Deterministic Edge Connectivity and Max Flow using Subquadratic Cut Queries

Aditya Anand [*]   Thatchaphol Saranurak[†]   Yunfan Wang [‡]

October 25, 2024

Abstract

We give the first deterministic algorithm that makes sub-quadratic queries to find the global min-cut of a simple graph in the cut query model. Given an n-vertex graph G, our algorithm makes $\widetilde{O}(n^{5/3})$ queries to compute the global min-cut in G. As a key ingredient, we also show an algorithm for finding s-t max-flows of size $\widetilde{O}(n)$ in $\widetilde{O}(n^{5/3})$ queries. We also show efficient cut-query implementations of versions of expander decomposition and isolating cuts, which may be of independent interest.

Contents

1 Introduction

Computing a global min-cut, also known as edge connectivity of a graph, is one of the most extensively studied problems in algorithmic graph theory [GH61, Gab91, NI92, HO94, Kar94, KS96, Kar00, KT18, HRW20, LP20, Li21, HLRW24]. In this problem, given an undirected graph $G = (V, E)$, we need to find a smallest set of edges $F \subset E$ such that $G \setminus F$ is not connected. The problem has also been explored in various computational models, including dynamic algorithms [Tho07, GHN+23, JST24], parallel algorithms [KM94, GG18, AB23], distributed algorithms [DHNS19, MN20, DEMN21, GZ22], and streaming algorithms [MN20, AD21].

One model which has attracted significant attention recently is the *cut-query* model. In this model, the edge set E is not known, instead we are allowed to make *cut queries*. A query is a set $S \subseteq V$, and an oracle returns $\text{CUT}(S)$, the number of edges across the cut $(S, V \setminus S)$.

A straightforward algorithm is to learn the entire graph in $\widetilde{O}(n^2)$ queries.[1] Rubinstein, Schramm, and Weinberg [RSW18] initiated the study of global min-cut in the cut-query model and showed a cut-query algorithm for simple graphs with only $\widetilde{O}(n)$ queries. Mukhopadhyay and Nanongkai [MN20] then matched this result for general weighted graphs. Recently, Apers et al. [AEG+22] improved this result to only $O(n)$ queries. However, all these algorithms are Monte Carlo randomized algorithms.

Hence, both [RSW18] and [AEG+22] posed the question if their algorithms can be made deterministic – the only known lower bound for deterministic cut queries is $\Omega(n)$ [HMT88, Har08]. However, the best known deterministic algorithm for edge connectivity (or global min-cut) [GK00] still takes $O(n^2/\log n)$ queries to *learn the whole graph*. This only slightly improves upon the trivial algorithm, which makes $\widetilde{O}(n^2)$ queries. Thus, there remains a big gap for deterministic algorithms, and the following conceptual question is still open:

> *Can we deterministically compute the edge connectivity of a graph*
> *without learning the whole graph?*

In this work, we make significant progress towards this problem by showing two results. We show the first deterministic cut-query algorithm for global min-cut in simple graphs with a sub-quadratic number of queries.

Theorem 1.1. *Given a simple undirected graph G, there is a deterministic algorithm that computes the global minimum cut in G using $\widetilde{O}(n^{5/3})$ cut queries.*

As a key ingredient, we show how to compute s-t max-flow using $\widetilde{O}(n^{5/3})$ queries.

Theorem 1.2. *Given a capacitated undirected graph G with integral edge weights bounded by W, and two vertices $s, t \in V$, there exists a deterministic algorithm that computes an s, t max-flow in $\widetilde{O}(n^{5/3}W)$ cut-queries.*

In particular, Theorem 1.2 implies that we can deterministically compute s-t max-flow in simple unweighted graphs in $\widetilde{O}(n^{5/3})$ cut queries. The previous algorithm for the same problem [RSW18] obtained this bound using a randomized Monte Carlo algorithm.

Towards obtaining these results we show efficient deterministic cut-query implementation of several tools: a version of expander decomposition, isolating cuts [LP20] and a deterministic algorithm that obtains a dominating set of size $\widetilde{O}(\frac{n}{\delta})$ in a graph with minimum degree δ. We believe

---

[1]Throughout the paper, we use $\widetilde{O}(\cdot)$ to hide a polylog(n) factor.

that the cut-query version of these simple and powerful tools, some of which have formed the basis for several fast algorithms in the last decade, may be of independent interest.

Other Query Models. Query models are widely studied in other settings as well. Many query models can be viewed as learning the properties of a matrix M through vector-matrix-vector $u^T M v$ queries. Different query models and the problems of concern within each model are introduced in [RWZ20].

[ACK21] considered a different setting; they studied the query complexity when we cannot use adaptive algorithms and can only perform a limited number of adaptive operations. [AAL21] gave a quantum minimum s-t cut algorithms using $\widetilde{O}(\sqrt{m}n^{\frac{2}{3}}W^{\frac{1}{3}})$ queries to the adjacency list of G, with integral edge weights bounded by W. [AL21] studied the query complexity of the connectivity problem for a graph G under the quantum query model.

	Connectivity		Edge Connectivity	
	Lower	Upper	Lower	Upper
Deterministic	$\Omega(n)$ [HMT88]	$O(\frac{n \log n}{\log \log n})$ [LC24]	$\Omega(n)$ [HMT88]	$\tilde{O}(n^{5/3})$ **(This paper)**
Zero-error, Randomized	$\Omega(\frac{n \log \log(n)}{\log n})$ [RS95]	$O(n)$ [AEG+22]	$\Omega(n)$	$\tilde{O}(n^{5/3})$ **(This paper)**
Bounded Error, Randomized	$\Omega(\frac{n}{\log n})$ [BFS86]	$O(n)$ [AEG+22]	$\Omega(\frac{n \log \log(n)}{\log n})$ [AD21]	$O(n)$ [AEG+22]
Quantum	$\Omega(1)$	$O(\log^5(n))$ [AL21]	$\Omega(1)$	$\tilde{O}(\sqrt{n})$ [AEG+22]

Table 1: The cut query complexity of connectivity and edge connectivity on simple graphs in various models. Before our work, the best deterministic and zero-error upper bound was from [GK00], they use $O(n^2/\log n)$ queries by learning the entire graph. Additionally, the lower bound for zero-error edge connectivity comes from an unpublished article by Troy Lee and Adi Shraibman: "On the communication complexity of edge connectivity". For more details, see [AEG+22].

2   Technical Overview

Maximum s-t flow: In this overview, we assume $W = 1$. That is, the input graph G has unit capacities, and the value of s-t maximum flow is at most $\widetilde{O}(n)$. Our algorithm is based on the classical Dinitz's algorithm [Din06], which we recall below.

The algorithm maintains a flow f and proceeds in iterations. In each iteration, we first construct the (directed) residual graph G_f for the flow f. Next, we obtain the layered graph G_L from G_f. The vertex set of G_L is V. Define layers $L_0, L_1 \ldots$, where L_i is the set of vertices at distance i from s in G_f. The edge set of the layered graph is the set of edges $\{u, v\}$ in $E(G_f)$, such that $u \in L_i$ and $v \in L_{i+1}$ for some i. A *blocking flow* f' is then a flow from s to t in G_L, such that on every flow path, some edge is saturated (that is, the flow through the edge equals its capacity). Let $\text{dist}(s, t)$

denote the length of the shortest (directed) path from s to t in G_L. Finally, we find a blocking flow f' and augment the flow f with f', and proceed to the next iteration. The crucial property of the algorithm is that, after each iteration, $\text{dist}(s,t)$ must strictly increase. Once we reach an iteration where a blocking flow of non-zero value does not exist, the obtained flow f is maximum. We will use the standard fact that once $\text{dist}(s,t) \geq n^{2/3}$, the total additional units of flow that can be sent in subsequent iterations is at most $O(n^{2/3})$.

To make only sub-quadratic cut queries, our key contribution is to show how to compute a blocking flow f' using only $\widetilde{O}(n + \text{val}(f')\text{dist}(s,t))$ queries where $\text{val}(f')$ denotes the value of f'. Our algorithm then keeps augmenting the flow via our blocking flow subroutine until the resulting flow is maximum.

To analyze the query complexity, we divide the algorithm into two phases. The first phase contains all iterations when $\text{dist}(s,t) \leq n^{2/3}$, and the second phase contains all subsequent iterations. In the first phase, we make $\widetilde{O}(n \cdot n^{2/3} + n \cdot n^{2/3}) = \widetilde{O}(n^{5/3})$ queries because (1) there are at most $n^{2/3}$ iterations since, after that, we must have $\text{dist}(s,t) > n^{2/3}$, (2) the total value of flow augmented during these iterations at most the maximum flow value, which we assumed to be at most $\widetilde{O}(n)$, and (3) $\text{dist}(s,t) \leq n^{2/3}$. In the second phase, we again make $\widetilde{O}(n \cdot n^{2/3} + n^{2/3} \cdot n) = \widetilde{O}(n^{5/3})$ queries because (1) there are at most $n^{2/3}$ iterations since, after that, f must already be maximum, (2) the total value of flow augmented during these iterations is at most $O(n^{2/3})$, and (3) $\text{dist}(s,t) \leq n$.

Global minimum cut: The starting point of our global min-cut algorithm is the algorithm of [LP20]. We give a high-level description of this algorithm below.

Given a set of terminals T, a Steiner min-cut is a minimum cut that separates some pair of terminals, i.e., a minimum size cut $(C, V \setminus C)$ that satisfies $C \cap T, T \setminus C \neq \emptyset$. Note that a Steiner min-cut when $T = V(G)$ is a global min-cut. Suppose $(C, V \setminus C)$ is the minimum Steiner Cut in the graph G. The algorithm of [LP20] is divided into two cases: one where $\min\{|T \cap C|, |T \setminus C|\} \leq \text{polylog}(n)$, and the other where $\min\{|T \cap C|, |T \setminus C|\} \geq \text{polylog}(n)$.

In the first case, they find the Steiner min-cut using the *isolating cuts* technique and finish. In the second case, they compute an expander decomposition and replace T by a smaller set $T' \subseteq T$ with $|T'| \leq \frac{|T|}{2}$, such that some minimum Steiner cut C for T still separates T'. Then they recursively run the same procedure with T'. Since they do not know which case actually happens, they run the isolating cut subroutine at every recursion, while continuing to sparsify T. There can be only $O(\log|T|)$ recursions since the size of T decreases by half after each recursion. Observe that if the initial terminal set $T = V(G)$, the algorithm must find a global mincut at some point.

