Incremental Approximate Maximum Flow
via Residual Graph Sparsification

Gramoz Goranci[1], Monika Henzinger[2], Harald Räcke[3], and A. R. Sricharan[4]

[1,4]Faculty of Computer Science, University of Vienna, Austria
[2]Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria
[3]Technical University Munich, Germany
[4]UniVie Doctoral School Computer Science DoCS

Abstract

We give an algorithm that, with high probability, maintains a $(1-\epsilon)$-approximate s-t maximum flow in undirected, uncapacitated n-vertex graphs undergoing m edge insertions in $\tilde{O}(m + nF^*/\epsilon)$ total update time, where F^* is the maximum flow on the final graph. This is the first algorithm to achieve polylogarithmic amortized update time for dense graphs ($m = \Omega(n^2)$), and more generally, for graphs where $F^* = \tilde{O}(m/n)$.

At the heart of our incremental algorithm is the residual graph sparsification technique of Karger and Levine [SICOMP '15], originally designed for computing exact maximum flows in the static setting. Our main contributions are (i) showing how to maintain such sparsifiers for approximate maximum flows in the incremental setting and (ii) generalizing the cut sparsification framework of Fung et al. [SICOMP '19] from undirected graphs to balanced directed graphs.

1 Introduction

The maximum s-t flow problem has been at the forefront of research in theoretical computer science and combinatorial optimization. Algorithms developed for maximum s-t flow and its dual problem, minimum s-t cut, have been highly influential due to their wide applicability [AMOR95] and their use as subroutines in other algorithms [AHK12, She09]. A long line of work has improved our understanding of how efficiently maximum flow can be solved in the static setting. Two landmark combinatorial results are the deterministic algorithm by Goldberg and Rao [GR98], which achieves a running time of $O(\min\{n^{2/3}, m^{1/2}\} \cdot m)$, and the randomized algorithm by Karger and Levine [KL15] which has a running time of $\widetilde{O}(m + nF^*)$,[1] where F^* is the value of the maximum flow. More recently, efforts building upon a novel blend of continuous optimization techniques, the Laplacian paradigm [Ten10] and graph-based data structures have led to even faster algorithms; notably, a randomized $\widetilde{O}(m/\epsilon)$ time approximate maximum flow algorithm on undirected graphs [She13, KLOS14, Pen16, She17], and a deterministic $\widehat{O}(m)$ time *exact* algorithm for min-cost flow (and thus maximum flow) even on directed graphs [CKL+22, CKL+24], which constitutes a major algorithmic breakthrough.

Recently, there has been growing interest in solving the maximum s-t flow problem in the challenging *dynamic* setting, where the goal is to maintain a flow (or its value) under edge insertions and deletions and answer queries about the maintained flow efficiently. For directed graphs, there are strong conditional hardness results [Dah16, HKNS15] showing a lower bound of $\Omega(n)$ and $\Omega(\sqrt{m})$ amortized update time for maintaining *exact* maximum flow, assuming the OMv conjecture. These strong polynomial lower bounds have motivated a shift in focus toward maintaining *approximate* maximum flows.

Research on maintaining approximate maximum flows in the dynamic setting can be categorized into the following three lines of works. The first line of work [CGH+20, GRST21, vdBCK+24a] deals

---

[1]In this paper, we use $\widetilde{O}(X)$ to denote $O(X \operatorname{polylog}(X))$ and $\widehat{O}(X)$ to denote $O(X^{1+o(1)})$.

Setting	Apx. Factor	Flow/Value	Directed	Weighted	Update Time	Reference
Incremental	1	Flow	Yes	Yes	$O(F^*)$	[Hen97, GK21]
	1	Flow	No	No	$\widetilde{O}(n^{2.5}m^{-1})$	[GH23]
	$1+\epsilon$	Flow	Yes	No	$\widehat{O}(m^{0.5}\epsilon^{-0.5})$	[GH23]
	$1+\epsilon$	Flow	Yes	Yes	$\widehat{O}(n^{0.5}\epsilon^{-1})$	[vdBLS23]
	$1+\epsilon$	Flow	No	Yes	$O(m^{o(1)}\epsilon^{-3})$	[vdBCK$^+$24b]
	$1+\epsilon$	Flow	Yes	Yes	$O(m^{o(1)}\epsilon^{-1})$	[CKL$^+$24]
	$1+\epsilon$	Flow	No	No	$\widetilde{O}(nF^*m^{-1}\epsilon^{-1})$	Theorem 1
Decremental	$1+\epsilon$	Value	Yes	Yes	$O(m^{o(1)}\epsilon^{-1})$	[vdBCK$^+$24a]
Fully dynamic	$\widetilde{O}(\log n)$	Value	No	Yes	$\widehat{O}(n^{0.667})$	[CGH$^+$20]
	$m^{o(1)}$	Value	No	No	$m^{o(1)}$	[GRST21]
	$m^{o(1)}$	Value	No	Yes	$m^{o(1)}$	[vdBCK$^+$24a]

Table 1: Results on dynamic max flow. The "Flow/Value" column indicates whether the algorithm maintains an actual flow or just the value of a flow. The update time stated is the amortized time over all updates.

with the fully dynamic setting, supporting both edge insertion and deletions and is based on dynamically maintaining tree-based cut approximations. However, these algorithms require at least a logarithmic loss in the quality of the maintained flow. The second line of work [Hen97, GK21, GH23] is purely combinatorial and works by repeatedly finding augmenting paths in an incremental residual graph together with a lazy rebuilding technique. While these algorithms achieve sub-linear update time, it is not clear how to use them to go beyond the $\sqrt{n}$ barrier on the update time. To address this challenge, the third line of work [vdBLS23, vdBCK$^+$24b, CKL$^+$24, vdBCK$^+$24a] leverages continuous optimization techniques to maintain a $(1-\epsilon)$-approximate max flow in the incremental (only edge insertions) and decremental (only edge deletions) settings in $m^{o(1)}$ update time. Table 1 offers a detailed summary of state-the-art results on dynamic maximum flow algorithms.

Focusing on the $(1-\epsilon)$-approximation regime, the recent partially dynamic maximum flow suffer from the following two main drawbacks: (i) they depend on the powerful yet intricate machinery of continuous optimization methods, such as interior point methods, monotone multiplicative weights updates, and dynamic min-ratio cycle/cut problems, (ii) all existing algorithms inherently incur an $m^{o(1)}$ factor in the update time, a limitation that arises in many modern dynamic graph-based data structures and appears difficult to overcome. This leads to the following fundamental question:

> Is there an incremental maximum flow algorithm that achieves $(1-\epsilon)$ approximation with polylogarithmic update time?

We answer this question in the affirmative for dense graphs that are undirected and uncapacitated, as summarized in the theorem below.

