of Theorem 1.5.

Conditional Lower Bounds. For the first statement of Theorem 1.6, we show that the *t-clique* problem can be reduced to the strengthened version of Problem 1.4, i.e., the problem of checking whether there exists a (x, ν, σ, k)-set for a given seed $x \in V$ even when σ only depends on t. See Section 8.3 for details. The similar reduction was shown in [FGK13] but in their reduction $\sigma = \Omega(n)$ is quite big, since they work in unweighted graphs.

Next, we explain the proof of the second statement of Theorem 1.6 based on the OMv conjecture [HKNS15]. In [HKNS15] they show that, assuming the conjecture, given a graph $G = (V, E)$ with n vertices and $\text{poly}(n)$ preprocessing time on G, then given a query $S \subset V$, there is no algorithm that checks whether S is an independent set in time $n^{2-\Omega(1)}$. We will show how to use a subroutine for Problem 1.4 to check if S is independent. Given S, imagine we artificially add a cycle C_S spanning S where every edge $e \in C_S$ has infinite weight. Let G_S be the resulting graph. Let $\sigma = |S| + 1$ and $k = \text{vol}(S)$. Fix an arbitrary vertex $x \in S$. If S is not independent, then S is an (x, σ, k)-extreme set in G_S because $|S| < \sigma$, $\delta(S) < \text{vol}(S) = k$, and any strict subset $S' \subset S$ has infinite cut value. On the other hand, if S is independent, then there is no (x, σ, k)-set in G_S. Suppose for contradiction that there is a (x, σ, k)-set T. We have that T cannot cross S, otherwise, T has an infinite cut value. Since $x \in T$, so $S \subseteq T$. But $|T| \leq \sigma - 1 = |S|$, so $T = S$. Now, we have $\delta(T) = \delta(S) = \text{vol}(S) = k$ because S is independent, which is contradiction. Lastly, note that we can easily simulate the adjacency list and matrix of G_S given those of G with just an additive $O(\sigma)$ overhead for any query set S which gets subsumed in $O(\sigma^{2-\varepsilon})$ time. Therefore, given the subroutine for Problem 1.4 with access to adjacency list and matrix of G with running time $\sigma^{2-\Omega(1)}$, we can then check if S is independent in $n^{2-\Omega(1)}$, refuting the OMv conjecture.

3 Preliminaries

We refer to the paragraph above the statement of Problem 1.4 for basic notations. Given a weighted undirected graph $G = (V, E, w)$ where $w \in \mathbb{Z}_{\geq 0}^E$, G is k-edge-connected if global minimum cut value, $\min_{\emptyset \neq S \subset V} w(\delta(S)) \geq k$. We give the proof of the following simple observation in the Appendix C.

Proposition 3.1. *The maximal k-edge-connected subgraphs $\{V_1, \ldots, V_z\}$ of any graph G are unique and form a partition of V.*

By Proposition 3.1, we will usually call our desired solution $\{V_1, \ldots, V_z\}$ as *maximal k-edge-connected partition* to emphasize that it forms a partition.

3.1 The Recursive-Mincut algorithm

In this subsection, we present a naive algorithm (see e.g. the appendix of [CKN20]) to find the maximal k-edge-connected partition of the graph as it helps the reader to familiarize with the ideas involved in our improved algorithm and serves as a warmup for the proofs that follow.

Suppose we have a cut in the graph G that is less than k, then any maximal vertex set k-edge-connected should be contained on one side of the cut. Otherwise, we can separate the graph induced on that vertex set with edges of weight at most k; thus, it is not k-edge-connected and thus not maximal k-edge-connected.

Hence, a straightforward algorithm finds cuts of size less than k if they exist and recur on both the graphs induced by the cut. If no cut of size less than k exists, then the graph is k-edge-connected, and we add the remaining vertices to the partition, and we are done.

Algorithm 1: RECMINCUT (G,k)

Data: $G = (V, E, w), k$
Result: Maximal k-edge-connected partition of vertex set V
1 Compute minimum cut $(S, V \setminus S)$ of graph G;
2 **if** $w(S, V \setminus S) \geq k$ **then**
3 $\quad$ return $\{V\}$
4 **else**
5 $\quad$ return RECMINCUT$(G[S], k) \cup$ RECMINCUT$(G[V \setminus S], k)$

Lemma 3.2. *The output of Algorithm 1 is the maximal k-edge-connected partition of G.*

Proof. Let $\mathcal{P}$ be the output of the algorithm. Observe that the sets output by Algorithm 1 form a partition of the vertex set V as no vertex is repeated and every vertex is included as the recursion would eventually bottom out at least with singleton vertex, because each vertex is k-edge-connected trivially.

Now, we argue that the subgraph induced on each set $S \in \mathcal{P}$ is a maximal k-edge-connected subgraph. Indeed, $G[S]$ k-edge-connected by how we stop the recursion. Next, we show the maximality of S. Assume for contradiction a set $T \supset S$ such that $G[T]$ is also k-edge-connected. At the start of the algorithm, set T is contained in V. Let V' be the last set during the algorithm's run on G that contains T. So the minimum cut of $G[V']$, of size less than k, separates the set T into two parts. Hence, $G[T]$ also has a cut of size less than k, which contradicts our assumption that $G[T]$ is k-edge-connected. Therefore, we conclude that $G[S]$ is a maximal k-edge-connected subgraph. $\qquad\square$

Lemma 3.3. *Algorithm 1 takes at most $\tilde{O}(mn)$ time.*

Proof. The recursion depth of Algorithm 1 is clearly at most n because each recursion reduces the number of vertices in the largest subgraph by at least one.

We use the Karger's algorithm from [Kar00] that takes a near-linear running time $O(m \log^3 n)$ to compute a minimum cut in the graph with m edges and n vertices. Since all the subgraphs at a particular recursion depth are disjoint and have at most m edges in total. Let G_i be one of the subgraph with m_i edges and n_i vertices. Hence, the total time to compute minimum cut on all the subgraphs is $\sum_i O(m_i \log^3 n_i) \leq \sum_i O(m_i \log^3 n) \leq \tilde{O}(m)$. So, the total running time over all the recursion levels is at most $\tilde{O}(mn)$. $\qquad\square$

4   Local algorithm for small volume cuts

The goal of this section is to prove Theorem 1.5 (1). We organize this section as follows. We introduce the key subroutine, LOCALPRIM of our local algorithm in Section 4.1. Given a *nice* rank function r with respect to an extreme set S (recall Lemma 2.2), LOCALPRIM would find an (x, ν, k)-set. In Section 4.2 we introduce the idea of random rank function and prove Lemma 2.2 using the idea of random contraction from [Kar93]. From Lemma 2.2 we know that a random rank function is nice with probability $\Omega(1/\sigma^2)$. In Section 4.3, we give our local algorithm which repeats the LOCALPRIM sub-routine with multiple random rank functions to amplify the probability of finding a *nice* rank function, which then solves the Problem 1.4 with high probability and prove Theorem 1.5 (1). Here, we restate the first part of the Theorem 1.5.

Theorem 4.1. *There exists a randomized algorithm that solves Problem 1.4 with high probability in time $O(\nu \sigma^2 \log^2 n)$ given the adjacency lists of the graph.*

4.1   Local Prim Algorithm

In this sub-section, we introduce the key sub-routine required for our local algorithm. To this end, we need to introduce some definitions. A *rank function* $r : E \to [0, 1]$ is a function that maps each edge to a number in $[0, 1]$. We call $r(e)$ the *rank of edge e*. Let $\mathrm{MST}_r(G)$ denote the minimum spanning tree of G with respect to the rank function r.

As we are always interested in the value of the rank $r(e)$ in comparision to other edges rather than the absolute rank of an edge, in the rest of the paper we call the rank value as the rank in short.

Definition 4.2 (Set respects rank function)**.** *Given a graph $G = (V, E, w)$ and a rank function $r : E \to [0, 1]$, a set $S \subset V$ respects the rank function r if $\mathrm{MST}_r(G) \cap E(S, \overline{S}) = e'$ and $r(e') > r(e)$ for all $e \in \mathrm{MST}_r(G) \cap G[S]$.*

In words, the set S respects r if the MST_r has only one edge e' in intersection with the cut $\delta(S)$, which makes $\mathrm{MST}_r \cap G[S]$ a minimum spanning tree of $G[S]$, and each edge of such minimum spanning tree contained in $G[S]$ should have rank lesser than the rank of edge e'. Recall the discussion below Lemma 2.2. We say a rank function r is *nice* w.r.t. a set S iff S respects r.

This definition is very critical to the correctness of our algorithm. Note that the minimum spanning tree used for analysis is with respect to rank function r. It can be very different from the one computed using the weight function w.

We consider the following problem.

Problem 4.3 (LOCALMST)**.** *Given a graph $G = (V, E, w)$, a vertex x, a rank function r and parameters ν, σ, k find a (x, ν, σ, k)-set S that respects the rank function r or return $\perp$ if no such (x, ν, σ, k)-set respecting r exists.*

10

We state the Local Prim algorithm that solves the Problem 4.3 and proves its correctness.

Algorithm 2: LOCALPRIM

Data: $G = (V, E, w), x, \nu, \sigma, k, r$ where r is a rank function $r : E \to [0, 1]$
Result: (x, ν, σ, k)-set respecting r if exists else $\perp$

1 $X = \{x\}$;
2 **while** $w(\delta(X)) \geq k$ *and* $\mathrm{vol}(X) < \nu$ *and* $|X| < \sigma$ **do**
3   Find the edge with minimum rank, $e = (u, v) \in E(X, V \setminus X) = \delta(X)$;
4   Update $X = X \cup \{v\}$;
5 **if** $w(\delta(X)) < k$ **then**
6   **return** X
7 **else**
8   **return** $\perp$

We expand the set X starting from the singleton vertex cut $\{x\}$. Since we only want the sets that respect the rank function, we consider the cut edge (u, v) with minimum rank where $u \in X$ and $v \notin X$, and expand X by setting $X \leftarrow X \cup \{v\}$. We expand the set X along the edges with minimum rank until we violate volume or cardinality constraints of the set X or find a set with a cut size less than k. If there exists a (x, ν, σ, k)-set S that respects the rank function r, then Algorithm 2 finds the minimal set such that no strict subset of it also respects the rank function r. We prove it in the following lemma.

Lemma 4.4. *If there exists a (x, ν, σ, k)-set S in $G = (V, E, w)$ that respects a rank function r and is minimal such that no strict subset respects r, then Algorithm 2 returns S and return $\perp$ only if no such respecting set S exists.*

Proof. Let S' be the set returned by the Algorithm 2. We claim S' cannot contain a vertex not in S. Since we start from $X = \{x\} \subset S$ and so, to get a vertex not in S, we need to expand set X along an edge from $\delta(S)$.

Let X be the current set before expanding along an edge from $\delta(S)$. If $X \subsetneq S$ then there exists an edge in $\delta(X)$ that is part of the minimum spanning tree of $G[S]$ that has smaller rank than any edge in $\delta(S)$ as S respects r. So $X = S$, before we can contract an edge from $\delta(S)$, but if $X = S$, $w(\delta(X)) = w(\delta(S)) < k$ as S is a (x, ν, σ, k)-set so LOCALPRIM terminates without expanding along an edge from $\delta(S)$ and returns S. So S' cannot contain a vertex outside S and thus $S' \subseteq S$. Since any subset of S is also (ν, σ)-set and according to the algorithm X always contains x. So we only terminate when cut size is less than k and thus any set returned is a (x, ν, σ, k)-set.

