Vertex Connectivity in Poly-logarithmic Max-Flows[*]

Jason Li
Carnegie Mellon University
USA
jmli@cs.cmu.edu

Danupon Nanongkai
University of Copenhagen, Denmark, and KTH, Sweden
danupon@gmail.com

Debmalya Panigrahi
Duke University
USA
debmalya@cs.duke.edu

Thatchaphol Saranurak
University of Michigan
USA
thsa@umich.edu

Sorrachai Yingchareonthawornchai
Aalto University
Finland
sorrachai.yingchareonthawornchai@aalto.fi

ABSTRACT

The vertex connectivity of an m-edge n-vertex undirected graph is the smallest number of vertices whose removal disconnects the graph, or leaves only a singleton vertex. In this paper, we give a reduction from the vertex connectivity problem to a set of maxflow instances. Using this reduction, we can solve vertex connectivity in $\tilde{O}(m^{\alpha})$ time for any $\alpha \geq 1$, if there is a m^{α}-time maxflow algorithm. Using the current best maxflow algorithm that runs in $m^{4/3+o(1)}$ time (Kathuria, Liu and Sidford, FOCS 2020), this yields a $m^{4/3+o(1)}$-time vertex connectivity algorithm. This is the first improvement in the running time of the vertex connectivity problem in over 20 years, the previous best being an $\tilde{O}(mn)$-time algorithm due to Henzinger, Rao, and Gabow (FOCS 1996). Indeed, no algorithm with an $o(mn)$ running time was known before our work, *even if we assume an $\tilde{O}(m)$-time maxflow algorithm.*

Our new technique is robust enough to also improve the best $\tilde{O}(mn)$-time bound for *directed* vertex connectivity to $mn^{1-1/12+o(1)}$ time

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**.

KEYWORDS

algorithmic graph theory, vertex connectivity

ACM Reference Format:
Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. 2021. Vertex Connectivity in Poly-logarithmic Max-Flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3406325.3451088

[*]The full version of this paper is available at https://arxiv.org/abs/2104.00104.

1 INTRODUCTION

The *vertex connectivity* of an undirected graph is the size of the minimum vertex cut, defined as the minimum number of vertices whose removal disconnects the graph (or becomes a singleton vertex). Finding the vertex connectivity of a graph is a fundamental problem in combinatorial optimization, and has been extensively studied since the 1960s. It is well-known that the related problem of an s-t vertex mincut, defined as the minimum vertex cut that disconnects a specific pair of vertices s and t, can be solved using an s-t maxflow algorithm. This immediately suggests a natural starting point for the vertex connectivity problem, namely use $O(n^2)$ maxflow calls to obtain the s-t vertex mincuts for all pairs of vertices, and return the smallest among them. It is against this baseline that we discuss the history of the vertex connectivity problem below. Following the literature, we use m, n, and k to respectively denote the number of edges, vertices, and the size of the vertex mincut in the input graph.

In the 60s and 70s, several algorithms [8, 18, 23], showed that for constant values of k, only $O(n)$ maxflow calls suffice, thereby improving the running time for this special case. The first unconditional improvement over the baseline algorithm was obtained by Becker *et al.* [3], when they used $O(n \log n)$ maxflow calls to solve the vertex connectivity problem. The following simple observation underpinned their new algorithm: if one were able to identify a vertex s that is not in the vertex mincut, then enumerating over the remaining $n - 1$ vertices as t in the s-t maxflow calls is sufficient. They showed that they could obtain such a vertex s whp[1] by a random sampling of vertices.

The next round of improvement was due to Linial, Lovász, and Wigderson (LLW) [20] who used an entirely different set of techniques based on matrix multiplication to achieve a running time bound of $O((n^{\omega} + nk^{\omega}) \log n)$, which is $O(n^{1+\omega} \log n)$ in the worst case of $k = \Theta(n)$; here, $\omega \approx 2.37$ is the matrix multiplication exponent. To compare this with the maxflow based algorithms, we note that the maxflow instances generated by the vertex connectivity problem are on unit vertex-capacity graphs, for which an $O(m\sqrt{n})$ algorithm has been known since the celebrated work of Dinic using blocking flows in the 70s [7]. Therefore, LLW effectively improved the running time of vertex connectivity from $O(n^{7/2})$ in the worst case to $O(n^{1+\omega})$.

[1]with high probability

A decade after LLW's work, Henzinger, Rao, and Gabow (HRG) [14] improved the running time further to $O(mn \log n)$ by reverting to combinatorial flow-based techniques. They built on the idea of computing $O(n)$ maxflows suggested by Becker *et al.* [3], but with a careful use of preflow push techniques [11] in these maxflow subroutines, they could amortize the running time of these maxflow calls (similar to, but a more refined version of, what Hao and Orlin had done for the edge connectivity problem a few years earlier [12]). The HRG algorithm remained the fastest unconditional vertex connectivity algorithm before our work.

We also consider the vertex connectivity problem on directed graphs. Here, the goal is to find a smallest set of vertices whose removal ensures that the remaining graph is not strongly connected. The HRG bound of $O(mn \log n)$ [13] generalizes to digraphs, and sets the current record for this problem as well.

In concluding our tour of vertex connectivity algorithms, we note that there has also been a large volume of work focusing on faster algorithms for the special case of *small* k. Nearly-linear time algorithms are known only when $k \leq 2$ [5, 10, 15, 16, 21, 24] until recently when [9, 22] give an $\tilde{O}(mk^2)$-time algorithm[2] for both undirected and directed graphs, which is nearly-linear for $k = \text{polylog}(n)$. Similarly, the question of *approximating* the vertex connectivity of a graph efficiently has received some attention, and a $(1+\epsilon)$-approximation is known in $\tilde{O}(\min\{mk/\epsilon, n^{\omega}/\epsilon^2\})$ time [9, 22] while a worse approximation factor of $O(\log n)$ can be achieved in near-linear time [4]. These two lines of work are not directly related to our paper.

1.1 Our Results

In this paper, we give the following result:

Theorem 1.1 (Main). *Given an undirected graph on m edges, there is a randomized, Monte Carlo vertex connectivity algorithm that makes s-t maxflow calls on unit capacity graphs that cumulatively contain $\tilde{O}(m)$ vertices and $\tilde{O}(m)$ edges, and runs in $\tilde{O}(m)$ time outside these maxflow calls.*

In other words, if maxflow can be solved in m^{α} time on unit capacity graphs, for any $\alpha \geq 1$, then we can solve the vertex connectivity problem in $\tilde{O}(m^{\alpha})$ time. In particular, using the current fastest maxflow algorithm on unit capacity graphs (Kathuria, Liu and Sidford [17]), we get a vertex connectivity algorithm for undirected graphs that runs in $m^{4/3+o(1)}$ time, which strictly improves on the previous best time complexity of $\tilde{O}(mn)$ achieved by the HRG algorithm. Even more ambitiously, if maxflow is eventually solved in $\tilde{O}(m)$ time, as is often conjectured, then our theorem will automatically yield an $\tilde{O}(m)$ algorithm for the vertex connectivity problem, which would resolve the long standing open question by Aho, Hopcroft and Ullman [1] since 1974 up to polylogarithmic factors. In contrast, even with an $\tilde{O}(m)$-time maxflow algorithm, no previous vertex connectivity algorithm achieves an $o(mn)$ running time bound.

We remark that the reduction in the theorem generates instances of the s-t vertex connectivity problem, i.e., a maximum set of vertex-disjoint paths between s and t in an undirected graph, which are solved by a maxflow call via a standard reduction. Also, we note that our algorithm is randomized (Monte Carlo) even if the maxflow subroutines are not. It is an interesting open question to match the running time bounds of this theorem using a deterministic algorithm, or even a Las Vegas one.

[2] $\tilde{O}(f(n)) = O(\text{polylog}(n) f(n))$.

We also generalize our new technique to work *directed* graphs and obtain a significant improvement upon the fastest $\tilde{O}(mn)$-time algorithm by HRG for the directed vertex connectivity problem

Theorem 1.2. *Given a directed graph with m edges and n vertices, there are randomized Monte Carlo vertex connectivity algorithms with*

- $mn^{1-1/12+o(1)}$ *time, or*
- $\tilde{O}(n^2)$ *time assuming that max flow can be solved in near-linear time.*

As the result on directed graphs is obtained by using our new technique in a less efficient way and does not give additional insight, we discuss it in the full version of this paper.

1.2 Technical Overview

Our main technical contribution is a new technique that we call *sublinear-time kernelization* for vertex connectivity. Namely, we show that under certain technical conditions, we can find a subgraph whose size is sublinear in n and preserves the vertex connectivity of the original graph. We use sketching techniques to construct such a subgraph in sublinear time. To the best of our knowledge, all previous techniques require $\Omega(n^3)$ time even in the extremely unbalanced case when the vertex mincut have size $\Omega(n)$, and the smaller side of the mincut contains $O(1)$ vertices. In contrast, sublinear-time kernelization allows us to reduce the problem in this case to maxflow calls of total size $\tilde{O}(m)$ in $\tilde{O}(m)$ time. Below, we elaborate on this new technique and discuss how it fits into the entire vertex connectivity algorithm.

Suppose the vertex mincut of the input graph G is denoted by (L, S, R), where $|L| \leq |R|$ are the two sides of the cut, and $|S| = k$ is the set of vertices whose removal disconnects L from R. For intuitive purposes, let us assume that we know the values of $|L|$ and $|R|$ and $|R| = \Omega(n)$. This allows us to obtain a vertex in R using just $O(\log n)$ samples. From now, we assume that we know a vertex $r \in R$. If we were also able to find a vertex $x \in L$, then we can simply compute an x-r maxflow to obtain a vertex mincut. But, in general, $|L|$ can be small, and obtaining a vertex in L whp requires $\tilde{O}(n/|L|)$ samples. Recall that we promised that the total number of edges in all the maxflow instances that we generate will be $\tilde{O}(m)$. One way of ensuring this would be to run each of the $\tilde{O}(n/|L|)$ maxflow calls on a graph containing only $\tilde{O}(k|L|)$ edges; then, the total number of edges in the max flow instances is $\tilde{O}((n/|L|) \cdot (k|L|)) = \tilde{O}(nk) = \tilde{O}(m)$ since the degree of every vertex is at least k. At first glance, this might sound impossible because the number of edges incident to $N(x)$ is already $\Omega(k^2)$. Nevertheless, our main technical contribution is in showing that in certain cases we can construct a graph H with just $\tilde{O}(k|L|)$ edges that gives us information about the vertex connectivity of G. We call such graph a *kernel*. In achieving this property, we need additional conditions on L and S, specifically on their relative sizes and the degrees of vertices in S. If these conditions do not hold, we give a different algorithm that uses a recent tool called the isolating cut lemma used in the edge connectivity problem [19] (we adapt the tool to vertex connectivity). More specifically, we consider three cases depending on the sizes of L and S_{low}, where

$$S_{\text{low}} = \{v \in S \mid \deg(v) \leq 8k\}.$$

It might not be intuitive now why we need S_{low}. Distinguishing cases using S_{low} is a crucial idea that makes everything fits together. Its role will be more clear in the discussion below. The use of our kernelization is in the last case (Case 3). We now discuss all the cases.

Case 1: Large L (details in Section 4). We first consider the easier case when L is not too small compared to S, i.e. $|L| > k/\text{polylog}(n)$. Consider the vertex set T where each vertex is included in T with probability $\frac{1}{\max\{|L|,k\}}$. Then, with probability at least $1/\text{polylog}(n)$, T contains *exactly one* vertex from L (call it x), no vertex from S, and the remaining vertices are from R. Assume that it is the case by repeating $\text{polylog}(n)$ times. Observe that a vertex mincut separating x from $T \setminus \{x\}$, denoted by $(x, T \setminus \{x\})$-vertex mincut, is a (global) vertex mincut of G.

The *isolating cut lemma* was recently introduced by Li and Panigrahi [19] for solving the edge connectivity problem deterministically. It says that in an undirected graph, given a set of terminal vertices T, we can make maxflow calls to graphs of total size $O(m \log |T|)$ and return for each terminal $t \in T$, the smallest *edge* cut separating t from $T \setminus \{t\}$. In particular, it returns us a $(x, T \setminus \{x\})$-edge mincut. If this lemma worked for vertex cuts, it would return us a $(x, T \setminus \{x\})$-vertex mincut and we would be done. It turns out that the isolating cuts lemma can be adapted to work for vertex connectivity, due to the submodularity property of vertex cuts.