Theorem 1. *Given any $\epsilon \in (0,1)$, there is an incremental randomized algorithm that maintains a $(1-\epsilon)$-approximate maximum s-t flow f under edge insertions on an undirected uncapacitated n-vertex graph G with high probability in total time $O(m\log(n)\alpha(n) + nF^*\log^3(n)/\epsilon)$, where m is the number of edge insertions, F^* is the value of the max flow after m insertions, and $\alpha(n)$ is the inverse Ackermann function.*

Note that in addition to dense graphs ($m = \Omega(n^2)$), our algorithm achieves polylogarithmic amortized update time even for graphs with small flow value $F^* = \widetilde{O}(m/n)$, regardless of their density. An important feature of our algorithm is that it builds upon arguably simple and classic combinatorial techniques, such as residual graph sparsification and cut sparsification.

1.1 Technical Overview

The main idea underlying our result is to dynamize the static algorithm by Karger and Levine [KL15] (abbrv. KL algorithm) that computes an *exact* max flow in $\widetilde{O}(m+nF^*)$ time. We start by reviewing their static construction and then identify the challenges in the dynamic setting, along with our approach to overcome them. We present the algorithm and its analysis in full in Section 3.

The KL algorithm proceeds by repeatedly sampling $\rho = O(n \log n)$ edges from the residual graph, searching for an augmenting path in this sample, and then augmenting along this path before resampling again. When the search fails to find an augmenting path, the number of sampled edges is doubled to 2ρ and this process is repeated until the sampled graph becomes as large as the original graph. Denoting the value of the maximum flow by F^*, they show that at least $F^*/2$ of the augmentations are performed on the sparsest graph with ρ edges, at least $F^*/4$ of the augmentations are performed on the graph with 2ρ edges and so on. After spending $\widetilde{O}(m)$ on pre-processing to compute the sampling probabilities, the total running time thus adds up to $\approx \sum_i (F^*/2^i) \cdot 2^{i-1}\rho$, which gives the required $\widetilde{O}(m + nF^*)$ bound. Crucially, the last augmentation from $F^* - 1$ to F^* is performed on a graph of size $\Omega(m)$.

This final augmentation, which requires a sample of size $\Omega(m)$, effectively invalidates any attempts to use the KL approach for obtaining an incremental *exact* max flow algorithm. To understand why, observe that after every edge insertion, the algorithm needs to check if this edge insertion creates a new $s \to t$ augmenting path in the sampled graph. As we discuss below, this can be done in two ways, but both lead to algorithmic dead ends.

We could try to use an incremental single source reachability data structure (e.g., the one by Italiano [Ita86]) to detect augmenting paths in the sampled graph, but since the actual augmentation reverses the direction of edges, we would need to reinitialize the incremental data structure on the sample after each augmentation. Since F^* augmentations are performed on samples of size $\Omega(m)$, this leads to a time bound of $\Omega(mF^*)$. As a concrete example, consider an initially empty bipartite graph $G = (S \cup T, \emptyset)$, with $s \in S$ and $t \in T$. In the first phase, insert all edges between $S \times T \setminus \{t\}$, then in the second phase, insert all the remaining edges of the type $\{(v, t)\}_{v \in S}$. Each edge insertion in the second phase increases the max flow by 1, and any algorithm based on a KL-type approach with an incremental reachability data structure needs to check an $\Omega(m)$-edge graph to perform each augmentation.

The second approach would be to use a *fully dynamic* directed $s \to t$ reachability data structure. This would solve the above problem, as an augmentation in the residual graph can be simulated using $O(n)$ directed edge deletions and insertions. However, fully dynamic $s \to t$ reachability is known to admit strong conditional lower bounds – for any $\eta > 0$, no algorithm can achieve $O(n^{1-\eta})$ worst-case update time and $O(n^{2-\eta})$ worst-case query time, assuming the OMv conjecture [HKNS15].

Interestingly, we show that the first approach can be extended to the incremental setting if we relax our algorithm to maintain a $(1 - \epsilon)$-*approximate* s-t max flow instead of an exact one. The high-level reason why is as follows: To show that at least $F^*/2^i$ of the augmentations are found in the sampled graph of size $2^{i-1}\rho$, Karger and Levine prove that when at least $F^*/2^i$ of the max flow still remains to be augmented, a sampled graph of size $2^{i-1}\rho$ suffices to preserve the existence of an augmenting path whp. For our setting, if we want to maintain a $(1 - \epsilon)$-approximate max flow, the above claim with $i = \log(1/\epsilon)$ tells us that it suffices to maintain a sparsifier with just $O(n \log(n)/\epsilon)$ edges incrementally. This then removes the need to augment on an $\Omega(m)$-edge graph which lets us circumvent the above barrier. While incrementally maintaining the sparsifier used by Karger and Levine would be highly non-trivial, we show that we can both efficiently maintain a different sparsifier incrementally, and that the new sparsifier is powerful enough to recover the guarantees required for the KL approach to work (at the cost of a $\log n$ factor in the sparsifier size).

The challenge of incrementally maintaining the KL sparsifier is due to its fragility under edge insertions. The KL sampling depends on a parameter called the *strong connectivity* of an edge [BK15], and in the incremental setting, a single edge insertion could modify the strong connectivities (and thus the sampling probabilities) of *all* other edges of the graph. As a result, no existing algorithm can even just maintain the sampling probabilities, let alone maintain an actual sample from the corresponding distribution. The same issue arises for other well-known connectivity measures used for sampling such

as edge-connectivity [FHHP19] and effective resistances [SS11], and even maintaining approximations to them is quite involved [DGGP18, DGGP19, CGH$^+$20, GLP21]. Instead, our key observation is to use *Nagamochi-Ibaraki (NI) indices* [NI92] for our sampling because of their resilience to incremental updates.

While the other connectivity parameters mentioned are unique for an edge, the NI index is determined by the forest packing constructed, and different packings gives rise to different indices. This flexibility in choosing the index allows us to maintain the same value even under edge updates, which we crucially use to maintain the sampling probabilities. The resilience property we use is as follows: when inserting an edge, the NI index of all other edges remains the same, and our task is to only determine the NI index of the newly inserted edge efficiently (which then doesn't change in the future). Efficient maintenance of NI indices is discussed in Section 4.

While we now maintain a set of sampling probabilities using NI indices, it is unclear how they can be used to replace strong connectivity in the KL algorithm. We extend the general result of Fung et al. [FHHP19] on using a wide variety of connectivity parameters for cut sparsification on *undirected* graphs to showing that the same parameters also lead to cut sparsification on balanced directed graphs, which cover residual graphs that arise in our algorithm as a special case. Their results sample each edge independently, which in our setting leads to a prohibitive $\Omega(m)$ time for sampling. We adapt this to repeated sampling from a probability distribution, where we only spend $O(\log n)$ time per edge for a total $O(n \log^2(n)/\epsilon)$ edges sampled. We present this result in full detail in Section 5.

Finally, we simplify our algorithm by not maintaining an exact sample from the distribution at each time step, but an oversample. Specifically, we obtain a sample of the correct size ($O(n \log^2(n)/\epsilon)$) right after an augmentation, but between two augmentations, we add edges directly to the sample instead of trying to maintain the distribution. This could blow up the number of edges in the sample to $\Omega(m)$ at some time steps, but since each edge is added this way to at most one sample throughout the algorithm, the total time bound of $\widetilde{O}(m + nF^*/\epsilon)$ still holds.