It remains to prove that LOCALPRIM cannot return $\perp$. Assume it returns $\perp$. We only return $\perp$ when we cannot find any (x, ν, σ, k)-set. However, since S respects r, expanding along minimum ranked edges is nothing but applying Prim's algorithm starting from x. According to the set respecting rank function definition, we expand along all the edges of the minimum spanning tree in $G[S]$ without expanding the set X along any edge from $\delta(S)$ as they all have a higher rank, making $X = S$. Thus LOCALPRIM either returns S as it exits the while loop or terminates early if it finds a subset that also respects r but cannot return $\perp$. $\qquad\square$

We now bound the running time of the LOCALPRIM algorithm.

Lemma 4.5. LOCALPRIM *subroutine takes at most $O(\nu \log \nu)$ time.*

Proof. Here, we describe how to implement LOCALPRIM. We create a dictionary D with an invariant that $D = E(X, V \setminus X)$ contains all the cut edges crossing X. Given this dictionary, we can

obtain a minimum rank edge $e = (u, v) \in E(X, V \setminus X)$ for Line 3 of Algorithm 2. This look-up operation occurs at most $|X| < \sigma$ times as we terminate whenever $|X| \geq \sigma$.

Whenever we expand $X \leftarrow X \cup \{v\}$, we update the dictionary by scanning through all edges of v in $O(1 + \deg(v))$ time. We delete from D all edges between v and X and insert into D all edges between v and $V \setminus X$. We also update the cut value of X while iterating over edges of v by subtracting the weight of the edges that are already incident to X and adding the weight of the edges that are not incident to X previously.

The total number of insertion and deletion operations is $O(\sum_{v \in X}(1 + \deg(v))) = O(\sigma + \nu)$ since we terminate whenever $\mathrm{vol}(X) \geq \nu$ or $|X| > \sigma$. Since each operation of the dictionary takes logarithmic time, the total time is at most $O((\sigma + \nu) \log \nu) = O(\nu \log \nu)$.

$\square$

According to above Lemma 4.4 given there is a (ν, σ, k)-set S in graph G. If we find a rank function r such that S respects r, then we can find a (ν, σ, k)-set by running LOCALPRIM with input as any vertex $x \in S$ using this rank function r in $O(\nu \log \nu)$ time.

The following sub-section will show how we construct the rank functions that help us find (ν, σ, k)-sets.

4.2 *Nice* Rank Function

In this subsection, we show how to construct nice rank functions introduced above and explain how it is related to the idea of using the randomized contraction technique of [Kar93] to find minimum cuts.

Constructing a rank function respected by any particular set S is easy. However, we do not know which subset is a (ν, σ, k)-extreme set, so we cannot explicitly construct a rank function that is respected by an unknown extreme set. We come up with a distribution R over rank functions and prove that any rank function chosen from this distribution is *nice* with good probability. The key idea used in coming up with the distribution R is the idea of random contraction.

To understand the construction of R we need to look at the process of Karger's random contraction from a different perspective. We begin with the process of how a random edge is sampled in Karger's mincut algorithm. It can be equivalently viewed as stated in the Lemma 6.2.1 of [KS96].

Claim 4.6 (Lemma 6.2.1 of [KS96])**.** *If we choose a rank function that maps each edge e to $r(e) = 1 - (1 - t)^{1/w(e)}$ computed by uniformly sampling a value $t \in [0, 1]$ where $w(e)$ is the weight of the edge, then the probability of an edge e having the minimum rank among all edges equals $\frac{w(e)}{\sum_{e \in E} w(e)}$.*

For completeness, we give the proof of Claim 4.6 with respect to our formalism and defer the proof to Appendix A. The rest of this section shows how a random rank function is a nice rank function with good probability.

From Claim 4.6, we know that choosing an edge that has the minimum rank value $(r(e))$ among all the edges simulates the process of picking an edge proportional to its weight. So Karger's random contraction [Kar93] algorithm is nothing but picking an edge with minimum rank $r(e)$ and contracting it. This process is the same as applying Kruskal's algorithm on the graph $G = (V, E, w)$ with r as the weight function. So, the set of all edges contracted during the running of Karger's random contraction process form MST_r of the graph G.

In the following lemma, we prove that any (x, ν, σ, k)-extreme set respects a random rank function described above with reasonable probability.

Remark 4.7. *Any (x, ν, σ, k)-set does not respect a random rank function with reasonable probability, and we need more structure for a set to respect a random rank function.*

We define formally the distribution of the rank functions we sample from. For each edge $e \in E$ of the graph $G = (V, E, w)$ we define a random variable R_e with cumulative distribution function (CDF) as $\Pr(R_e \leq t) = 1 - (1 - t)^{w(e)}$. Distribution R is the joint probability distribution of the random variables for all edges in E, $R = (R_{e_1}, R_{e_2}, \cdots)$. Note that each R_e is independent of another edge's random variable R_f when $e \neq f$. So, choosing a random rank function r from the distribution R is nothing but each edge $e \in E$ choosing a value according to R_e which is the rank value $r(e)$ for that edge.

We now prove a key lemma that is crucial for proving the correctness of our local algorithm.

Lemma 4.8. *Let r be a rank function sampled from the distribution R and S be any (x, ν, σ, k)-extreme set in the graph $G = (V, E, w)$ then the probability that S respects r is at least $\frac{2}{\sigma(\sigma-1)}$.*

Proof. We need to lower bound the probability that an extreme set S respects the rank function r.

Let $G/\overline{S} = (V_S, E_S, w_S)$ be the graph which is obtained by contracting the set $V \setminus S$ into a single super node t. $V_S = S \cup \{t\}$. From definition of the (x, ν, σ, k)-extreme set $(S, \{t\})$ is the unique minimum cut of $G/\overline{S}$.

Using Karger's random contraction algorithm on graph $G/\overline{S}$, the probability of successful contractions to find the minimum cut is at least $\frac{2}{\sigma(\sigma-1)}$ as number of vertices in the graph $G/\overline{S}$ are bounded by σ.

As noted in Claim 4.6 sampling of an edge in Karger's random contraction process can be simulated using a rank function chosen from R restricted on the graph $G/\overline{S}$ i.e. considering only the edges in $G/\overline{S}$. So rank function r succeeds in finding the unique minimum cut $(S, \{t\})$ with probability at least $\frac{2}{\sigma(\sigma-1)}$ i.e. all the vertices of the set S are contracted into a single node say s and finally we have the cut $(\{s\}, \{t\})$ as it is the unique minimum cut.

Now we show that such a succeeding rank function chosen from R is respected by the set S in graph $G/\overline{S}$. Since the rank function, r should contract the whole S without contracting any edge from the minimum cut $\delta_{G/\overline{S}}(S)$. The MST of $G/\overline{S}$ with respect to the rank function r should cross the minimum cut S in $G/\overline{S}$ only once and also should span $G[S]$ before crossing the set S.

So we conclude that all edges in the minimum cut $(S, \{t\})$ of $G/\overline{S}$ have a higher rank than the edges that belong to the spanning tree restricted to $G[S]$. So S respects r in G whenever the random rank function finds the unique minimum cut in $G/\overline{S}$.

Since probability of success of random contraction process finding minimum cut in $G/\overline{S}$ is at least $\frac{2}{\sigma(\sigma-1)}$, probability of S respecting random rank function is also at least $\frac{2}{\sigma(\sigma-1)}$. $\qquad \square$

4.3 Local Subroutine for (ν, σ, k)-set

In the Section 4.1, we have seen how to find a (x, ν, σ, k)-set that respects a given rank function r by using LOCALPRIM sub-routine. Section 4.2 showed that any (ν, σ, k)-extreme set respects a random rank function with good probability. In this sub-section we combine both to give an algorithm that solves Problem 1.4 and proves Theorem 1.5(1).

From Lemma 4.8 we know that a random rank function is respected by an (x, ν, σ, k)-extreme set with $\Omega(1/\sigma^2)$ probability. To find a rank function that is respected with high probability, we need to independently repeat this process $O(\sigma^2 \log n)$ times.

So the algorithm to solve Problem 1.4 would mainly be iterating over $(O(\sigma^2 \log n))$ many independent random rank functions. Since the (x, ν, σ, k)-extreme set respects at least one random rank function among $O(\sigma^2 \log n)$ and so we are guaranteed to return a (x, ν, σ, k)-set by Lemma 4.4.

We formalize the above argument and give an algorithm that solves Problem 1.4. We can fix all the random rank functions $r^i, i \in [O(\sigma^2 \log n)]$ before the algorithm starts with out loss of generality.

Algorithm 3: LOCALKCUT

Data: $G = (V, E, w), x, \nu, \sigma, k$
Result: (x, ν, σ, k)-set or $\perp$ if no (x, ν, σ, k)-extreme set exists

1 for $i \in [O(\sigma^2 \log n)]$ **do**
2 $X = \text{LOCALPRIM}(G, x, \nu, \sigma, k, r^i)$;
3   **if** $X \neq \perp$ **then**
4 $\lfloor$ **return** X;

5 return $\perp$;

Lemma 4.9. *Algorithm 3 solves Problem 1.4 correctly with high probability.*

Proof. From Lemma 4.8 we know that any (x, ν, σ, k)-extreme set should respect a random rank function with probability at least $\frac{2}{\sigma(\sigma-1)}$. If there exists an (x, ν, σ, k)-extreme set S, then S will respect at least one of the $\Theta(\sigma^2 \log n)$ random rank function. The probability that none of them is respected by S,

$$\Pr(S \text{ does not respect any } r^i) \leq \left(1 - \frac{2}{\sigma(\sigma-1)}\right)^{\Theta(\sigma^2 \log n)} \leq \frac{1}{n^{\Theta(1)}}$$

Since there is a rank function that is respected by the (x, ν, σ, k)-extreme set. According to Lemma 4.4 we find a (x, ν, σ, k)-set and we are done.

However if we do not find a (x, ν, σ, k)-set by any of the random rank functions then we know that no (x, ν, σ, k)-extreme set existed and we are only wrong with negligible probability $\frac{1}{n^{\Theta(1)}}$. $\square$

Lemma 4.10. *Rutime of the Algorithm 3 is*

$$T(\text{LOCALKCUT}) = O(\sigma^2 \log n) \cdot T(\text{LOCALPRIM}). \tag{2}$$

Proof. Each iteration of the for loop takes $T(\text{LOCALPRIM})$ time and so the total time taken by LOCALKCUT is $O(\sigma^2 \log n) \cdot T(\text{LOCALPRIM})$. $\square$

We now prove the Theorem 4.1 which is same as proving Theorem 1.5(1).

Proof of Theorem 4.1: From Lemma 4.5 we have $T(\text{LOCALPRIM}) = O(\nu \log \nu)$ which is at most $O(\nu \log n)$. According to Lemma 4.10, $T(\text{LOCALKCUT}) = O(\nu \sigma^2 \log^2 n)$. From Lemma 4.9 we have the Algorithm 3 that solves Problem 1.4 with high probability in running time at most $O(\nu \sigma^2 \log^2 n)$ proving Theorem 4.1. $\square$

Since any (ν, σ, k)-extreme set of volume at most ν cannot have more than ν vertices as they need to be connected hence a trivial upper bound on the number of vertices contained in the set would be at most ν. Substituting the trivial upper bound for number of vertices σ in Eq (2) we get $T(\text{LOCALKCUT}) = O(\nu^3 \log^2 n)$. We can now eliminate the parameter σ from Theorem 4.1, which gives the following theorem.

