Deterministic Minimum Steiner Cut in Maximum Flow Time

Matthew Ding
University of California, Berkeley, CA, USA

Jason Li
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We devise a deterministic algorithm for minimum Steiner cut, which uses $(\log n)^{O(1)}$ maximum flow calls and additional near-linear time. This algorithm improves on Li and Panigrahi's (FOCS 2020) algorithm, which uses $(\log n)^{O(1/\epsilon^4)}$ maximum flow calls and additional $O(m^{1+\epsilon})$ time, for $\epsilon > 0$. Our algorithm thus shows that deterministic minimum Steiner cut can be solved in maximum flow time up to polylogarithmic factors, given any black-box deterministic maximum flow algorithm. Our main technical contribution is a novel deterministic graph decomposition method for terminal vertices that generalizes all existing s-strong partitioning methods, which we believe may have future applications.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases graph algorithms, minimum cut, deterministic

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.46

Related Version *Full Version*: `https://arxiv.org/abs/2312.16415` [4]

Acknowledgements Jason Li: This work was done while visiting the Simons Institute for the Theory of Computing. We want to thank Monika Henzinger, Satish Rao, and Di Wang for helpful discussions related to Lemma 11.

1 Introduction

The minimum cut (or "min-cut") of a weighted graph is the smallest weighted subset of edges whose deletion disconnects the graph. The problem of finding the minimum cut is one of the most fundamental problems in combinatorial optimization and theoretical computer science as a whole. It also has important applications such as network optimization [1] and image segmentation [3]. Thus, finding faster algorithms for this problem will have far-reaching applications for a wide variety of fields. Recently, there has been a large amount of groundbreaking work in the field, including deterministic almost-linear time[1] algorithms for both minimum cut [10] and maximum flow [2].

1.1 Minimum Steiner Cut Background

A classic extension of the min-cut problem is the minimum Steiner cut (or "Steiner min-cut") problem. In this problem, we are given an undirected, weighted graph $G = (V, E)$ and a subset $T \subseteq V$ of terminals. A Steiner cut is a subset of edges whose removal disconnects at least one pair of terminals in the graph. The minimum Steiner cut is the Steiner cut with the minimum total weight of cut edges. This problem generalizes both $s - t$ minimum cut ($T = \{s, t\}$) and global minimum cut ($T = V$) and is therefore a fundamental problem in graph algorithms.

[1] Near-linear algorithms have runtime $\tilde{O}(m)$ and almost-linear algorithms have runtime $m^{1+o(1)}$. We use $\tilde{O}(\cdot)$ to hide polylogarithmic factors.

The classical algorithm to solve minimum Steiner cut uses $|T|-1$ max-flow computations. Li and Panigrahi [11] give a randomized algorithm which reduces minimum Steiner cut in near-linear time to just polylogarithmic number of max-flow computations. They additionally provide a deterministic algorithm which takes, for any parameter $\epsilon > 0$, $(\log n)^{O(1/\epsilon^4)}$ max-flow calls with $O(m^{1+\epsilon})$ additional running time. Given the currently known fastest deterministic maximum flow algorithm in almost-linear time [2], the two results combined give an almost-linear time algorithm for global minimum cut, which matches the algorithm of Li [10]. We remark that a very recent work [5] has improved the running time of deterministic *global* minimum cut to near-linear, i.e., $\tilde{O}(m)$. However, since minimum Steiner cut is at least as hard as $s-t$ minimum cut, traditionally solved through $s-t$ max-flow, a near-linear time minimum Steiner cut algorithm remains elusive without an equally fast max-flow algorithm.

1.2 Our Contributions

We show that a deterministic, near-linear time max-flow algorithm is the *only* obstacle towards obtaining a deterministic, near-linear time minimum Steiner cut algorithm. More precisely, we introduce a new deterministic algorithm that finds the minimum Steiner cut in polylogarithmic $s-t$ max flow calls and near-linear additional processing time.