2   Preliminaries

2.1   Graphs and flows

Flows   Our algorithmic results are on undirected uncapacitated graphs $G = (V, E)$. We denote $n = |V|$ and $m = |E|$. For two vertices $s, t \in V$, an s-t flow $f \in \mathbb{R}^m$ assigns a value f_e to each edge such that $\sum_{e \sim v} f_e = 0$ for all $v \neq s, t$ with $|f_e| \leq 1 \; \forall e$. The value of a flow f is $F(f) = \sum_{e \sim s} f_e$, and the maximum flow $f^* = \text{argmax}_{s\text{-}t \text{ flows } f} F(f)$, with value $F^* = F(f^*)$. We use F without the argument f when the flow is clear from context. For any $\alpha \leq 1$, an s-t flow f is an α-approximate flow if $F \geq \alpha \cdot F^*$. If $\alpha > 1$, then the statement holds with $\alpha' = 1/\alpha$.

Model   In the incremental model, we start with an empty graph with no edges, and the graph is revealed as a sequence of edge insertions $e_1, e_2, \ldots$, leading to graphs $G_1, G_2, \ldots$. Our goal is to maintain, after each edge insertion e_i, an s-t flow f that is a $(1 - \epsilon)$ approximation to the max s-t flow f_i^* in the current graph G_i. We use F_i^* to denote the value of the flow f_i^*, and we use F^* to denote the final max s-t flow value after all edge insertions.

Directed graphs   We denote by $\vec{G} = (V, \vec{E})$ a directed (multi-)graph without edge capacities. An edge of integer capacity $x \in \mathbb{Z}^+$ is represented using x parallel edges. For any subset of vertices $S \subseteq V$, $\bar{S} = V \setminus S$ is the complement of S, and $\partial_{\vec{G}}(S)$ denotes the set of edges leaving S, which is also denoted by $\vec{C}(S) = \partial_{\vec{G}}(S)$. Similarly, we use $\bar{\vec{C}}(S) = \partial_{\vec{G}}(\bar{S})$ to denote the set of edges entering S. We use $\vec{C}$ and $\bar{\vec{C}}$ without the argument S when the subset is clear from context. The capacity $u(\vec{C}) = |\vec{C}|$ of the cut $\vec{C}$ is the number of edges in $\vec{C}$.

Underlying undirected graph   For any directed (multi-)graph $\vec{G}$, we denote by G the underlying undirected (multi-)graph obtained from $\vec{G}$ by forgetting edge orientations, and $\partial_G(S)$ is the set of edges leaving S, which is also denoted by $C(S) = \partial_G(S)$. For a directed cut $\vec{C} = \partial_{\vec{G}}(S)$, we use $C = \partial_G(S)$ to denote the underlying undirected cut in G.

Residual graph   For any undirected, uncapacitated graph G, the corresponding residual graph for the zero flow $f = 0^m$ is given by G_{0^m} where each edge (u, v) in G is replaced by two directed edges $\vec{uv}$ and $\vec{vu}$. For any G and a non-zero flow f, the residual graph G_f is constructed the following way: start with the graph G_{0^m} as above, and for every edge e that carries flow in the direction $u \to v$, replace the edge $\vec{uv}$ in G_{0^m} with the edge $\vec{vu}$. In that case there are two edges $\vec{vu}$ in G_f. A flow f^* is a maximum flow if and only if G_{f^*} contains no $s \to t$ paths.

2.2   Sampling parameters

We define our importance parameter for sampling edges next. Since the residual graph has multiple edges between the same vertex pairs, we consider multi-graphs (which allow parallel edges) for the following definition.

Definition 1 (NI index [NI92]). Given an undirected, unweighted (multi-)graph G, an ordered sequence of edge-disjoint spanning forests $T_1, T_2, \dots$ of is said to be an NI forest packing of G if each T_i is a maximal spanning forest on $G \setminus \cup_{1 \le j < i}\{T_j\}$, and $\cup_i\{T_i\}$ is a partition of all the edges of G. The NI index ℓ_e of an edge e is the index i of the forest T_i that e belongs to.

While the NI indices are the connectivity parameters that we will use for our sampling scheme due to their resilience to incremental graph updates, the following connectivity parameter is required for our directed sparsification proofs and cut-counting.

Definition 2 (Edge-connectivity). Given an undirected unweighted (multi-)graph G and two vertices u and v, the edge connectivity between u and v is defined to be the value of the minimum cut (and thus also the maximum flow) between u and v in G. For an edge $e = (u, v)$, we define the edge-connectivity of e to be the edge connectivity between u and v.

As we perform *non-uniform* sampling of the edges, we maintain connectivity parameters λ_e that influence the sampling probabilities. The NI index of an edge and the edge-connectivity of an edge are examples of such connectivity parameters. We will show concentration bounds for classes of edges whose λ_es are close to each other, which motivates the following definition.

Definition 3 (Connectivity class). Given an undirected graph G and connectivity parameters λ_e for each edge, taking $\Lambda = \max_e \lceil \log \lambda_e \rceil + 1$, the connectivity classes $\mathcal{F} = \{F_i : 1 \le i \le \Lambda\}$ are given by

$$F_i = \{e : 2^{i-1} \le \lambda_e \le 2^i - 1\}.$$

For any connectivity class F_i, the graph (V, F_i) does not have the same edge-connectivity properties that $G = (V, E)$ did. An edge $e \in F_i$ could have edge-connectivity $\Omega(n)$ in G and 1 in (V, F_i), e.g., a graph with a single (s, t) edge and $\Omega(n)$ parallel paths from s to t of length two. The following definitions from [FHHP19] are used to construct a subgraph of G that is slightly larger than F_i that certify connectivity properties of edges in F_i.

Definition 4 (Π-connected decomposition [FHHP19]). Given an undirected graph G and connectivity parameters λ_e, a decomposition $\mathcal{G} = (G_i = (V, E_i))_{1 \le i \le \Lambda}$ is a sequence of subgraphs of G that satisfy $F_i \subseteq E_i$ for all i with $\Lambda = \max_e \lceil \log \lambda_e \rceil + 1$. Further, for a sequence of parameters $\Pi = (\pi_i)_{1 \le i \le \Lambda}$, the decomposition satisfies Π-connectivity if all edges $e \in F_i$ have edge-connectivity at least π_i in G_i for all i.

Note that the same edge is allowed to appear in multiple sets E_i.

Definition 5 (γ-overlap [FHHP19]). For any $\gamma \geq 1$, an undirected graph G and its Π-connected decomposition $\mathcal{G}$ satisfies γ-overlap if for all i and for all cuts $C = \partial_G(S)$,

$$\sum_{i=0}^{\Lambda} u(C \cap E_i) \cdot \frac{2^{i-1}}{\pi_i} \leq \gamma \cdot u(C)$$

2.3 Balance and cut counting

Imbalance of directed graphs   Preserving various graph properties by subsampling is difficult on general directed graphs. However, sampling algorithms can be successfully applied when the directed graph exhibits some nice structure, for example, they have a similar number of edges crossing a cut in both directions. Eulerian graphs have exactly the same number of edges crossing any cut in both directions, and more pertinently, the residual graph of a flow that is not a $(1 - \epsilon)$ approximate max flow has at least an $O(\epsilon)$ fraction of edges crossing any cut in one direction over the other.