Several challenges arise when implementing this algorithm in the cut-query model. First of all, one needs to compute isolating cuts and expander decomposition in this model. But, superficially, we can be hopeful because the common primitive for these techniques is a *s-t* max-flow algorithm, and we have a new efficient algorithm when the max-flow value is at most $\widetilde{O}(n)$.

Unfortunately, the flow instances when computing isolating cuts and expander decomposition may have max-flow value $\Omega(m)$, where m is the number of edges, even when the input graph is simple. For example, the flow instances for computing isolating cuts have the following structure: The super source s connects to many terminals v, and the capacity of edge (s, v) is the degree of the vertex v. This is similar for the super sink t. When $T = V(G)$, for example, the max-flow value of this flow instance can be as large as $\Omega(m)$.

To overcome the above challenge, our solution is that we choose the initial terminal set T to be, instead of the whole vertex set $V(G)$, a dominating set D of size $\widetilde{O}(n/\delta)$, where δ is the minimum

3

degree. The correctness of this is based on our new and simple structural lemma (Lemma 5.3): every cut $(C, V \setminus C)$ of size at most $\delta - 1$ must separate D, i.e., we must have $D \cap C, D \setminus C \neq \emptyset$. Therefore, assuming that global min-cut has size at most $\delta - 1$ (otherwise, it has size δ, which is trivial to compute), we conclude that the minimum Steiner Cut with respect to D is the global min-cut. To initialize our terminal set as D, we show how to compute D using $\widetilde{O}(n)$ queries.

Why does starting with D help? This is because we will set up all flow instances such that only vertices in D are connected to super source s or super sink t and the capacity of each edge incident to s and t is at most $\delta + 1$.[2]

So the max-flow value is only $\widetilde{O}(n/\delta) \cdot (\delta + 1) = \widetilde{O}(n)$, but we prove that this new flow instance can still be used for computing isolating cuts and expander decomposition (in Section 5.3 and appendix A respectively). Therefore, we can employ our max-flow algorithm with $\widetilde{O}(n^{5/3})$ queries to compute both isolating cuts and a version of expander decomposition in only $\widetilde{O}(n^{5/3})$ queries.

3 Preliminaries

In this section, we will formally define terms and notations which we shall adopt throughout the paper.

Graphs: Given an undirected simple graph $G = (V(G), E(G))$, we define δ_G to be the *minimum degree* of G and λ_G to be the *size of the minimum cut* in G. For a vertex set $S \subseteq V$, we let $G[S]$ denote the subgraph induced on the set of vertices S. For a set of vertices S, we denote by $\partial_G(S)$ the set of edges with exactly one endpoint in S. For a vertex u, we denote by $N_G(u)$ the set of vertices v with $\{u, v\} \in E(G)$. For a set of vertices U, we denote by $N_G(U)$ the set $\bigcup_{u \in U} N_G(u) \setminus U$, the set of neighbors of vertices in U outside U. We denote a cut in G using the notation $(S, V(G) \setminus S)$ for $S \subseteq V(G)$. For simplicity, we shall refer to $V(G) \setminus S$ as $\overline{S}$. We say that a vertex set $S \subseteq V(G)$ is a *dominating set* of G if, for any vertex $v \in V(G)$, either $v \in S$ or v has at least one neighbor in S.

Given a directed graph H, we use $\mathrm{dist}_H(s, t)$ to represent the distance (length of the shortest path) between two vertices $s, t \in V(H)$. We drop the subscripts when the graph is clear from the context.

Flows and residual graphs: For a flow f from s to t, we use $\mathrm{val}(f)$ to denote the *size of the flow f*. We describe the flow by flow values $f(u, v), f(v, u)$ for every edge $\{u, v\} \in E$ with $f(u, v) = -f(v, u)$. Given a graph G with non-negative edge capacities $c : E \to \mathbb{R}_{\geq 0}$ and a flow f, we define the residual graph as the directed capacitated graph G_f, whose vertex set is $V(G_f) = V$. For an edge $\{u, v\} \in E$ with flow $0 \leq f(u, v) \leq c(u, v)$, the *residual capacity* is defined as $c_f(u, v) = c(u, v) - f(u, v) \geq 0$. We also define $c_f(v, u) = c(v, u) + f(u, v)$. Then we add two edges in G_f: if $c_f(u, v) > 0$, we add an edge from u to v with capacity $c_f(u, v)$ and if $c_f(v, u) > 0$, we add an edge from v to u with capacity $c_f(v, u)$.

Types of queries: Next, we formally define cut queries and bipartite independent set queries.

---

[2] Strictly speaking, our max-flow algorithm takes $\widetilde{O}(n^{5/3})$ queries only if edges have unit capacity. But, we can simulate an edge from super source s to vertex a of capacity τ by replacing it with τ paths: $s \to v_1 \to a, \ldots, s \to v_\tau \to a$. We do similarly for the super sink t. The total number of newly added vertices is $\widetilde{O}(n/\delta) \cdot (\delta + 1) = \widetilde{O}(n)$.

Definition 3.1 (Cut Query). For an undirected capacitated graph G with capacities $c : E(G) \to \mathbb{R}^+ \cup \{0\}$, given a set $S \subseteq V(G)$, the cut query oracle for graph G returns the total capacity of cut-edges $\text{CUT}(S) := c(S, \overline{S}) = \sum_{a \in S, b \in \overline{S}} c(a, b)$.

Definition 3.2 (Bipartite Independent Set (BIS) Query). Given a (possibly directed) capacitated graph H with capacities $c : E(H) \to \mathbb{R}^+ \cup \{0\}$, sets $A, B \subseteq V(H)$ as input such that $A \cap B = \emptyset$, the bipartite independent set query oracle returns the boolean value which indicates if $c(A, B) = \sum_{a \in A, b \in B} c(a, b) > 0$.

BIS queries were introduced by [BHPNR+18] and have been extensively studied in the context of problems on undirected graphs. In an undirected graph H, a BIS query between A and B can be simulated with 3 cut queries since

$$c(A, B) = \frac{1}{2} \left(\text{CUT}(A) + \text{CUT}(B) - \text{CUT}(A \cup B) \right)$$

We will require BIS queries on the residual graph for our maximum flow algorithm. The following lemma demonstrates how to simulate BIS queries on the residual graph using cut queries.

Lemma 3.3. *Given an undirected capacitated graph G and an explicit flow f, we can simulate a BIS query on the residual graph G_f by using 3 cut queries in G.*

Proof. Given $A, B \subseteq V$, the BIS query on G_f will return whether there exists $a \in A$, $b \in B$ such that $c_f(a, b) > 0$. This is equivalent to

$$\sum_{a \in A} \sum_{b \in B} c_f(a, b) > 0$$

Notice that

$$\sum_{a \in A} \sum_{b \in B} c_f(a, b) = \sum_{a \in A} \sum_{b \in B} c(a, b) - f(a, b)$$

$$= \sum_{a \in A} \sum_{b \in B} c(a, b) - \sum_{a \in A} \sum_{b \in B} f(a, b)$$

$$= \frac{1}{2} [\text{CUT}(A) + \text{CUT}(B) - \text{CUT}(A \cup B)] - \sum_{a \in A} \sum_{b \in B} f(a, b)$$

We can use 3 cut queries to get $\text{CUT}(A)$, $\text{CUT}(B)$ and $\text{CUT}(A \cup B)$. Since f is explicit, we can obtain $\sum_{a \in A} \sum_{b \in B} f(a, b)$ in zero queries. Therefore, a BIS query in G_f can be simulated by 3 cut queries in G. □

4 Flow Algorithm

In this section, we prove our result on computing an s-t max flow of bounded size (Theorem 1.2). Section 4.1 describes how to obtain simple primitives of the graph using BIS queries. Section 4.2 recalls Dinitz's algorithm and its analysis. Section 4.3 describes how to implement Dinitz's algorithm via BIS queries.

4.1 Simple Primitives using BIS Queries

Lemma 4.1. *Given a graph G and a flow f in G, for two disjoint subsets $A, B \subseteq V(G)$, if $E_{G_f}(A, B) \neq \emptyset$, we can find a neighbor of A in B (a vertex $b \in B$ such that there exists a vertex $a \in A$ with $(a, b) \in E(G_f))$ in the residual graph G_f in $O(\log |B|)$ BIS queries on G_f.*

Proof. Do a binary search on B. Divide B into two parts B_1, B_2 with roughly equal size, i.e. $|B_1| \approx |B_2|$. We know that either $E_{G_f}(A, B_1) \neq \emptyset$ or $E_{G_f}(A, B_2) \neq \emptyset$ because $E_{G_f}(A, B) \neq \emptyset$. We check which one of these is true using one BIS query. If $E_{G_f}(A, B_1) \neq \emptyset$, then we recursively find a neighbor in B_1. Otherwise, we recurse on B_2. So, we use $\mathcal{O}(\log |B|)$ BIS queries on G_f. □

Corollary 4.2. *For a vertex $u \in V$ or a set of vertices $U \subseteq V$, we can learn $N_{G_f}(u)$ and $N_{G_f}(U)$ in $\widetilde{O}(|N_{G_f}(u)|)$ and $\widetilde{O}(|N_{G_f}(U)|)$ BIS queries on G_f, respectively.*

Proof. We first show this for a single vertex u. Let $B = V \setminus \{u\}$. By Lemma 4.1 we can find a neighbor of u in B in G_f using $O(\log n)$ BIS queries. Next, delete v by setting $B = B \setminus \{v\}$, and invoke Lemma 4.1 to find another neighbor of u. Inductively, we learn the set $N_{G_f}(u)$ using $O(|N_{G_f}(u)| \log n)$ BIS queries.

The argument for a set of vertices U is obtained by treating the set U as one super vertex. □

Corollary 4.3. *Given graph G and a flow f, we can find a BFS tree in G_f with root s in $\widetilde{O}(n)$ BIS queries on G_f.*

Proof. We build the tree T in layers. Let $V(T) = \{s\}$, and $E(T) = \emptyset$. Additionally, we will maintain the distance from s for each vertex $u \in V$, denoted by $d(u)$. We initialize $d(s) = 0$, and $d(u) = \infty$ for every other vertex.

In every iteration, we find a vertex in $V(T)$ with the smallest distance to s, say u. We find all neighbors u' of u in $V \setminus V(T)$ using Corollary 4.2. For each such neighbor u' of u, we update the distance as $d(u') = d(u) + 1$, and add u' to $V(T)$ and the edge (u, u') to $E(T)$. Let x be the number of neighbors of u (in G_f) in $V \setminus V(T)$. Then we spend $\widetilde{O}(x)$ queries for this step. We charge these queries to the x vertices which are added to $V(T)$ in this step. Repeating the same process for at most $n - 1$ iterations, we correctly obtain the BFS tree T.