Theorem 4.11. *There exists a randomized algorithm given a graph $G = (V, E, w)$, a vertex x and parameters ν, k finds either a (x, ν, k)-set or guarantees that no (x, ν, k)-extreme set exists with high probability in time $O(\nu^3 \log^2 n)$.*

5 $\tilde{O}(m^{1.75})$ algorithm for maximal k-connected partitions

In the previous section, we introduced the notion of small local cuts. We also gave a local algorithm that finds small volume cuts whose cut size is smaller than a given parameter k. This section introduces an algorithm that uses the local sub-routine to improve recursion depth and finds maximal k-edge-connected partition in running time $\tilde{O}(m^{1.75})$. Although this algorithm can be slower than $\tilde{O}(mn)$ when the graph is dense, in the next section, we introduce another algorithm that improves the running time $\tilde{O}(mn)$ for finding maximal k-edge-connected partitions.

The main goal of this section is to prove the first running time stated in Theorem 1.1. We restate it here for completeness.

Theorem 5.1. *There exists a randomized algorithm that takes a graph $G = (V, E, w)$ with m edges and n vertices and finds the maximal k-edge-connected partition with high probability in at most $O(m^{1.75} \log^{3.75} n)$ time.*

5.1   Global Algorithm using LocalKCut

Let us see how to use the LocalKCut subroutine described above to improve the recursion depth. For the rest of this section, we omit the cardinality constraint σ. The main idea is to find as many (ν, k)-sets as possible using LocalKCut. However, it might happen that removing a (ν, k)-set might give rise to new extreme sets in the graph. So, at every stage, we maintain a list of candidate vertices (L as given in MaximalPartition) which might be inside a (ν, k)-extreme set. We remove all these extreme sets by using LocalKCut with input as vertices x from the candidate list, which either finds a (ν, k)-set containing x or guarantees that no extreme set containing x exists.

Exhausting the candidate list ensures the residual graph has no small extreme sets, which would imply a ν-balanced minimum cut. Let us prove this below.

Lemma 5.2. *If there are no (ν, k)-extreme sets in the graph $G = (V, E, w)$, then either*

- *minimum cut is at least k in which case graph G is k-edge-connected.*

- *minimum cut is less than k in which case it is ν-balanced.*

Proof. The first part is trivial; if the graph's minimum cut is at least k, it is k-edge-connected.

If the minimum cut of the graph is less than k and if every inclusion-wise minimal cut is ν-balanced then we are done. Assume there exists an inclusion-wise minimum cut $(A, \overline{A})$ that is not ν-balanced. Without loss of generality $vol(A) < \nu$. Because A is inclusion wise minimum cut, A is the unique minimum cut in $G/\overline{A}$. So A is a (ν, k)-extreme set which is contradiction. So $vol(A) \geq \nu$. So every inclusion wise minimal cut has volume $\geq \nu$ and so every minimum cut is ν-balanced. $\qquad\square$

From above Lemma 5.2 we have proved that if there are no (ν, k)-extreme sets in the graph and the graph is not k-edge-connected, then every minimum cut is ν-balanced and thus remove at least ν volume from the graph. It only takes at most $O(m/\nu)$ recursion depth for the maximum sized sub-graph to reduce to half the original volume $m/2$. Given the high-level overview of the

algorithm, we now formally define and analyze it below.

Algorithm 4: KCUTPARTITION

    Data: $G = (V, E, w), \nu, \sigma, k, L, m$

    Result: k-cut partition of G

1 $\hat{G} = G$;

2 $\mathcal{R} = \{\}$;

3 **while** *There exists $x \in L$* **do**

4 $S = \text{LOCALKCUT}(\hat{G}, x, \nu, \sigma, k)$;

5   **if** $S \neq \perp$ **then**

6 $\mathcal{R} = \mathcal{R} \cup \{S\}$;

7 $L = (L \cup V(\delta_{\hat{G}}(S)) \setminus S$;

8 $\hat{G} = \hat{G} \setminus S$;

9   **else**

10 $L = L \setminus \{x\}$;

11 $(A, B) = \text{MINCUT}(\hat{G})$;

12 **if** $\lambda(G) \geq k$ **then**

13 $\mathcal{R} = \mathcal{R} \cup \{\hat{V}\}$;

14 **else if** $vol(\hat{G}[A]) > m$ **then**

15 $\mathcal{R} = \mathcal{R} \cup \text{KCUTPARTITION}(\hat{G}[A], \nu, \sigma, k, V(\delta_{\hat{G}}(A)) \cap A, m)$

16 **else if** $vol(\hat{G}[B]) > m$ **then**

17 $\mathcal{R} = \mathcal{R} \cup \text{KCUTPARTITION}(\hat{G}[B], \nu, \sigma, k, V(\delta_{\hat{G}}(B)) \cap B, m)$;

18 **else**

19 $\mathcal{R} = \mathcal{R} \cup \{A, B\}$;

20 **return** $\mathcal{R}$;

KCUTPARTITION partitions the graph $G = (V, E, w)$ using cuts of size less than k, but doesn't give us the maximal k-edge-connected partition. It reduces the problem to smaller-sized sub-graphs. This is proved in the following Lemma 5.3.

Lemma 5.3. *Let $\mathcal{P}$ be the maximal k-edge-connected partition of G and $\mathcal{R}$ be the result returned by Algorithm 4. For all $U \in \mathcal{P}, \exists U' \in \mathcal{R}$ such that $U \subset U'$. In other words, $\mathcal{P}$ is a refinement of the partition $\mathcal{R}$. For all $U' \in \mathcal{R}, \text{vol}(G[U']) = \text{vol}(G)/2$.*

Proof. As seen in the proof of Lemma 3.2, any maximal k-edge-connected set of vertices U cannot be separated by a cut of size less than k. Every cut used to partition the graph in KCUTPARTITION is of size less than k. Hence every maximal k-edge-connected set of G has to be strictly contained in one set belonging to $\mathcal{R}$.

Since we always recurse on the larger volume side of the minimum cut, the smaller side has less than half the original volume. Any subgraphs formed by local cuts have volume less than ν, which we later choose to be less than $m/2$. Thus all subgraphs have at most $\text{vol}(G)/2$. □

Note that every maximal k-edge-connected set of the original graph is also a maximal k-edge-connected set of the smaller sub-graph resulting from KCUTPARTITION. Thus naturally, the algo-

rithm for finding the maximal k-edge-connected partition is as follows.

Algorithm 5: MAXIMALPARTITION

Data: $G = (V, E, w), k$

Result: Maximal k-edge-connected partition of G

1   $\mathcal{R} = \text{KCUTPARTITION}(G, \nu(|E|), \sigma(|V|), k, V, |E|)$;

2   $\mathcal{P} = \{\}$;

3   **for** $U \in \mathcal{R}$ **do**

4       $\mathcal{P} = \mathcal{P} \cup \text{MAXIMALPARTITION}(G[U], k)$;

5   **return** $\mathcal{P}$;

The size of the graphs for which we find the maximal k-edge-connected partition is at most half the original volume. So, after at most $O(\log m)$ recursive applications of KCUTPARTITION we find every maximal k-edge-connected set $U \in \mathcal{P}$. Since any graph that is k-edge-connected cannot broken further by KCUTPARTITION, or we end up with a single vertex which is trivially k-edge-connected. In the following Lemma 5.4 we prove the correctness of the Algorithm 5.

Lemma 5.4. *Algorithm 5 computes the maximal k-edge-connected partition of the graph G.*

Proof. From Lemma 5.3, we have seen that any maximal k-edge-connected set is strictly contained in one set of the partition returned by KCUTPARTITION. However, according to the guarantee of KCUTPARTITION volume of each component is reduced by half of the original graph. Hence every component has volume at most $m/2^i$ at a recursion depth of i. So any particular set S from maximal k-edge-connected partition with $vol(G[S]) \in [\frac{m}{2^{i+1}}, \frac{m}{2^i}]$ is identified atleast at a recursion depth of $i+1$, as none of the cuts break the set S and so after $O(\log n)$ depth each component either has become k-edge-connected and so has become part of the partition $\mathcal{P}$ or the volume becomes 0 i.e., a singleton vertex which is trivially k-edge-connected. $\qquad\square$

5.2   Runtime analysis

In this subsection, we analyze the run time of the Algorithm 4 and then Algorithm 5.

We divide the running time of KCUTPARTITION into two parts: the invocation of the local subroutines and the invocations of the global minimum cut algorithm. For the first part, since we only apply local subroutine with seed vertices from the candidate list L in graph KCUTPARTITION. We can bound the running time by bounding the number of vertices added to L. We start with candidate list $L = V$ initially as (ν, k)-extreme sets can contain any vertex in V.

Once a cut of size less than k is found (using either LOCALKCUT or MINCUT), we remove the smaller side from the graph $\hat{G}$ and find cuts in the residual graph. Because we are removing some edges from the graph $\hat{G}$, it can lead to the origin of new extreme sets that might not be present before. So we need to update L to account for new extreme sets that might arise due to the removal of the cut edges. It is unwarranted and time-consuming to iterate over all vertices to find new extreme sets. Similar to the idea from [CHI$^+$17], we prove that any new extreme sets that arise due to the removal of an edge from the graph should contain at least one of the edge's endpoints. We add this vertex to the candidate list so that we can find a (ν, k)-set surrounding this vertex if the new extreme set is (ν, k)-extreme set.

Claim 5.5. *Any new extreme sets introduced in the graph by removing an edge must contain at least one endpoint of that edge.*

Proof. Let G be the graph before removing the edge e and H be the graph after removing. Removal of an edge e may lead to the formation of new extreme set A only if it belongs to the cut edges of

the new extreme set i.e., $e \in \delta_G(A) = \delta_{G/\overline{A}}(A)$. If it is an internal edge of the new extreme set, i.e., $e \in G[A]$, then even before the removal of the edge, it is an extreme set, and hence it is not formed by removing the edge e. If it belongs to $G[\overline{A}]$, it does not affect whether A is an extreme set or not as we consider the graph $G/\overline{A}$ in deciding whether A is an extreme set. Since it is the cut edge of the extreme set, then there exists an endpoint of the edge e in the set, proving the claim. $\qquad \square$

Once the candidate list is exhausted, we are sure (with high probability) that there are no extreme sets in the residual graph; thus, we apply the global minimum cut to get a ν-balanced cut and recurse on the larger side, as shown in Lines 15 and 17. We now bound the recursion depth of KCUTPARTITION in the below lemma.

Claim 5.6. *Algorithm 4 has a recursion depth of at most $O(m/\nu)$ for a m edge graph.*

Proof. At the end of the while loop, we have the guarantee that there are no (ν, k)-extreme sets, which means from Lemma 5.2 we either have that the minimum cut is at least k in which case we are done by adding the vertex set of the current graph to the $\mathcal{R}$. Otherwise, the minimum cut is ν-balanced. So the largest component is decreased by at least ν volume when a global mincut is called. Since initially we have volume m, and at each recursion depth, we reduce the volume of the maximum component by at least ν, so it only takes at most $O(m/\nu)$ recursion depth for the volume of the largest component to be less than $m/2$. We stop the recursion once all the components are of volume at most half the original volume by just adding both sides of the minimum cut to $\mathcal{R}$. $\quad \square$

We analyze the total running time of KCUTPARTITION by separating the calls to LOCALKCUT local subroutine from the remaining part of the KCUTPARTITION.