Definition 6 (balance [EMPS16]). Given a directed (multi-)graph $\vec{G} = (V, \vec{E})$, the (im)balance of a cut $\vec{C} = \partial_{\vec{G}}(S)$ is defined as

$$\beta(\vec{C}) = \frac{u(\vec{C})}{u(\overleftarrow{C})}$$

where $\overleftarrow{C} = \partial_{\vec{G}}(\bar{S})$ is the set of edges in the other direction of the cut.

Counting cut projections   Karger shows that there are at most $n^{2\alpha}$ distinct cuts whose size is $\leq \alpha\lambda$, where λ is the global mincut. We, like Fung et al. [FHHP19], look at larger cuts in the graph and require a tighter bound than the one given by Karger, which requires the following definition.

Definition 7 (directed k-projection). Let $\vec{G} = (V, \vec{E})$ be a directed (multi-)graph and let G be the corresponding undirected graph. The k-projection of any cut $\vec{C} = \partial(S)$ in $\vec{G}$ is $\vec{C} \cap H_k$ where

$$H_k = \left\{ \vec{e} \in \vec{E} : \text{the edge-connectivity of e in } G \text{ is at least } k \right\}$$

2.4 Data Structures

We also need three classic data structures for our results, which we produce here.

Fact 1 (UNIONFIND [TvL84]). There is a data structure that supports the following operations

- ADD(u): Adds $\{u\}$ to the set of sets in $O(1)$ time.
- FIND(u): Finds the representative of the set that u belongs to in amortized $O(\alpha(n))$ time.
- UNION(u, v): Replaces the sets that u and v belong to with their union in $O(1)$ time.

where n is the number of elements present at that time, and $\alpha(n)$ is the inverse Ackermann function.

Fact 2 (SINGLESOURCEREACHABILITY [Ita86]). There is a data structure that supports the following operations on a graph $\vec{G}$ with n vertices

- INITIALIZE($\vec{G}, s$): Initializes a data structure on $\vec{G}$ with special vertex s in $O(m + n)$ time.
- INSERT($e = \vec{uv}$): Inserts the directed edge $\{e = \vec{uv}\}$ into $\vec{G}$ in amortized $O(1)$ time.
- REACHABLE(t): Returns "True" if t is reachable from s in $\vec{G}$ in amortized $O(1)$ time.
- GETPATH(t): Returns a directed $s \to t$ path in $\vec{G}$ in time proportional to the length of the path.

Fact 3 (BINARYSEARCHTREE). There is a data structure that supports the following operations

- INITIALIZE: Initializes the binary search tree data structure.
- INSERT(e, x): Inserts the key e with value x in amortized $O(\log n)$ time.
- SEARCH(x): Returns the largest key e with value $\leq x$ in amortized $O(\log n)$ time.

where n is the number of items inserted into the data structure at time of operation.

High probability bounds   Finally, we say that an algorithm satisfies a guarantee *with high probability* if the guarantee holds with probability at least $1 - 4/n$.

3   Incremental Maximum Flow

Our main result in this section is the following:

Theorem 1. *Given any $\epsilon \in (0, 1)$, there is an incremental randomized algorithm that maintains a $(1-\epsilon)$-approximate maximum s-t flow f under edge insertions on an undirected uncapacitated n-vertex graph G with high probability in total time $O(m \log(n)\alpha(n) + nF^* \log^3(n)/\epsilon)$, where m is the number of edge insertions, F^* is the value of the max flow after m insertions, and $\alpha(n)$ is the inverse Ackermann function.*

Algorithm 1: Algorithm for incremental approximate maximum flow on undirected, uncapacitated graphs

---

1 Function INITIALIZE(ϵ)**:**
2 $\rho \leftarrow 5390 \cdot n \log^2(n)/\epsilon$
3 $f \leftarrow \emptyset, F \leftarrow 0, G_f \leftarrow (V, \emptyset), H \leftarrow G_f$
4 $D \leftarrow$ SINGLESOURCEREACHABILITY.INITIALIZE(H, s)
5 $K \leftarrow$ INCNISAMPLE.INITIALIZE   ▷ Algorithm 4
6 Function INSERT($e = (u, v)$)**:**
7   Add $\vec{uv}$ and $\vec{vu}$ to H and D
8 K.INSERT(e)
9   **if** D.REACHABLE(t) **then**
10 $p \leftarrow D$.GETPATH(t)
11   update f and G_f by augmenting along p, and update $F \leftarrow F + 1$
12   reset $H \leftarrow (V, \emptyset)$
13   **for** $i = 1, 2, \ldots, \rho$ **do**
14 $(u, v) \leftarrow K$.SAMPLE
15   **for** $\vec{ab} \in \{\vec{uv}, \vec{vu}\}$ **do**
16   **if** $\vec{ab} \in G_f$ **then**
17   Add $\vec{ab}$ to H
18   reinitialize $D \leftarrow$ SINGLESOURCEREACHABILITY.INITIALIZE(H, s)
19 Function QUERY(*flow/value*)**:**
20   **if** query = flow **then return** f **else return** F

We present the algorithm achieving the guarantees in Algorithm 1. The algorithm works in phases. At the start of a new phase, we sample $O(n \log^2(n)/\epsilon)$ edges from the residual graph using a specific probability distribution which we will discuss later, and add them to the *sampled graph H*. On top of H, we run an incremental directed single-source reachability algorithm, called D, starting from s. At the beginning, and after each edge insertion, we query D if there exists an $s \to t$ path in the sample. As long as D does not return "Yes", for each edge insertion $e = (u, v)$, we add edges $\vec{uv}$ and $\vec{vu}$ to D. When D returns "Yes", this corresponds to an $s \to t$ path in the residual graph, and we retrieve this path from D, augment along this path in G_f, and start a new phase.

We need to argue about the time bounds and the correctness. For correctness, we show that whenever an edge insertion increases the max flow to a value larger than a $(1 - \epsilon)^{-1}$ factor over the current flow maintained by the algorithm, the existence of an $s \to t$ path in the sampled graph H is guaranteed with high probability. Since H is a subsample of the residual graph G_f, this provides us an augmenting path in G_f along which we augment the flow. With a union bound over the at most n times when this can happen (since the max flow value is at most n and each augmentation increases the flow by 1), this will imply that at the end of every time step, the algorithm maintains a valid $(1 - \epsilon)$-approximate s-t flow with high probability. To show this claim, we first show that an extension of Fung et al.'s high probability result [FHHP19] on independent sampling for cut sparsification on undirected

---

Algorithm 2: Algorithm for directed balanced sparsification

    Input: Directed graph $\vec{G} = (V, \vec{E})$, connectivity parameters $\{\lambda_e\}_{e \in E}$, parameters $\beta, \gamma \geq 1$, and $\epsilon < 1$
    Output: Sparsified graph $\vec{H} = (V, \vec{F}, w)$