Note that each vertex $v \in V(T)$ receives $O(1)$ charge, and the number of vertices in T is n. Therefore, the total query complexity is $\widetilde{O}(n)$. □

4.2 Reminder of Dinitz's Algorithm

Here we recall Dinitz's blocking flow algorithm [Din06] and its analysis.

Definition 4.4 (Layered Graph). Let G be an undirected graph and let s and t be source and sink vertices in $V(G)$. Further, suppose that f is a s-t flow in G, and let G_f be the residual graph of G with respect to f. The layered graph G_L of G with respect to f is the decomposition of the vertices of G into layers defined by their distance from the source in G_f. The i-th layer is defined as $L_i = \{v \mid \text{dist}_{G_f}(s, v) = i\}$. The vertex set of the layered graph is $V(G_L) = V(G)$, and the edge set is the set of edges (u, v) of G_f such that $u \in L_i$ and $v \in L_{i+1}$ for some layer i.

Note that we always have $L_0 = \{s\}$. For simplicity, we will assume that the last layer contains only the sink vertex t - if there are other vertices in this layer, we will drop them.

Definition 4.5 (Blocking Flow). Given a graph G, the source and sink vertices vertex s, t, and an s-t flow f in G, let G_L be the layered graph G_L of G with respect to f. An s-t flow f' in G_L is called a blocking flow, if for every s-t path in G_L, some edge is saturated by f'

Algorithm 1 Dinitz's blocking flow algorithm[Din06]

Input: Graph G, two vertices s and t
Output: An s-t maximum flow f in G
 1: Initialize an empty flow f from s to t.
 2: Construct the residual graph G_f from G and f
 3: Construct the layered graph G_L from G_f and the source vertex s.
 4: **if** $\text{dist}_{G_L}(s, t) > n$ **then**
 5:     Terminate and output the flow f.
 6: **else**
 7:     Find a blocking flow f' from s to t in G_L
 8:     Augment f by f' to obtain a new flow f''. Update $f \leftarrow f''$, and repeat step 2.
 9: **end if**

The next lemma gives some simple facts about maximum flows which will be useful for the subsequent analysis.

Fact 4.6. *We have the following:*

1. *A flow f is maximum if and only if there is no augmenting path in the residual graph G_f.*

2. *If there is an s-t cut of capacity B in G_f, then*

   the maximum flow from s to t in G is at most $\text{val}(f) + B$.

Next, we need the following property of Dinitz's algorithm (Algorithm 1).

Lemma 4.7 ([Din06]). *After augmenting the flow f with a blocking flow f' to obtain a new flow f'', the distance from s to t in the residual graph increases by at least 1. Concretely, $\text{dist}_{G_{f''}}(s, t) \geq \text{dist}_{G_f}(s, t) + 1$.*

Thus Dinitz's algorithm must terminate after n iterations of blocking flow and give us a maximum flow, as after these many iterations, there cannot be an s-t path of length n. However, this is too expensive for us. Instead, we will use a standard optimization that actually shows that the algorithm terminates in $O(n^{2/3}W)$ iterations. We provide the proof here for completeness.

Lemma 4.8. *Given a capacitated graph G where each edge weight is an integer in the range $[1, W]$ and two vertices s, t, Algorithm 1 finds an s-t maximum flow after $O(n^{2/3}W)$ iterations of blocking flow.*

Proof. For the purpose of analysis, consider the s-t flow f_i obtained after i iterations, where i is the smallest subscript such that $\text{dist}_{G_{f_i}}(s, t) > d := n^{2/3}$. By Lemma 4.7, since the distance between s and t in the residual graph after i iterations exceeds d, it follows that $i \leq d$. This bounds the number of iterations required.

Since the graph contains n vertices and there are at least d layers in the layered graph G_L, by the pigeonhole principle, there must exist a pair of consecutive layers L_j and L_{j+1} such that $|L_j| +$

7

$|L_{j+1}| \leq O(\frac{n}{d})$. Therefore, the total edge weight between these layers is bounded by $O((\frac{n}{d})^2 W) = O(n^{2/3}W)$. By the definition of the layered graph, the set of edges $E(L_j, L_{j+1})$ forms a cut between s and t in G_{f_i}. Thus, by Fact 4.6, the total flow from s to t is at most $\mathrm{val}(f_i) + O(n^{2/3}W)$.

Consequently, the remaining flow size in the residual graph G_{f_i} is at most $O(n^{2/3}W)$. Since each iteration of blocking flow increases the flow value by at least 1, it follows that the algorithm can perform at most $O(n^{2/3}W)$ additional iterations before termination. Therefore, the algorithm terminates after at most $O(n^{2/3}W)$ iterations in total. □

4.3 Implementing Dinitz's Algorithm via BIS Queries

Now we implement Dinitz's blocking flow algorithm [Din06] in the cut query model.

Claim 4.9. *Given a flow f such that $\mathrm{dist}_{G_f}(s,t) = d'$, we can compute a blocking flow f', augment the flow f with f', and recompute the residual graph using $\widetilde{O}(n + d' \cdot \mathrm{val}(f'))$ BIS queries on G_f.*

Proof. We first construct the layered graph G_L of G_f by running BFS in $\widetilde{O}(n)$ queries using Corollary 4.3. This gives us $L_0 = \{s\}, ..., L_{d'} = \{t\}$. The goal is to find an s-t path in G_L efficiently using a stack-based method.

Next, we maintain a stack to track our search for an s-t path. Initially, the stack contains $\{s\}$. At each step, let the top element of the stack be u, where $u \in L_i$. We then attempt to find $v \in L_{i+1}$ such that $c_f(u, v) > 0$. If such v exists, we can find it using $\widetilde{O}(1)$ queries by Lemma 4.1, and we push v onto the stack. If no such v exists, we pop u from the stack and remove u from the layered graph, as u cannot send any flow to the next layer.

When the stack reaches a length of $d' + 1$, this indicates that we have found an s-t path. We add this flow path to f', adjust the residual capacities along the path, and then reset the stack to $\{s\}$ to search for another path. The process terminates when the stack is empty, meaning there is no further flow from s to any vertex in the next layer.

Now we analyze the query complexity. In each step, we either push a new vertex onto the stack or pop one off and remove it from the graph. Each push or pop requires $\widetilde{O}(1)$ queries. Whenever the stack reaches length $d'+1$, we send flow from s to t, and the stack is reset, reducing its length by d'. The total length of the stack can increase by at most $n + O(d') \cdot \mathrm{val}(f')$, as this is the maximum number of vertices and flow pushes in the layered graph. Therefore, the total number of queries is at most $\widetilde{O}(n + d' \cdot \mathrm{val}(f'))$. □

We are now ready to prove the main result, Theorem 1.2.

Proof of Theorem 1.2. By Claim 4.9, a single blocking flow iteration requires $\widetilde{O}(n + d' \cdot \mathrm{val}(f'))$ queries, where $\mathrm{val}(f')$ represents the current flow value, and d' is the distance from s to t in the layered graph. Therefore, to analyze the algorithm's query complexity, we should consider three factors: the total number of iterations, the flow value, and the distance. Balancing these factors is essential for optimizing the overall query complexity.

By Lemma 4.8, we know that the number of iterations is at most $O(n^{2/3}W)$. When the distance is less than $n^2/3$, the flow value is at most $O(nW)$. Conversely, when the distance exceeds $n^{2/3}$, the remaining flow value is at most $O(n^{2/3}W)$.

Thus, the total query complexity can be expressed as:

$$\sum \widetilde{O}(n + d' \cdot \mathrm{val}(f')) \leq O(n^{\frac{2}{3}}W) \cdot \widetilde{O}(n) + \sum \widetilde{O}(d' \cdot \mathrm{val}(f'))$$

8

Breaking it down further:

$$= \widetilde{O}(n^{\frac{5}{3}}W) + \sum_{d'<n^{\frac{2}{3}}} \widetilde{O}(d' \cdot \mathrm{val}(f')) + \sum_{d'\geq n^{\frac{2}{3}}} \widetilde{O}(d' \cdot \mathrm{val}(f'))$$

For the two cases:

1. For $d' < n^{\frac{2}{3}}$, we have

$$\sum_{d'<n^{\frac{2}{3}}} \widetilde{O}(d' \cdot \mathrm{val}(f')) \leq n^{\frac{2}{3}} \cdot O(nW) = \widetilde{O}(n^{\frac{5}{3}}W)$$

2. For $d' \geq n^{\frac{2}{3}}$, we have

$$\sum_{d'\geq n^{\frac{2}{3}}} \widetilde{O}(d' \cdot \mathrm{val}(f')) \leq n \cdot O(n^{\frac{2}{3}}W) = \widetilde{O}(n^{\frac{5}{3}}W)$$

It now follows that the total query complexity is at most $\widetilde{O}(n^{5/3}W)$, which proves the result. □

5   Global Min-cut Algorithm

In this section, we present our algorithm for finding global min-cut, proving Theorem 1.1. We will focus on obtaining the following threshold version of the result.

Theorem 5.1. *Given a graph G and a parameter $\tau \leq \delta - 1$, there is an algorithm that either*

- *returns a cut $(C, V \setminus C)$ with size $|\partial C|$ at most τ, or*
- *certifies that the global min-cut in G must have a size larger than τ*

in $\widetilde{O}(n^{5/3})$ cut queries.

Note that Theorem 1.1 now follows immediately by binary search. If the global min-cut has size δ, we simply return the cut corresponding to of the minimum-degree vertex. We fix the parameter τ throughout this section.

The rest of this section is dedicated to proving Theorem 5.1. We first give a crucial yet simple observation that any dominating set must be separated by every cut of size at most $\delta - 1$ in Section 5.1, and show how to efficiently compute the dominating set in Section 5.2. Then, we show how to efficiently compute minimum isolating cuts of size at most τ in the cut query model in Section 5.3.

Given these three subsections, we are ready to implement the global minimum cut algorithm of [LP20] in the cut query model. In high level, the algorithm starts by computing a dominating set R and has two cases. If the minimum cut is unbalanced with respect to R, then we will compute minimum isolating cuts with respect to subsets of R (Section 5.4). Otherwise, if the minimum cut is balanced with respect to R, then we will compute expander decomposition with respect to R in the cut query model (Appendix A) and use the decomposition to "sparsify" R to be a smaller subset $R' \subseteq R$ that is still separated by minimum cuts (Section 5.5).