Claim 5.7. *Algorithm 4 invokes LOCALKCUT at most $O(m)$ times during the running of the whole algorithm i.e. over all $O(m/\nu)$ recursion levels.*

Proof. Local subroutine LOCALKCUT is always invoked on vertices that belong to L. So it is convenient to bound the number of vertices added to the candidate list during the whole algorithm course.

We initially start with the $L = V$, the whole vertex set of size n. When each (ν, k)-set is separated from $\hat{G}$, we add the endpoints of the cut edges on the larger side to the candidate list. From *Claim* 5.5, we know that any new extreme sets introduced due to removing cut edges should contain endpoints of cut edges. Hence it is enough to check for (ν, k)-extreme sets just around the cut edges endpoints. As per the guarantee of LOCALKCUT we find the (ν, k)-sets with high probability if any (ν, k)-extreme sets exist. Since we are removing the (ν, k)-set from the graph, the extreme set is not present in the residual graph.

Once a cut edge is removed from the graph, it cannot be part of any of the sub-graphs present as a part of the later recursion. So every edge introduces a new vertex into list L, and no edge becomes a cut edge again. Hence a trivial bound on the number of vertices added to the candidate list is, at most, the number of edges. So total number of invocations of LOCALKCUT subroutine is at most $m + n$ which is $O(m)$. $\qquad \square$

Since number of invocations to LOCALKCUT is $O(m)$, we can union bound the failure probability over all the invocations of LOCALKCUT which is $\frac{O(m)}{n^c}$ still leading to a total error probability of at most $\frac{1}{\text{poly}(n)}$ for sufficiently large constant c. Similarly, we can bound the error probability for $O(m/\nu)$ invocations of the global mincut algorithm. Since the sizes of the graphs during $O(m/\nu)$ invocations of the algorithm are at least $\Omega(n^{O(1)})$. If the error probability of global mincut algorithm is at most $\frac{1}{n^c}$ for some sufficiently large constant c then error probability over all the $O(m/\nu)$

invocations is at most $\frac{1}{\text{poly}(n)}$ as required. This proves that the error probability of Algorithm 4 is at most $\frac{1}{\text{poly}(n)}$.

Lemma 5.8. *Algorithm 4 runs in time at most $O(m^{1.75} \log^{2.75} n)$ for an m edge n vertex weighted graph.*

Proof. From Claim 5.7 we have bounded the number of calls to the LOCALKCUT subroutine which is at most $O(m\nu^3 \log^2 n)$ from Theorem 4.11.

At each recursion level of KCUTPARTITION, we apply the global mincut algorithm on a graph with edges less than m and vertices less than n. Hence we can trivially bound the run time due to the global mincut being at most $O(m/\nu) \cdot O(m \log^3 n)$.

So the total run time of the KCUTPARTITION is at most $O(m\nu^3 \log^2 n + \frac{m^2 \log^3 n}{\nu})$. Optimizing with $\nu = O((m \log n)^{1/4})$ the run time of KCUTPARTITION would be $O(m^{1.75} \log^{2.75} n)$. $\square$

Now, we analyze the running time of Algorithm 5.

Lemma 5.9. *Algorithm 5 runs in time at most $O(m^{1.75} \log^{3.75} n)$ on a m edge n vertex weighted graph.*

Proof. From the guarantee of the Algorithm 4, we have that the volume of the graph induced on each set that belongs to $\mathcal{R}$ is half the original volume. Hence the recursion depth of the MAXIMALPARTITION subroutine is at most $\log m = O(\log n)$.

Let us then bound the run time taken by MAXIMALPARTITION at every recursion level. The volume of all the graphs induced on the partitioned sets is bounded by m. Let $m_i (< m/2), n_i$ be the number of edges and vertices in the ith graph induced on the partition of vertices after applying KCUTPARTITION subroutine. Because these form a partition of vertices of the original graph and we only consider induced graphs on all these partitions $\sum_i n_i = n$, total volume is bounded by m, i.e. $\sum_i m_i \leq m$.

Hence we have the following recurrence equation.

$$T(m) = \sum_i T(m_i) + O(m^{1.75} \log^{2.75} n)$$

$$m_i \leq m/2 \text{ for all i}$$

Since $\sum_i O(m_i^{1.75} \log^{2.75} n_i) \leq O(m^{1.75} \log^{2.75} n)$. Hence it takes $O(m^{1.75} \log^{2.75} n)$ at each recursion level and there are at most $O(\log n)$ recursion level. Therefore the run time of the MAXIMALPARTITION subroutine is at most $O(m^{1.75} \log^{3.75} n)$. $\square$

Proof of Theorem 5.1. From Lemma 5.4 we have Algorithm 5 that computes the maximal k-edge-connected partition of the graph $G = (V, E, w)$ with parameter k as input, which takes running time at most $O(m^{1.75} \log^{3.75} n)$ from Lemma 5.9 proving Theorem 5.1. $\square$

6 Local algorithm for small cardinality cuts

The main goal of this section is to prove Theorem 1.5 (2). In this section, we only care about (σ, k)-sets, small cardinality cuts of small cut size. However, they can have an arbitrarily high volume when the degree of the vertices is large. Hence applying Algorithm 3 would lead to a large running time. Thus we overcome this by doing additional book-keeping to find (σ, k)-sets in $\tilde{O}(\sigma^4)$ time.

We describe the data structure called sorted adjacency list for book-keeping in Section 6.1 and then use it to speed up the LOCALKCUT in Section 6.2.

6.1 Sorted adjacency list

Given a graph $G = (V, E, w)$ with a rank function $r : E \to [0, 1]$ undergoing edge deletions we maintain adjacency lists of each vertex sorted according to rank. Along with a sorted adjacency list, we also maintain an adjacency matrix to check whether an edge is present or not in the graph in $O(1)$ time. We initially start with empty adjacency lists $\mathcal{L}$ and empty adjacency matrix $\mathcal{M}$, and we can modify our data structure using the following operations.

- INSERT $(e = (u, v))$:

  1. Add edge to the adjacency matrix $\mathcal{M}$.
  2. Insert edge to adjacency lists of $\mathcal{L}[u], \mathcal{L}[v]$. As the list is sorted according to rank and of size at most Δ, it takes at most $O(\log \Delta)$ time.

- DELETE $(e = (u, v))$:

  1. Mark the edge deleted in adjacency matrix $\mathcal{M}$.
  2. We can store the pointer to edge e in the adjacency matrix thus we can find it in $O(1)$ time and can also delete in adjacency lists $\mathcal{L}[u], \mathcal{L}[v]$ in $O(1)$ time if we maintain $\mathcal{L}[u], \mathcal{L}[v]$ as doubly linked lists.

- NEXTEDGE (X):

  1. Return $\min\{x \in X : \min(\mathcal{L}[x])\}$ in $O(|X|)$ time.

We construct the sorted adjacency list of the graph G with respect to a rank function r by inserting (INSERT) each edge into the data structure, which takes a total run time of at most $O(m \log \Delta) = O(m \log n)$.

As seen in LOCALKCUT to boost the probability of finding the extreme sets, we repeat the LOCALPRIM sub-routine $O(\sigma^2 \log n)$ times with independent rank functions. So we will need to individually maintain the above data structure for all the $O(\sigma^2 \log n)$ rank functions.

It takes a total time $O(m \log n) \cdot O(\sigma^2 \log n) = O(m\sigma^2 \log^2 n)$ to construct the data structure with respect to all rank functions. In the next subsection, we see how to use the data structure to speed up LOCALPRIM.

6.2 Speedup LocalPrim

We will formalize how to use the sorted adjacency list below to improve the run time of LOCALPRIM. We omit the volume constraint ν while expanding the set X in the algorithm, as we only care about

small cardinality cuts. The rest of the algorithm remains the same.

Algorithm 6: LOCALPRIM

> **Data:** $G = (V, E, w), x, \sigma, k, r$ where r is a rank function $r : E \to [0, 1]$
> **Result:** (x, σ, k)-set respecting r if exists else $\perp$

1 $X = \{x\}$;
2 **while** $w(\delta(X)) \geq k$ ~~and $vol(X) < \nu$~~ and $|X| < \sigma$ **do**
3 $\quad$ Find the edge with minimum rank, $e = (u, v) \in E(X, V \setminus X) = \delta(X)$;
4 $\quad$ Update $X = X \cup \{v\}$;
5 **if** $w(\delta(X)) < k$ **then**
6 $\quad$ **return** X
7 **else**
8 $\quad$ **return** $\perp$

In Claim 6.1, we analyze the running time of LOCALPRIM that uses the sorted adjacency list.

Claim 6.1. LOCALPRIM *takes at most $O(\sigma^2)$ time using sorted adjacency list.*

Proof. In each iteration of the while loop, we find the next edge to expand the set X along using NEXTEDGE, which takes $O(|X|)$ time. Let v be the endpoint of the edge that is not in X. To maintain $w(\delta(X))$ we need to add the deg(v) (weighted degree) which takes at most $O(1)$ time and delete the weight of the edges that are incident on v from vertices currently present in X. So it takes at most $O(|X|)$ to compute the cut size after contracting v.

After expanding the set by including the vertex v, we need to delete the edges whose both endpoints are in X as the next edge to expand set X should not be from the internal edges of set X. After expanding set X with v, there are at most $|X|$ many edges that are becoming internal edges (edges whose both endpoints lie in X) and need to be deleted (DELETE) from the data structure. So it takes at most $O(|X|)$ time.

So each iteration of the while loop takes $O(|X|)$ time. As $|X|$ ranges from 1 to at most σ the total run time is at most $O(\sigma^2)$. $\qquad \square$

Note that we are modifying the sorted adjacency list data structure for a particular rank function r. So we maintain a journal of the edits made to the data structure, which are at most $O(\sigma^2)$ as we only delete the internal edges. Thus we can undo the changes once we have completed the sub-routine LOCALPRIM. So it still takes $O(\sigma^2)$ time to run the sub-routine and maintains the data structure intact.

Similar to LOCALPRIM, we modify LOCALKCUT to find small cardinality local cuts of small cut size. Note that the LOCALPRIM we use here is the one for finding small cardinality cuts.

Algorithm 7: LOCALKCUT

> **Data:** $G = (V, E, w), x, \sigma, k$
> **Result:** (x, σ, k)-set or $\perp$ if no (x, σ, k)-extreme set exists

1 **for** $i \in [O(\sigma^2 \log n)]$ **do**
2 $\quad$ $X = \text{LOCALPRIM}(G, x, \sigma, k, r^i)$;
3 $\quad$ **if** $X \neq \perp$ **then**
4 $\quad\quad$ **return** X;
5 **return** $\perp$;

From Lemma 4.10 we can conclude the following Theorem 6.2.

Theorem 6.2. *There exists a randomized algorithm given a graph $G = (V, E, w)$, a vertex x and parameters σ, k finds either a (x, σ, k)-set or guarantees that no (x, σ, k)-extreme set exists with high probability in time $O(\sigma^4 \log n)$.*

Proof. Since the probability of finding the (σ, k)-set using LOCALPRIM is only dependent on number of vertices in the set and it is at least $\frac{2}{\sigma(\sigma-1)}$ as seen in Lemma 4.8. Hence we can repeat the LOCALPRIM subroutine $O(\sigma^2 \log n)$ times to improve the probability thus taking a total run time of $O(\sigma^4 \log n)$. $\qquad\square$

This proves the Theorem 1.5(2).