1   $\rho \leftarrow \frac{128\gamma(\beta+1)\ln n}{0.38\epsilon^2} \cdot \sum_{e \in E} \lambda_e^{-1}$

2   $\vec{H} \leftarrow (V, \emptyset, w)$

3   **for** $i \in 1, 2, \ldots, \rho$ **do**

4 | Sample an edge e from the probability distribution $\{p_e = \lambda_e^{-1} / \sum_e \lambda_e^{-1}\}$

5 | Insert e into $\vec{H}$ with weight $w_e = (\rho \cdot p_e)^{-1}$ (additively increasing its weight if e already exists in $\vec{H}$)

---

graphs also extends to repeated sampling for cut sparsification on balanced directed graphs. When combined with the fact that the residual graph of a flow that is not a $(1 - \epsilon)$-approximate max flow is $\Omega(\epsilon)$-balanced, this shows that sampling approximately $n \log^2(n)/\epsilon$ edges based on the probability distribution discussed below will preserve directed cuts, and thus also the existence of an augmenting path in residual graphs of non-$(1-\epsilon)$-approximate flows. Our initial sample at the beginning of a phase already satisfies the size requirement above, and since we add further edges directly to the maintained sample, we always oversample the rate required for the augmenting path preservation guarantee.

For the time bounds, assuming that the sampling can be done in time approximately $O(\log n)$ per sample and that the reachability data structure runs in total time proportional to the number of edges in D, we will show that the algorithm runs in $\widetilde{O}(m + nF^*/\epsilon)$ time. If we had maintained the size of the sampled graph at $\widetilde{O}(n/\epsilon)$ throughout a phase, then the claimed time bound would follow since D runs on a graph of size $\widetilde{O}(n/\epsilon)$ in each phase, at the end of which the value of the flow increases by 1, and the max flow is bounded by F^* which bounds the number of phases by F^* as well. However, the size of the sampled graph could increase to $\Omega(m)$ if no paths are found by D over all edge insertions, so this argument does not immediately work. The bound follows from a slight modification to the above argument: in each phase, denote an edge to be "old" when it is inserted into H due to sampling, and "new" when it is inserted into H directly on insertion in G. Then the claimed time bound follows since each edge appears as "new" in at most one phase throughout the algorithm, and the previous argument still holds for the "old" edges of each phase since there are at most $\widetilde{O}(n/\epsilon)$ of them per phase.

Finally, we maintain NI indices incrementally for the probability distribution, which we discuss in Section 4. We use the following definition in our proofs.

Definition 8 (Phase). Let $t_0 = 0$, and let t_i be the time step when the ith augmenting path is found. Let m_k be the number of edges inserted in time steps $(t_{k-1}, t_k]$. Phase k refers to the computations performed after the $(k-1)^{th}$ augmentation finishes until the kth augmentation ends.

We also use the following two statements directly, whose proofs can be found in Section 4 and Section 5 respectively.

Lemma 1. *There is an incremental data structure that maintains an NI forest packing (and thus the NI index ℓ_e of each edge) of an undirected graph under edge insertions in total time $O(m \cdot \log m \cdot \alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function and m is the number of edges in the final graph. At any point of time, it also allows querying for an edge sampled from the probability distribution $\left\{\ell_e^{-1}/L\right\}_{e \in E}$ where $L = \sum_e \ell_e^{-1}$ in amortized time $O(\log n)$ for each sample.*

Theorem 2. *Let $\vec{G} = (V, \vec{E})$ be any directed (multi-)graph, and G be the underlying undirected graph. Let $\{\lambda_e\}_{e \in E}$ be some connectivity parameters in G, and $\beta, \gamma \geq 1$ and $\epsilon < 1$ be input parameters. Let $\vec{H} = (V, \vec{F}, w)$ be the random graph with $O\left(\gamma\beta \ln(n) \cdot \sum_{e \in E} \lambda_e^{-1}/\epsilon^2\right)$ edges generated as in Algorithm 2.*

If there exists a Π-connected decomposition $\mathcal{G} = \{G_i = (V, E_i) : 1 \leq i \leq \Lambda\}$ of G that satisfies γ-overlap, then for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ of balance at least β^{-1} in $\vec{G}$, it holds simultaneously with probability $\geq 1 - 4/n^2$ that

$$(1 - \epsilon) \cdot u(\vec{C}) \leq w(\vec{C}) \leq (1 + \epsilon) \cdot u(\vec{C}).$$

3.1 Running time

Lemma 2. *Phase k takes total time $O(m_k \log(n)\alpha(n) + n\log^3(n)/\epsilon)$, where m_k is the number of edges inserted in Phase k and $\alpha(n)$ is the inverse Ackermann function.*

Proof. TOPROVE 0 □

Lemma 3. *The algorithm takes total time $O(m\log(n)\alpha(n) + n\log^3(n)F^*/\epsilon)$ where F^* is the value of the final max flow after all edge insertions.*

Proof. TOPROVE 1 □

3.2 Correctness

We first collect a few general statements that will be useful in the final proof.

Lemma 4. *Let f be any s-t flow in an undirected graph G of value F, and let G_f be its corresponding residual graph. Suppose $F \le (1-\epsilon)F^*$, where F^* is the value of the maximum flow. Then every cut in G_f is at least $\epsilon/2$-balanced.*

Proof. TOPROVE 2 □

Lemma 5. *Let $\{\ell_e\}_{e\in E}$ be a set of NI indices for an undirected (multi-)graph G with m edges. Then $\sum_e \ell_e^{-1} \le 2n\log m$.*

Proof. TOPROVE 3 □

Since we work with the residual graph which has two copies of each edge of G, we prove the following lemma to show that an NI forest packing of G can be easily transformed into an NI forest packing of the undirected residual graph.

Lemma 6. *Let $\{\ell_e\}_{e\in E}$ be a set of NI indices for an undirected, simple graph $G = (V, E)$. Let E' be a copy of the edges in E. Then, for the multigraph $H = (V, E\cup E')$ which doubles the edges in E, assigning $h_e = 2\ell_e - 1$ and $h_{e'} = 2\ell_e$ is a set of NI indices for H.*

Proof. TOPROVE 4 □

Lemma 7 ([FHHP19]). *Let $T_1, T_2, \dots$ be an NI forest packing of an undirected graph G leading to NI indices ℓ_e, and let $\mathcal{F} = \{F_i\}$ be the corresponding connectivity classes. Then $E_1 = F_1$ and $E_i = F_{i-1}\cup F_i$ for $i \ge 2$ is a decomposition that satisfies Π-connectivity and γ-overlap for $\pi_i = 2^{i-1}$ and $\gamma = 2$, i.e., for all i and for all cuts $C = \partial_G(S)$,*

$$\sum_{i=0}^{\Lambda} u(C\cap E_i) \le 2\cdot u(C).$$

Using these, we will first show that sampling $\Omega(n\log^2(n)/\epsilon)$ edges from the residual graph of a non-$(1-\epsilon)$-approximate flow preserves an augmenting path with high probability.