5.1 Dominating Sets are Separated by Small Cuts

Below, we show a simple and crucial observation: any dominating set must be separated by a cut of size at most $\delta - 1$.

Definition 5.2. Given a graph G, a vertex $R \subseteq V$ is c-separated if for every cut $(C, V \setminus C)$ of size $|\partial C| \leq c$, we have $C \cap R, R \setminus C \neq \emptyset$. That is, R hits both sides of the cut C.

Lemma 5.3. *For any dominating set $R \subseteq V$, R is $\delta - 1$-separated, where δ is the minimum degree.*

Proof. Suppose for contradiction that this is not the case. Then there is a cut $(C_1, C_2 = V \setminus C_1)$ of size at most $\delta - 1$ where $R \subseteq C_1$. Now we have $\delta - 1 \geq |E(C_1, C_2)| \geq |E(R, C_2)| \geq |C_2|$ where the last inequality is because R is dominating. But every vertex in C_2 has degree at least δ, therefore we must have $|E(C_1, C_2)| \geq |C_2|(\delta - (|C_2| - 1)) \geq \delta$ for $1 \leq |C_2| \leq \delta - 1$, which leads to a contradiction. $\square$

We remark that the above lemma is the crucial place where we exploit that the graph is simple.

5.2 Computing a Dominating Set

In this section, we show how to compute a dominating set efficiently in the cut-query model.

Theorem 5.4. *There is a deterministic algorithm that finds a dominating set $R \subseteq V$ of size $|R| \leq \widetilde{O}(\frac{n}{\delta})$ using $\widetilde{O}(n)$ cut-queries.*

Proof. We will first state our algorithm, then prove its correctness, and finally analyze its complexity under the cut query model.

Algorithm We construct R iteratively. Initially, we set $G' = G$, $R = \emptyset$. We first apply the following reduction rule – as long as there is a vertex in $v \in G'$ whose degree in G' is larger than $\frac{\delta}{2}$, we add it into R and delete $v \cup N_{G'}(v)$ from G'.

After exhaustively applying the above reduction rule, if $V(G') \neq \emptyset$ and there is no vertex in G' with a degree larger than $\frac{\delta}{2}$, then we do the following. Consider the bipartite graph H whose vertex set is the bipartition $W_1 \cup W_2$, where $W_1 = N_G(R)$ and $W_2 = V(G')$. The edge set of H is the edge set of G restricted to the edges between W_1 and W_2.

First, observe that $\forall u \in W_2$, $|E(W_1, u)| = \deg(u) - |E(u, W_2 \setminus \{u\})| \geq \frac{\delta}{2}$, otherwise, this vertex would have been considered in the previous step. Thus, we have $|E(W_1, W_2)| = \sum_{u \in W_2} |E(W_1, u)| \geq \sum_{u \in W_2} \frac{\delta}{2} \geq |W_2| \frac{\delta}{2}$. Then there exists a $v \in W_1$ such that $|E(v, W_2)| \geq \frac{|E(W_1, W_2))|}{|W_1|} \geq \frac{|W_2|\delta}{2|W_1|} \geq \frac{|W_2|\delta}{2n}$. We add v into R and delete all its neighbors in G'. We repeat until $V(G') = \emptyset$, hence R must be a dominating set.

Correctness It remains to bound the size of R. When there is a vertex in G' with degree larger than $\frac{\delta}{2}$, we reduce $|V(G')|$ by at least $\frac{\delta}{2}$ each time. So we can add at most $\frac{2n}{\delta}$ such vertices into R.

When no vertex in G has degree larger than $\frac{\delta}{2}$, we remove at least $|V(G')| \frac{\delta}{2n}$ vertices from G'. Thus the size of G' reduces by at least a $1 - \frac{\delta}{2n}$ fraction. Thus after $O(\frac{n}{\delta})$ iterations, the size of G' reduces by a constant factor, and hence this step can happen at most $O(\frac{n}{\delta} \log n) = \widetilde{O}(\frac{n}{\delta})$ many times. In total, we have $|R| \leq \frac{2n}{\delta} + O(\frac{n}{\delta} \log(\frac{n}{\delta})) = \widetilde{O}(\frac{n}{\delta})$.

Complexity First, we want to find a vertex v whose degree in G' is at least $\frac{\delta}{2}$. Given a vertex w, we can find the degree of w in G' in $\mathcal{O}(1)$ queries. We enumerate each vertex $w \in V(G')$, and find its degree $\deg(w)$. If $\deg(w) < \frac{\delta}{2}$, then we mark w as irrelevant and continue. If $\deg(w) > \frac{\delta}{2}$, then we find all its neighbors in G' using $\widetilde{O}(\deg(w))$ queries using Lemma 4.1. Note that we delete all these neighbors of w from G'. Let us charge the query cost uniformly to each vertex in $\deg(w)$, and note that each vertex receives a charge of $\widetilde{O}(1)$. Observe that the degree of a vertex in G' is non-increasing, thus irrelevant vertices remain irrelevant.

Now it remains to analyze the case when no vertex of G' has degree $> \frac{\delta}{2}$ in G'. Recall that we want to find a vertex $v \in W_1$ such that $|E(v, W_2)| \geq \frac{|E(W_1, W_2))|}{|W_1|}$. We will use binary search. Divide W_1 into two (roughly) equal sized groups W_{11} and W_{12}. By Lemma 4.1, using $\widetilde{O}(1)$ queries, we can find both $E(W_{11}, W_2)$ and $E(W_{12}, W_2)$. By simple averaging, either $E(W_{11}, W_2) \geq \frac{|E(W_1, W_2))|}{|W_1|}|W_{11}|$ or $E(W_{12}, W_2) \geq \frac{|E(W_1, W_2))|}{|W_1|}|W_{12}|$. Without loss of generality, assume the former. We now recurse on W_{11}, and repeat the same process, till we reach a singleton vertex $v \in W_1$, which must satisfy $|E(v, W_2)| \geq \frac{|E(W_1, W_2))|}{|W_1|}$.

The total number of cut queries is at most $\widetilde{O}(1)$, since in each iteration we use $\widetilde{O}(1)$ queries and there are at most $\widetilde{O}(1)$ iterations of the binary search. Therefore, the total query complexity of finding the dominating set is dominated by the query complexity of the reduction rule, and hence is at most $\widetilde{O}(n)$. □

5.3   Computing Minimum Isolating Cuts

This section shows an efficient algorithm for computing a minimum isolating cut. Let us first recall its definition below.

Definition 5.5 (Minimum Isolating Cut). For any set of vertices $R \subseteq V$ and $r \in R$, the minimum isolating cut of r is an $\{r\}$-$(R \setminus \{r\})$ min-cut. The minimum isolating cut (of R) is the minimum sized cut among the minimum isolating cuts over all $r \in R$.

Theorem 5.6. *Given an unweighted graph G and a set of vertices $R \subseteq V$ such that $|R|$ is at most $\widetilde{O}(\frac{n}{\tau})$, there is an algorithm that either*

- *outputs a minimum isolating cut of R of size at most τ, or*

- *certifies that the minimum isolating cut of R has a size larger than τ*

in $\widetilde{O}(n^{5/3})$ cut queries.

We remark that the algorithm can be modified easily to output an isolating cut for each $r \in R$ whose min-isolating cut has size at most τ.

Proof. First, we will state the algorithm, which follows the approach of [LP20], adapted to our setting. Then we will discuss its correctness, and finally argue about implementation and the query cost. We will use the concept of *closest min-cuts*. For two sets S and T, we say that an S-T min-cut $(X, V \setminus X)$ is closest to S if for any other S-T min-cut $(X', V \setminus X')$, we have $X \subseteq X'$.

Algorithm: Encode each vertex in R using a unique $\mathcal{O}(\log|R|)$ dimensional bit vector. Next, compute $O(\log|R|)$ different bi-partitions of the vertices in R, where each bi-partition is obtained by selecting a coordinate c of the bit vector and grouping the vertices based on whether they have 0 or 1 in the cth coordinate. Let $\mathcal{C}$ be the set of these bi-partitions. Note that each pair of distinct vertices $u, v \in R$ will be separated by at least one bi-partition in $\mathcal{C}$.

For each of these $O(\log|R|)$ bi-partitions $(A, B) \in \mathcal{C}$ of R, $A \cap B = \emptyset$, set up the following flow problem. Add two vertices $s_{\text{source}}, s_{\text{sink}}$ to the vertex set of G. For each vertex $a \in A$, add an edge (s_{source}, a) with capacity $\tau + 1$. Similarly, for each $b \in B$, add an edge (b, s_{sink}) with capacity $\tau + 1$. Every edge of G has a capacity of 1. Call the modified graph H. Find an s_{source}-s_{sink} max-flow in H.

Let $f_{A,B}$ be this max-flow and $(C_A, C_B = V(G) \setminus C_A)$ be the restriction of the minimum cut corresponding to this maximum flow to G (we simply remove s and t from the cut to obtain C_A, C_B). Consider the graph G' after deleting every such cut, that is, we delete the edges ∂C_A for every bi-partition $(A, B) \in \mathcal{C}$. For each $r \in R$, define $T_r \subseteq V(G)$ to be the set of vertices reachable from r in G'. Further, let $R' \subseteq R$ be the set of $r \in R$ which satisfy the following property – for every bi-partition $(A, B) \in \mathcal{C}$, if $r \in A$, then $r \in C_A$, and if $r \in B$, then $r \in C_B$. In other words, R' is the subset of R whose edges to s_{source} or s_{sink} are not saturated in any max-flow $f_{A,B}$ across all $(A, B) \in \mathcal{C}$ (see Claim 5.7).

Let $T_r' = \{r\} \cup (T_r \setminus R)$ for each $r \in R'$, we obtain a new capacitated graph G_r from G as follows. Contract all vertices of $V \setminus T_r'$ into vertex s_r while keeping parallel edges. Then, we compute the min-cut between r and s_r in G_r. We check if it is an isolating cut for r. Let λ_r be the size of this cut (and let $\lambda_r = \infty$ if no such cut exists). Then we check if there exists an $r \in R'$ with $\lambda_r \leq \tau$. If so, we output the corresponding cut. Otherwise, we declare that the minimum isolating cut for R has a size larger than τ. This concludes the description of the algorithm.

Correctness: We now show the correctness of our algorithm. Note that our algorithm is similar to [LP20], but we need to argue its correctness a bit more carefully since whenever we compute an A,B min-cut for a bi-partition, we set the capacities of the edges incident to the source and sink vertices as $\tau + 1$ (instead of ∞ as in their setting). On a high level, this still works since we are only interested in the isolating cuts of size at most τ; if there is no isolating cut of size $\leq \tau$, we will simply output that this is the case. The next few lemmas formally show that our algorithm is indeed correct.