7 $\tilde{O}(mn^{0.8})$ algorithm for maximal k-connected partitions

We have seen in the Section 5 how a local subroutine LOCALKCUT is used to improve the recursion depth and thus improve the run time, at least when the graph is sparse. However, $\tilde{O}(m^{1.75})$ is still high when m is in dense regime. Let us see an improved version of the above algorithm below by using the fast LOCALKCUT from Section 6 and changing the parameter to σ instead of ν, with the general framework of the algorithm remaining intact. This results in the second running time claimed in Theorem 1.1. The main result that we prove in this section is the following theorem.

Theorem 7.1. *There exist a randomized algorithm that takes a graph $G = (V, E, w)$ with m edges and n vertices and finds the maximal k-edge-connected partition with high probability in at most $O(mn^{0.8} \log^{3.6} n)$ time.*

7.1 Parametrize over n

In the previous local subroutine, we have parameterized over the volume of the local cut ν, thus resulting in at most $O(m/\nu)$ recursion depth. When we parameterize over σ, we can change the recursion depth, which would be at most $O(n/\sigma)$. We formally prove it in the following lemma.

Lemma 7.2. *Recursion depth of the KCUTPARTITION using updated LOCALKCUT, LOCALPRIM subroutine is at most n/σ.*

Proof. At each level, when all the (σ, k)-extreme sets are removed and if the residual graph is still not k-edge-connected, then the minimum cut found using Karger's MINCUT has at least σ vertices on both sides. If not, we would find a (σ, k)-extreme set contradicting (with high probability) that no (σ, k)-extreme set exists. Since the minimum cut is a σ-balanced cut, the number of vertices in the large component reduces by at least σ. So, it takes at most $O(n/\sigma)$ recursion depth to reduce the number of vertices in the largest component to half its original size, i.e., $n/2$. Hence the recursion depth of KCUTPARTITION is at most $O(n/\sigma)$. $\qquad\square$

7.2 Run time analysis

The analysis is very similar to that of the previous algorithm.

Lemma 7.3. *The run time of KCUTPARTITION is at most $O(mn^{0.8} \log^{2.6} n)$.*

Proof. As proven in Claim 5.7 we invoke LOCALKCUT at most $O(m)$ times, each of which takes $O(\sigma^4 \log n)$ time. So it takes $O(m\sigma^4 \log n)$ time for the local subroutine calls.

The recursion depth of KCUTPARTITION is at most $O(n/\sigma)$ and the run time required for applying Karger's MINCUT at each recursive level is at most $O(m \log^3 n)$.

We also need to account for the time taken to construct the data structure for all the $O(\sigma^2 \log n)$ rank functions which amount to at most $O(m\sigma^2 \log^2 n)$.

Hence the total runtime is $O(m\sigma^2 \log^2 n) + O(m\sigma^4 \log n) + O(m \log^3 n) \cdot O(\frac{n}{\sigma})$. Optimizing with $\sigma = O((n \log^2 n)^{1/5})$ the total run time is at most $O(mn^{0.8} \log^{2.6} n)$. $\qquad\square$

Since MAXIMALPARTITION calls KCUTPARTITION which breaks the graph into partition in which each set is of size at most half of the original size. Hence recursion depth of MAXIMALPARTITION is at most $\log n$. With similar argument stated in Lemma 5.9 we conclude that the run time taken to find maximal k-edge-connected partition is at most $O(mn^{0.8} \log^{3.6} n)$.

This proves the Theorem 1.1. Note that $\tilde{O}(mn^{4/5})$ is an improvement over $\tilde{O}(m^{1.75})$ in the dense regime where $m = \Omega(n^{16/15})$.

8   Conditional lower bounds

As we have seen from the above sections, the main bottleneck in our framework to find maximal k-edge-connected partitions is the sub-routine to find a set of cut size less than k having small cardinality or volume. A natural question is "Can we improve the running time dependence on parameters ν, σ for these problems?". In this section, we give lower bounds for problems LOCALMST and LOCALKCUT. We use the conjectured hardness of the OMv problem to give running time lower bounds for these problems.

We state the OMv conjecture below. Let M be a boolean matrix of dimension $n \times n$. We have to compute the matrix-vector product of $v_1, \cdots v_n$ one after another in an online fashion. The naive running time to compute the vector product of all the vectors would be $O(n^3)$. It is conjectured that even after allowing a polynomial amount of pre-processing time, one cannot polynomially improve over the naive running time.

Conjecture 8.1 (OMv Conjecture). *There is no algorithm that computes matrix-vector products of n vectors coming in an online fashion in a total time of $n^{3-\Omega(1)}$ with an error probability of at most $1/3$.*

In particular, we use the hardness of the independent set query problem proved in [HKNS15] to give lower bounds to Problem 1.4 and Problem 4.3. More precisely, By combining Theorems 2.7 and 2.12 from [HKNS15], we obtain the following hardness result:

Theorem 8.2 ([HKNS15]). *Assuming Conjecture 8.1, there is no algorithm that computes whether any subset, $S \subset V$ is independent or not in the graph $G = (V, E, w)$ with error probability at most $1/3$ in $O(|S|^{2-\Omega(1)})$ time even after allowing a polynomial pre-processing on the graph G.*

The rest of the section is organized as follows. In Section 8.1 we prove the lower bound for the cardinality-constrained small cut containing x, (x, σ, k)-LOCALKCUT problem. In Section 8.2 we show that even the simple problem of cardinality-constrained small cut containing x and respecting a given rank function r, (x, σ, k)-LOCALMST cannot have a faster algorithm when only adjacency list and matrix of graph G are given. Finally, in Section 8.3 we extend the NP-hardness reduction given in [FGK13] to give $W[1]$-hardness albeit in the weighted case.

8.1   Lower bound for LocalKCut

Recall the (x, ν, σ, k)-LOCALKCUT problem.

Problem 1.4 (LOCALKCUT). *Given a graph $G = (V, E, w)$, a vertex x, and parameters ν, σ and k, either find an (x, ν, σ, k)-set or return $\bot$ indicating that no (x, ν, σ, k)-extreme set exists.*

We have given two different local algorithms in Section 4 and Section 6 respectively for different parameter variations of LOCALKCUT that run in time $\tilde{O}(\nu\sigma^2)$ and $\tilde{O}(\sigma^4)$. We would prove a conditional lower bound for (x, σ, k)-LOCALKCUT as stated below.

Theorem 8.3. *Assuming Conjecture 8.1, there is no algorithm that, given access to the adjacency lists and matrix of G (with no further preprocessing) solves (x, σ, k)-LOCALKCUT in $O(\sigma^{2-\Omega(1)})$ time.*

Before proving the above theorem, we will first show the reduction that connects the problem of finding the (σ, k)-extreme set to the problem of deducing whether a set is independent.

Given a graph $G = (V, E, w)$ and a set S of size σ we can find the volume of S in $O(\sigma)$ time. Let $k = \text{vol}_G(S), \sigma = |S|$. Define a new graph $G_S = (V, E', w')$, which is an exact copy of G but also has an additional weighted cycle consisting of all the vertices in S with each edge of weight $k + 1$. If an edge of the newly added cycle already exists in E, then just increment the weight of that edge by $k + 1$ in w' to keep G_S simple graph. We replace the infinite value in the introduction with $k + 1$ as it is enough to get the desired result. We prove the following equivalence:
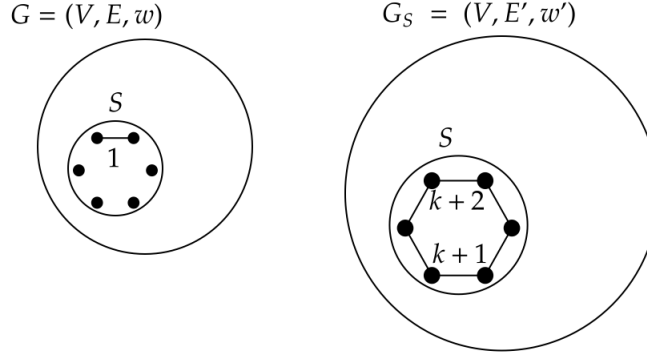

Figure 1: Lowerbound for (x, σ, k)-LOCALKCUT

Lemma 8.4. $E(S, S) \neq \phi$ *in G iff S is $(\sigma + 1, k)$-extreme set in G_S.*

Proof. ($\Longleftarrow$) If S is $(\sigma + 1, k)$-extreme set in G_S then we have $\delta_{G_S}(S) < k$ from the definition of $(\sigma + 1, k)$-extreme set. According to our construction of G_S we have $\delta_G(S) = \delta_{G_S}(S)$. Hence, $\delta_G(S) = \delta_{G_S}(S) < k = \text{vol}_G(S)$. Since, $\delta_G(S) < \text{vol}_G(S)$ we have $E(S, S) \neq \phi$.

($\Longrightarrow$) If $E(S, S) \neq \phi$ then $\delta_G(S) < k$ and $|S| < \sigma + 1$, hence S is a $(\sigma + 1, k)$-set. S is also an extreme set in G_S as it is unique minimum cut of $G_S/\overline{S}$ as any other cut of $G_S/\overline{S}$ would consist an edge from the cycle of weight $k + 1$ and hence the weight of the cut is more than k. $\square$

Observation 8.5. *For any $x \in S$, S is the only $(x, \sigma + 1, k)$-set in G_S which is also extreme.*

Observation 8.5 follows from the above construction of G_S. Now, we prove the lower bound for the problem of (x, σ, k)-LOCALKCUT.

Proof of Theorem 8.3. Assume for contradiction that there is an algorithm $\mathcal{A}$ that solves (x, σ, k)-LOCALKCUT and runs in $O(\sigma^{2-\varepsilon})$ time for some constant $\varepsilon > 0$. We can use this algorithm to find whether a given set S is independent or not in running time $O(\sigma^{2-\varepsilon})$ as follows.

Given a set S to check whether it is independent or not in graph G, construct the graph G_S as described above in $O(\sigma)$ time. Choose an arbitrary vertex $x \in S$ and use $\mathcal{A}$ to find $(\sigma + 1, k)$-set containing x in G_S. According to the runtime guarantee of the algorithm, this takes at most $O((\sigma+1)^{2-\varepsilon}) = O(\sigma^{2-\varepsilon})$ and finds whether there is a $(x, \sigma+1, k)$-set or return $\bot$ if no $(x, \sigma+1, k)$-extreme set exists.

If $\mathcal{A}$ returns a $(x, \sigma+1, k)$-set with any $x \in S$ as input, then it has to be S from Observation 8.5. If S is the $(\sigma+1, k)$-set then S is not independent.

If $\mathcal{A}$ returns $\bot$, then it implies that there is no $(x, \sigma + 1, k)$-extreme set in G_S. From Observation 8.5, the only $(x, \sigma + 1, k)$-set is S which is also extreme. Since, $\mathcal{A}$ implies no $(x, \sigma + 1, k)$-extreme set, there is no $(x, \sigma + 1, k)$-set which implies S is not a $(x, \sigma + 1, k)$-set. Since S contains x and has size less than $\sigma + 1$, the only condition that violates is the cut size constraint, so $\delta_G(S) = \delta_{G_S}(S) \geq k = \mathrm{vol}_G(S)$. Hence S is independent.