Case 2: Small L, small S_{low} (details in Section 4). From now on, we assume that $|L| < k/\text{polylog}(n)$. Note that for every vertex $x \in L$, all neighbors of x are inside $L \cup S$ and so $\deg(x) \leq |L| + k < 2k$. Let V_{low} be all vertices whose degrees are less than $8k$. We know that $L \subseteq V_{\text{low}}$ and $S_{\text{low}} = S \cap V_{\text{low}}$ by definition. It is also easy to show that $|R \cap V_{\text{low}}| \geq |L|$ (see Claim 4.5). So, if $|S_{\text{low}}| < |L| \cdot \text{polylog}(n)$, then, by sampling from V_{low} instead of V with probability $1/(|L| \,\text{polylog}(n))$, we can obtain a random sample that includes exactly one vertex from L, some vertices from R, and none from S, as in the previous case. In this case, we again can apply the isolating cuts lemma.

Case 3: Small L, large S_{low} (details in Section 3). The above brings us to the crux of our algorithm, where the isolating cuts lemma is no longer sufficient. Namely, L is much smaller than the cut S and S contains many vertices with low degree, i.e.

$$|L| < k/\text{polylog}(n) \text{ and } |S_{\text{low}}| > |L| \cdot \text{polylog}(n). \tag{1}$$

Let us first sample $\tilde{O}(n/|L|)$ vertices; at least one of these vertices is in L whp. Now, for each vertex x in the sample, we will invoke a maxflow instance on $\tilde{O}(k|L|)$ edges that returns the vertex mincut if $x \in L$. This suffices because $\tilde{O}((n/|L|) \cdot (k|L|)) = \tilde{O}(nk)$ can be bounded by $\tilde{O}(m)$, noting that the degree of every vertex is at least k. Thus, we can reduce our problem to the following goal:

> *Given a vertex $x \in L$, describe a procedure to create a maxflow instance on $\tilde{O}(k|L|)$ edges that returns a vertex mincut.*

In other words, assuming that we have a vertex $x \in L$, we want to construct a small graph H and two vertices s and t in H such that the (s,t)-maxflow in H tells us about the vertex mincut in the original input graph. The graph H corresponds to the concept of *kernel* in parameterized algorithms. A challenge is that it is not clear if a small kernel exists for vertex connectivity; it is not even clear if it is possible to reduce the number of edges at all. The entire description below aims to show that it is possible to reduce the number of edges to $\tilde{O}(k|L|)$. We ignore the time complexity for this process for a moment.

The key step is to define the following set T_x. First, let T be a set such that every vertex is in T with probability $1/|L|$. Then, T_x is defined from T by *excluding* x and its neighbors, i.e. $T_x = T \setminus N_G[x]$, where $N_G[x] = N_G(x) \cup \{x\}$ and $N_G(v)$ denotes the set of neighbors of v. (We drop G when the context is clear). We exploit a few properties of T_x. First, we claim that $T_x \subseteq R$ with $\Omega(1)$ probability. To see this, note that $N[x] \subseteq L \cup S$ for any $x \in L$. Since $|N[x]| > k$ but $|L \cup S| \leq |L| + k$, it must be the case that

$$|(L \cup S) \setminus N[x]| < |L|. \tag{2}$$

Now, $T_x \subseteq R$ iff none of vertices from $(L \cup S) \setminus N[x]$ is sampled to T. As $|(L \cup S) \setminus N[x]| < |L|$ and the sampling probability is $1/|L|$, so $T_x \subseteq R$ with $\Omega(1)$ probability.

From now we assume that $T_x \subseteq R$. Consider contracting vertices in T_x into a single node t_x. Since $T_x \subseteq R$, an (x, t_x)-maxflow call would return a vertex mincut of the original graph. However, the contracted graph might still contain too many edges. To resolve this issue, we make the following important observations:

(1) any vertex v neighboring to both x and t_x must be in S, and
(2) there exists a collection of k vertex disjoint paths between x and t_x where each path contains exactly one neighbor of x and exactly one neighbor of t_x.

The observations above simply follow from the fact that x and t_x are on the different side of the vertex mincut. The first observation allows us to remove all common neighbors of x and t_x and add them back to the vertex mincut later. The second observation allows us to remove all edges between neighbors of x and all edges between neighbors of t_x without changing the (x, t_x) vertex connectivity. Further, after all these removals, neighbors of t_x of degree one (i.e. they are adjacent only to t_x) can be removed without changing the (x, t_x) vertex connectivity. Interestingly, these removals are already enough for us to show that there are $\tilde{O}(k|L|)$ vertices and edges left!

Small kernel. We call the remaining graph from above a *kernel* and denote it by H. We now show that H contains $\tilde{O}(k|L|)$ edges whp. Note that H consists of the terminals x and t_x, disjoint sets $N_x \triangleq N_H(x)$ and $N_t \triangleq N_H(t_x)$, and all other vertices in a set that we call F' (for "far"). We illustrate this in Figure 1.

Recall that we have already discarded all internal edges in N_x and N_t; hence, we have three types of edges in H:

(E1) edges in $N_x \times (F' \cup N_t)$, i.e. edges with one endpoint in N_x and the other in F' or N_t,
(E2) edges in $F' \times (F' \cup N_t)$, i.e. edges with one endpoint in F' and the other in F' or N_t, and
(E3) edges incident to terminals x and t_x.

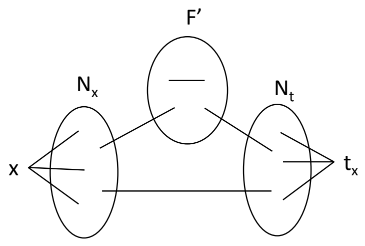

Figure 1: Kernel H of graph $G = (V, E)$.

We count the number of edges in (E1) and (E2) by charging them to its endpoint in N_x and F' respectively. We will show that there are $\tilde{O}(k|L|)$ such edges in total. It then follows that there are $\tilde{O}(k|L|)$ edges in (E3), since there are at most $k + |L|$ edges incident to x and each vertex in N_t must be incident to some edge in (E1) or (E2) (otherwise, we would have already deleted such vertex). The claimed $\tilde{O}(k|L|)$ bound on the number of edges in (E1) or (E2) follows immediately once we show that whp

(a) every vertex in $N_x \cup F'$ is charged by $\tilde{O}(L)$ edges, and
(b) there are $O(k)$ vertices in $N_x \cup F'$.

To prove (a), consider any vertex $v \neq x$ in G with $\deg_{G \setminus N[x]}(v) > |L| \cdot \text{polylog}(n)$, i.e. v has many neighbors outside $N_G[x]$, the neighborhood of x. Then, one of these neighbors must have been sampled to T whp, and would be retained in T_x. This implies that such v is in N_t whp. This implies further that every vertex $v \in N_x \cup F'$ has at most $|L| \cdot \text{polylog}(n)$ edges to vertices in $F' \cup N_t$ (since the latter vertices are all outside of $N_G[x]$). This establishes (a).

To prove (b), first note that $|N_x| < |L| + |S| < 2k$ since $N_x \subseteq N_G(x) \subseteq L \cup S$; so, it is left to show that $|F'| = O(k)$. The key statement that we need is that

$$\Omega(|S_{\text{low}}|) \text{ neighbors of every vertex } v \in F' \text{ are in } S_{\text{low}}. \quad (3)$$

Given this, as we know that vertices in S_{low} are incident to $O(k|S_{\text{low}}|)$ edges in total, we have $|F'| = O(\frac{k|S_{\text{low}}|}{|S_{\text{low}}|}) = O(k)$ as desired. To prove (3), we essentially use the following facts (for precise quantities, see Figure 2).

(b1) There are less than $k + |L|$ vertices in N_x, i.e. $|N_x| < k + |L|$ (we just proved this above).
(b2) All but $|L|$ of vertices in S_{low} are in N_x. This follows from (2).
(b3) Whp, every vertex in F' has at least $k - |L| \,\text{polylog}\, n$ neighbors in N_x. This follows from the argument in the proof of (a).

This means that each $v \in F'$ has at least the following number of neighbors in S_{low}:

$$k - |L| \,\text{polylog}(n) - (|N_x| - (|S_{\text{low}}| - |L|)) \geq$$
$$k - |L| \,\text{polylog}(n) - (k + |L|) + (|S_{\text{low}}| - |L|) = \Omega(|S_{\text{low}}|)$$

where the last equality holds as k's cancel each other and $|S_{\text{low}}|$ dominates other terms. This is the crucial place where we need that S_{low} is large as stated in (1). Without this guarantee, we could not have bounded the size of H and this explains the reason why we need to introduce Case 2 above. This completes the proof of (b).

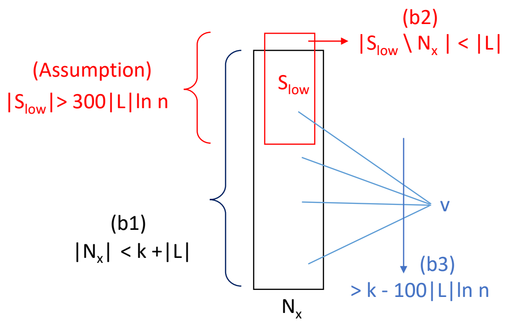

Figure 2: Facts (b1), (b2) and (b3).

Building kernels in sublinear time. So far, we only bound the size of the kernel H. Below, we discuss how to actually build it in sublinear time. Note that we will end up building a *subgraph* of H instead of H.

Consider the following BFS-like process: Initialize the queue of the BFS with vertices in N_x. Whenever v is visited, if $v \notin N_t$, we add $N(v) \setminus N[x]$ into the queue. This process will explore the "relevant" subgraph of $H \setminus \{x, t_x\}$ because the part that is not even reached from N_x cannot be relevant to (x, t_x)-vertex connectivity in H and so we ignore it. The kernel graph that our algorithm actually constructs is obtained by adding $E(x, N_x)$ and $E(N_t, t_x)$ into the above explored subgraph of H. Our goal is to implement this process in $\tilde{O}(k|L|)$ time. There are two main challenges.

(c1) For all $O(k)$ visited vertices $v \notin N_t$, we must list $N(v) \setminus N[x]$ in $\tilde{O}(|L|)$ time. Note that simply listing neighbors of v already takes $\deg(v) \geq k$ time which is too expensive.
(c2) For all $\tilde{O}(k|L|)$ visited vertices v, we must test if $v \in N_t$ (i.e. if its neighborhood in G overlaps with T_x) in polylog(n) time.

We address both challenges by implementing our BFS-like process based on linear sketches from the streaming algorithm community, and so we call our technique *sketchy search*. The key technique for (c1) is *sparse recovery sketches*: An s-sparse recovery sketch *linearly* maps a vector $\vec{a} \in \mathbb{Z}^n$ to a smaller vector $\text{sk}_s(\vec{a}) \in \mathbb{Z}^{\tilde{O}(s)}$ in $\tilde{O}(\|\vec{a}\|_0)$ time so that, if $\vec{a}$ has at most s non-zero entries, then we can recover $\vec{a}$ from $\text{sk}_s(\vec{a})$ in $\tilde{O}(s)$ time. For any vertex v, let $\mathbb{1}_{N(v)}$ and $\mathbb{1}_{N[v]}$ be the indicator vectors of $N(v)$ and $N[v]$ respectively. We observe two things: (1) non-zero entries in $\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}$ correspond to the symmetric difference $N(v) \triangle N[x]$, and (2) $|N(v) \triangle N[x]| = \Theta(|N(v) \setminus N[x]| + |L|)$ (formally proved in (6)).

This motivates the following algorithm. Set $s \leftarrow |L| \,\text{polylog}(n)$ and precompute $\text{sk}_s(\mathbb{1}_{N(v)})$ and $\text{sk}_s(\mathbb{1}_{N[v]})$ for all vertices v. This takes $\sum_v \tilde{O}(\deg(v)) = \tilde{O}(m)$ time. Now, given any v, we can compute in $\tilde{O}(s)$ time $\text{sk}_s(\mathbb{1}_{N(v)}) - \text{sk}_s(\mathbb{1}_{N[x]}) = \text{sk}_s(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})$, where the equality is because the map is linear. If $v \notin N_t$, then we have argued previously that $|N(v) \setminus N[x]| \leq |L| \,\text{polylog}(n)$ and so $\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}$ has at most s non-zero entries. Thus, from $\text{sk}_s(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})$ we can obtain $N(v) \triangle N[x]$ which contains the desired set $N(v) \setminus N[x]$ in $\tilde{O}(s) = \tilde{O}(|L|)$ time.

To address (c2), recall that if $|N(v) \setminus N[x]| \geq |L| \operatorname{polylog}(n)$, then $v \in N_t$. This condition can be checked in $O(\log n)$ time using another linear sketch (called *norm estimation*) for estimating $\|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_2$ which is proportional to $|N(v) \setminus N[x]| + |L|$. However, there can still be some $v \in N_t$ but $|N(v) \setminus N[x]| \leq |L| \operatorname{polylog}(n)$. Fortunately, there are only $O(k)$ such vertices in N_t (using the same argument that bounds $|F| = O(k)$ in the proof of (b)). For those vertices, we have enough time to list $N(v) \setminus N[x]$ in $\tilde{O}(|L|)$ time using the sparse recovery sketches and check if $t_x \in N(v) \setminus N[x]$, which holds iff $v \in N_t$.