▶ **Theorem 1.** *Given an undirected, weighted graph $G = (V, E)$ with n vertices and m edges, polynomially bounded edge weights, and a set of terminal vertices $T \subseteq V$, there is a deterministic minimum Steiner cut algorithm that makes polylog(n) maximum flow calls on undirected, weighted graphs with $O(n)$ vertices and $O(m)$ edges, and runs in $\tilde{O}(m)$ time outside of these maximum flow calls.*

Specifically, a hypothetical **deterministic near-linear time algorithm for $s-t$ max-flow also implies a deterministic near-linear time algorithm for minimum Steiner cut**. This result was not known from the work of [11], given the additional $m^{1+\epsilon}$ running time in their deterministic algorithm. In fact, for the case of polylogarithmic maximum flow calls ($\epsilon = \Omega(1)$), their algorithm has runtime $m^{1+\Omega(1)}$, even slower than almost-linear. Our algorithm also matches the running time of the randomized minimum Steiner cut algorithm from [11] up to polylogarithmic factors.

Terminal-Based Partitioning Methods

Expander decompositions have been a powerful tool for solving minimum cut problems in recent years [10, 11]. However, the current state-of-the-art deterministic expander decomposition takes almost-linear time [12], and it is an open problem whether this can be improved.

A key tool for deterministic near-linear time algorithms is the decomposition into s-strong clusters used by recent minimum cut algorithms on *simple* graphs [6, 7]. However, recent work [5] has applied the concept of s-strong clusters to weighted graphs to obtain a deterministic near-linear global minimum cut algorithm. Our main technical contribution is to extend this decomposition framework on general weighted graphs and apply it towards a decomposition that *specifically partitions clusters of terminals with a boundary proportional to the size of the terminal set.* We state this result informally below.

▶ **Theorem 2** (Informal). *Given an undirected, weighted graph $G = (V, E)$ with n vertices and m edges, polynomially bounded edge weights, a set of terminal vertices $T \subseteq V$, sparsity parameter $0 < \psi < 1$, and cut size parameter $\delta > 0$, there is a deterministic algorithm which returns a vertex partitioning of clusters $V_1, V_2, ..., V_\ell$ such that the following hold:*

1. *For every cluster, any cut with weight less than δ splits the cluster with at most* polylog(n) *terminals on at least one side.*
2. *For every cluster, any cut with weight less than δ either does not split the cluster or has at least $\geq \delta/\text{polylog(n)}$ weight of cut edges inside the cluster.*
3. *The total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$.*

The algorithm makes $O(\log^2 n)$ maximum flow calls on undirected, weighted graphs with $O(n)$ vertices and $O(m)$ edges and runs in $\tilde{O}(m)$ time outside of these maximum flow calls.

Properties 1 and 3 together directly generalize the notion of s-strong partitions to the terminal regime. Property 2 provides an additional guarantee useful for the Steiner algorithm of [11] and can be achieved with only polylogarithmic loss elsewhere in the decomposition.

2 Preliminaries

In this paper, all graphs are undirected and weighted. For simplicity, all weights are assumed to be polynomially bounded (however, this can be relaxed by adding logarithmic dependence on the maximum edge weight).

We begin by introducing standard definitions and tools from previous works that we will utilize for our algorithm. We also define our new modification of s-strong clusters to terminals.

We use a standard definition of induced subgraphs using self-loops for boundary edges, which preserves degrees of vertices in subgraphs. We denote the induced subgraph of the vertex set $S \subset V$ on graph $G(V, E)$ as $G[S]$.

2.1 Sparsity and Strength

Sparsity is a specific measure of how connected a graph is. For a cut $(U, \overline{U})$, where $\overline{U} = V \setminus U$, we define $\partial U = \partial \overline{U}$ as the boundary of the cut, which is the set of edges between U and $\overline{U}$.

▶ **Definition 3** (Sparsity). *The sparsity of a cut $(U, \overline{U})$ is defined as*

$$\Psi(U) = \frac{w(\partial U)}{\min\{|U|, |\overline{U}|\}} = \Psi(\overline{U}) \tag{1}$$

We also introduce the new definition of terminal-sparsity for Steiner cuts with the terminal set T.

▶ **Definition 4** (Terminal-sparsity). *The terminal-sparsity of a cut $(U, \overline{U})$ is defined as*

$$\Psi_T(U) = \frac{w(\partial U)}{\min\{|U \cap T|, |\overline{U} \cap T|\}} = \Psi_T(\overline{U}) \tag{2}$$

We use the terms *ψ-sparse* and *ψ-terminal-sparse* to refer to cuts with sparsity and terminal-sparsity $< \psi$, respectively.