Lemma 8. *Consider an undirected, uncapacitated graph G with some s-t flow f of value F and a max s-t flow of value F^* such that $F \le (1-\epsilon)F^*$. Let ℓ_e be a NI index of edge e. If a sample of $5390\cdot n\log^2(n)/\epsilon$ edges is chosen i.i.d. with probability $\left\{\ell_e^{-1}/\sum_e \ell_e^{-1}\right\}_{e\in G_f}$ from the residual graph G_f, then there exists an augmenting path in the sample with probability $1 - 4/n^2$.*

Proof. TOPROVE 5 □

We will now show that at any fixed time step, the augmenting path guarantee holds with high probability.

Lemma 9. *At the beginning of any phase k, if $F \leq (1 - \epsilon)F^*_{t_{k-1}}$, then there exists an augmenting path in H with probability $1 - 4/n^2$. Here, F^*_i is the max flow at time i, and t_{k-1} is the time step when the $(k-1)$-th augmentation ends.*

Proof. TOPROVE 6 □

Lemma 10. *At each time step i, whenever $F < (1 - \epsilon)F^*_i$, there exists an augmenting path in H with probability $1 - 4/n^2$. Here, F^*_i is the value of the max flow at time i.*

Proof. TOPROVE 7 □

Finally, we take a simple union bound to extend the guarantee to all time steps.

Lemma 11. *Algorithm 1 maintains a $(1 - \epsilon)$-approximate s-t max flow at all times with probability $1 - 4/n$.*

Proof. TOPROVE 8 □

4 Incremental Nagamochi-Ibaraki indices

In this section, we show how to incrementally maintain (and sample from) Nagamochi-Ibaraki indices ℓ_e for each edge e. The main result of this section is the following:

Lemma 1. *There is an incremental data structure that maintains an NI forest packing (and thus the NI index ℓ_e of each edge) of an undirected graph under edge insertions in total time $O(m \cdot \log m \cdot \alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function and m is the number of edges in the final graph. At any point of time, it also allows querying for an edge sampled from the probability distribution $\{\ell_e^{-1}/L\}_{e \in E}$ where $L = \sum_e \ell_e^{-1}$ in amortized time $O(\log n)$ for each sample.*

We maintain Nagamochi-Ibaraki indices instead of other sampling parameters because of their stability under incremental updates. When an edge $e = (u, v)$ is inserted into a graph G, it could change other connectivity parameters (such as edge-connectivity, strong-connectivity, or the effective resistance) of *every existing edge* in the graph. However, the NI index of every other edge remains the same on edge insertion, which is a very desirable property for dynamic algorithms. This happens because while other graph properties are unique for an edge in a given graph, the NI index depends heavily on the particular forest packing used. It could vary wildly for the same edge, and could range from 1 up to n depending on the packing.

The incremental algorithm involves maintaining a collection of forests, where each forest is a union-find data structure on the vertices currently in that forest. On an edge insertion, we binary search across the forests to find the first one where its endpoints are disconnected, and insert the edge into that forest (and also adding the endpoints to the forest or initializing a new forest if necessary).

Due to the subtleties of implementing this, we present the data structure in full detail below. In particular, if we naively "initialized" a new forest by inserting *all* the vertices into the forest, then this would lead to a running time of $\Omega(n)$ per forest, which could be prohibitive. For the extreme case where n parallel edges are inserted between the same two vertices u and v, this necessitates initializing $\Omega(n)$ forests, which leads to $\Omega(n^2)$ total time for the naive implementation, as opposed to adding a vertex to a forest only when necessary (as done below with the *last_tree* variable).

Lemma 12. *Algorithm 3 maintains the NI forest packing at all times, and runs in total time $O(m \cdot \log m \cdot \alpha(n))$*

Proof. TOPROVE 9 □

Algorithm 3: Maintaining Incremental Nagamochi-Ibaraki Indices (IncNIIndex)

1 Function Initialize:
2 $\quad k \leftarrow 0$ ▷ Number of forests maintained
3 $\quad tree \leftarrow []$ ▷ Collection of forests
4 $\quad last\_tree[v] \leftarrow 0$ for all $v \in V$ ▷ last forest that v belongs to
5 Function Insert$(e = (u, v))$:
6 $\quad i \leftarrow$ FindTree$(u, v, \min(last\_tree(u), last\_tree(v)))$ ▷ find correct forest for e
7 $\quad$ **if** $i > k$ **then** $k \mathrel{+}= 1, tree[i] \leftarrow$ UnionFind.Initialize   ▷ add new forest
8 $\quad$ **for** $x \in \{u, v\}$ **do**
9 $\quad\quad$ **if** $i > last\_tree(x)$ **then** $last\_tree[x] \mathrel{+}= 1, tree[i]$.Add$(x)$ ▷ add x to forest i
10 $\quad tree[i]$.Union(u, v) ▷ connect u and v in forest i
11 $\quad$ **return** i
12 Function FindTree$(u, v, upper\_bound)$:
13 $\quad L \leftarrow 0, R \leftarrow upper\_bound$
14 $\quad$ **while** $L \leq R$ **do**
15 $\quad\quad M \leftarrow \lfloor (L + R)/2 \rfloor$ ▷ binary search for forest
16 $\quad\quad$ **if** IsConnected(u, v, M) **then** $L \leftarrow M + 1$ **else** $R \leftarrow M - 1$
17 $\quad$ **return** L
18 Function IsConnected(u, v, i):
19 $\quad$ **if** $i = 0$ **then return** True **else return** $tree[i]$.Find$(u) = tree[i]$.Find(v)

Algorithm 4: Incremental Nagamochi-Ibaraki sampling (IncNISample)

1 Function Initialize:
2 $\quad X \leftarrow$ IncNIIndex.Initialize
3 $\quad Y \leftarrow$ BinarySearchTree.Initialize
4 $\quad L \leftarrow 0$
5 Function Insert$(e = (u, v))$:
6 $\quad \ell_e \leftarrow$ X.Insert(e)
7 $\quad Y$.Insert(e, L)
8 $\quad L \leftarrow L + \ell_e^{-1}$
9 Function Sample:
10 $\quad z \leftarrow$ uniform random number in $[0, L]$
11 $\quad e \leftarrow Y$.Search(z)
12 $\quad$ **return** e

We quickly recall an example of a binary search tree insertion sequence before we prove the next lemma. In a binary search tree, if there are three consecutive insertions of (key, value) pairs $(a, 0)$, $(b, 1/3)$, and $(c, 1/2)$, then searching for any $z \in [0, 1/3)$ returns a, searching for $z \in [1/3, 1/2)$ returns b, and searching for $z \in [1/2, \infty)$ returns c.

Lemma 13. *Algorithm 4 maintains a sample from the correct distribution at all times, each insertion takes time $O(\log(m)\alpha(n))$ and each sample takes time $O(\log n)$.*

Proof. TOPROVE 10   □

Lemma 1 now follows from Lemma 12 and Lemma 13.

5 Balanced sparsification

We remark that while Cen et al. [CCPS21] obtain a cut sparsification result for balanced directed graphs, their result only works with strong connectivity and cannot be used with NI indices, which we need for our algorithm. In this section, we show the following result.