Claim 5.7. *For any bipartition $(A, B) \in \mathcal{C}$ and any $a \in A$, we say that a is saturated if in the maximum flow $f_{A,B}$, the edge (s_{source}, a) is saturated. Then if a is not saturated by $f_{A,B}$, then $a \in C_A$. The same holds analogously for any $b \in B$.*

Proof. If $a \notin C_A$, then the edge (s_{source}, a) is cut by the $(s_{\text{source}}, s_{\text{sink}})$ min-cut. But since a min-cut is saturated in any max-flow, this means that a must be saturated, which is a contradiction. $\square$

Claim 5.8. *For each bi-partition (A, B), (C_A, C_B) is a minimum cut in G that separates $C_A \cap A$ and $C_B \cap B$.*

Proof. Since (C_A, C_B) is a restriction of the s_{source}-s_{sink} min-cut, it follows that there exists a feasible flow in G from vertices of $C_A \cap A$ to vertices of $C_B \cap B$ saturating every edge of ∂C_A. This flow certifies that (C_A, C_B) must be a minimum cut between $C_A \cap A$ and $C_B \cap B$ in G. $\square$

Claim 5.9. *Consider some terminal $r \in R$ and let $(C_r, V \setminus C_r)$ be the min-cut between $\{r\}$ and $R \setminus \{r\}$ where $r \in C_r$, which is closest to r (that is, the closest isolating cut for r). If $|E(C_r, V \setminus C_r)| \leq \tau$, then the following statements must hold for every bipartition $(A, B) \in \mathcal{C}$.*

- *r cannot be saturated in the flow $f_{A,B}$.*

- *If $r \in A$, $C_r \subseteq C_A$. Likewise, if $r \in B$, we must have $C_r \subseteq C_B$.*

Proof. Consider a bi-partition $(A, B) \in \mathcal{C}$. Let us assume without loss of generality that $r \in A$, the other case is symmetric. Since $(C_r, V \setminus C_r)$ is a cut of size at most τ between r and $R \setminus \{r\}$, r cannot be saturated. Therefore, by Claim 5.7, we have $r \in C_A$.

Now, we will show that $C_r \subseteq C_A$. Suppose, for the sake of contradiction, that this is not the case. Then by submodularity of cuts, we have:

$$\partial(C_r) + \partial(C_A) \geq \partial(C_r \cap C_A) + \partial(C_r \cup C_A).$$

Since C_r is the minimum cut separating r and $R \setminus \{r\}$, it follows that $\partial(C_r) \leq \partial(C_r \cap C_A)$. Similarly, since $(C_A, C_B = V \setminus C_A)$ is the minimum cut separating $C_A \cap A$ and $C_B \cap B$ (see Claim 5.8), we must have $\partial(C_A) \leq \partial(C_r \cup C_A)$. Combining these results, we obtain the equalities:

$$\partial(C_r) = \partial(C_r \cap C_A) \text{ and } \partial(C_A) = \partial(C_r \cup C_A).$$

This implies that $C_r \cap C(A)$, which is a subset of C_r, is also a minimum cut separating r from $R \setminus \{r\}$. This contradicts the fact that C_r is the minimum cut between r and $R \setminus \{r\}$ that is closest to r. $\square$

The following lemma shows that across all vertices $r \in R'$, the sets T_r are disjoint.

Lemma 5.10. *For every distinct vertices $r, r' \in R'$, we have $T_r \cap T_{r'} = \emptyset$, $r \in T_r$, and $r' \in T_{r'}$.*

Proof. Recall that each pair of distinct vertices in R' is separated by at least one of the bi-partitions defined by bit vectors. Without loss of generality, we can assume that there exists a bi-partition $(A, B) \in \mathcal{C}$ such that $r \in A$ and $r' \in B$. Since r and r' are not saturated, by Claim 5.7, we have $r \in C_A$ and $r' \in C_B$. But then the (C_A, C_B) min-cut separates r from r', and hence $T_r \subseteq C_A$ ad $T_{r'} \subseteq C_B$. It follows that $T_r \cap T_{r'} = \emptyset$. $\square$

We are now ready to prove the correctness of our algorithm. Suppose there exists a terminal $r^* \in R$ whose minimum isolating cut $(C_{r^*}, V \setminus C_{r^*})$ satisfies $E(C_{r^*}, V \setminus C_{r^*}) \leq \tau$. Using Claim 5.9, it follows that we must have $C_{r^*} \subseteq T_{r^*}$ (Recall that T_{r^*} is defined as the set of vertices reachable from r^* in the graph obtained after deleting the minimum cuts for every bi-partition in $\mathcal{C}$). Furthermore, by the definition of isolating cut, we know that $C_{r^*} \cap R = \{r^*\}$, therefore $C_{r^*} \subseteq T'_{r^*}$ Hence, the minimum r^*-s_{r^*} cut in G_{r^*} is of size at most τ. Also by the definition of R', we must have $r^* \in R'$. Since our algorithm finds an r-s_r min-cut for each $r \in R'$, and we have $r^* \in R'$, the output cut must be a minimum isolating cut for some $r \in R$ of size at most τ.

Query complexity: For each bi-partition $(A, B) \in \mathcal{C}$, we construct the auxiliary graph H and compute an s_{source}-s_{sink} maximum flow. For each edge (s_{source}, a), $a \in A$ with capacity $\tau + 1$, we replace it with $\tau + 1$ parallel edges each with capacity 1. We then sub-divide each of these edges by adding an additional vertex. For each edge (b, s_{sink}) where $b \in B$, we do the same. Then the resulting graph is simple, has at most $O(n + |R|\tau) = O(n)$ nodes, and every edge has unit capacity. By Theorem 1.2, we can compute an s_{source}-s_{sink} max-flow in this graph using $\widetilde{O}(n^{5/3})$ queries.

The (restriction to G of the) min-cut (C_A, C_B) can be obtained as follows. Once we obtain a maximum flow $f_{A,B}$ we let C_A be the set of vertices reachable from s in the residual graph $G_{f_{A,B}}$, and let $C_B = V(G) \setminus C_A$. We can obtain C_A by applying Corollary 4.3 to find a BFS tree with root node s_{source} in $G_{f_{A,B}}$ with $\widetilde{O}(n)$ queries. Once we know (C_A, C_B) for each bi-partition $(A, B) \in \mathcal{C}$, we can find the sets T_r for each $r \in R$. We can then identify the set R'. For each $r \in R'$ we construct the graph G_r, and find an r-s_r closest min-cut in G_r. To find this min-cut, we obtain a max-flow and find the set of vertices reachable from r. The next lemma shows that we can do this without using too many queries.

Lemma 5.11. *We can compute r-s_r max flow (min cut) in G_r using $\widetilde{O}(|T_r'|^{5/3})$ queries.*

Proof. All flow goes from r to s_r can be divided into two cases

1. Directly goes from r to s_r through edge (r, s_r)

2. First go through an edge (r, u), where $u \in T_r' \setminus \{r\}$, and then goes from u to s_r

It's relatively easy to compute the flow in the first case by calculating the weight of (r, s_r), which is $|E(r, V \setminus T_r)|$ and can be determined with $O(1)$ queries. Notice that in the residual graph, all flows are the second case. The flow size of the second case is at most $|E(r, T_r' \setminus \{r\})| \leq |T_r'| - 1$, since each flow must use an edge between r to $C_r \setminus \{r\}$. Then by Theorem 1.2, we can compute it in $\widetilde{O}(|T_r' + 1|^{5/3}) = \widetilde{O}(|T_r|^{5/3})$ queries. □

By Lemmas 5.10 and 5.11, the total query complexity for finding r-s_r min-cuts for every $r \in R'$ is bounded by $\sum_{r \in R'} \widetilde{O}(|T_r|^{5/3}) = \widetilde{O}(n^{5/3})$. The final equality holds since the sets T_r, $r \in R'$, are disjoint from each other, ensuring $\sum_r |T_r| \leq n$. This concludes the proof of Theorem 5.6. □

5.4 Unbalanced Case

In this section, we show how to find a cut of size at most τ when this cut is *unbalanced* with respect to a given terminal set $R \subseteq V$ that is τ-separated.

Definition 5.12 (Unbalanced/Balanced Cut). For any set of vertices $R \subseteq V$, a cut $C = (C_1, C_2 = V \setminus C_1)$ with size at most τ, and a parameter $\phi \geq \text{poly}(\frac{1}{\log n})$, we say that C is ϕ-unbalanced for R if $\min\{|C_1 \cap R|, |C_2 \cap R|\} \leq (\frac{1}{\phi})^3 + \frac{1}{\phi}$, otherwise we say that C is ϕ-balanced for R.

The goal of this subsection is to prove the following theorem.

Lemma 5.13. *Let $R \subseteq V$ be τ-separated and $|R| \leq \widetilde{O}(\frac{n}{\tau})$. If there is a cut $(C, V \setminus C)$ of size at most τ that is ϕ-unbalanced for R for some $\phi \geq \text{poly}(\frac{1}{\log n})$, then we can find a cut with size at most τ in $\widetilde{O}(n^5/3)$ queries.*

We need the following tool called *splitter* for essentially deterministic subsampling vertices.

14

Lemma 5.14 (Theorem 4.3 from [LP20]). *For every positive integer n and $k < n$, there is a deterministic algorithm that constructs a family $\mathcal{F}$ of subsets of $[n]$ such that, for each subset $S \subseteq [n]$ of size at most k, there exists a set $S' \in \mathcal{F}$ with $|S \cap S'| = 1$. The family $\mathcal{F}$ has size $k^{O(1)} \log n$ and contains only sets of size at least 2.*

Proof of Lemma 5.13. Let $\mathcal{F}$ be the family of subsets of R from Lemma 5.14 such that, for any unbalanced cut C, there is a $R^* \in \mathcal{F}$ with $|R^* \cap C| = 1$. Note that $|\mathcal{F}| = [(\frac{1}{\phi})^3 + \frac{1}{\phi}]^{O(1)} \log |R| = \widetilde{O}(1)$.

Then we run the algorithm of Theorem 5.6 for each set in $\mathcal{F}$. Since $R^* \in \mathcal{F}$, the algorithm will successfully conclude that the isolating cut for R^* is of size at most τ. The total number of queries is $|\mathcal{F}| \cdot \widetilde{O}(n^{5/3}) = \widetilde{O}(n^{5/3})$. $\qquad\square$

5.5 Balanced Case

We say that R is ϕ-*balanced* if every cut $(C, V \setminus C)$ of size $|\partial C| \leq \tau$ is ϕ-balanced for R. In the previous section, if R is τ-separated, but not ϕ-balanced, then Lemma 5.13 will find a cut of size at most τ for us. In this section, we handle the case when R is τ-separated and ϕ-balanced using the following lemma.