Remarks: Note that all we have established is that in any one of the many invocations of the sampling processes being used, we will return a vertex mincut. For the sake of correctness, we carefully argue later that in all the remaining calls, i.e., when sampling does not give us the properties we desire, we actually return some vertex cut in the graph. This allows us to distinguish the vertex mincut from the other cuts returned, since it has the fewest vertices.

2 PRELIMINARIES

Let $G = (V, E)$ be an undirected graph. For any set T of vertices, we let $N_G(T) = \{v \notin T \mid \exists u \in T \text{ and } (u, v) \in E\}$ and $N_G[T] = T \cup N_G(T)$. If $T = \{v\}$, we also write $N(v)$ and $N[v]$. The set $E_G(A, B)$ denote the edges with one endpoint in A and another in B. If $A = \{v\}$, we write $E_G(v, B)$. We usually omit the subscript when G refers to the input graph. For any graph H, we use $V(H)$ to denote the set of vertices of H, and $E(H)$ to denote the set of edges of H. Whenever we contract a set of vertices in a graph, we remove all parallel edges to keep the graph simple. This is because parallel edges does not affect vertex connectivity.

A *vertex cut* (L, S, R) of a graph $G = (V, E)$ is a partition of V such that $L, R \neq \emptyset$ and $E_G(L, R) = \emptyset$. We call S the corresponding *separator* of (L, S, R). The size of a vertex cut is the size of its separator $|S|$. A vertex cut (L, S, R) is an (s, t)-vertex cut if $s \in L$ and $t \in R$. A vertex mincut is a vertex cut with minimum size. An (s, t)-vertex mincut is defined analogously. If (L, S, R) is an (s, t)-vertex mincut, we say that S is an (s, t)-min-separator. For disjoint subsets $A, B \subset V$, a vertex cut (L, S, R) is an (A, B)-vertex cut if $A \subseteq L$ and $B \subseteq R$. (A, B)-separator and (A, B)-min-separator are defined analogously. Throughout the paper, we assume wlog that

$$|L| \leq |R|.$$

In Section 3.3, we will employ the following standard linear sketching techniques. We state the known results in the form which is convenient for us below. We prove them in the Appendix. In both theorems below, an input vector v is represented in a sparse representation, namely a list of (index,value) of non-zero entries. The number of non-zero entries of v is denoted as $\|v\|_0$.

Theorem 2.1 (Norm Estimation). *For any number n, there is an algorithm that preprocesses in $\tilde{O}(n)$ time and then, given any vector $v \in \mathbb{R}^n$, return a sketch $\operatorname{sk}_{\ell_2}(v) \in \mathbb{R}^{O(\log n)}$ in $\tilde{O}(\|v\|_0)$ time such that $\|v\|_2 \leq \|\operatorname{sk}_{\ell_2}(v)\|_2 \leq 1.1\|v\|_2$ whp. Moreover, the sketch is* linear, *i.e.* $\operatorname{sk}_{\ell_2}(u + v) = \operatorname{sk}_{\ell_2}(v) + \operatorname{sk}_{\ell_2}(u)$ *for any $u, v \in \mathbb{R}^n$.*

Theorem 2.2 (Sparse Recovery). *For any numbers n and s, there is an algorithm that preprocesses in $\tilde{O}(s)$ time and then, given any vector $v \in \{-1, 0, 1\}^n$, return a sketch $\operatorname{sk}_s(v) \in \mathbb{Z}^{\tilde{O}(s)}$ in $\tilde{O}(\|v\|_0)$ time and guarantees the following whp (as long as the number of recovery operations is* $\operatorname{poly}(n)$*).*[3]

- *If $\|v\|_0 \leq s$, then we can recover v from $\operatorname{sk}_s(v)$ in $\tilde{O}(s)$ time. (More specifically, we obtain all non-zero entries of v together with their indices).*
- *Otherwise, if $\|v\|_0 > s$, then the algorithm returns $\perp$.*

Moreover, the sketch is linear, *i.e.* $\operatorname{sk}_s(u + v) = \operatorname{sk}_s(v) + \operatorname{sk}_s(u)$ *for any $u, v \in \mathbb{Z}^n$.*

3 USING SUBLINEAR-TIME KERNELIZATION

We say that a vertex cut (L, S, R) is a *k-scratch* if $|S| < k$, $|L| \leq k/(100 \log n)$ and $|S_{\text{low}}| \geq 300|L| \ln n$ where $|S_{\text{low}}| = \{v \in S \mid \deg(v) \leq 8k\}$. This kind of cuts is considered in Case 3 of Section 1.2. In this section, we show that if a graph has k-scratch, then we can return some vertex cut of size less than k.

Lemma 3.1. *There is an algorithm that, given an undirected graph G with n vertices and m edges and a parameter k, returns a vertex cut (L, S, R) in G. If G has a k-scratch, then $|S| < k$ w.h.p. The algorithm makes s-t maxflow calls on unit-vertex-capacity graphs with $\tilde{O}(m)$ total number of vertices and edges and takes $\tilde{O}(m)$ additional time.*

Throughout this section, we assume that minimum degree of G is at least k, otherwise the lemma is trivial. The rest of this section is for proving the above lemma. Assume that a k-scratch exists, let (L, S, R) be an arbitrary k-scratch. We start with a simple observation which says that, given a vertex $x \in L$, the remaining part of $L \cup S$ outside $N[x]$ has size at most $|L|$ which is potentially much smaller than k.

Proposition 3.2. *For any $x \in L$, $|(L \cup S) \setminus N[x]| < |L|$.*

Proof. Note that $N[x] \subseteq L \cup S$ as $x \in L$. The claim follows because $|L \cup S| < |L| + k$ and $|N[x]| > k$ as the minimum degree is at least k. □

We will use $\tilde{\ell}$ as an estimate of $|L|$ (since $|L|$ is actually unknown to us). Let T be obtained by sampling each vertex with probability $1/(8\tilde{\ell})$. Let $T_x \triangleq T \setminus N[x]$ for any $x \in V$. Below, we show two basic properties of T.

Proposition 3.3. *For any $x \in V$, we have the following whp.*

$$\text{For every } v \notin N[T_x],\ |N(v) \setminus N[x]| \leq 40\tilde{\ell} \ln n \qquad (4)$$

Proof. It suffices to prove that, for any $v \in V$, if $|N(v) \setminus N[x]| > 40\tilde{\ell} \ln n$, then v is incident to T_x whp. Indeed, v is not incident to T_x is with probability at most $(1 - \frac{1}{8\tilde{\ell}})^{|N(v) \setminus N[x]|} < n^{-5}$. □

Proposition 3.4. *Suppose $|L|/4 \leq \tilde{\ell} \leq |L|$. For each $x \in L$, $\emptyset \neq T_x \subseteq R$ with constant probability.*

Proof. Note that $\emptyset \neq T_x \subseteq R$ iff none of vertices from $(L \cup S) \setminus N[x]$ is sampled to T *and* some vertex from $R \setminus N[x]$ is sampled to T. Observe that $|(L \cup S) \setminus N[x]| < |L|$ by Proposition 3.2 and $|R \setminus N[x]| = |R| \geq |L|$.

To rephrase the situation, we have two disjoint sets A_1 and A_2 where $|A_1| < |L|$ and $|A_2| \geq |L|$ and each element is sampled

[3]The algorithm works for larger range $[-\operatorname{poly}(n), \operatorname{poly}(n)]$ of integers, but the range $\{-1, 0, 1\}$ is sufficient for our purpose.

with probability $\frac{1}{8\widetilde{\ell}} \in [\frac{1}{8|L|}, \frac{1}{2|L|}]$. No element is A_1 is sampled with probability at least $(1 - \frac{1}{2|L|})^{|L|} \geq 0.5$. Some element in A_2 is sampled with probability at least $1 - (1 - \frac{1}{8|L|})^{|L|} \geq 1 - e^{1/8} \geq 0.1$. As both events are independent, so they happen simultaneously with probability at least 0.05. That is, $\emptyset \neq T_x \subseteq R$ with probability at least 0.05. □

For intuition, let us see why these observations above can be useful. Suppose $\widetilde{\ell} \approx |L|$ and we can guess $x \in L$. Then, Proposition 3.4 says that $\emptyset \neq T_x \subseteq R$ with some chance. This implies that any (x, T_x)-vertex mincut must have size at most $|S| < k$ and we so could return it as the answer of Lemma 3.1. However, directly computing a (x, T_x)-vertex mincut in G is too expensive. One initial idea is to contract T_x into a single vertex t_x (denoted the contracted graph by $G'_{x,T}$) and then compute a (x, t_x)-vertex mincut in the smaller graph $G'_{x,T}$. Now, Equation (4) precisely means that, for every vertex v in $G'_{x,T}$ not incident to the sink t_x and not t_x itself, the neighbor set of v outside $N[x]$ is at most $40\widetilde{\ell}\ln n$.

This fact that many vertices in $G'_{x,T}$ has "degree outside $N[x]$" at most $\tilde{O}(\widetilde{\ell})$ is the key structural property used for constructing a small graph $G_{x,T}$ with $\tilde{O}(k\widetilde{\ell})$ edges such that a (x, t_x)-vertex mincut in $G_{x,T}$ corresponds to a (x, T_x)-vertex mincut in G. The graph $G_{x,T}$ fits into the notion of *kernel* in parameterized algorithms and hence we call it a *kernel* graph. The graph $G_{x,T}$ will be obtained from $G'_{x,T}$ by removing further edges and vertices.

The following key lemma further shows that, given a set X, we can build the kernel graph $G_{x,T}$ for each $x \in X$ in $\tilde{O}(k\widetilde{\ell})$ time, which is sublinear time.

Lemma 3.5 (Sublinear-time Kernelization). *Let G and k be the input of Lemma 3.1. Let $\widetilde{\ell} \leq k/(100\log n)$. Let X be a set of vertices. Let T be obtained by sampling each vertex with probability $1/(8\widetilde{\ell})$ and $T_x \triangleq T \setminus N[x]$ for any $x \in X$. There is an algorithm that takes total $\tilde{O}(m + |X|k\widetilde{\ell})$ time such that, whp, for every $x \in X$, either*

- *outputs a* kernel *graph $G_{x,T}$ containing x and t_x as vertices where $|E(G_{x,T})| = O(k\widetilde{\ell}\log n)$ together with a vertex set $Z_{x,T}$ such that a set Y is a (x, t_x)-min-separator in $G_{x,T}$ if and only if $Y \cup Z_{x,T}$ is a (x, T_x)-min-separator in G, or*
- *certifies that $T_x = \emptyset$ or that there is no k-scratch (L, S, R) where $\emptyset \neq T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2, |L|]$ and $x \in L$.*

Below, we prove the main result of this section using the key lemma (Lemma 3.5) above.

Proof of Lemma 3.1. For each $i = 1, \ldots, \lg(k/(100\log n))$, let $\widetilde{\ell}^{(i)} = 2^i$. Let $T^{(i,1)}, \ldots, T^{(i,O(\log n))}$ be independently obtained by sampling each vertex with probability $1/(8\widetilde{\ell}^{(i)})$ and let $X^{(i)}$ be a set of $O(n\log n/\widetilde{\ell}^{(i)})$ random vertices. We invoke Lemma 3.5 with parameters $(\widetilde{\ell}^{(i)}, X^{(i)}, T^{(i,j)})$ for each $j = 1, \ldots, O(\log n)$. For each $x \in X^{(i)}$ where the kernel graph $G_{x,T^{(i,j)}}$ is returned, we find (x, t_x)-min-separator in $G_{x,T^{(i,j)}}$ by calling the maxflow subroutine and obtain a $(x, T_x^{(i,j)})$-min-separator in G by combining it with $Z_{x,T^{(i,j)}}$. Among all obtained $(x, T_x^{(i,j)})$-min-separators (over all i, j, x), we return the one with minimum size as the answer of Lemma 3.1. Before returning such cut, we verify in $O(m)$ time that it is indeed a vertex cut in G (as Lemma 3.5 is only correct whp.). If not, we return an arbitrary vertex cut of G (e.g., $N_G(v)$ where v is a minimum degree vertex). Also, if there is no graph $G_{x,T^{(i,j)}}$ returned from Lemma 3.5 at all, then we return an arbitrary vertex cut of G as well.