So, we can compute whether S is independent or not in $O(\sigma^{2-\varepsilon})$ time using $\mathcal{A}$, which contradicts Conjecture 8.1. Hence, our assumption that there exists a $O(\sigma^{2-\varepsilon})$ algorithm for Problem 1.4 is false for any constant $\varepsilon > 0$. $\qquad\square$

Although Theorem 8.3 rules out any polynomially faster algorithm than $O(\sigma^2)$ for any general σ. It still does not rule out an algorithm that only works when the size of the set S is small compared to the vertex set V. This is important because according to our parameters setting $\sigma = \tilde{O}(n^{1/5})$. So we have to rule out a better algorithm for Problem 1.4 even in this smaller parameter regime. Below we give a strong lower bound, which rules out an algorithm that performs polynomially better than $O(\sigma^2)$, even when we are guaranteed that the query set's size, $|S| = \sigma \ll n$.

Theorem 8.6. *Assuming Conjecture 8.1, there is no algorithm that, given access to adjacency list and matrix of G (with no further preprocessing) solves Problem 1.4 in $O(\sigma^{2-\Omega(1)})$ even when $\sigma \leq n^\gamma$ for every constant $\gamma \in [0, 1)$.*

Proof. Let $\mathcal{A}$ be an algorithm that solves Problem 1.4 in $O(\sigma^{2-\varepsilon})$ for some constant $\varepsilon > 0$ when $\sigma \leq n^\gamma$. We can create an algorithm that solves independent set query problem polynomially faster than $O(|S|^2)$ for set S of vertices in any graph of size n.

Given a graph $G = (V, E)$, construct an arbitrary graph $H = (V_H, E_H)$ of vertex set size $|V_H| = n^{1/\gamma}$ and embed the graph G on some arbitrarily chosen subset of n vertices in H which is of size at most $|V_H|^\gamma$.

Given any set $S \subseteq V$ in graph G construct the corresponding subset S_H of vertices in H on which graph G is embedded and add the cycle along the vertices S_H with weight $\mathrm{vol}_G(S) + 1$ as constructed in Theorem 8.3. We can now use $\mathcal{A}$ for finding $(\sigma + 1, k)$-extreme set in H because the queried set S has size $|S| \leq n = |V_H|^\gamma$. So as proved in Theorem 8.3 we can find whether the set S_H is an independent set or not in H which is the same as finding whether S is independent in G in run time $O(|S|^{2-\varepsilon})$ contradicting Conjecture 8.1. $\qquad\square$

8.2 Lower bound for LocalMST

Recall the problem of LocalMST. In this subsection, we give a lower bound for this problem.

Problem 4.3 (LocalMST)**.** *Given a graph $G = (V, E, w)$, a vertex x, a rank function r and parameters ν, σ, k find a (x, ν, σ, k)-set S that respects the rank function r or return $\bot$ if no such (x, ν, σ, k)-set respecting r exists.*

In Section 4 we have seen an algorithm for solving Problem 4.3 which finds a (ν, σ, k)-set that respects the rank function in $O(\nu \log \nu)$ time. In Section 6 we have seen another algorithm that

finds a (σ, k)-set that respects a given rank function in $O(\sigma^2)$ time. Below we will show that even this simple Problem 4.3 of finding a (x, σ, k)-set that respects the given rank function r cannot be solved in run time polynomially better than $O(\sigma^2)$.

Again as in the previous lower bound, we will show that no algorithm solves Problem 4.3 polynomially faster than $O(\sigma^2)$ for any parameter range σ. We can use the same argument as in Theorem 8.6 for proving the lower bound for any algorithm that only works in the smaller range of parameter σ. We state the theorem below.

Theorem 8.7. *Assuming Conjecture 8.1, given a graph $G = (V, E, w)$ with adjacency lists and matrix, parameters σ, k a vertex $x \in V$ and a rank function r, there is no algorithm that solves Problem 4.3 in time $O(\sigma^{2-\Omega(1)})$.*

Given a graph $G = (V, E, w)$ and a query set S, we create G_S by adding a cycle of weight $k + 1$, where $k = \text{vol}_G(S)$ as in the lower bound for LocalKCut. We define a rank function r_S with rank value 0 on the edges added as part of the cycle, and all other edges have rank 1. From the construction, it is clear that S respects the rank function r_S in G_S as it has a spanning tree in $G_S[S]$ where each edge has rank value 0 and any edge that belongs to cut $E(S, V \setminus S)$ has rank value 1 which is more than 0.

Proof of Theorem 8.7. Assume we have an algorithm $\mathcal{A}$ that solves Problem 4.3 in time $O(\sigma^{2-\varepsilon})$ for some $\varepsilon > 0$. Given any graph G we do no pre-processing and whenever we get a query set S we construct G_S and the corresponding rank function r_S (implicitly).

Hence we can use the algorithm $\mathcal{A}$ on graph G_S with any arbitrary $x \in S$ and rank function r_S to find a $(x, \sigma + 1, k)$-set in $O(\sigma^{2-\varepsilon})$ time, where $\sigma = |S|$ and $k = \text{vol}_G(S)$.

If $\mathcal{A}$ finds a $(x, \sigma + 1, k)$-set respecting r_S in G_S then it has to be S, because from Observation 8.5 we know that S is the only $(x, \sigma + 1, k)$-set. Hence S is not an independent set.

Else $\mathcal{A}$ finds no set and confirms that S is independent. It only takes $O(|S|^{2-\varepsilon})$ time thus contradicting Conjecture 8.1. So no such algorithm $\mathcal{A}$ exists. $\qquad\square$

8.3   Lower bound for SmallLocalSeparation

[FGK13] proves Problem 8.8 is NP-complete when the graph is unweighted by giving a reduction from CLIQUE problem in regular graphs. We extend the theorem and prove that Problem 8.8 is $W[1]$-hard for weighted graphs when parameterized by σ. It is important to note that Problem 8.8 has a stronger guarantee than the Problem 1.4. We call it SmallLocalSeparation.

Problem 8.8 (SmallLocalSeparation)**.** *Given a graph $G = (V, E, w)$, parameters σ, k and a vertex $x \in V$, either find a (x, σ, k)-set or guarantee that no such set exists.*

It is worthwhile to note that the LocalKCut problem is a relaxation of the SmallLocalSeparation. Although in both problems we find a (x, σ, k)-set, the guarantee that we give when we do not find such a set is weaker in LocalKCut as we are only ruling out (x, σ, k)-extreme sets, while stronger in SmallLocalSeparation as we have to rule out all (x, σ, k)-sets. This relaxation is crucial to the tractability of LocalKCut as we exploit the structure involved in extreme sets.

Lemma 8.9. *Problem 8.8 is $W[1]$-hard in parameter σ.*

At a high level, the proof directly extends from the NP-complete proof of Problem 8.8 in the unweighted case in [FGK13], by replacing the base clique of dn nodes with a single vertex and joining it to all the vertices of the original graph directly with edges of weight d. The base clique

is needed in their construction to keep the graph unweighted, however, we chose to prove for the weighted case at the cost of getting a better hardness i.e., proving $W[1]$-hard.

We know that t-CLIQUE problem is $W[1]$-hard when parameterized by t, where t is the number of vertices in the clique, even on d-regular graphs.

We define the construction of graph G' from G that reduces t-CLIQUE problem to Problem 8.8, that is used in proving Lemma 8.9.

Let a d-regular unweighted graph $G = (V, E)$ be the input of the k-Clique problem. The input for SMALLLOCALSEPARATION $G' = (V', E', w')$ is constructed as follows. Vertex set V' consists of the vertex x to be used as a terminal in SMALLLOCALSEPARATION, so we call it "terminal node" and set of nodes corresponding to the vertices and edges of graph G, call them "vertex nodes" and "edge nodes" respectively. Hence $|V'| = 1 + n + m$. Add an edge of weight d from x to all vertex nodes. Hence the degree of x would be dn. For every edge $e = (u, v) \in E$, add two edges, each of weight one, from the edge node to the corresponding vertex nodes in V'. Below we prove that the graph G' consists of a small local cut of size σ if and only if G has a t-clique and $\sigma = \text{poly}(t)$ as required for proving fixed-parameter intractability.
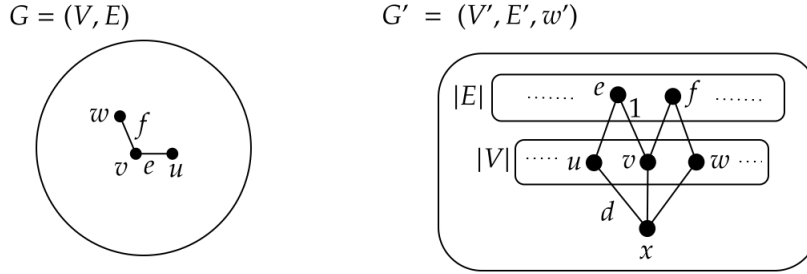

Figure 2: $W[1]$-hardness of SMALLLOCALSEPARATION

Lemma 8.10. *Let $\sigma = 2 + t + \binom{t}{2} = \text{poly}(t)$ and $k = dn - 2\binom{t}{2} + 1$. (x, σ, k)-set exists in G' iff a clique of size t exists in G.*

Proof. ($\Longleftarrow$) Let S be a clique of size t in G. Construct the set S' as follows. It consists of the terminal node x, vertex nodes corresponding to the set S in V' and all the edge nodes corresponding to the edges $E(S, S)$. Since S is a clique in G, the size of S' is $1 + |S| + |E(S, S)| = 1 + t + \binom{t}{2} < \sigma$.

The size of the cut $\delta_{G'}(S')$ is the total number of edges going out of S'. The weight of the cut edges incident on x is $d(n - |S|) = d(n - t)$. The degree of vertex nodes excluding the edges to the terminal node is d according to our construction. Hence, the total degree of vertex nodes is $d|S|$. However, since all the edge nodes $E(S, S)$ are included in the set S', the degree to them is subtracted. Since each such edge is counted twice because of both endpoints. The weight of the cut edges due to vertex nodes is $d|S| - 2|E(S, S)| = dt - 2\binom{t}{2}$. Since all the edges from edge nodes are inside the set S' they do not contribute to cut edges. Hence, the cut size of S' is $d(n - t) + dt - 2\binom{t}{2} = dn - 2\binom{t}{2} < k$. This proves that S' is a (x, σ, k)-set.

($\Longrightarrow$) If there exists a (x, σ, k)-set S' in G', then it contains x by definition. Let V_o, E_o be the vertex nodes and edge nodes present in the set S'. V_o cannot be empty as every edge node adds a degree 2 and the cut size would be $dn + 2|E_o|$, which violates the cut size constraint. Hence,

$V_o \neq \emptyset$. Assuming $E_o = \emptyset$. The cut size would be $|\delta_{G'}(V_o \cup \{x\})| = d(n - |V_o|) + d|V_o| = dn$ which violates the cut size constraint. Hence $E_o \neq \emptyset$.

There are three types of edges in G with respect to the vertex set V_o chosen, internal edges $E_G(V_o, V_o)$, cut edges $E_G(V_o, V \setminus V_o)$ and external edges $E_G(V \setminus V_o, V \setminus V_o)$. (Note that we misuse the notation V_o, what we meant is the corresponding vertex set in G.)