The concept of strength, introduced by Kawarabayashi and Thorup [7], is a relaxed notion of edge expanders. At a high level, a vertex subset $U \subseteq V$ is s-strong if every cut $(C, \overline{C})$ of weight at most δ satisfies $\min\{\mathbf{vol}(C \cap U), \mathbf{vol}(\overline{C} \cap U)\} \leq s$, where the volume $\mathbf{vol}(S)$ is the sum of degrees of vertices in S. In [7], the parameter δ is chosen as the minimum degree of all vertices, which serves as an upper bound on the min-cut.

One of our main conceptual contributions is translating s-strength to the terminal setting and providing necessary generalizations to handle the Steiner min-cut problem. First, we work from a *sparsity* viewpoint, which bounds the minimum cardinality of intersection

$\min\{|C \cap U|, |\overline{C} \cap U|\}$ instead of volume, which is more handy when we start introducing terminals. Second, we can no longer choose δ as the minimum degree since it no longer upper bounds the Steiner min-cut. One natural choice is the minimum (weighted) degree of all vertices in set S, but instead, for technical reasons, we set δ closer to the minimum Steiner cut λ itself. For now, we keep δ as a free parameter and provide our s-strong guarantees in terms of δ. Finally, we need an additional requirement that if the cut $(C, \overline{C})$ cuts any edges inside a cluster, then it must cut sufficiently many such edges, and we introduce another parameter γ to capture this condition.

▶ **Definition 5** ((s, δ, γ)-strength). *A vertex subset $U \subseteq V$ (called a* cluster*) is* (s, δ, γ)-strong in G *if every cut $(C, \overline{C})$ of graph G with at most weight δ satisfies* $\min\{|C \cap U|, |\overline{C} \cap U|\} \leq s$*, and moreover, if* $\min\{|C \cap U|, |\overline{C} \cap U|\} > 0$ *then* $w(\partial_{G[U]}C) \geq \gamma \cdot \delta$*.*

Next, we introduce the notion of (s, δ, γ)-terminal-strength, where the "size" of a set of vertices is only determined by its number of terminals. This property is necessary to deal with cuts separating terminal vertices instead of just regular ones.

▶ **Definition 6** ((s, δ, γ, T)-terminal-strength). *A vertex subset $U \subseteq V$ (called a cluster) is (s, δ, γ, T)-terminal-strong in G if every Steiner cut $(C, \overline{C})$ of graph G with at most weight δ satisfies* $\min\{|C \cap U \cap T|, |\overline{C} \cap U \cap T|\} \leq s$*, and moreover, if* $|C \cap U \cap T| > 0$ *and* $|\overline{C} \cap U \cap T|\} > 0$*, then* $w(\partial_{G[U]}C) \geq \gamma \cdot \delta$*.*

*For the rest of the paper, we sometimes omit the "*in G*" and the terminal set T from the definition whenever they are apparent from the context.*

An important property of both $(s, \delta, 0)$-strength and terminal strength is that the property is inherited by subgraphs, i.e., if $G(V, E)$ is $(s, \delta, 0)$-strong or terminal-strong, then $G[A]$ is as well for all $A \subseteq V$. This property holds for s-strength [7], and is straightforward to verify that the same is true for our $(s, \delta, 0)$-strength definitions.

Lastly, we also define terminal-strong decompositions, which are analogous to s-strong and expander decompositions, except that we split our graph into (s, δ, γ)-terminal-strong components as opposed to s-strong sets and expanders, respectively.

▶ **Definition 7** ((s, δ, γ, T)-terminal-strong decomposition). *A set of disjoint vertex clusters $V_1, V_2, ..., V_\ell$ is an (s, δ, γ, T)-terminal-strong decomposition if each cluster V_i is (s, δ, γ, T)-terminal-strong, and if the total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$.*

Our primary new technical tool is a fast algorithm for computing an (s, δ, γ, T)-terminal-strong decomposition with a small bounded weight of intercluster edges (Theorem 2), which is presented in detail in Section 4. At a high level, we use a (non-terminal) (s, δ, γ)-strong decomposition to devise an algorithm that finds an (s, δ, γ, T)-terminal-strong decomposition through the *cut-matching game* framework of Khandekar, Rao, and Vazirani [9], which we outline in the following subsection.

Finally, given such a decomposition, we use the framework of [11], replacing their expander decomposition step with our (s, δ, γ)-strong decomposition. We leave the details to Section 5.