For correctness, it is clear that the algorithm always returns some vertex cut of G with certainty. Now, suppose that G has a k-scratch (L, S, R). Consider i such that $\widetilde{\ell}^{(i)} \in [|L|/2, |L|]$. Then, there exists $x \in X^{(i)}$ where $x \in L$ whp. Also, by Proposition 3.4, there is j where $\emptyset \neq T_x^{(i,j)} \subseteq R$ whp. Therefore, a $(x, T_x^{(i,j)})$-min-separator must have size less than k and we must obtain it by Lemma 3.5.

Finally, we bound the running time. As we call Lemma 3.5 $O(\log^2 n)$ times, this takes time at most

$$\tilde{O}(\sum_i (m + |X_i|k\widetilde{\ell}^{(i)})) = \tilde{O}(m + \sum_i \frac{n}{\widetilde{\ell}^{(i)}} k\widetilde{\ell}^{(i)}) = \tilde{O}(m).$$

The total size of maxflow instances is at most

$$\sum_{i,j} \sum_{x \in X^{(i)}} |E(G_{x,T^{(i,j)}})| = \sum_i \tilde{O}(k\widetilde{\ell}^{(i)} \cdot (n/\widetilde{\ell}^{(i)})) = \tilde{O}(m).$$

This completes the proof.

Organization of this section. We formally show the existence of $G_{x,T}$ in Section 3.2 (using the help of reduction rules shown in Section 3.1). Next, we give efficient data structures for efficiently building each $G_{x,T}$ in Section 3.3 and then use them to finally prove Lemma 3.5 in Section 3.4.

3.1 Reduction Rules for (s, t)-Vertex Mincut

In this section, we describe a simple and generic "reduction rules" for reducing the instance size of the (s, t)-vertex mincut problem. We will apply these rules in Section 3.2. Let $H = (V, E)$ be an arbitrary simple graph with source s and sink t where $(s, t) \notin E$.

The first rule helps us identify vertices that must be in every mincut and hence we can remove them. More specifically, we can always remove common neighbors of both source s and sink t and work on the smaller graph.

Proposition 3.6 (Identify rule). *Let $H' = H \setminus N(s) \cap N(t)$. Then, S' is an (s, t)-min-separator in H' iff $S = S' \cup (N(s) \cap N(t))$ is an (s, t)-min-separator in H'.*

Proof. Let $v \in N(s) \cap N(t)$. Observe that v is contained in *every* (s, t)-separator in H. So S' is an (s, t)-min-separator in $H \setminus \{v\}$ iff $S' \cup \{v\}$ is an (s, t)-min-separator in H. The claim follows by applying the same argument on another vertex $v' \in N(s) \cap N(t) \setminus \{v\}$ in $H \setminus \{v\}$ and repeating for all vertices in $N(s) \cap N(t)$. □

The second rule helps us "filter" useless edges and vertices w.r.t. (s, t)-vertex connectivity.

Proposition 3.7 (Filter rule). *There exists a maximum set of (s, t)-vertex-disjoint paths $P_1, \ldots, P_z$ in H such that no path contains edges or vertices that satisfies any of the following properties.*

(1) an edge e with both endpoints in $N(s)$ or both in $N(t)$.
(2) a vertex v where $t \in N(v) \subseteq N[t]$.
(3) a vertex v where s cannot reach v in $H \setminus N[t]$.

Therefore, by maxflow-mincut theorem, the size of (s, t)-vertex mincut in H stays the same even after we remove these edges and vertices from H.

Proof. (1): Suppose there exists $P_i = (s, \ldots, u_1, u_2, \ldots, t)$ where $(u_1, u_2) \in N(s) \times N(s)$. We can replace P_i with $P'_i = (s, u_2, \ldots, t)$ which is disjoint from other paths P_j. The argument is symmetric for $N(t)$.

(2): Let v be such that $t \in N(v) \subseteq N[t]$. We first apply rule (1). This means that $N(v) = \{t\}$. It is clear that there is no simple s-t path through v.

(3): Suppose $v \in P_i$. There must exist $t' \in N(t)$ where $P_i = (s, \ldots, t', \ldots, v, \ldots, t)$ because s could not reach v if $N[t]$ was removed. Then, we can replace P_i with $P'_i = (s, \ldots, t', t)$ which does not contain v and is still disjoint from other paths P_j. □

3.2 Structure of Kernel $G_{x,T}$

Let G and k be the input of Lemma 3.1. Throughout this section, we fix a vertex x and a vertex set $T \neq \emptyset$. The goal of this section is to show the existence of the graph $G_{x,T}$ as needed in Lemma 3.5 and state its structural properties which will be used later in Sections 3.3 and 3.4.

Recall that $T_x \triangleq T \setminus N[x]$ and also the graph $G'_{x,T}$ is obtained from G by contracting T_x into a *sink* t_x. We call x a *source*. Clearly, every (x, t_x)-vertex cut in $G'_{x,T}$ is a (x, T_x)-vertex cut in G.

Let $G_{x,T}$ be obtained from $G'_{x,T}$ by first applying Identify rule from Proposition 3.6. Let $Z_{x,T} = N_{G'_{x,T}}(x) \cap N_{G'_{x,T}}(t_x)$ be the set removed from $G'_{x,T}$ by Identify rule. We also write $Z = Z_{x,T}$ for convenience. After removing $Z_{x,T}$, we apply Filter rule from Proposition 3.7. We call the resulting graph the *kernel* graph $G_{x,T}$. The reduction rules from Propositions 3.6 and 3.7 immediately imply the following.

Lemma 3.8. *Any set Y is a (x, t_x)-min-separator in $G_{x,T}$ iff $Y \cup Z_{x,T}$ is a (x, T_x)-min-separator in G.*

Let us partition vertices of $G_{x,T}$ as follows. Let $N_x = N_{G_{x,T}}(x)$ be the neighborhood of source x. Let $N_t = N_{G_{x,T}}(t_x)$ be the neighbor of sink t_x. Note that N_x and N_t are disjoint by Identify rule. Let $F = V(G_{x,T}) \setminus (N_x \cup N_t \cup \{x, t_x\})$ be the rest of vertices, which is "far" from both x and t_x. By Filter rule(1), $G_{x,T}$ has no internal edges inside N_x nor N_t. So, the edges of $G_{x,T}$ can be partitioned to

$$\begin{aligned} E(G_{x,T}) = & E_{G_{x,T}}(x, N_x) \cup E_{G_{x,T}}(N_x, F \cup N_t) \\ & \cup E_{G_{x,T}}(F, F \cup N_t) \cup E_{G_{x,T}}(N_t, t_x). \qquad (5) \end{aligned}$$

Below, we further characterize each part in $G_{x,T}$ in term of sets in $G = (V, E)$. See Figure 3 for illustration.

Lemma 3.9. *We have the following:*

(1) $Z = N(x) \cap N(T_x)$ and $N_x = N(x) \setminus N(T_x)$. So, Z and N_x partition $N(x)$.
(2) $F = \{v \in V \setminus (N[x] \cup N[T_x]) \mid v$ is reachable from N_x in $G \setminus N[T_x]\}$.
(3) $N_t = \{v \in N(T_x) \setminus N[x] \mid v$ is incident to F or $N_x\}$

Proof. (1): Observe that $N'_t \triangleq N_{G'_{x,T}}(t_x) = N(T_x)$ and $N'_x \triangleq N_{G'_{x,T}}(x) = N(x)$ because $G'_{x,T}$ is simply G after contracting T_x. So $Z = N(x) \cap N(T_x)$. After removing Z from $G'_{x,T}$ via Identify rule, the remaining neighbor set of x is $N(x) \setminus N(T_x)$. Since Filter rule never further removes any neighbor of the source x, we have $N_x = N(x) \setminus N(T_x)$.

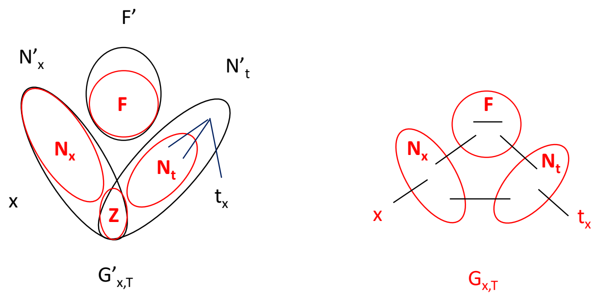

Figure 3: $G'_{x,T}$ (left) is obtained from G by contracting T_x. $G_{x,T}$ (right) is obtained from $G'_{x,T}$ by applying Identify rule and Filter rule, respectively. The set Z is identified using Identify rule. Note that $N_x = N'_x \setminus Z$. The vertices in $F' \setminus F$ cannot be reached from N_x in $G'_{x,T} \setminus N_{G'_{x,T}}(t_x)$. The vertices in $N'_t \setminus (N_t \cup Z)$ have edges only to N_t or t_x.

(2): Let $F' = V \setminus (N[x] \cup N[T_x])$. Note that F' is precisely the set of vertices in $G'_{x,T}$ that is not dominated by source x or sink t_x. As F is an analogous set for $G_{x,T}$ and $G_{x,T}$ is a subgraph of $G'_{x,T}$, we have $F \subseteq F'$. Observe that only Filter rule(3) may remove vertices from F'. (Identify rule and Filter rule(1,2) do not affect F'). Now, Filter rule(3) precisely removes vertices in F' that are not reachable from source x in $G'_{x,T} \setminus N_{G'_{x,T}}[t_x]$. Equivalently, it removes those that are not reachable from N_x in $G \setminus N[T_x]$. Hence, the remaining part of F' in $G_{x,T}$ is exactly F.

(3): Let $N''_t = N(T_x) \setminus N[x]$. N''_t precisely contains neighbors of sink t_x in $G'_{x,T}$ outside $N[x]$. As N_t is the neighbor set of t_x in $G_{x,T}$ and $Z = N(x) \cap N(T_x)$ is removed from $G_{x,T}$, we have that $N_t \subseteq N''_t$. Now, only Filter rule(2) may remove vertices from N''_t, and it precisely removes those that are not incident to F or N_x. Therefore, the remaining part of N''_t in $G_{x,T}$ is exactly N_t. □

Next, we show we bound the size of $|E(G_{x,T})|$.

Lemma 3.10. *Suppose Equation (4) holds. Then, $|E(G_{x,T})| = O((k + |F|)\widetilde{\ell} \log n)$.*

Proof. We bound $|E(G_{x,T})|$ by bounding each term in Equation (5). First, $|E(x, N_x)| \leq |N_x| \leq |L \cup S| \leq 2k$. Next, for any $v \in V(G_{x,T}) \setminus N_t$, we have $|E_{G_{x,T}}(v, F \cup N_t)| \leq 40\widetilde{\ell} \ln n$ by Equation (4). So $|E_{G_{x,T}}(N_x, F \cup N_t) \cup E_{G_{x,T}}(F, F \cup N_t)| \leq (|N_x| + |F|) \cdot 40\widetilde{\ell} \ln n = O((k + |F|)\widetilde{\ell} \log n)$ because $|N_x| \leq 2k$. Lastly, each vertex in N_t must have a neighbor in either N_x or F by Lemma 3.9(3). So $|E_{G_{x,T}}(N_t, t_x)| \leq |N_t| \leq |E_{G_{x,T}}(N_t, N_x \cup F)|$ which can be charged to either $E_{G_{x,T}}(N_x, F \cup N_t)$ or $E_{G_{x,T}}(F, F \cup N_t)$ whose size are $O((k + |F|)\widetilde{\ell} \log n)$. To conclude, $|E(G_{x,T})| = O((k + |F|)\widetilde{\ell} \log n)$. □

Since $|E(G_{x,T})|$ depends on $|F|$, we will bound $|F|$ as follows. We show that the set $F'_{\text{relax}} = \{v \in V \setminus N[x] \mid |N(v) \setminus N[x]| \leq 100\widetilde{\ell} \log n\}$ is a superset of F and then bound $|F'_{\text{relax}}|$. The bound on $|F'_{\text{relax}}|$ will be also used later in Proposition 3.18 for proving efficiency of our algorithm.

Lemma 3.11. *Suppose Equation* (4) *holds and there is a k-scratch (L, S, R) where $\emptyset \neq T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2, |L|]$, and $x \in L$. Then, we have that $F \subseteq F'_{\text{relax}}$ and $|F'_{\text{relax}}| \leq 16k$.*

Proof. Let $F' = V \setminus (N[x] \cup N[T_x])$, which is precisely the set of vertices in $G'_{x,T}$ that is not dominated by source x or sink t_x. We have $F \subseteq F'$ because $G_{x,T}$ is a subgraph of $G'_{x,T}$ and we have $F' \subseteq F'_{\text{relax}}$ because of Equation (4). Now, we bound $|F'_{\text{relax}}|$. Recall that the definition of a k-scratch (L, S, R) says that $|S_{\text{low}}| \geq 300|L| \ln n$ where $|S_{\text{low}}| = \{v \in S \mid \deg(v) \leq 8k\}$. Let $N_{\text{low}} = N(x) \cap S_{\text{low}}$. We will show that $|F'_{\text{relax}}| \cdot |S_{\text{low}}|/2 \leq |E_G(N_{\text{low}}, F'_{\text{relax}})| \leq 8k|S_{\text{low}}|$. This would imply that $|F'_{\text{relax}}| \leq 16k$ and complete the proof of the claim.