Adding an internal edge node to S' would reduce the cut size by two, a cut edge node to S' would not change the cut size, and an external edge node to S' would increase the cut size by two. Hence to reduce the cut size from dn to below $k = dn - 2\binom{t}{2} + 1$ we should add at least $\binom{t}{2}$ internal edges. To have $|E_G(V_o, V_o)| \geq \binom{t}{2}$, the size of V_o must be at least t. Since the number of nodes in the set S' is bounded by $1 + t + \binom{t}{2}$, the number of vertices in V_o cannot exceed t making it equal to t. It implies $|E(V_o, V_o)| = \binom{t}{2}$. Thus the set of vertices corresponding to V_o in G have to induce a clique of size t. □

Given the above reduction, we can now prove the $W[1]$-hardness of the SMALLLOCALSEPARATION. Observe that the set S' is not extreme in G' as the size of degree cut of any edge node is 2 which is less than the size of the cut $\delta_{G'}(S')$. Hence, this hardness does not work for Problem 1.4.

Proof of Lemma 8.9. Given an instance of t-CLIQUE, d-regular unweighted graph $G = (V, E)$ and a parameter t, we construct an instance of weighted graph $G' = (V', E', w')$ as shown above and set parameters $x, \sigma = \text{poly}(t), k$.

From Lemma 8.10 any FPT algorithm in parameter σ for Problem 8.8 can be turned into an FPT algorithm for t-CLIQUE as $\sigma = \text{poly}(t)$. Since t-CLIQUE is $W[1]$-hard, Problem 8.8 is $W[1]$-hard with parameter as σ. □

9 Approximating edge strength

In this section we prove the Corollary 1.2.

Definition 9.1 (Edge Strength). *The strength of an edge $e = (u, v)$ in a graph G is the largest k for which there exists a set $S \subseteq V$ containing u, v such that $G[S]$ is k-edge-connected.*

Hence, if two vertices of an edge belong to the same set in the maximal k-edge-connected partition then the edge has strength at least k. As we know, the maximum possible strength of any edge is $\Delta \leq (n-1)W = \text{poly}(n)$ where W is the maximum weight of an edge in the graph. As any graph can be disconnected by removing $n - 1$ edges. Assuming that W is bounded by $\text{poly}(n)$, edge strength is also bounded by $\text{poly}(n)$.

Lemma 9.2. *Given a maximal k-edge-connected partition of vertices:*

1. *Strength of edge contained in a subgraph is at least k.*

2. *Strength of edge that is across two subgraphs is less than k.*

Proof. Let $\mathcal{R} = \{V_1, V_2, \cdots\}$ be the maximal k-edge-connected partition of vertices. First part trivially follows from the definition.

Any edge $e = (u, v)$ across two partitions cannot have a k-edge-connected component surrounding it. Assume for contradiction that a subgraph H exists which is k-edge-connected and containing u, v both.

Since the edge is across two components in the maximal k-edge-connected partition, let V_i be the component that contains the vertex u. Since H is k-edge-connected and $G[V_i]$ is k-edge-connected,

hence $H \cup G[V_i]$ is also k-edge-connected which contradicts the maximality of the k-edge-connected subgraph induced on V_i. Thus such a subgraph H does not exist. Hence the strength of any inter-partition edge is less than k, proving the second part of the lemma. $\qquad \square$

Hence we approximate the strength of every edge followed by upper and lower bounding it in an interval of $(1 + \varepsilon)$.

Proof of Corollary 1.2: The maximum value of k for which a graph can be k-edge-connected is bounded by $\mathrm{poly}(n)$. Hence divide the space in powers of $(1 + \varepsilon)$ leading to $O(\frac{1}{\varepsilon} \log n)$ many values.

We invoke the subroutine given in *Algorithm* 5 with k taking all powers of $(1 + \varepsilon)$. Using Lemma 9.2, we can find upper and lower bounds to the strength of each edge up to a factor of $(1 + \varepsilon)$.

So the total runtime is at most $\tilde{O}(m \cdot \min(m^{3/4}, n^{4/5})/\varepsilon)$. $\qquad \square$

Acknowledgements

A   Rank function and its guarantees

Proof of Claim 4.6. Let us start with a warm up first. Let e_1, e_2, e_3 be three edges that sample values x, y, z in $[0, 1]$. Then the probability of x being minimum among the three values x, y, z is

$$\Pr(x = \min(x, y, z)) = \int_0^1 \int_x^1 \int_x^1 dz \, dy \, dx = \frac{1}{3}$$

Hence choosing the edge with a minimum rank among x, y, z is the same as choosing a uniform edge. Now let us see how to extend this to edges with weights.

Since weights are integral, each edge now chooses as many samples as its weight, and we choose the edge with the minimum random value as in warm-up. So similar to the warm-up, the probability of a particular sample being minimum is

$$\text{Probability of any sample value is minimum} = \frac{1}{\sum_f w(f)}$$

However, since each edge has multiple samples, we choose an edge e if any of its $w(e)$ samples of the edge are minimum. Hence the probability of choosing an edge e is

$$\text{Probability of any } w(e) \text{ samples being minimum} = \frac{w(e)}{\sum_f w(f)}$$

.

It is the same as choosing an edge with a weighted probability. Note that the number of random samples needed to sample is $\sum_f w(f)$, which can be very large when the weights are large.

We can overcome this problem by simulating the random distribution of the minimum of multiple random samples by sampling a single random variable. Let $X = \min(S_1, S_2, \cdots, S_w)$ be the random variable that is needed to be computed for an edge with weight w where S_i are uniform

random samples in $[0, 1]$. The cumulative density function of X is as follows.

$$\Pr(X > t) = \Pr(S_1 > t \text{ and } S_2 > t \text{ and } \cdots S_w > t)$$
$$= (1 - t)^w$$
$$\Pr(X \le t) = 1 - \Pr(X > t)$$
$$= 1 - (1 - t)^w$$

We can simulate the distribution of X for an edge with weight w as follows. Let R be a uniform random variable in $[0, 1]$. $\Pr(R \le p) = p$ for any $p \in [0, 1]$. Defining $Y = f(R) = 1 - (1 - R)^{1/w}$, we have

$$\Pr(R \le p) = p = \Pr(Y \le 1 - (1 - p)^{1/w}) \qquad \text{(as } Y = f(R))$$
$$\Pr(Y \le t) = 1 - (1 - t)^w \qquad \text{(taking } 1 - (1 - p)^{1/w} = t)$$

Hence we have a new random variable Y whose cumulative density function is the same as that of X and uses a single random sample by just plugging the single random sample R in the function given, which is nothing but the inverse probability distribution of the random variable X. $\qquad \square$

We have shown that a random weighted edge can be sampled using the rank function. However, we have considered a random sampling of a real value in the interval $[0, 1]$ which is not possible using a computer, so we need to discretize the space and sample a discrete value in $[0, 1]$ which results in a sampling error.

We show that dividing the space into very small discrete parts $\frac{1}{\text{poly}(n)}$ would bound the error in the rank with high probability. Since the error in the rank of the edges is small, the probability with which order of ranks of any two edges would change due to this error is small too. Bounding the probability of error over all the pairs of edges we prove that the probability of order of rank of edges changing due to discretizing the space $[0, 1]$ is very small.

Let us first discretize the space $[0, 1]$ into n^{10} parts and sample a number in $t \in [0, n^{10}]$ and return $\frac{t}{n^{10}}$. Recall that we sample a uniform random value $x \in [0, 1]$ and use the function $r(e) = 1 - (1 - x)^{1/w(e)}$. The error in the sample x due to discretizing the space is at most $\frac{1}{n^{10}}$. Hence, $dx = \frac{1}{n^{10}}$.

$$\frac{dr(e)}{dx} = \frac{1}{w(e)} \cdot (1 - x)^{\frac{1}{w(e)} - 1}$$
$$= \frac{1 - r(e)}{w(e)(1 - x)}$$
$$\le \frac{1}{1 - x} \qquad (w(e) \ge 1 \text{ and } r(e) \in [0, 1])$$
$$\le n^3 \qquad (\text{when } x \le 1 - \frac{1}{n^3})$$
$$dr(e) \le \frac{1}{n^7}$$

To bound the error in $r(e)$ we ignore some range of the x, the probability of we coming up with such an x is at most $\frac{1}{n^3}$. This has to be true for the uniform samples chosen for all the edges

which can be at most $O(n^2)$. Hence union bound over all those edges the probability of error in $r(e)$ exceeding over $\frac{1}{n^7}$ is at most $\frac{m}{n^3} \leq \frac{1}{n}$. Hence with high probability we can assume that error in $r(e)$ is very small due to discretizing the space.

We are not yet done, although the error in rank is small we have to bound the probability of the event in which the order of the edges according to rank is changed due to the error in the rank of the edges. We show that it occurs with very low probability. The high level idea is, because the error in rank is small for another edge to change the order it has to fall in the range of the small error and it occurs with very low probability.

Let e, f are two edges and we have proved above that error in $r(e)$ is at most $\frac{1}{n^7}$. For f to change order with respect to e, $r(f)$ has to lie in a range of $\frac{1}{n^7}$ of $r(e)$. Hence we have to bound the size of the domain for which the range is small. We have $dr(f) \leq \frac{c}{n^7}$ for some constant c.

$$
\begin{aligned}
\frac{dr(f)}{dx} &= \frac{1}{w(f)} \cdot (1-x)^{\frac{1}{w(f)}-1} \\
dx &= w(f) \cdot (1-x)^{1-\frac{1}{w(f)}} \cdot dr(f) \\
&\leq \frac{c}{n^5} \qquad\qquad\qquad\qquad\qquad (1 \leq w(f) \leq n^2)
\end{aligned}
$$

Applying a union bound over all possible pairs of edges we have at most $O(n^4)$ pairs and thus error probability is at most $O(n^4) \cdot \frac{c}{n^5} \leq \frac{c}{n}$. Hence with high probability no pair of edges change their order of rank due to discretizing the space.

So the total error probability of the event in which the error in rank is high or any pair of edges swap their order due to the error in rank is at most $O(\frac{1}{n})$. Thus we say that order of edges with respect to rank does not change due to discretizing space with high probability. Since all we care about is the order of the edges with respect to rank while choosing an edge to contract, the process outputs a random edge with high probability.

B Local algorithm for minimal extreme set

From [CLP22] we know that all extreme sets in the graph form a laminar family. So all extreme sets containing a particular vertex say x are contained in one another. So we define a minimal k-extreme set containing a particular vertex x as follows.

Definition B.1 (Minimal (x, k)-extreme set). *A k-extreme set S containing the vertex x is called minimal (x, k)-extreme set if no strict subset of S containing x is a k-extreme set.*

Note that this is guaranteed to exist if the graph G is not k-connected i.e., the minimum cut of G is $< k$.

We can define *minimal (x, ν, σ, k)-extreme set* analogously. However, this may not be guaranteed to exist as the volume of the k-extreme set need not be $< \nu$ or the cardinality need not be $< \sigma$.

Before we present an algorithm for finding the minimal (x, ν, k)-extreme set in case if it exists, we need to observe some properties of the extreme sets that help us to prove the correctness of the algorithm. From [CLP22] we know that the extreme sets in a graph form a laminar family. Hence all extreme sets containing x are contained in one another. Out of all possible extreme sets containing x let us consider those extreme sets that are (ν, σ, k)-extreme sets i.e. they satisfy the volume, cardinality, and cut size constraints. If there exists at least one such extreme set containing

x, then there exists a rank function respected by that (x, ν, σ, k)-extreme set among the $O(\sigma^2 \log n)$ random rank functions. Hence we can find the set using an algorithm similar to Algorithm 3.