Theorem 2. *Let $\vec{G} = (V, \vec{E})$ be any directed (multi-)graph, and G be the underlying undirected graph. Let $\{\lambda_e\}_{e \in E}$ be some connectivity parameters in G, and $\beta, \gamma \geq 1$ and $\epsilon < 1$ be input parameters. Let $\vec{H} = (V, \vec{F}, w)$ be the random graph with $O\big(\gamma\beta\ln(n) \cdot \sum_{e \in E} \lambda_e^{-1}/\epsilon^2\big)$ edges generated as in Algorithm 2.*

If there exists a Π-connected decomposition $\mathcal{G} = \{G_i = (V, E_i) : 1 \leq i \leq \Lambda\}$ of G that satisfies γ-overlap, then for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ of balance at least β^{-1} in $\vec{G}$, it holds simultaneously with probability $\geq 1 - 4/n^2$ that

$$(1 - \epsilon) \cdot u(\vec{C}) \leq w(\vec{C}) \leq (1 + \epsilon) \cdot u(\vec{C}).$$

This is an extension of the result of [FHHP19] to balanced directed graphs, with the sampling scheme changed from sampling each edge independently with probability $\approx \text{polylog}(n)/(\lambda_e\epsilon^2)$ to sampling $\approx O(n\,\text{polylog}(n)/\epsilon^2)$ edges with probability distribution proportional to λ_e^{-1}. The proof proceeds by reducing the problem to showing concentration for each connectivity class where the connectivity parameters are close to each other. Inside each connectivity class F_i, the cuts are then collected by the size of the underlying undirected cut in G_i and concentration is shown for each collection separately. This involves combining the concentration for each cut in the collection, along with a cut-counting argument similar to the one on [FHHP19] (which is an extension of Karger's cut-counting). Finally, the concentration of each cut is shown using a standard Chernoff bound.

Concretely, we will show later that the following concentration holds for every connectivity class.

Lemma 14. *It holds for all $i \in [\Lambda]$ and for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ simultaneously with probability at least $1 - 4/n^2$ that*

$$\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| \leq \frac{\epsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

where $C = \partial_G(S)$ is the underlying undirected cut of $\vec{C}$ in G.

We first use it to prove Theorem 2.

Proof. TOPROVE 11 $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ $\square$

To show Lemma 14, we partition the cuts in a connectivity class based on the size of the underlying undirected cut, and show concentration in each of them separately. Formally, we will show later that

Lemma 15. *Let $\mathcal{C}_{ij}$ be the collection of all cuts $\vec{C} = \partial_{\vec{G}}(S)$ such that for the corresponding undirected cut $C = \partial_G(S)$ in G,*

$$\pi_i \cdot 2^j \leq u(C \cap E_i) \leq \pi_i \cdot 2^{j+1} - 1$$

Then for all cuts $\vec{C}$ in $\mathcal{C}_{ij}$ simultaneously, it holds with probability $1 - 2/n^{4 \cdot 2^j}$ that

$$\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| \leq \frac{\epsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

We first use this to prove Lemma 14.

Lemma 14. *It holds for all $i \in [\Lambda]$ and for all cuts $\vec{C} = \partial_{\vec{G}}(S)$ simultaneously with probability at least $1 - 4/n^2$ that*

$$\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| \leq \frac{\epsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

where $C = \partial_G(S)$ is the underlying undirected cut of $\vec{C}$ in G.

Proof. TOPROVE 12 $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ $\square$

To show Lemma 15, we show the bound for a single cut in Lemma 16 and use this with a directed cut counting argument as in Lemma 17.

Lemma 16. *For any single cut $\vec{C}$ in $\mathcal{C}_{ij}$, it holds with probability $1 - 2/n^{8 \cdot 2^j}$ that*

$$\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| \leq \frac{\epsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

Lemma 17. *Let $\vec{G} = (V, \vec{E})$ be a directed (multi-)graph, and G be the underlying undirected graph. Let $k \geq \lambda$ be any real number, where λ is the value of the global minimum cut in G. Consider all the cuts $\vec{C} = \partial_{\vec{G}}(S)$ in $\vec{G}$ such that $u(C) \leq \alpha k$ for the corresponding undirected cut $C = \partial_G(S)$, and let $\mathcal{C}_{\alpha}^{\downarrow k}$ be the set of all directed k-projections of these cuts. Then $|\mathcal{C}_{\alpha}^{\downarrow k}| \leq 2 \cdot n^{2\alpha}$.*

We first use them to show Lemma 15.

Lemma 15. *Let $\mathcal{C}_{ij}$ be the collection of all cuts $\vec{C} = \partial_{\vec{G}}(S)$ such that for the corresponding undirected cut $C = \partial_G(S)$ in G,*
$$\pi_i \cdot 2^j \leq u(C \cap E_i) \leq \pi_i \cdot 2^{j+1} - 1$$
Then for all cuts $\vec{C}$ in $\mathcal{C}_{ij}$ simultaneously, it holds with probability $1 - 2/n^{4 \cdot 2^j}$ that

$$\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| \leq \frac{\epsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

Proof. TOPROVE 13 □

We use a Chernoff bound of the following form to prove Lemma 16.

Lemma 18. *Let $X_1, \ldots, X_n$ be n independent random variables that take values in the range $[0, M]$ for some $M > 0$. Then, for any $\epsilon \in (0, 1)$ and $N \geq E[\sum_i X_i]$, we have*

$$\Pr\left[\left| \sum_{i=1}^n X_i - E\left[\sum_{i=1}^n X_i \right] \right| > \epsilon N \right] \leq 2 \exp(-0.38 \epsilon^2 N / M)$$

Lemma 16. *For any single cut $\vec{C}$ in $\mathcal{C}_{ij}$, it holds with probability $1 - 2/n^{8 \cdot 2^j}$ that*

$$\left| w(\vec{C} \cap F_i) - u(\vec{C} \cap F_i) \right| \leq \frac{\epsilon}{2} \cdot \left(\frac{u(C \cap E_i) \cdot 2^{i-1}}{\pi_i \cdot \gamma \cdot (\beta + 1)} + u(\vec{C} \cap F_i) \right)$$

Proof. TOPROVE 14 □

We use the following version of Chernoff bound from [FHHP19] to obtain Lemma 18.

Lemma 19 ([FHHP19]). *Let $Y_1, \ldots, Y_n$ be n independent random variables each taking values in $[0, 1]$. Let $\mu = E[\sum_i Y_i]$. Then for all $\delta > 0$, we have*

$$\Pr\left[\left| \sum_{i=1}^n Y_i - \mu \right| > \delta \mu \right] \leq 2 \exp\left(-0.38 \min\{1, \delta\} \delta \mu \right)$$

Lemma 18. *Let $X_1, \ldots, X_n$ be n independent random variables that take values in the range $[0, M]$ for some $M > 0$. Then, for any $\epsilon \in (0, 1)$ and $N \geq E[\sum_i X_i]$, we have*

$$\Pr\left[\left| \sum_{i=1}^n X_i - E\left[\sum_{i=1}^n X_i \right] \right| > \epsilon N \right] \leq 2 \exp(-0.38 \epsilon^2 N / M)$$

Proof. TOPROVE 15 □

Finally, we use the undirected projection result of [FHHP19] to prove Lemma 17.