The upper bound on $|E_G(N_{\text{low}}, F'_{\text{relax}})|$ follows because $N_{\text{low}} \subseteq S_{\text{low}}$ and each vertex in S_{low} has degree at most $8k$. To prove the lower bound on $|E_G(N_{\text{low}}, F'_{\text{relax}})|$, we will actually show that for every $v \in F'_{\text{relax}}$, $|E_G(v, N_{\text{low}})| \geq |S_{\text{low}}|/2$. We have

$$\begin{aligned}|E_G(v, N_{\text{low}})| &\geq |E_G(v, N(x))| - |N(x) \setminus N_{\text{low}}| \\ &\geq k - 100\widetilde{\ell} \ln n - (k + |L| - |N_{\text{low}}|) \\ &= |N_{\text{low}}| - 100\widetilde{\ell} \ln n - |L| \\ &\geq |S_{\text{low}}| - |L| - 100\widetilde{\ell} \ln n - |L| \\ &\geq |S_{\text{low}}| - 102|L| \ln n \geq |S_{\text{low}}|/2.\end{aligned}$$

To see the second inequality, we have $|E_G(v, N(x))| \geq k - 100\widetilde{\ell} \ln n$ because $\deg_G(v) \geq k$ but $|E_G(v, V \setminus N(x))| \leq 100\widetilde{\ell} \ln n$ by definition of F'_{relax}. Also, $|N(x)| \leq |L \cup S| \leq k + |L|$. The third inequality follows because $|S_{\text{low}}| \leq |N_{\text{low}}| + |S_{\text{low}} \setminus N_{\text{low}}| \leq |N_{\text{low}}| + |L|$ by Proposition 3.2 (the part of S outside $N[x]$ has size less than L, and so the part of S_{low} outside N_{low} has size less than L as well). This completes the proof of the claim. □

3.3 Data Structures

In this section, we show fast data structures needed for proving Lemma 3.5. Throughout this section, let $(G, k, \widetilde{\ell}, T)$ denote the input given to Lemma 3.5. We treat them as global variables in this section. Moreover, as the guarantee from Equation (4) holds whp, **we will assume that Equation (4) holds in this section.**

There are two steps. First, we build an oracle that, given any vertices x and v, lists all neighbors of v outside $N[x]$ if the set is small. Second, given an arbitrary vertex x, we use this oracle to perform a BFS-like process that allows us to gradually build $G_{x,T}$ without having an explicit representation of $G_{x,T}$ in the beginning. We show how to solves these tasks respectively in the subsections below.

3.3.1 *An Oracle for Listing Neighbors Outside $N[x]$.*

In this section, we show the following data structure.

Lemma 3.12 (Neighbor Oracle). *There is an algorithm that preprocesses $(G = (V, E), k, \widetilde{\ell})$ in $\tilde{O}(m)$ time and supports queries* OutNeighbor(x, v) *for any vertex x where $|N[x]| \leq k + 2\widetilde{\ell}$ and $v \in V \setminus \{x\}$.*

OutNeighbor(x, v) either returns the neighbor set of v outside $N[x]$, i.e. $N(v) \setminus N[x]$, in $\tilde{O}(\widetilde{\ell})$ time, or report "too big" in $O(\log n)$ time. If $|N(v) \setminus N[x]| \leq 40\widetilde{\ell} \ln n$, then $N(v) \setminus N[x]$ is returned. If $|N(v) \setminus N[x]| > 100\widetilde{\ell} \ln n$, then "too big" is reported. Whp, every query is answered correctly.

For any vertex set $V' \subseteq V$, let the indicator vector $\mathbb{1}_{V'} \in \{0, 1\}^V$ of V' be the vector where $\mathbb{1}_{V'}(u) = 1$ iff $u \in V'$. In this section, we always use sparse representation of vectors, i.e. a list of (index,value) of non-zero entries of the vector.

The algorithm preprocesses as follows. Set $s \leftarrow 100\widetilde{\ell} \ln n$. For every vertex $v \in V$, we compute the sketches $\mathrm{sk}_s(\mathbb{1}_{N(v)})$, $\mathrm{sk}_s(\mathbb{1}_{N[v]})$, $\mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)})$, and $\mathrm{sk}_{\ell_2}(\mathbb{1}_{N[v]})$ using Theorems 2.1 and 2.2. Observe the following.

Proposition 3.13. *The preprocessing time is $\tilde{O}(m)$.*

Proof. Theorems 2.1 and 2.2 preprocess in $\tilde{O}(n)$ time. The total time to compute the sketches is $\sum_{v \in V} \tilde{O}(\deg(v)) = \tilde{O}(m)$. □

Now, given a vertex x where $|N[x]| \leq k + 2\widetilde{\ell}$ and $v \in V \setminus \{x\}$, observe that the non-zero entries of $\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]} \in \{-1, 0, 1\}^V$ corresponds to the symmetric difference $(N(v) \setminus N[x]) \cup (N[x] \setminus N(v))$. We will bound the size of $N[x] \setminus N(v)$ as follows:

$$\begin{aligned}|N[x] \setminus N(v)| &\leq |N[x]| - |N[x] \cap N(v)| \\ &\leq k + 2\widetilde{\ell} - (k - |N(v) \setminus N[x]|) \\ &= |N(v) \setminus N[x]| + 2\widetilde{\ell} \end{aligned} \tag{6}$$

where the second inequality is because $k \leq |N(v)| = |N(v) \cap N[x]| + |N(v) \setminus N[x]|$. Therefore, we have

$$|N(v) \setminus N[x]| \leq \|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_0 \overset{(6)}{\leq} 2|N(v) \setminus N[x]| + 2\widetilde{\ell}.$$

Since $\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]} \in \{-1, 0, 1\}^V$, we have

$$|N(v) \setminus N[x]| \leq \|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_2 \leq 2|N(v) \setminus N[x]| + 2\widetilde{\ell}. \tag{7}$$

Now, we describe how to answer the query. First, we compute $\mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)}) - \mathrm{sk}_{\ell_2}(\mathbb{1}_{N[x]}) = \mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})$ in $O(\log n)$ time. If $\|\mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})\|_2 > s$, then we report "too big". Otherwise, we have $s \geq \|\mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})\|_2 \geq \|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_2 = \|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_0$ by Theorem 2.1 and because $\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]} \in \{-1, 0, 1\}^V$. So, we can compute

$$\mathrm{sk}_s(\mathbb{1}_{N(v)}) - \mathrm{sk}_s(\mathbb{1}_{N[x]}) = \mathrm{sk}_s(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})$$

and obtain the set $N(v) \setminus N[x]$ inside $(N(v) \setminus N[x]) \cup (N[x] \setminus N(v))$ using Theorem 2.2 in $\tilde{O}(s) = \tilde{O}(\widetilde{\ell})$ time.

To see the correctness, if $|N(v) \setminus N[x]| \leq 40\widetilde{\ell} \ln n$, then

$$\begin{aligned}\|\mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})\|_2 &\leq 1.1\|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_2 \\ &\overset{(7)}{\leq} 1.1 \cdot (2|N(v) \setminus N[x]| + 2\widetilde{\ell}) \\ &\leq 100\widetilde{\ell} \ln n = s.\end{aligned}$$

So the set $N(v) \setminus N[x]$ must be returned. If $|N(v) \setminus N[x]| > 100\widetilde{\ell} \ln n$, then

$$\begin{aligned}\|\mathrm{sk}_{\ell_2}(\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]})\|_2 &\geq \|\mathbb{1}_{N(v)} - \mathbb{1}_{N[x]}\|_2 \\ &\overset{(7)}{\geq} |N(v) \setminus N[x]| \\ &> 100\widetilde{\ell} \ln n.\end{aligned}$$

and so "too big" is reported in $O(\log n)$ time. Every query is correct whp because of the whp guarantees from Theorems 2.1 and 2.2. This completes the proof of Lemma 3.12.

3.3.2 *Building $G_{x,T}$ by Sketchy Search.*

In this section, we show how to use the oracle from Lemma 3.12 to return the kernel graph $G_{x,T}$. As the oracle is based on linear sketching and we use it in a BFS-like process, we call this algorithm *sketchy search.*

Lemma 3.14 (Sketchy Search). *There is an algorithm that preprocesses $(G,k,\widetilde{\ell},T)$ in $O(m)$ time and guarantees the following whp.*

Given a query vertex $x \in X$, by calling the oracle from Lemma 3.12, return either $\perp$ or the kernel graph $G_{x,T}$ with $O(k\widetilde{\ell}\log n)$ edges together with the set $Z_{x,T}$ (defined in the beginning of Section 3.2) in $\widetilde{O}(k\widetilde{\ell})$ time. If $T_x \neq \emptyset$ and there is a k-scratch (L,S,R) where $T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2,|L|]$, and $x \in L$, then the algorithm must return $G_{x,T}$ and $Z_{x,T}$.

The remaining part of this section is for proving Lemma 3.14. In the preprocessing step, we just compute $V_{\text{bad}} = \{v \mid T \subseteq N[v]\}$ by trivially checking if $T \subseteq N[v]$ on each vertex v using total $\sum_v \deg(v) = O(m)$ time. Observe that $x \in V_{\text{bad}}$ iff $T_x = \emptyset$.

Next, if there is a k-scratch (L,S,R) where $x \in L$ and $\widetilde{\ell} \in [|L|/2,|L|]$, then we must have $|N[x]| \leq k + |L| \leq k + 2\widetilde{\ell}$. So, given a query vertex x, if $x \in V_{\text{bad}}$ or $|N[x]| > k + 2\widetilde{\ell}$, we can just return $\perp$. From now, we assume that $T_x \neq \emptyset$ and $|N[x]| \leq k + 2\widetilde{\ell}$.

Before showing how to construct $G_{x,T}$, we recall the definitions of $G_{x,T}$ and $Z_{x,T}$ from Section 3.2. Equation (5) says that edges of $G_{x,T}$ can be partitioned as

$$\begin{aligned} E(G_{x,T}) = &E_{G_{x,T}}(x,N_x) \cup E_{G_{x,T}}(N_x, F \cup N_t) \\ &\cup E_{G_{x,T}}(F, F \cup N_t) \cup E_{G_{x,T}}(N_t, t_x) \end{aligned}$$

where $N_x = N_{G_{x,T}}(x)$, $N_t = N_{G_{x,T}}(t_x)$, and $F = V(G_{x,T}) \setminus (N_x \cup N_t \cup \{x, t_x\})$. We write $Z \triangleq Z_{x,T} = N_{G'_{x,T}}(x) \cap N_{G'_{x,T}}(t_x)$ where $G'_{x,T}$ is obtained from G by contracting T_x into a single vertex t_x.

Our strategy is to exploit OutNeighbor queries from Lemma 3.12 to perform a BFS-like process on $G_{x,T}$ that allows us to gradually identify $G_{x,T}$ and $Z_{x,T}$ without having an explicit representation of $G_{x,T}$ in the beginning. The algorithm initializes $\widetilde{Z}, \widetilde{N}_x, \widetilde{E}_{N_x}, \widetilde{N}_t, \widetilde{F}, \widetilde{E}_F = \emptyset$. At the end of the algorithm, these sets will become $Z, N_x, E_{G_{x,T}}(N_x, F \cup N_t), N_t, F, E_{G_{x,T}}(F, F \cup N_t)$ respectively.

Observe that once we know all these sets we can immediately deduce $E_{G_{x,T}}(x,N_x)$ and $E_{G_{x,T}}(N_t,t_x)$, and hence we obtain all parts in $E(G_{x,T})$. So we can return $G_{x,T}$ and $Z_{x,T}$ as desired.

The algorithm has two main loops. After the first loop, $\widetilde{Z}, \widetilde{N}_x$, and $\widetilde{E}_{N_x}$ become Z, N_x, and $E_{G_{x,T}}(N_x, F \cup N_t)$ respectively. After the second loop, $\widetilde{N}_t, \widetilde{F}$, and $\widetilde{E}_F$ become N_t, F, and $E_{G_{x,T}}(F, F \cup N_t)$ respectively. Let CountList $= 0$ initially. We use CountList to count the number of times that OutNeighbor(x,v) lists neighbors of v (not just reports "too big"). In Algorithm 1, we describe this BFS-like process in details.