Lemma 5.15. *Suppose that R is τ-separated and ϕ-balanced. Then, we can make $\widetilde{O}(n^{5/3})$ cut queries and either*

- *find a set $\widetilde{R}$ such that $|\widetilde{R}| \leq O(\phi|R| \log^6 n) + \frac{|R|}{\log n}$ and $\widetilde{R}$ is τ-separated, or*

- *find a cut in G with cut-size at most τ.*

Towards proving this key lemma, we begin by introducing the notion of expanders and expander decomposition that we need.

Definition 5.16 ((ϕ, R)-expander). Given a graph G and a terminal set $R \subseteq V$, G is a (ϕ, R)-expander if for every cut $(S, V \setminus S)$, we have

$$\Phi(S) = \frac{|\partial S|}{\min\{|R \cap S|, |R \setminus S|\}} \geq \phi(\tau + 1)$$

We work with graphs with a slightly weaker notion of expansion, which we call (ϕ, R)-almost expanders.

Definition 5.17. Given a graph $G = (V, E)$ and a terminal set $R \subseteq V$, G is a (ϕ, R) almost-expander with core R' if $R' \subseteq R$, $|R'| \geq |R|(1 - \frac{1}{\log n})$, and for any cut $(S, V \setminus S)$ in G, we have

$$\Phi(S) = \frac{|\partial S|}{\min\{|R' \cap S|, |R' \setminus S|\}} \geq \phi(\tau + 1)$$

Our next and crucial step in this section is to obtain a decomposition into (ϕ, R)-almost expanders. We defer this proof to Appendix A since this proof is mostly standard.

At a high level, the algorithm combines an implementation of the cut-matching game [KKOV07] together with the expander pruning technique [SW19]. The cut player trivially needs zero queries since the auxiliary graph will be explicitly maintained. The matching player applies our algorithm for s-t maximum flow. This takes $\widetilde{O}(n^{5/3})$ queries. Similar to the flow instance in Theorem 5.6, we send flow between terminal set R of size $\widetilde{O}(\frac{n}{\tau})$ and each terminal sends at most $\tau + 1$ units of flow. So the maximum flow is at most $\widetilde{O}(n)$ and, hence, our flow algorithm from Theorem 1.2 uses $\widetilde{O}(n^{5/3})$ queries. The formal statement is summarized below:

Lemma 5.18. *Given a graph $G = (V, E)$ and a terminal set $R \subseteq V$ with $|R| \leq \widetilde{O}(\frac{n}{\tau})$ and a parameter $\phi = \text{poly}(\frac{1}{\log n})$, one can find a partition of the vertex set V into subsets $V_1, V_2, \ldots, V_k$, where $R_i = V_i \cap R$ for each $i \in [k]$, and further obtain sets $R'_1 \subseteq R_1, R'_2 \subseteq R_2, \ldots, R'_k \subseteq R_k$ using $\widetilde{O}(n^{5/3})$ queries such that*

1. *Each $G[V_i]$ is a (ϕ, R_i) almost expander with core R'_i.*

2. *The number of crossing edges $\sum_{i \neq j} E(V_i, V_j) = O(\phi|R|(\tau + 1) \log^6 n)$.*

After obtaining this decomposition, we classify each part as follows.

Definition 5.19. For each set V_i, $i \in [k]$, we say that V_i is

1. empty, if $R'_i = \emptyset$

2. small, if $1 \leq |R'_i| \leq (\frac{1}{\phi})^2$

3. large, if $|R'_i| > (\frac{1}{\phi})^2$

Now, we construct $\widetilde{R}$ as follows:

1. Include every vertex of $R_i \setminus R'_i$ in $\widetilde{R}$, for each $i \in [k]$.

2. For each V_i, $i \in [k]$,

   - if it is empty, do nothing;
   - if it is small, add an arbitrary vertex of R'_i to $\widetilde{R}$;
   - else if it is large, add $1 + \frac{1}{\phi}$ arbitrary vertices of R'_i to $\widetilde{R}$.

Lemma 5.20. *Suppose that R is τ-separated and ϕ-balanced. If $|\partial V_i| \geq \tau + 1$ for each V_i, then $|\widetilde{R}| \leq O(\phi|R| \log^6 n) + \frac{|R|}{\log n}$ and $\widetilde{R}$ is τ-separated.*

Proof. Since $|\partial V_i| \geq \tau + 1$ and $\sum_{i \neq j} E(V_i, V_j) = O(\phi|R|(\tau + 1) \log^6 n)$, the number of subsets is bounded by $k \leq O(\phi|R| \log^6 n)$. Therefore, the total number of vertices in $\widetilde{R} \cap R'_i$ across all small V_i is at most $O(\phi|R| \log^6 n)$.

For each large V_i, observe that we pick only an $O(\phi)$ fraction of R'_i to add to $\widetilde{R}$. Additionally, for each i, we have $|R_i \setminus R'_i| \leq \frac{|R_i|}{\log n}$. Combining these results, we have

$$|\widetilde{R}| \leq \frac{|R|}{\log n} + O(\phi|R| \log^6 n)$$

as desired. Now we show that $\widetilde{R}$ must be τ-separated.

Assume, for the sake of contradiction, that $\widetilde{R}$ is not τ-separated. This implies the existence of a cut $C = (C_1, C_2 = V(G) \setminus C_1)$ with cut-size $\partial C \leq \tau$ such that $\widetilde{R} \subseteq C_1$. We will demonstrate that this cut C must be ϕ-unbalanced for R, contradicting our assumption that R is ϕ-balanced.

Before we proceed further, we will show that, in this case, C_2 must contain the "smaller" side of every large V_i. The next lemma will clarify this point.

Claim 5.21. *If $C_2 \cap \widetilde{R} = \emptyset$, then for any large V_i, $i \in [k]$, we must have $|R'_i \cap C_2| \leq |R'_i \cap C_1|$.*

16

Proof. Since V_i is a (ϕ, R_i) almost-expander with core R_i' and (C_1, C_2) is a cut with cut-size at most τ, we must have $\min\{|R_i' \cap C_1|, |R_i' \cap C_2|\} \le \frac{\tau}{\phi(\tau+1)} \le \frac{1}{\phi}$.

Thus if $|R_i' \cap C_1| \le |R_i' \cap C_2|$, then $|R_i' \cap C_1| \le \frac{1}{\phi}$. Recall that we picked $1 + \frac{1}{\phi}$ vertices from R_i' and added them to $\widetilde{R}$. It then follows that at least one vertex of $R_i' \cap C_2$ was added to $\widetilde{R}$, and hence $\widetilde{R} \cap C_2 \ne \emptyset$, which is a contradiction. $\qquad\square$

Now, we show that C is ϕ-unbalanced for R. First, observe that $\sum_{\text{large } V_i} |R_i' \cap C_2| \le \frac{1}{\phi}$; otherwise,

$$|\partial C| \ge \sum_{\text{large } V_i} \phi(\tau + 1)\min\{|R_i' \cap C_1|, |R_i' \cap C_2|\} = \phi(\tau+1)\sum_{\text{large } V_i}|R_i' \cap C_2| \ge \tau + 1.$$

where equality holds by Claim 5.21.

Second, the number of small V_i satisfying $R_i' \cap C_2 \ne \emptyset$ must be at most $\frac{1}{\phi}$; otherwise,

$$|\partial C| \ge \sum_{\text{small } V_i} \phi(\tau + 1)\min\{|R_i' \cap C_1|, |R_i' \cap C_2|\} \ge \sum_{\text{small } V_i, R_i' \cap C_2 \ne \emptyset} \phi(\tau + 1) \ge \tau + 1$$

where we use the fact that $|R_i' \cap C_1| \ge 1$, otherwise we must pick a vertex in $R_i' \cap C_2$ into $\tilde{R}$, which is a contradiction.

In total, we have

$$|R \cap C_2| = \sum_{i \in [k]} |R_i' \cap C_2| \le \sum_{\text{large } V_i} |R_i' \cap C_2| + \sum_{\text{small } V_i, |R_i' \cap C_2| \ne \emptyset} |R_i' \cap C_2| \le \frac{1}{\phi} + \left(\frac{1}{\phi}\right)^3$$

where the equality holds because $(R_i \setminus R_i') \cap C_2 = \emptyset$ for any $i \in [k]$; otherwise, we have $C_2 \cap \widetilde{R} \ne \emptyset$. This implies that C is ϕ-unbalanced for R, contradicting that R is ϕ-balanced. $\qquad\square$

We are now ready to prove Lemma 5.15.

Proof of Lemma 5.15. First, we compute a (ϕ, R) almost-expander decomposition using Lemma 5.18, which results in the sets $V_1, V_2, \dots, V_k, R_1, R_2, \dots, R_k$ and the cores $R_i' \subseteq R_i$ for each $i \in [k]$.

Next, we make $\widetilde{O}(n)$ queries to determine if there exists a set V_i such that $|\partial V_i| \le \tau$. If such a set exists, we return this cut and terminate.

If no such set exists, we construct $\widetilde{R} \subseteq R$ as described previously. In this case, since $|\partial V_i| \ge \tau + 1$ for each $i \in [k]$, Lemma 5.20 guarantees that $|\widetilde{R}| \le O(\frac{1}{\phi}|R|\log^6 n) + \frac{|R|}{\log n}$ and that $\widetilde{R}$ is τ-separated. Then, we can return $\widetilde{R}$ as desired. $\qquad\square$

Theorem 5.1 now follows by combining Lemma 5.13 and Lemma 5.15.

Proof of Theorem 5.1. We begin by computing a dominating set R using Theorem 5.4.

Now let $\phi = 1/\log^{10} n$. Using both Lemmas 5.13 and 5.15 with the set R as the terminal set, we proceed in one of two ways: if we find a cut with size at most τ in $\widetilde{O}(n^{5/3})$ queries, then we are done. Otherwise, if we find a subset $R' \subseteq R$ such that $|R'| \le \frac{1}{2}|R|$ and R' is still τ-separated. We replace R by R' and recursively continue the algorithm till we either find a cut of size at most τ, or confirm that no such cut exists.