Lemma 20 ([FHHP19]). *Let $G = (V, E)$ be an undirected (multi-)graph. Let $k \geq \lambda$ be any real number, where λ is the value of the global minimum cut in G. Consider all the cuts $C = \partial_G(S)$ in G such that $u(C) \leq \alpha k$, and let $\mathcal{K}_\alpha^{\downarrow k}$ be the set of all undirected k-projections of these cuts, where the undirected k-projection of a cut $C = \partial_G(S)$ is $C \cap H_k$ and H_k is the set of edges with edge-connectivity at least k. Then $|\mathcal{K}_\alpha^{\downarrow k}| \leq n^{2\alpha}$.*

Lemma 17. *Let $\vec{G} = (V, \vec{E})$ be a directed (multi-)graph, and G be the underlying undirected graph. Let $k \geq \lambda$ be any real number, where λ is the value of the global minimum cut in G. Consider all the cuts $\vec{C} = \partial_{\vec{G}}(S)$ in $\vec{G}$ such that $u(C) \leq \alpha k$ for the corresponding undirected cut $C = \partial_G(S)$, and let $\mathcal{C}_\alpha^{\downarrow k}$ be the set of all directed k-projections of these cuts. Then $|\mathcal{C}_\alpha^{\downarrow k}| \leq 2 \cdot n^{2\alpha}$.*

Proof. TOPROVE 16 □

Acknowledgements

References (with cited pages)

AHK12 Sanjeev Arora, Elad Hazan, and Satyen Kale.
The multiplicative weights update method: a meta-algorithm and applications.
Theory Comput. 2012, 8(1):121–164. (1)

AMOR95 Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, and MR Reddy.
Applications of network optimization.
Handbooks in Operations Research and Management Science 1995, 7:1–83. (1)

BK15 András A. Benczúr and David R. Karger.
Randomized approximation schemes for cuts and flows in capacitated graphs.
SIAM J. Comput. 2015, 44(2):290–319. (3)

CCPS21 Ruoxu Cen, Yu Cheng, Debmalya Panigrahi, and Kevin Sun.
Sparsification of directed graphs via cut balance.
In *International Colloquium on Automata, Languages, and Programming (ICALP)* 2021. (11)

CGH$^+$20 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak.
Fast dynamic cuts, distances and effective resistances via vertex sparsifiers.
In *Foundations of Computer Science (FOCS)* 2020. (1, 2, and 4)

CKL$^+$22 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva.
Maximum flow and minimum-cost flow in almost-linear time.
In *Foundations of Computer Science (FOCS)* 2022. (1)

CKL$^+$24 Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg.
Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow.
In *Symposium on Theory of Computing (STOC)* 2024. (1 and 2)

Dah16 Søren Dahlgaard.
On the hardness of partially dynamic graph problems and connections to diameter.
In *International Colloquium on Automata, Languages, and Programming (ICALP)* 2016. (1)

DGGP18   David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng.
Fully dynamic effective resistances.
CoRR 2018. (4)

DGGP19   David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng.
Fully dynamic spectral vertex sparsifiers and applications.
In *Symposium on Theory of Computing (STOC)* 2019. (4)

EMPS16   Alina Ene, Gary L. Miller, Jakub Pachocki, and Aaron Sidford.
Routing under balance.
In *Symposium on Theory of Computing (STOC)* 2016. (6)

FHHP19   Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi.
A general framework for graph sparsification.
SIAM J. Comput. 2019, 48(4):1196–1223. (4, 5, 6, 7, 9, 12, 13, and 14)

GH23   Gramoz Goranci and Monika Henzinger.
Efficient data structures for incremental exact and approximate maximum flow.
In *International Colloqium on Automata, Languages, and Programming (ICALP)* 2023. (2)

GK21   Manoj Gupta and Shahbaz Khan.
Simple dynamic algorithms for maximal independent set, maximum flow and maximum matching.
In *Symposium on Simplicity in Algorithms (SOSA)* 2021. (2)

GLP21   Yu Gao, Yang P. Liu, and Richard Peng.
Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao.
In *Foundations of Computer Science, (FOCS)* 2021. (4)

GR98   Andrew V. Goldberg and Satish Rao.
Beyond the flow decomposition barrier.
J. ACM 1998, 45(5):783–797. (1)

GRST21   Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan.
The expander hierarchy and its applications to dynamic graph algorithms.
In *Symposium on Discrete Algorithms (SODA)* 2021. (1 and 2)

Hen97   Monika Rauch Henzinger.
A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity.
J. Algorithms 1997, 24(1):194–220. (2)

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 *Symposium on Theory of Computing (STOC)* 2015. (1 and 3)

Ita86   Giuseppe F. Italiano.
Amortized efficiency of a path retrieval data structure.
Theor. Comput. Sci. 1986, 48(3):273–281. (3 and 6)

KL15   David R. Karger and Matthew S. Levine.
Fast augmenting paths by random sampling from residual graphs.
SIAM J. Comput. 2015, 44(2):320–339. (1 and 3)

KLOS14 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford.
An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations.
In *Symposium on Discrete Algorithms (SODA)* 2014. (1)

NI92 Hiroshi Nagamochi and Toshihide Ibaraki.
A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph.
Algorithmica 1992, 7(5&6):583–596. (4 and 5)

Pen16 Richard Peng.
Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time.
In *Symposium on Discrete Algorithms (SODA)* 2016. (1)

She09 Jonah Sherman.
Breaking the multicommodity flow barrier for o(vlog n)-approximations to sparsest cut.
In *Foundations of Computer Science (FOCS)* 2009. (1)

She13 Jonah Sherman.
Nearly maximum flows in nearly linear time.
In *Foundations of Computer Science (FOCS)* 2013. (1)

She17 Jonah Sherman.
Area-convexity, l_∞ regularization, and undirected multicommodity flow.
In *Symposium on Theory of Computing (STOC)* 2017. (1)

SS11 Daniel A. Spielman and Nikhil Srivastava.
Graph sparsification by effective resistances.
SIAM J. Comput. 2011, 40(6):1913–1926. (4)

Ten10 Shang-Hua Teng.
The laplacian paradigm: Emerging algorithms for massive graphs.
In *Theory and Applications of Models of Computation (TAMC)* 2010. (1)

TvL84 Robert Endre Tarjan and Jan van Leeuwen.
Worst-case analysis of set union algorithms.
J. ACM 1984, 31(2):245–281. (6)

vdBCK$^+$24a Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva.
Almost-linear time algorithms for decremental graphs: Min-cost flow and more via duality.
In *Foundations of Computer Science (FOCS)* 2024. (1 and 2)

vdBCK$^+$24b Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford.
Incremental approximate maximum flow on undirected graphs in subpolynomial update time.
In *Symposium on Discrete Algorithms (SODA)* 2024. (2)

vdBLS23 Jan van den Brand, Yang P. Liu, and Aaron Sidford.
Dynamic maxflow via dynamic interior point methods.
In *Symposium on Theory of Computing (STOC)* 2023. (2)