Before prove the correctness of Algorithm 1, we observe the following simple fact.

Fact 3.15. *$N(v) \setminus N[x]$ intersects T_x iff v is incident to T_x.*

Proof. As $T_x \cap N[x] = \emptyset$, we have $N(v) \setminus N[x]$ intersects T_x iff $N(v)$ intersects T_x iff $v \in N(T_x)$. □

That is, the condition in Steps 1(c)i and 2(c)ii is equivalent to checking if v is incident to T_x. Now, we prove the correctness of the first loop.

Algorithm 1: An algorithm for building $G_{x,T}$

```
(1) For each $v \in N(x)$,
  (a) Set visit$(v)$ = true.
  (b) If OutNeighbor$(x,v)$ returns "too big", add $v$ to $\widetilde{Z}$.
  (c) Else, OutNeighbor$(x,v)$ returns the set $N(v) \setminus N[x]$.
    (i) If $N(v) \setminus N[x]$ intersects $T_x$, add $v$ to $\widetilde{Z}$.
    (ii) Else, (1) add $v$ to $\widetilde{N}_x$ and edges between $v$ and
         $N(v) \setminus N[x]$ to $\widetilde{E}_{N_x}$, and (2) add
         $\{w \in N(v) \setminus N[x] \mid$ visit$(w) \neq$ true$\}$ to Queue.
(2) While $\exists v \in$ Queue,
  (a) Remove $v$ from Queue. Set visit$(v)$ = true.
  (b) If OutNeighbor$(x,v)$ returns "too big", add $v$ to $\widetilde{N}_t$.
  (c) Else, OutNeighbor$(x,v)$ returns the set $N(v) \setminus N[x]$.
    (i) CountList $\leftarrow$ CountList + 1.
    (ii) If $N(v) \setminus N[x]$ intersects $T_x$, then add $v$ to $\widetilde{N}_t$.
    (iii) Else, (1) add $v$ to $\widetilde{F}$ and edges between $v$ and
          $N(v) \setminus N[x]$ to $\widetilde{E}_F$, and (2) add
          $\{w \in N(v) \setminus N[x] \mid$ visit$(w) \neq$ true$\}$ to Queue.
    (iv) If CountList > $16k$, return $\perp$ and terminate.
```

Proposition 3.16. *After the for loop in Step 1, $\widetilde{Z}$, $\widetilde{N}_x$, and $\widetilde{E}_{N_x}$ become Z, N_x, and $E_{G_{x,T}}(N_x, F \cup N_t)$, respectively.*

Proof. By Lemma 3.9(1), $N(x) = Z \dot{\cup} N_x$ where $Z = N(x) \cap N(T_x)$ and $N_x = N(x) \setminus N(T_x)$. After the for loop, every $v \in N(x)$ is added to either $\widetilde{Z}$ or $\widetilde{N}_x$. If v is added to $\widetilde{Z}$ in Step 1(c)i, then Lemma 3.12 implies that $|N(v) \setminus N[x]| > 40\widetilde{\ell}\ln n$ and so $v \in N(T_x)$ by Equation (4), which means $v \in Z$. If v is added to $\widetilde{Z}$ in Step 1(c)i, then we directly verify that $v \in N(T_x)$ (see Fact 3.15) and so $v \in Z$ again. Lastly, if v is added to $\widetilde{N}_x$ in Step Item 1(c)ii, then $v \notin N(T_x)$ and so $v \in N_x$. This means that indeed $\widetilde{Z} = Z$ and $\widetilde{N}_x = N_x$ after the for loop. Lastly, every time v is added to $\widetilde{N}_x$, we add $E_G(v, V \setminus N[x]) = E_{G_{x,T}}(v, F \cup N_x)$ into $\widetilde{E}_{N_x}$. So $\widetilde{E}_{N_x}$ also collects all edges in $E_{G_{x,T}}(N_x, F \cup N_t)$ after the for loop. □

Next, we prove the correctness of the second loop. The proof is similar to the first one but more complicated.

Proposition 3.17. *Suppose $\perp$ is not returned by Algorithm 1. Then, at end of the while loop in Step 2, $\widetilde{N}_t$, $\widetilde{F}$ and $\widetilde{E}_F$ become N_t, F and $E_{G_{x,T}}(F, F \cup N_t)$ respectively.*

Proof. We will prove by induction on time that (1) $\widetilde{N}_t \subseteq N_t$, (2) $\widetilde{F} \subseteq F$, and (3) if $w \in$ Queue at some point of time, then $w \in N_t \cup F$.

For the base case, consider the time before the while loop is executed. We have $\widetilde{N}_t = \emptyset$ and $\widetilde{F} = \emptyset$. If $w \in$ Queue, then $w \in N(v) \setminus N[x]$ for some $v \in N_x$. There are two cases: if $w \in N(T_x)$, then $w \in N(T_x) \setminus N[x]$ and w is incident to $v \in N_x$, which means that $w \in N_t$ by Lemma 3.9(3). Otherwise, if $w \notin N(T_x)$, then $w \notin N[x] \cup N(T_x)$ and (v,w) is a path from N_x to w in $G \setminus N[T_x]$, which means that $w \in F$ by Lemma 3.9(2).

For the inductive step, consider that iteration where we visit v. We prove the three statements below one by one.

(1) Suppose v is added to $\widetilde{N}_t$. If v is added at Step 2b, then Lemma 3.12 implies that $|N(v) \setminus N[x]| > 40\widetilde{\ell}\ln n$ and so $v \in N(T_x)$ by Equation (4). If v is added at Step 2(c)ii, then

we directly verify that $v \in N(T_x)$ (see Fact 3.15). In both cases, $v \in N(T_x)$. As $v \in N_t \cup F$ by induction, v must be in N_t. So $\widetilde{N}_t \subseteq N_t$ holds.

(2) Suppose v is added to $\widetilde{F}$, which only happens at Step 2(c)iii. We directly verify that $v \notin N(T_x)$. As $v \in N_t \cup F$ by induction, v must be in F and so $\widetilde{F} \subseteq F$ holds.

(3) Suppose w is added into Queue at Step 2(c)iii. There are two cases. If $w \in N(T_x)$, then $w \in N(T_x) \setminus N[x]$ and w is incident to $v \in F$, which means that $w \in N_t$ by Lemma 3.9(3). Otherwise, if $w \notin N(T_x)$, then $w \notin N[x] \cup N(T_x)$. As $v \in F$, there exists a path p_v from N_x to v in $G \setminus N[T_x]$. Now, observe that the concatenated path $p_w = p_v \circ (v, w)$ is a path from N_x to w in $G \setminus N[T_x]$. So, $w \in F$ by Lemma 3.9(2). In either case, we have $w \in N_t \cup F$.

To show that $\widetilde{N}_t = N_t$ and $\widetilde{F} = F$ at the end, we argue that all vertices in $N_t \cup F$ must be visited at some point. Observe that our algorithm simulate a BFS algorithm on $G \setminus Z$ when we start the search from vertices in N_x. Moreover, it never continues the search once it reaches vertices in N_t. By Lemma 3.9(2), vertices in F are reachable from N_x in $G \setminus N[T_x] \subseteq G \setminus Z$. So all vertices from F must be visited. Also, because $N_t \subseteq N(T_x) \setminus N[x]$ and every vertex in N_t is incident to F or N_x, all vertices from N_t must be visited as well. This completes the proof that $\widetilde{N}_t = N_t$ and $\widetilde{F} = F$ at the end of the while loop.

Finally, every time v is added to $\widetilde{F}$, we add $E_G(v, V \setminus N[x]) = E_{G_{x,T}}(v, F \cup N_t)$ into $\widetilde{E}_F$. So $\widetilde{E}_F$ collects all edges in $E_{G_{x,T}}(F, F \cup N_t)$ after the while loop. □

Let v be a visited vertex in some iteration of the for loop or the while loop. We say that v's iteration is *fast* if OutNeighbor(x, v) returns "too big", otherwise we say that v's iteration is *slow*.

Proposition 3.18. *Algorithm 1 takes $\tilde{O}(k\widetilde{\ell})$ time.*

Proof. By Lemma 3.12, each fast iteration takes $O(\log n)$ time. For each slow iteration, the bottle necks are (i) listing vertices in $N(v) \setminus N[x]$, and (ii) checking if $N(v) \setminus N[x]$ intersects T_x. The former takes $\tilde{O}(\widetilde{\ell})$ time by Lemma 3.12. The latter also takes $|N(v) \setminus N[x]| = O(\widetilde{\ell})$ time because we can simply check, for every $w \in N(v) \setminus N[x]$, if $w \in T_x$ which happens iff $w \in T$.

Observe the number of slow iterations is at most

$$|N(x)| + \text{CountList} \leq 2k + 16k$$

by the condition in Item 2(c)iv. So the total time on slow iterations is at most $\tilde{O}(k\widetilde{\ell})$. We claim the number of fast iterations is most CountList $\cdot\, 100\widetilde{\ell} \ln n$, which would imply that the total running time is $\tilde{O}(k\widetilde{\ell})$.

To prove that claim, we say that w is a *child* of v if w is added to Queue at v's iteration. If v's iteration is fast, then $v \in \widetilde{Z} \cup \widetilde{N}_t$ and so v has no child. If v's iteration is slow, then v has at most $|N(v) \setminus N[x]| \leq 100\widetilde{\ell} \ln n$ children by Lemma 3.12. This implies that there are at most CountList $\cdot\, 100\widetilde{\ell} \ln n$ fast iterations as desired. □

Proposition 3.19. *If Algorithm 1 returns $\perp$, then there is no k-scratch (L, S, R) where $\emptyset \neq T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2, |L|]$, and $x \in L$.*

Proof. Recall $F'_{\text{relax}} = \{v \in V \setminus N[x] \mid |N(v) \setminus N[x]| \leq 100\widetilde{\ell} \log n\}$ defined above Lemma 3.11. Observe that if CountList is incremented in v's iteration, then $|N(v) \setminus N[x]| \leq 100\widetilde{\ell} \log n$ by Lemma 3.12. As $v \in V \setminus N[x]$, we have $v \in F'_{\text{relax}}$. So, if CountList $> 16k$, then $|F'_{\text{relax}}| > 16k$. As we assume that Equation (4) holds, Lemma 3.11 implies that there is no k-scratch (L, S, R) where $\emptyset \neq T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2, |L|]$, and $x \in L$. □

Now, we conclude with the proof of Lemma 3.14.

Proof of Lemma 3.14. Let $(G, k, \widetilde{\ell}, T)$ be given. In the preprocessing step, we compute V_{bad} which takes $O(m)$ time. Given a query $x \in V$, if $x \in V_{\text{bad}}$ or $|N[x]| > k + 2\widetilde{\ell}$, we return $\perp$ and we are done. Otherwise, we execute Algorithm 1 which takes $\tilde{O}(k\widetilde{\ell})$ time by Proposition 3.18. The algorithm either returns $\perp$ and otherwise correctly constructs all parts of $G_{x,T}$ by Propositions 3.16 and 3.17 whp. Using these sets, we can build $G_{x,T}$ via Equation (5) and obtain $Z_{x,T} = Z$ in $\tilde{O}(k\widetilde{\ell})$ time. Note that $|F| = |\widetilde{F}| \leq 16k$ by the condition in Step 2(c)iv. So $|E(G_{x,T})| = O(k\widetilde{\ell} \log n)$ by Lemma 3.10.

Finally, if $T_x \neq \emptyset$ and there is a k-scratch (L, S, R) where $\emptyset \neq T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2, |L|]$, and $x \in L$, then we have $x \notin V_{\text{bad}}$ and $N[x] \leq k + 2\widetilde{\ell}$, so $\perp$ is not returned before running Algorithm 1. By Proposition 3.19, Algorithm 1 cannot return $\perp$ as well. So $G_{x,T}$ and $Z_{x,T}$ must be returned.

3.4 Proof of Lemma 3.5 (Sublinear-Time Kernelization)

Let $(G, k, \widetilde{\ell}, T, X)$ be given as input. We first initialize the oracle from Lemma 3.12 and the BFS-like process from Lemma 3.14. This takes $\tilde{O}(m)$ time. For each $x \in X$, we query x to the algorithm from Lemma 3.14. Lemma 3.14 guarantees that each query takes $\tilde{O}(k\widetilde{\ell})$ time and returns either $\perp$ or $(G_{x,T}, Z_{x,T})$. Therefore, the total running time is $\tilde{O}(m + |X|k\widetilde{\ell})$.