Since at each step $|R'| \le \frac{1}{2}|R|$ by the choice of ϕ, so the recursion can proceed for at most $\mathcal{O}(\log |R|) = \widetilde{O}(1)$ iterations. This process guarantees that we will eventually find a cut of size at most τ or declare that no such cut exists. Finally, note that the total query complexity is $\widetilde{O}(n^{5/3})$ since we make at most $\widetilde{O}(1)$ calls to the subroutines in Lemmas 5.13 and 5.15. $\qquad\square$

6 Conclusion

We show the first subquadratic deterministic algorithms for the *s-t* minimum cut and global minimum cut problems in simple graphs, both using $\widetilde{O}(n^{5/3})$ cut queries. Nevertheless, there remains a considerable gap between our results and the current lower bound of $\Omega(n)$ [HMT88]. Improving the lower bound to $\omega(n)$ is very interesting as it would separate deterministic and randomized cut query complexity for the global min-cut problem.

For the upper bound side, algorithms using $\widetilde{O}(n)$ queries would be exciting. This algorithm must be very different from ours because we explicitly compute a maximum *s-t* flow and the flow of value ν may have representation size as large as $\Omega(n\sqrt{\nu}) = \Omega(n^{1.5})$ even on simple unweighted graphs. In fact, it is known that, given any *simple* unweighted graph G where the maximum *s-t* flow value is ν, there exists a subgraph $H \subseteq G$ with $O(n\sqrt{\nu})$ edges such that the maximum *s-t* flow in H has the same value as the one in G and this bound is tight (see, e.g., [KL98, RSW18]). Thus, it is interesting whether there is a (near-optimal) algorithm for explicitly computing maximum *s-t* flow in a simple unweighted graph using $\widetilde{O}(n\sqrt{\nu})$ cut queries. Through our framework, this would immediately imply a global min-cut algorithm using $\widetilde{O}(n^{1.5})$ cut queries, reaching the barrier of this approach.

It is interesting to generalize our result to weighted graphs or just non-simple unweighted graphs. Our technique does not work because Lemma 5.3 is specific to simple unweighted graphs.

References

[AAL21]   Simon Apers, Arinta Auza, and Troy Lee. A sublinear time quantum algorithm for st minimum cut on dense simple graphs. *arXiv preprint arXiv:2110.15587*, 2021. 2

[AB23]   Daniel Anderson and Guy E Blelloch. Parallel minimum cuts in o (m log2 n) work and low depth. *ACM Transactions on Parallel Computing*, 10(4):1–28, 2023. 1

[ACK21]   Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and or queries. In *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021. 2

[AD21]   Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 172–180. SIAM, 2021. 1, 2

[AEG+22] Simon Apers, Yuval Efron, Paweł Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 507–518. IEEE, 2022. 1, 2

[AL21]   Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *arXiv preprint arXiv:2109.02115*, 2021. 2

[ARV09]   Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):1–37, 2009. 23

[BFS86]   László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 337–347. IEEE, 1986. 2

[BHPNR⁺18] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 5

[DEMN21]   Michal Dory, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Distributed weighted min-cut in nearly-optimal time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1144–1153, 2021. 1

[DHNS19]   Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 343–354, 2019. 1

[Din06]   Yefim Dinitz. Dinitz'algorithm: The original version and even's version. In *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 218–240. Springer, 2006. 2, 6, 7, 8

[Gab91]   Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 112–122, 1991. 1

[GG18]   Barbara Geissmann and Lukas Gianinazzi. Parallel minimum cuts in near-linear work and low depth. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 1–11, 2018. 1

[GH61]   Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. 1

[GHN⁺23]   Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. Fully dynamic exact edge connectivity in sublinear time. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 70–86. SIAM, 2023. 1

[GK00]   Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28:104–124, 2000. 1, 2

[GZ22]   Mohsen Ghaffari and Goran Zuzic. Universally-optimal distributed exact min-cut. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 281–291, 2022. 1

[Har08]   Nicholas James Alexander Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, 2008. 1

[HLRW24]   Monika Henzinger, Jason Li, Satish Rao, and Di Wang. Deterministic near-linear time minimum cut in weighted graphs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3089–3139. SIAM, 2024. 1

[HMT88]   András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 186–191, 1988. 1, 2, 18

[HO94]   JX Hao and James B Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17(3):424–446, 1994. 1

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

[JST24]   Wenyu Jin, Xiaorui Sun, and Mikkel Thorup. Fully dynamic min-cut of superconstant size in subpolynomial time. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2999–3026. SIAM, 2024. 1

[Kar94]   David R Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 648–657, 1994. 1

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

[KKOV07]   Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 6(7):12, 2007. 15, 22

[KL98]   David R Karger and Matthew S Levine. Finding maximum flows in undirected graphs seems easier than bipartite matching. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 69–78, 1998. 18

[KM94]   David R Karger and Rajeev Motwani. Derandomization through approximation: An nc algorithm for minimum cuts. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of Computing*, pages 497–506, 1994. 1

[KS96]   David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996. 1

[KT18]   Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM (JACM)*, 66(1):1–50, 2018. 1

[LC24]   Hang Liao and Deeparnab Chakrabarty. Learning spanning forests optimally in weighted undirected graphs with cut queries. In *International Conference on Algorithmic Learning Theory*, pages 785–807. PMLR, 2024. 2

[Li21]   Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–395, 2021. 1

[LP20]   Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 85–92. IEEE, 2020. 1, 3, 9, 11, 12, 15

[MN20]   Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 496–509, 2020. 1

[NI92]   Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of ak-connected graph. *Algorithmica*, 7(1):583–596, 1992. 1

[RS95]   Ran Raz and Boris Spieker. On the "log rank"-conjecture in communication complexity. *Combinatorica*, 15:567–588, 1995. 2

[RSW18]   Aviad Rubinstein, Tselil Schramm, and S Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*, 2018. 1, 18

[RWZ20] Cyrus Rashtchian, David P Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (AP-PROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. 2

[SW19]   Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2635. SIAM, 2019. 15, 24

[Tho07]   Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. 1

A   Computing Expander Decomposition: Proof of Lemma 5.18

Definitions. Given a graph $H = (V(H), E(H))$, define the *sparsity* of a cut $(S, V(H) \setminus S)$ as $\frac{|E(H, V(H) \setminus S)|}{\min\{|S|, |V(H) \setminus S|\}}$. Further, given a terminal set R, we define the *sparsity with respect to R* of a cut $(S, V(H) \setminus S)$ as $\frac{|E(H, V(H) \setminus S)|}{\min\{|S \cap R|, |(V(H) \setminus S) \cap R|\}}$. We say that the set R is ϕ-expanding in H if there is no cut $(S, V(H) \setminus S)$ in H which has sparsity at most ϕ with respect to R. Further, H is ϕ-expanding if $V(H)$ is ϕ-expanding in H. We say that a cut $(S, V(H) \setminus S)$ is b-balanced with respect to R if both $S \cap R$ and $(V(H) \setminus S) \cap R$ have size at least $b|R|$. For a set of vertices $S \subseteq V(H)$, we define the volume of S, $\mathrm{vol}_H(S)$, as the sum of the degrees of the vertices in S in H. The conductance of a cut $(S, V \setminus S)$ in H is defined as $\frac{E(S, V \setminus S)}{\min\{\mathrm{vol}_H(S), \mathrm{vol}_H(V \setminus S)\}}$. The conductance of H is the minimum conductance across all such cuts. We drop the subscripts when the graph H is clear from the context.

The following is the key subroutine we need for finding a decomposition into (ϕ, R) almost-expanders.

Lemma A.1 (One-step of Expander Decomposition)*. Given a graph $G = (V, E)$, terminal set $R \subseteq V$ and parameters $\phi = \text{poly}(\frac{1}{\log n})$ and $\tau \geq 1$ such that $|R| \leq \widetilde{O}(\frac{n}{\tau})$, using $\widetilde{O}(n^{\frac{5}{3}})$ queries we can either*

- *return a cut $(S, \overline{S})$ that is $(\frac{1}{\log^5 n})$-balanced and $(\phi\tau)$-sparse with respect to R.*

- *return a set $R' \subseteq R$ of size $|R'| \geq (1 - \frac{1}{\log n})|R|$ such that R' is $\frac{\phi\tau}{\log^5 n}$-expanding in G.*

Before showing this crucial lemma, let us first show why it implies Lemma 5.18.

Proof of Lemma 5.18. We start with the input graph G and run the algorithm of Lemma A.1. If Lemma A.1 returns a cut $(S, \overline{S})$, then we recursively apply Lemma A.1 on $G[S]$ and $G[\overline{S}]$. On the other hand, if Lemma A.1 returns a set R', then G is a (ϕ, R) almost expander with core R'. The recursion depth for the first case is at most $\log^5 n \log |R|$, and every leaf of the recursion tree, where the second case occurs, together give us the decomposition into (ϕ, R) almost expanders.

The argument about the number of crossing edges is a standard charging argument: Every time we obtain a cut $(S, \overline{S})$, that is $\frac{1}{\log^5 n}$ balanced and $\phi(\tau + 1)$ sparse with respect to R, we charge the number of cut edges to the terminals on the side of the cut with smaller number of terminals. Since each cut is $\phi(\tau + 1)$ sparse, each terminal $r \in R$ receives a charge of at most $\phi(\tau + 1)$. Since every cut we find is $\frac{1}{\log^5 n}$ balanced, the recursion depth is at most $\log^5 n \log |R|$, and hence a terminal is charged at most $\log^5 n \log |R| \leq \log^6 n$ times. It follows that the number of cut edges is at most $\mathcal{O}(\phi|R|(\tau + 1) \log^6 n)$. $\qquad\square$

We now proceed to prove Lemma A.1. The key ingredient is the cut-matching game of [KKOV07]. Here, we state the cut-matching game where in each round the matching player returns a perfect *b-matching* instead of a perfect matching. A perfect b-matching is a set of edges such that each vertex v has exactly b edges in the set containing v. A perfect b-matching between two sets A and B is a perfect b-matching in which every edge is between a vertex of A and a vertex of B. We now describe the cut-matching framework.

- Start with an empty graph X with n vertices.

- In round i, the cut player chooses a bisection (A, B) of $V(X)$.

- Then, the matching player returns an arbitrary perfect b-matching M_i between A and B, and the edges in M_i are added to $E(X)$.