Cactus graph is a graph in which every edge of the graph belongs to at most one simple cycle. From [DKL76] we know that all minimum cuts of a graph G can be represented in the form of a cactus graph H where each vertex of the cactus maps to a disjoint set of vertices in the original graph G and can also map to an empty set. Every cut induced in the cactus graph by the removal of a single non-cycle edge or two cycle edges that are part of same cycle will induce a corresponding cut in the original graph. Each such cut is minimum cut in the original graph. Cactus graph is succinct representation of all minimum cuts of the original graph. When we have a unique minimum cut in the graph, the cactus representation of minimum cuts of such a graph would be a single edge. From [PK09] we know that cactus representation of the graph can be found in near linear time. Hence we can check if a graph is having a unique minimum cut or not in near linear time. Let ISEXTREME be the sub-routine to check if a graph is extreme set or not.

We formally define the problem as follows.

Problem B.2. *Given a graph $G = (V, E, w)$, parameters ν, σ, k and a vertex $x \in V$, find the (x, ν, σ, k)-minimal extreme set or return $\perp$ if no such set exists.*

We will now present the local algorithm that solves Problem B.2. The algorithm is similar to Algorithm 3 except that we also check if the returned set is extreme or not.

Algorithm 8: MINIMALEXTREMESET

Data: $G = (V, E, w), x, \nu, \sigma, k$
Result: (x, ν, σ, k)-minimal extreme set or $\perp$ if no (x, ν, σ, k)-extreme set exists

1 $S = V$;
2 **for** $i \in [O(\sigma^2 \log n)]$ **do**
3 $\quad$ $X = \{x\}$;
4 $\quad$ **while** $vol(X) < \nu$ *and* $|X| < \sigma$ **do**
5 $\quad\quad$ **if** $w(\delta(X)) < k$ *and* ISEXTREME$(G/\overline{X})$ *and* $X \subset S$ **then**
6 $\quad\quad\quad$ $S = X$;
7 $\quad\quad\quad$ Break the while loop;
8 $\quad\quad$ Find the edge with minimum rank r^i, $e = (u, v) \in E(X, V \setminus X) = \delta(X)$;
9 $\quad\quad$ Update $X = X \cup \{v\}$;

10 **if** $S \neq V$ **then**
11 $\quad$ **return** S;
12 **else**
13 $\quad$ **return** $\perp$;

Lemma B.3. *Algorithm 8 solves the Problem B.2 with high probability.*

Proof. If there is no (x, ν, σ, k)-extreme set then the algorithm would not update S from V, we return $\perp$ at the end.

If there exists an (x, ν, σ, k)-extreme set S then as proved in Lemma 4.8, S would respect at least one of the $O(\sigma^2 \log n)$ random rank functions, say r^*. Thus set S would be one among the $\sigma - 1$ sets obtained using random contractions with respect to r^*. Since any other (x, ν, σ, k)-extreme set strictly contains S as extreme sets form a laminar family. We would find S and return it without being replaced by any other extreme set. $\qquad \square$

Lemma B.4. *Algorithm 8 runs in at most $\tilde{O}(\nu \sigma^3)$ time.*

Proof. For each of the $O(\sigma^2 \log n)$ rank functions, we perform σ many edge contractions and each time we check whether the set is extreme or not, which takes near linear in the number of edges present in $G/\overline{X}$ which is at most $\tilde{O}(\nu)$. It also takes $O(\nu \log \nu)$ for finding the minimum rank edges. So in total it takes at most $\tilde{O}(\nu\sigma) + O(\nu \log \nu) = \tilde{O}(\nu\sigma)$ per rank function. Since we do not know which rank function would be respected by the minimal extreme set, we have to run for every rank function making the running time at most $\tilde{O}(\nu\sigma^3)$. $\square$

C   Uniqueness of maximal k-edge-connected partition

The goal of this section is to mainly prove the Proposition 3.1.

Claim C.1. *A maximal k-edge-connected subgraphs $\{V_1, \ldots, V_z\}$ of any graph G is unique and form a partition of V.*

Proof. First we prove that the maximal k-edge-connected subgraphs form a partition. Any maximal k-edge-connected subgraph H is always an *induced subgraph* of G. That is, $H = G[S]$ for some $S \subseteq V$. Otherwise, there is an edge $e \in G[S] \setminus E(H)$ and $H \cup e$ is a strict supergraph of H that is k-edge-connected, contradicting the maximality of H.

So if two maximal k-edge-connected graphs $G[A], G[B]$ have intersection i.e., $A \cap B \neq \emptyset$. Since both $G[A]$ and $G[B]$ are k-edge-connected hence $G[A \cup B]$ is also k-edge-connected as $A \cap B \neq \emptyset$. This contradicts that $G[A]$ is maximal k-edge-connected. Hence the sets are disjoint. Each vertex by itself is k-edge-connected hence, should belong to a maximal k-edge-connected subgraph. Thus they form a partition.

Now we prove that it is unique. Assume for contradiction that there exists at least two such partitions. Then there exists at least one vertex v such that it belongs to different sets in the two maximal connected partitions. Again let A, B be the two sets from two different partitions containing v and both $G[A], G[B]$ are k-edge-connected. Using same argument as above we prove that the partition is unique. $\square$

D   Counter example for edge strength estimation

Benczur and Karger [BK02] gave an algorithm that provides estimates $\tilde{k}_e$ for the strength of edges k_e such that $\tilde{k}_e \leq k_e$ for all $e \in E$ and $\sum_{e \in E} \frac{1}{\tilde{k}_e} = O(n)$. These estimates are obtained by running ESTIMATION$(G, 1)$ sub-routine described in Lemma 4.9 of [BK02]. To get a c-approximation for edge strengths, we need that $\tilde{k}_e \geq \frac{k_e}{c}$. However, the second condition given above is different and does not necessarily provide a good approximation guarantee. Below we give a counter example where these estimates are off from the actual values by a large factor.

Consider the following counter-example: a lollipop graph G that has a path of length $n - n^{1/3}$ connected to a clique of size $n^{1/3}$. Note that G contains less than $2(n-1)$ edges. Thus, the procedure ESTIMATION$(G, 1)$ will assign the edge-strength estimate of all edges to 1. However, all edges from the clique have strength at least $n^{1/3}$. So the estimate of edge strength for the edges inside the clique is off by a $n^{1/3}$ factor.

References

[ACL06]   Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006. 4

[AIY13]   Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 909–918, 2013. 3

[AKL+21]   Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Gomory-hu tree in subcubic time. *arXiv preprint arXiv:2111.04958*, 2021. 6

[BK02]   András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *CoRR*, cs.DS/0207078, 2002. 4, 33

[CCPS21]   Ruoxu Cen, Yu Cheng, Debmalya Panigrahi, and Kevin Sun. Sparsification of directed graphs via cut balance. In *International Conference on Automata, Languages and Programming*, 2021. 4

[CDK+21]   Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P Liu, Richard Peng, Mark Sellke, and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1206–1225. SIAM, 2021. 4

[CFK+15]   Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. 5

[CHI+17]   Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F Italiano, Veronika Loitzenbauer, and Nikos Parotsidis. Faster algorithms for computing maximal 2-connected subgraphs in sparse directed graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1900–1918. SIAM, 2017. 3, 6, 7, 17

[CKN20]   Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72. IEEE, 2020. 4, 9

[CLN+22]   Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. Minimum cuts in directed graphs via partial sparsification. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1147–1158. IEEE, 2022. 3

[CLP22]   Ruoxu Cen, Jason Li, and Debmalya Panigrahi. Edge connectivity augmentation in near-linear time, 2022. 31

[CX18]   Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018. 4

[DKL76]   E. Dinic, Alexander Karzanov, and M. Lomonosov. The system of minimum edge cuts in a graph. *In book: Issledovaniya po Diskretnoĭ Optimizatsii (Engl. title: Studies in Discrete Optimizations), A.A. Fridman, ed., Nauka, Moscow, 290-306, in Russian,*, 01 1976. 32

[ES80]   Ya M Erusalimskii and GG Svetlov. Bijoin points, bibridges, and biblocks of directed graphs. *Cybernetics*, 16(1):41–44, 1980. 3

[FGK13]   Fedor V. Fomin, Petr A. Golovach, and Janne H. Korhonen. On the parameterized complexity of cutting a few vertices from a graph. *CoRR*, abs/1304.6189, 2013. 8, 23, 26

[FHHP19]   Wai-Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM Journal on Computing*, 48(4):1196–1223, 2019. 4

[FNY⁺20]   Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2046–2065. SIAM, 2020. 3, 4, 6, 7

[Gab95]   Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995. 3

[GR97]   Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 406–415, 1997. 4, 5

[HKL15]   Monika Henzinger, Sebastian Krinninger, and Veronika Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In *International Colloquium on Automata, Languages, and Programming*, pages 713–724. Springer, 2015. 3

[HKNS15]   Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015. 5, 8, 23

[HRG00]   Monika R Henzinger, Satish Rao, and Harold N Gabow. Computing vertex connectivity: new bounds from old techniques. *Journal of Algorithms*, 34(2):222–250, 2000. 3

[HRW20]   Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM Journal on Computing*, 49(1):1–36, 2020. 4

[Jab16]   Raed Jaberi. On computing the 2-vertex-connected components of directed graphs. *Discrete Applied Mathematics*, 204:164–172, 2016. 3

[JHSZ22]   Han Jiang, Shang-En Huang, Thatchaphol Saranurak, and Tian Zhang. Vertex sparsifiers for hyperedge connectivity, 2022. 4

[Kar93]   David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. 5, 7, 10, 12

[Kar00]   David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000. 3, 10

[KS96]   David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996. 12

[LNP+21]   Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthaworncai. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 317–329, 2021. 3

[Mak88]   Seishi Makino. An algorithm for finding all the k-components of a digraph. *International journal of computer mathematics*, 24(3-4):213–221, 1988. 3

[NSWN17]   Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 950–961. IEEE, 2017. 4

[NSY19]   Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthaworncai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 241–252, 2019. 4, 6

[OR11]   Yaron Orenstein and Dana Ron. Testing eulerianity and connectivity in directed sparse graphs. *Theoretical Computer Science*, 412(45):6390–6408, 2011. 4, 6

[OZ14]   Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1267–1286. SIAM, 2014. 4, 6

[PK09]   Debmalya Panigrahi and David Karger. A near-linear time algorithm for constructing a cactus representation of minimum cuts, 01 2009. 32

[PR02]   Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002. 4, 5

[ST13]   Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013. 4

[ST14]   Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. 4

[SW19]   Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2635. SIAM, 2019. 4

[Tar72]   Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972. 3

[YI10]   Yuichi Yoshida and Hiro Ito. Testing k-edge-connectivity of digraphs. *Journal of systems science and complexity*, 23(1):91–101, 2010. 4, 6

[YI12]   Yuichi Yoshida and Hiro Ito. Property testing on k-vertex-connectivity of graphs. *Algorithmica*, 62(3):701–712, 2012. 4

[ZLY⁺12]   Rui Zhou, Chengfei Liu, Jeffrey Xu Yu, Weifa Liang, Baichen Chen, and Jianxin Li. Finding maximal k-edge-connected subgraphs from a large graph. In *Proceedings of the 15th international conference on extending database technology*, pages 480–491, 2012. 3