For each query $x \in X$, Equation (4) holds whp by Proposition 3.3. So we will assume it and conclude the following whp. By Lemma 3.14, if $\perp$ is returned, then we can correctly certify that $T_x = \emptyset$ or there is no k-scratch (L, S, R) where $\emptyset \neq T_x \subseteq R$, $\widetilde{\ell} \in [|L|/2, |L|]$, and $x \in L$. If $(G_{x,T}, Z_{x,T})$ is returned, then we have that $|E(G_{x,T})| = O(k\widetilde{\ell} \log n)$. By Lemma 3.8, any set Y is a (x, t_x)-min-separator in $G_{x,T}$ iff $Y \cup Z_{x,T}$ is a (x, T_x)-min-separator in G. as desired.

4 USING ISOLATING CUTS LEMMA

We say that a vertex cut (L, S, R) is a *k-non-scratch* if it has size less than k but it is not a k-scratch. That is, (L, S, R) is such that (1) $|S| < k$ and $|L| > k/100 \log n$, or (2) $|S| < k$, $|L| \leq k/100 \log n$, and $|S_{\text{low}}| < 300|L| \ln n$. Recall that $S_{\text{low}} = S \cap V_{\text{low}}$ and $V_{\text{low}} = \{v \mid \deg(v) \leq 8k\}$. In the previous section, we can report that a mincut has size less than k if a graph contains a k-scratch. In this section, we solves the opposite case; we will report that a mincut has size less than k if a graph contains a k-non-scratch. More formally, we prove the following.

Lemma 4.1. *There is an algorithm that, given an undirected graph G with n vertices and m edges and a parameter k where $m \leq nk$, returns a vertex cut (L, S, R) in G. If G has a k-non-scratch, then $|S| < k$ w.h.p. The algorithm makes s-t maxflow calls on unit-vertex-capacity graphs with $O(m \log^5 n)$ total number of vertices and edges and takes $\tilde{O}(m)$ additional time.*

Note that the lemma above only applies on graphs with at most nk edges, but we can easily and will ensure this when we use the lemma in Section 5. The rest of this section is for proving Lemma 4.1. The key tool in this section is the *isolating cuts lemma* which was introduced in [19]. We show how to adapt it for vertex connectivity as follows.

Lemma 4.2 (Isolating Cuts Lemma)**.** *There exists an algorithm that takes as inputs $G = (V, E)$ and an independent set $I \subset V$ of size at least 2, and outputs, for each vertex $v \in I$, a $(v, I \setminus v)$-min-separator C_v. The algorithm makes s-t maxflow calls on unit-vertex-capacity graphs with $O(m \log |I|)$ total number of vertices and edges and takes $O(m)$ additional time.*

We will prove Lemma 4.2 at the end of this section in Section 4.1. Below, we set up the stage so that we can use it to prove Lemma 4.1. First, we need the following concept:

Definition 4.3. For any vertex set T, a vertex cut (L, S, R) *isolates* a vertex x in T if

$$L \cap T = \{x\}, S \cap T = \emptyset, \text{and } R \cap T \neq \emptyset.$$

For any $p \in [0, 1]$, we let $V(p)$ be obtained by sampling each vertex in V with probability p. Similarly, let $V_{\text{low}}(p)$ be obtained by sampling each vertex in V_{low} with probability p. The following observation says that, for any a k-non-scratch (L, S, R), we can obtain a random set that (L, S, R) isolates a vertex in it with good probability.

Proposition 4.4. *Suppose that G has a k-non-scratch (L, S, R). Then, with probability $\Omega(1/\log^2 n)$, there is $i \in \{1, \dots, \log n\}$ where (L, S, R) isolates a vertex in $V(\frac{1}{2^i})$ or isolates a vertex in $V_{\text{low}}(\frac{1}{2^i})$.*

Proof. There are two cases. Suppose that $|S| < k$ and $|L| > k/100 \log n$. Consider $p = 1/2^i$ such that $1 \le p(2|L| + |S|) \le 2$. As $|S| < 100|L| \log n$, we have $|L|p = \Omega(1/\log n)$. Therefore, (L, S, R) isolates a vertex in $V(p)$ with probability

$$\begin{aligned}
&\mathbb{P}[|L \cap V(p)| = 1] \cdot \mathbb{P}[|S \cap V(p)| = 0] \cdot \mathbb{P}[|R \cap V(p)| \ge 1] \\
&\ge \mathbb{P}[|L \cap V(p)| = 1]^2 \cdot \mathbb{P}[|S \cap V(p)| = 0] \\
&= (|L|p \cdot (1-p)^{|L|-1})^2 (1-p)^{|S|} \\
&\ge (|L|p)^2 (1-p)^{2|L|+|S|} = \Omega(1/\log^2 n)
\end{aligned}$$

where the first inequality is because $|R| \ge |L|$ and the last inequality follows because $p(2|L| + |S|) = \Theta(1)$ and $|L|p = \Omega(1/\log n)$.

Consider another case where $|S| < k$, $|L| \le k/100 \log n$, and $|S_{\text{low}}| \le 300|L| \ln n$. The argument is similar to the previous case, but we first need this claim:

Claim 4.5. *Let $L_{\text{low}} = L \cap V_{\text{low}}$ and $R_{\text{low}} = R \cap V_{\text{low}}$. We have $L_{\text{low}} = L$ and $|R_{\text{low}}| \ge |L_{\text{low}}|$.*

Proof. For each $x \in L$, $N[x] \subseteq L \cup S$. So $\deg(x) \le |L| + |S| \le 2k$ and thus $x \in L_{\text{low}}$. So $L_{\text{low}} = L$. To see why $|R_{\text{low}}| \ge |L_{\text{low}}|$, if $k \ge n/8$, then $V_{\text{low}} = V$ and so $|R_{\text{low}}| = |R| \ge |L| = |L_{\text{low}}|$. Otherwise, $k < n/8$. So $|L \cup S| \le 2k \le n/4$ and then $|R| \ge 3n/4$. As $8k|V \setminus V_{\text{low}}| \le \sum_v \deg(v) \le 2nk$, we have $|V \setminus V_{\text{low}}| \le n/4$ and so $|V_{\text{low}}| \ge 3n/4$. Therefore, $|R_{\text{low}}| = |R \cap V_{\text{low}}| \ge n/2 \ge |L_{\text{low}}|$. ◻

Consider $p = 1/2^i$ such that $1 \le p(2|L| + |S_{\text{low}}|) \le 2$. As $|S_{\text{low}}| < 300|L| \ln n$, we have $|L|p = \Omega(1/\log n)$. Therefore, (L, S, R) isolates a vertex in $V_{\text{low}}(p)$ with probability

$$\begin{aligned}
&\mathbb{P}[|L \cap V_{\text{low}}(p)| = 1] \cdot \mathbb{P}[|S \cap V_{\text{low}}(p)| = 0] \cdot \mathbb{P}[|R \cap V_{\text{low}}(p)| \ge 1] \\
&= \\
&\mathbb{P}[|L_{\text{low}} \cap V(p)| = 1] \cdot \mathbb{P}[|S_{\text{low}} \cap V(p)| = 0] \cdot \mathbb{P}[|R_{\text{low}} \cap V(p)| \ge 1] \\
&\ge \mathbb{P}[|L \cap V(p)| = 1]^2 \cdot \mathbb{P}[|S_{\text{low}} \cap V(p)| = 0] \\
&= (|L|p \cdot (1-p)^{|L|-1})^2 (1-p)^{|S_{\text{low}}|} \\
&\ge (|L|p)^2 (1-p)^{2|L|+|S_{\text{low}}|} = \Omega(1/\log^2 n)
\end{aligned}$$

where the first inequality by Claim 4.5 and the last inequality follows because $p(2|L| + |S_{\text{low}}|) = \Theta(1)$ and $|L|p = \Omega(1/\log n)$. ◻

The last observation we need is about maximal independent sets of an isolated set.

Proposition 4.6. *Suppose that a vertex cut (L, S, R) isolates a vertex x in a set T. Let I be an maximal independent set of T. Then (L, S, R) also isolates x in I.*

Proof. Note that $|T| \ge 2$ because $L \cap T = \{x\}$ and $R \cap T \neq \emptyset$. As x is not incident to any other vertex in T, we have $x \in I$. So $L \cap I = \{x\}$. Also some vertex in $R \cap T$ must remain in I because $S \cap T = \emptyset$. So $R \cap I \neq \emptyset$. This means that (L, S, R) isolates x in I. ◻

Now, we are ready to prove Lemma 4.1.

Proof of Lemma 4.1. The algorithm for Lemma 4.1 is as follows. For each $i \in \{1, \dots, \log n\}$ and $j \in \{1, \dots, O(\log^3 n)\}$, we independently sample $T^{(i,j)} = V(\frac{1}{2^i})$ and $T^{(i,j)}_{\text{low}} = V_{\text{low}}(\frac{1}{2^i})$ and compute maximal independent sets $I^{(i,j)}$ of $T^{(i,j)}$ and $I^{(i,j)}_{\text{low}}$ of $T^{(i,j)}_{\text{low}}$ respectively. Next, we invoke Lemma 4.2 on $(G, I^{(i,j)})$ if $|I^{(i,j)}| \ge 2$ and on $(G, I^{(i,j)}_{\text{low}})$ if $|I^{(i,j)}_{\text{low}}| \ge 2$. Among all separators that Lemma 4.2 returns, we return the one with minimum size and its corresponding vertex cut. If $|I^{(i,j)}|, |I^{(i,j)}_{\text{low}}| < 2$ for all i, j, we return an arbitrary vertex cut.

It is clear the algorithm makes s-t maxflow calls on unit-vertex-capacity graphs with $O(m \log^5 n)$ total number of vertices and edges and takes $\tilde{O}(m)$ additional time because we invoke Lemma 4.2 $O(\log^4 n)$ times.

To see the correctness, suppose there is a k-non-scratch (L, S, R), then by Proposition 4.4, there exist i and j such that (L, S, R) isolates a vertex in either $T^{(i,j)}$ or $T^{(i,j)}_{\text{low}}$ whp. By Proposition 4.6, (L, S, R) must also isolate a vertex in either $I^{(i,j)}$ or $I^{(i,j)}_{\text{low}}$ whp. Suppose that (L, S, R) isolates a vertex x in $I^{(i,j)}$. Then, (L, S, R) is a $(x, I^{(i,j)} \setminus x)$-separator. So the call of Lemma 4.2 on $(G, I^{(i,j)})$ must return a separator of size at most $|S| < k$. The argument is the same if (L, S, R) isolates a vertex x in $I^{(i,j)}_{\text{low}}$.

4.1 Proof of Lemma 4.2 (Isolating Cuts Lemma)

The goal of this section is to prove Lemma 4.2. We follow the proof of Theorem II.2 of [19]. Order the vertices in I arbitrarily from 1 to $|I|$, and let the *label* of each $v \in I$ be its position in the ordering, a number from 1 to $|I|$ that is denoted by a unique binary string of length $\lceil \lg |I| \rceil$. Let us repeat the following procedure for each

$i = 1, 2, \ldots, \lceil \lg |I| \rceil$. Let $A_i \subseteq I$ be the vertices in I whose label's i'th bit is 0, and let $B_i \subseteq I$ be the vertices whose label's i'th bit is 1. Compute a (A_i, B_i)-min-separator $C_i \subseteq V$ (for iteration i). Note that since $I = A_i \cup B_i$ is an independent set in G, the set $V \setminus (A_i \cup B_i)$ is an (A_i, B_i)-separator, so an (A_i, B_i)-min-separator exists.