The goal of the cut player is to guarantee that, after a few rounds, the graph X (whose edge set is the union $\bigcup_i M_i$) has sparsity at least b, i.e., for every $S \subseteq V(X)$, we have

$$E_X(S, V(X) \setminus S) \geq b \cdot \min\{|S|, |V(X) \setminus S|\}.$$

Theorem A.2 ([KKOV07])*. In the cut-matching game where the matching player always returns a perfect b-matching in each round, there exists an algorithm for the cut player that, after $r = O(\log n)$ rounds, guarantees that X has sparsity at least b.*

We remark that the running time of Theorem A.2 is exponential, but since we will know the entire graph X along with its edge set $E(X)$ whenever we invoke this theorem, we spend zero cut queries to run this algorithm, and hence it does not affect the query complexity of our algorithm. However, if we insist on a polynomial time implementation, we can still guarantee that $\bigcup_{i=1}^{r} M_i$ has sparsity at least $b/O(\sqrt{\log n})$ using [ARV09]. For the rest of the section, for simplicity, we work with the exponential time cut player as in Theorem A.2.

Before we prove Lemma A.1, we need an implementation for the matching player. Towards this, let us define the notion of an *almost perfect b-matching*.

Definition A.3. Given sets A, B, parameters β, b and a graph H, a set of edges F is a β-almost perfect A-B b-matching, if there exists a set of βb edges F', such that $F \cup F'$ is a perfect b-matching between A and B.

The matching player, in each round, either finds a sufficiently balanced and sparse cut with respect to the terminal set R, or returns an embedding of an almost perfect b-matching. In every round, if we find a balanced and sparse cut, we return this cut. Else, we use the almost perfect matching, together with a small number of fake edges, to obtain a perfect matching and make progress for the cut-matching game.

Before stating this theorem, we define the notion of a flow embedding. Given a set of edges M, we say that $M \preceq^{\text{flow}} \frac{1}{\phi} G$ to mean that there is a multicommodity flow in G that can simultaneously send one unit of flow across each edge in M with congestion $\frac{1}{\phi}$.

Theorem A.4 (Matching Player). *Suppose we are given a graph $G = (V, E)$, integers $\tau, \beta \geq 1$, a terminal set $R \subseteq V$ such that $|R| \leq \widetilde{O}(\frac{n}{\tau})$ and a parameter $\phi = \text{poly}(\frac{1}{\log n})$. Given a bisection (A, B) of R, s.t. $|A| = |B| = \frac{|R|}{2}$ there is an algorithm that either*

(1) *finds a cut $(S^*, V \setminus S^*)$ with $\min\{|S^* \cap A|, |(V \setminus S^*) \cap B|\} \geq \beta$, such that $(S^*, V \setminus S^*)$ is $\phi(\tau + 1)$ sparse with respect to R.*

(2) *obtains a flow embedding of a β-almost perfect A-B $(\tau+1)$-matching M such that $M \preceq^{\text{flow}} \frac{1}{\phi} G$*

using $\widetilde{O}(n^{\frac{5}{3}})$ queries.

Proof. Add two vertices s_{source} and s_{sink} to the original graph. Connect s_{source} to all vertices in A with capacity $\tau + 1$, and s_{sink} to all vertices in B with capacity $\tau + 1$, and we give all edges in G with capacity $\frac{1}{\phi}$.

Then since $|R| \leq \widetilde{O}(\frac{n}{\tau})$, the max-flow from s to t is at most $\widetilde{O}(n)$. We can then compute an s_{source}-s_{sink} max flow in $\widetilde{O}(n^{\frac{5}{3}})$ queries by Theorem 1.2[3]. If the maximum flow is of size $\geq (|A| - \beta)(\tau + 1)$, then using the set of flow paths from the flow algorithm, we obtain a β-almost perfect A-B $(\tau + 1)$ matching M such that $M \prec^{\text{flow}} \frac{1}{\phi} G$. Otherwise, after finding the max flow f^* from s_{source} to s_{sink}, we run BFS (see Corollary 4.3) from s_{source} in the residual graph G_{f^*} and find all vertices that can be reached by s_{source}, denoted as S^*. Then $|S^* \cap A| \geq \beta$, since there are at least β vertices in A not sending all the demands. Similarly we can show that $|\overline{S^*} \cap B| \geq \beta$, where $\overline{S^*} = V \setminus S^*$. Now we show that $(S^*, \overline{S^*})$ is $\phi(\tau + 1)$ sparse with respect to R. Without loss of generality, let us assume that $|S^* \cap R| \leq |\overline{S^*} \cap R|$. Recall that the capacity of every edge in E is

---

[3]As in previous instances, we can add parallel edges and subdivide the edges incident to the source and sink to ensure that all edges have unit capacity

set to be $\frac{1}{\phi}$. Since the flow is feasible, and every edge of the min-cut is saturated in the maximum flow, we must have $\frac{|\partial S^*|}{\phi} \leq |S \cap R|(\tau + 1)$, which means that $|\partial S^*| \leq \phi|S \cap R|(\tau + 1)$. It follows that $(S^*, \overline{S^*})$ is a $\phi(\tau + 1)$-sparse cut with respect to R. $\qquad\square$

Proof of Lemma A.1. We run the cut player algorithm in Theorem A.2, and the matching player algorithm in Theorem A.4 with $\beta = \frac{|R|}{\log^5 n}$. If in any round the matching player returns a cut that is $\phi(\tau + 1)$ sparse with respect to R with both sides having at least β terminals, we simply return this cut as this cut satisfies the requirement of Lemma A.1. Otherwise, in every round, the matching player returns an β-almost perfect $(\tau + 1)$-matching F, so that there exists a set F' with $|F'| \leq \beta(\tau + 1)$ such that $F \cup F'$ is a perfect b-matching and we add the edges in $F \cup F'$ to X to continue the cut-matching game. Mark the edges F' as *fake edges*.

After $r = \mathcal{O}(\log n)$ rounds of the cut-matching game, if the matching player did not return a balanced and sparse cut in any round, Theorem A.2 guarantees that X has sparsity at least $b = \tau + 1$. Note that the total number of fake edges is at most $\beta(\tau + 1)r$. Also, since each matching flow embeds in $\frac{1}{\phi}G$, and there are at most $O(\log n)$ rounds, we must have $X \preceq^{\text{flow}} \frac{\mathcal{O}(\log n)}{\phi}G$.

Consider any cut $(S, V(X) \setminus S)$ with $|S| \leq |V(X) \setminus S|$ in X. Since the sparsity is at least $(\tau + 1)$, it follows that $|E(S, V(X) \setminus S)| \geq (\tau + 1)|S|$. Note that in X, every vertex has degree $(\tau + 1)r = \mathcal{O}((\tau + 1) \log n)$.

This means that $\text{vol}(S) \leq \mathcal{O}(|S|(\tau + 1) \log n)$, and hence the conductance of every cut $(S, V(X) \setminus S)$ is at least $\Omega(\frac{1}{\log n})$.

We will now show that there exists a set R' such that $R' \subseteq R$, $|R'| \geq |R|(1 - \frac{1}{\log n})$, and R' is $\phi(\tau + 1)$ expanding in X. The construction of R' is simple. On a high level, our goal is to try to certify the expansion of X after removing the fake edges F'. Therefore, a natural step is to apply expander pruning to the graph X after deleting the set of fake edges F'.

Theorem A.5 ([SW19], Theorem 1.3). *Given a graph H in which every cut has conductance at least ϕ', and a set of deletion edges F', there exists a prune set $P \subseteq V(H)$ with $\text{vol}_H(P) \leq \frac{8}{\phi'}|F'|$, such that in the induced subgraph on $V(H) \setminus P$ after removing F', in $H[V(H) \setminus P] \setminus F'$, every cut $(S, (V(H) \setminus P) \setminus S)$ has conductance at least $\frac{\phi'}{6}$.*

Apply expander pruning to graph X, with the set of fake edges F' as the deletion set, and the conductance parameter $\phi_X = \Omega(\frac{1}{\log n})$ to obtain a prune set P. Here we recall that the graph X is explicitly known, so we spend zero queries to obtain this set P. Note that $\text{vol}(P) \leq \frac{8|F'|}{\phi_X} \leq \mathcal{O}(\beta(\tau + 1)r \log n) = \mathcal{O}(\beta(\tau + 1) \log^2 n) \leq \frac{|R|(\tau + 1)}{\log^2 n}$ since $\beta = \frac{|R|}{\log^5 n}$. But each vertex has degree $r(\tau + 1)$ in H, hence the size of P, $|P| \leq \frac{\text{vol}(P)}{r(\tau + 1)} \leq \mathcal{O}(\beta \log n) \leq \frac{|R|}{\log^2 n}$. We define R' as all the vertices $v \in R \setminus P$, which have a combined at most $\frac{\tau + 1}{10}$ number of edges incident to P and edges incident in F'. Note that $|F'| + \text{vol}(P) \leq 2\text{vol}(P) \leq \frac{2|R|(\tau + 1)}{\log^2 n}$.

By Markov's inequality, the number of $v \in R$ for which there are more than $\frac{\tau + 1}{10}$ edges in F or whose other endpoint is in P, must be at most $20\frac{|R|}{\log^2 n} \leq \frac{|R|}{2 \log n}$. But this means $|R'| \geq |R|(1 - \frac{1}{2 \log n}) - |P| \geq |R|(1 - \frac{1}{\log n})$ since $|P| \leq \frac{|R|}{\log^2 n}$.

Since each vertex $v \in R$ has degree $r(\tau + 1)$, it follows that each vertex in R' has degree at least $\frac{r}{10}(\tau + 1)$ in $X[V(X) \setminus P] \setminus F'$. The next lemma proves the desired expansion property of R'.

Lemma A.6. R' is $\frac{\phi(\tau + 1)}{\log^5 n}$ expanding in G.

24

Proof. Consider a cut $(S, V \setminus S)$ in G with $|S \cap R'| \leq |(V \setminus S) \cap R'|$. Let $R_1 = S \cap (R \setminus P)$ and $R_2 = (V(X) \setminus S) \cap (R \setminus P)$. Since the induced subgraph on $V(X) \setminus P \setminus F$ is a $\Omega(\frac{1}{6 \log n}) \geq \frac{1}{\log^2 n}$ conductance expander, there are at least $\frac{\min(\text{vol}(R_1), \text{vol}(R_2))}{6 \log^2 n} \geq \frac{|S \cap R'|(\tau+1)}{60 \log^2 n}$ edges in $E(X)$ between the sets R_1 and R_2, where the inequality follows since each vertex of R' has degree at least $\frac{\tau+1}{10}$ in $X[V(X) \setminus P] \setminus F'$. Recall that $X \preceq^{\text{flow}} O(\frac{\log n}{\phi})G$. Thus the number of edges across $(S, V \setminus S)$ in G is at least $\Omega\left(\frac{|S \cap R'|\phi(\tau+1)}{\log^3 n}\right) \geq \frac{\phi(\tau+1)|S \cap R'|}{\log^5 n}$. $\square$

$\square$