First, we show that $G \setminus \bigcup_i C_i$ partitions the set of vertices into connected components each of which contains at most one vertex of I. Let U_v be the connected component in $G \setminus \bigcup_i C_i$ containing $v \in I$. Then:

Claim 4.7. *$U_v \cap I = \{v\}$ for all $v \in I$.*

Proof. By definition, $v \in U_v \cap I$. Suppose for contradiction that $U_v \cap I$ contains another vertex $u \neq v$. Since the binary strings assigned to u and v are distinct, they differ in their j'th bit for some j. Assume without loss of generality that $u \in A_j$ and $b \in B_j$. Since $C_j \subseteq V$ is a (A_j, B_j)-min-separator, there cannot be a u-v path whose vertices are disjoint from C_j, contradicting the assumption that u and v belong in the same connected component of $G \setminus \bigcup_i C_i$. □

Claim 4.8 (Submodularity of vertex cuts)**.** *For any subsets $A, B \subseteq V$, we have*

$$|N(A)| + |N(B)| \geq |N(A \cup B)| + |N(A \cap B)|.$$

Proof. We consider the contribution of each vertex $v \in V$ to the LHS $|N(A)| + |N(B)|$ and the RHS $|N(A \cup B)| + |N(A \cap B)|$ separately. Each vertex $v \in N(A) \cap N(B)$ contributes 2 to the LHS and at most 2 to the RHS. Each vertex $v \in N(A) \setminus N(B)$ contributes 1 to the LHS, and 1 to the RHS because $v \in N(A \cup B)$ and $v \notin N(A \cap B)$. A symmetric case covers each vertex $v \in N(B) \setminus N(A)$. Finally, each vertex $v \notin N(A) \cup N(B)$ contributes 0 to both sides. □

Now, for each vertex $v \in I$, let λ_v be the size of a $(v, I \setminus v)$-min-separator. For each $(v, I \setminus v)$-min-separator C, we can consider the set $S \subseteq V$ of vertices in the connected component of $G \setminus C$ containing v, which necessarily satisfies $N(S) = C$. Let $S_v^* \subseteq V$ be an inclusion-wise minimal set such that $N(S_v^*)$ is a $(v, I \setminus v)$-separator. Then, we claim the following:

Claim 4.9. *$U_v \supseteq S_v^*$ for all $v \in I$.*

Proof. Fix a vertex $v \in I$ and an iteration i. Let $T_v^i \subseteq V$ be the vertices in the connected components of $G \setminus C_i$ that contain at least one vertex the same color as v (on iteration i). By construction of C_i, the set T_v^i does not contain any vertex of the opposite color. We now claim that $S_v^* \subseteq T_v^i$. Suppose for contradiction that $S_v^* \setminus T_v^i \neq \emptyset$. Note that $(S_v^* \cap T_v^i) \cap I = \{v\}$ and

$$N(S_v^* \cap T_v^i) \cap I \subseteq (N(S_v^*) \cup N(T_v^i)) \cap I \subseteq (N(S_v^*) \cup C_i) \cap I = \emptyset,$$

where the first inclusion holds because $N(S \cap T) \subseteq N(S) \cup N(T)$ for any $S, T \subseteq V$, and the second inclusion holds because $N(T_v^i) \subseteq C_i$ by construction of T_v^i. Therefore,

$$|N(S_v^* \cap T_v^i)| \geq \lambda_v = |N(S_v^*)|.$$

Indeed, by our choice of S_v^* to be inclusion-wise minimal, we can claim the strict inequality:

$$|N(S_v^* \cap T_v^i)| > \lambda_v = |N(S_v^*)|.$$

But, by Claim 4.8 we have:

$$|N(S_v^* \cup T_v^i)| + |N(S_v^* \cap T_v^i)| \leq |N(S_v^*)| + |N(T_v^i)|.$$

Therefore, we get:

$$|N(S_v^* \cup T_v^i)| < |N(T_v^i)|.$$

Now observe that $(S_v^* \cup T_v^i) \cap I = T_v^i \cap I$ since $(S_v^* \setminus T_v^i) \cap I = \emptyset$. In particular, $S_v^* \cup T_v^i$ contains all vertices in A_i and no vertices in B_i. Also, since $N(S_v^*) \cap I = \emptyset$ and $N(T_v^i) \cap I = \emptyset$, we also have $N(S_v^* \cup T_v^i) \cap I = \emptyset$. Then,

$$|N(S_v^* \cup T_v^i)| < |N(T_v^i)| \leq |C_i|,$$

so $N(S_v^* \cup T_v^i)$ is a smaller (A_i, B_i)-separator than C_i, a contradiction.

For each iteration i, since $S_v^* \subseteq T_v^i$, none of the vertices in S_v^* are present in C_i. Note that $G[S_v^*]$ is a connected subgraph; therefore, it is a subgraph of the connected component U_v of $G \setminus \bigcup_i C_i$ containing v. This concludes the proof of Claim 4.9. □

Fact 4.10. *Given a graph $G = (V, E)$ and distinct vertices $s, t \in V$, and given a s-t vertex maxflow, we can compute in $O(|V| + |E|)$ time a set $S \subseteq V$ with $S \cap \{s, t\} = \{s\}$ such that $N(S)$ is a (s, t)-min-separator.*

It remains to compute the desired set S_v given the property that $U_v \supseteq S_v$. Construct the graph G_v as follows. Start from the induced graph $G[U_v \cup N_G(U_v)]$, remove all edges with both endpoints in $N_G(U_v)$, and then add a vertex t connected to all vertices in $N_G(U_v)$. We compute a v-t vertex maxflow in G_v and then apply Fact 4.10, obtaining a set S_v such that $N_{G_v}(S_v)$ is a (v, t)-min-separator. Since $t \notin N_{G_v}(S_v)$, we must have $S_v \cap N_{G_v}(U_v) = \emptyset$, so by construction of G_v, we have $N_{G_v}(S_v) = N_G(S_v)$. In particular, $N_G(S_v) = N_{G_v}(S_v) \subseteq U_v \cup N_G(U_v)$, and along with $v \in S_v$, we obtain $N_G(S_v) \cap I = \emptyset$.

Claim 4.9 implies that $N_{G_v}(S_v^*) \subseteq U_v \cup N_{G_v}(U_v)$, so $N_{G_v}(S_v^*) = N_G(S_v^*)$ and $t \notin N_{G_v}(S_v^*)$. Therefore, $N_{G_v}(S_v^*)$ is a (v, t)-separator in G_v of size λ_v. Since $N_{G_v}(S_v)$ is a (v, t)-*min*-separator in G_v, we have $|N_G(S_v)| = |N_{G_v}(S_v)| \leq |N_{G_v}(S_v^*)| = \lambda_v$. Define $C_v = N(S_v)$, which satisfies the desired properties in the statement of the lemma.

We now bound the total size of the graphs G_v over all $v \in I$. By construction of the graphs G_v, each edge in E joins at most one graph G_v. Each graph G_v has $|N_G(U_v)|$ additional edges adjacent to t, but since each vertex in $N_G(U_v)$ is adjacent to some vertex in U_v via an edge originally in E, we can charge the edges in G_v adjacent to t to the edges originating from E. Therefore, the total number of edges over all graphs G_v is $O(m)$. Each of the graphs G_v is connected, so the total number of vertices is also $O(m)$. Finally, to compute (A_i, B_i)-min-separator for all i, the total size of the maxflow instances is $O(m \log |I|)$. To bound the additional time, by Fact 4.10, recovering the sets S_v and the values $|N(S_v)|$ takes time linear in the number of edges of G_v, which is $O(m)$ time over all $v \in I$. This completes the proof of Lemma 4.2.

5 PUTTING EVERYTHING TOGETHER

For any k, we can detect if G has vertex mincut of size less than k as follows. First, compute a k-connectivity certificate H of G which preserves all vertex cuts of size less than k and H has at most nk edges (so H is applicable for Lemma 4.1). This can be done in linear time using the algorithm by Nagamochi and Ibaraki [21]. Then, we apply Lemmas 3.1 and 4.1 on H with parameter k. If H has a vertex cut of size less than k, that cut is either a k-scratch or k-non-scratch, and so one of the algorithms of Lemmas 3.1 or 4.1 must return a vertex cut of size less than k whp. If vertex mincut of G is at least k, then any of the algorithms in Lemma 3.1 and Lemma 4.1 always returns a vertex cut of size at least k. Theorem 1.1 follows immediately by a binary search on k.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672 and No 759557. Nanongkai was also partially supported by the Swedish Research Council (Reg. No. 2019-05622). Panigrahi has been supported in part by NSF Awards CCF 1750140 and CCF 1955703.

A PROOFS OF LINEAR SKETCHING

Proof of Theorem 2.1. We can use F_2-moment frequency estimation by [2]. Although their work focus on estimating on positive entries, their algorithm is linear, and thus it is possible to estimate norm of the difference between two vectors x, y: $\|x - y\|_2$.

Given a vector $v \in \mathbb{R}^n$, we compute sketch of v by viewing it in a streaming setting as follows. We start with a zero vector $x = 0$, and feed a sequence of update (i, v_i) for each non-empty entry in v of total $\|v\|_0$ updates. Each update can be performed in $\log^{O(1)}(n)$ time.

Proof of Theorem 2.2. The sparse recovery algorithm is described in Section 2.3 in [6] (Section 2.3.1 and Section 2.3.2 in particular). In order for their algorithm to work efficiently, we need a standard assumption that the vector x that we compute the sketch from satisfies $x_i \in \mathbb{Z} \cap [-n^{O(1)}, n^{O(1)}]$ for all $i \in [n]$ so that all arithmetic operations in this algorithm can be computed in $O(\log n)$ time.

Given a vector $v \in \{-1, 0, 1\}^n$, we compute sketch of v by viewing it in a streaming setting as follows. We start with a zero vector $x = 0$, and feed a sequence of update (i, v_i) for each non-zero entry in v of total $\|v\|_0$ updates. Each update can be performed in $\log^{O(1)}(n)$ time according to their sparse recovery algorithm.

REFERENCES

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.* 58, 1 (1999), 137–147. https://doi.org/10.1006/jcss.1997.1545

[3] Michael Becker, W. Degenhardt, Jürgen Doenhardt, Stefan Hertel, Gerd Kaninke, W. Kerber, Kurt Mehlhorn, Stefan Näher, Hans Rohnert, and Thomas Winter. 1982. A Probabilistic Algorithm for Vertex Connectivity of Graphs. *Inf. Process. Lett.* 15, 3 (1982), 135–136.

[4] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. 2014. Distributed connectivity decomposition. In *PODC*. ACM, 156–165.

[5] Joseph Cheriyan and Ramakrishna Thurimella. 1991. Algorithms for Parallel k-Vertex Connectivity and Sparse Certificates (Extended Abstract). In *STOC*. ACM, 391–401.

[6] Graham Cormode and Donatella Firmani. 2014. A unifying framework for ℓ_0-sampling algorithms. *Distributed Parallel Databases* 32, 3 (2014), 315–335.

[7] E. A. Dinic. 1970. Algorithm for solution of a problem of maximal flow in a network with power estimation. 11 (1970), 1277–1280.

[8] Shimon Even and Robert Endre Tarjan. 1975. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.* 4, 4 (1975), 507–518.

[9] Sebastian Forster, Danupon Nanongkai, Thatchaphol Saranurak, Liu Yang, and Sorrachai Yingchareonthawornchai. 2020. Computing and Testing Small Connectivity in Near-Linear Time and Queries via Fast Local Cut Algorithms. In *SODA*. SIAM, 2046–2065.

[10] Loukas Georgiadis. 2010. Testing 2-Vertex Connectivity and Computing Pairs of Vertex-Disjoint s-t Paths in Digraphs. In *ICALP (1) (Lecture Notes in Computer Science, Vol. 6198)*. Springer, 738–749.

[11] Andrew V. Goldberg and Robert Endre Tarjan. 1988. A new approach to the maximum-flow problem. *J. ACM* 35, 4 (1988), 921–940.

[12] Jianxiu Hao and James B. Orlin. 1994. A Faster Algorithm for Finding the Minimum Cut in a Directed Graph. *J. Algorithms* 17, 3 (1994), 424–446.

[13] Monika Henzinger, Satish Rao, and Di Wang. 2017. Local Flow Partitioning for Faster Edge Connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 1919–1938. https://doi.org/10.1137/1.9781611974782.125

[14] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. 2000. Computing Vertex Connectivity: New Bounds from Old Techniques. *J. Algorithms* 34, 2 (2000), 222–250. Announced at FOCS'96.

[15] John E. Hopcroft and Robert Endre Tarjan. 1973. Dividing a Graph into Triconnected Components. *SIAM J. Comput.* 2, 3 (1973), 135–158.

[16] Arkady Kanevsky and Vijaya Ramachandran. 1991. Improved Algorithms for Graph Four-Connectivity. *J. Comput. Syst. Sci.* 42, 3 (1991), 288–306. announced at FOCS'87.

[17] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. 2020. Unit Capacity Maxflow in Almost $O(m^{4/3})$ Time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020.* 119–130. https://doi.org/10.1109/FOCS46700.2020.00020

[18] D Kleitman. 1969. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory* 16, 2 (1969), 232–233.

[19] Jason Li and Debmalya Panigrahi. 2020. Deterministic Min-cut in Poly-logarithmic Max-Flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society.

[20] Nathan Linial, László Lovász, and Avi Wigderson. 1988. Rubber bands, convex embeddings and graph connectivity. *Combinatorica* 8, 1 (1988), 91–102. Announced at FOCS'86.

[21] Hiroshi Nagamochi and Toshihide Ibaraki. 1992. A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph. *Algorithmica* 7, 5&6 (1992), 583–596.

[22] Danupon Nanongkai, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. 2019. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *STOC*. ACM, 241–252.

[23] VD Podderyugin. 1973. An algorithm for finding the edge connectivity of graphs. *Vopr. Kibern* 2 (1973), 136.

[24] Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160. Announced at FOCS'71.