2.2 Cut-Matching Game

We start with an overview of the cut-matching game.

1. The cut player chooses a bisection $(S, \overline{S})$ of the graph H_{t-1} based on a given strategy.
2. The matching player chooses a perfect matching of the bisection based on a given strategy.
3. The cut player adds the edges of the perfect matching to graph H_{t-1}, forming graph H_t.

The game continues until graph H_t is an edge-expander. The key insight of the cut-matching game is that there is always a strategy for the cut player that finishes the game in few rounds.

We use the cut-matching game to reduce our problem from one with terminals ((s, δ, γ)-terminal-strong decomposition) to one without terminals ((s, δ, γ)-strong decomposition). We then adapt the $(s, \delta, 0)$-strong decomposition algorithm of [5] to obtain an (s, δ, γ)-strong decomposition for large enough γ.

3 Minimum Steiner Cut Algorithm Overview

The following is an overview of our algorithm (Algorithm 1) to solve minimum Steiner cut on an undirected, weighted graph G deterministically in near-linear time (i.e., $\tilde{O}(m)$) plus polylogarithmic maximum flow calls. Throughout, we assume that we have guessed the value of the Steiner mincut up to factor 2 (which we denote $\tilde{\lambda}$) by, for example, guessing all powers of 2 (incorrect guesses may return an overestimate of the minimum Steiner cut, but we can take the minimum cut ever found at the end).

1. We use the "unbalanced case" of [11] to find the Steiner minimum cut $(C, \overline{C})$ if $\min\{|C \cap T|, |\overline{C} \cap T|\} \leq \text{polylog}(n)$. This algorithm is described in the full version of the paper [4].
2. In the "balanced case", we find an (s, δ, γ)-terminal-strong decomposition on the graph. To do this, we use the cut-matching game on a graph H containing only the terminals of the original graph, with $s = \text{polylog}(n)$, $\delta = \tilde{\lambda}$, and $\gamma = 1/\text{polylog}(n)$. This algorithm is described in Algorithm 2 (Section 4). At a high level, we use a (non-terminal) (s', δ', γ')-strong decomposition to find an (s, δ, γ)-terminal-strong decomposition for appropriate parameters $s', \delta', \gamma', s, \delta, \gamma$.
3. Using our (s, δ, γ)-terminal-strong decomposition, we find a set $T' \subseteq T$ and $|T'| \leq |T|/2$ such that the minimum Steiner cut of G with terminal set T' is the same as with terminal set T. In this case, we recursively apply our minimum Steiner cut algorithm on graph G with terminal set T' (Section 5).

Algorithm 1 Minimum-Steiner-Cut(G, T).

```
Input  : Undirected weighted graph $G$, terminal set $T \subseteq V$, $\gamma = 1/\text{polylog}(n)$,
         $k = \text{polylog}(n)$
for $i \leftarrow 1$ to $O(\log n)$ do
    $U \leftarrow T$
    $\tilde{\lambda} \leftarrow 2^i$
    do
        Run algorithm from Unbalanced Case algorithm with terminal set $U$
            // Unbalanced Case
        Run Terminal-Decomp$(V, U, \tilde{\lambda}, \gamma)$
        Find sparsified set $U' \subset U$ // Theorem 22, Balanced Case
        $U \leftarrow U'$
    while $|U| > k$;
return minimum weight Steiner cut over all iterations of Line 5
```

We give a high-level analysis of the runtime, which we formally prove in the following sections. Each call of terminal decomposition takes polylogarithmic max-flow computations and at most near-linear time with respect to the graph outside of the max-flows. Since the sparsification procedure halves the terminal set each iteration, it adds at most a $\log n$ extra factor in runtime. Along with the additional $\log n$ factor for guessing $\tilde{\lambda}$, this gives us our claimed runtime.

4 Terminal Decomposition Using Cut-Matching Game

The goal of the cut-matching game is to try to certify that the entire vertex set V is (s, δ, γ, T)-terminal-strong in G by iteratively constructing our cut-graph H to be $(s, \tilde{O}(\delta), \gamma)$-strong. This may not always be possible, but throughout the cut-matching game, the algorithm may also verify that V is (s, δ, γ, C)-terminal-strong for a subset $C \subseteq T$ with $|C| \geq 2|T|/3$. In that case, we apply a *trimming* procedure similar to [13]. Otherwise, if this is also not possible, we will be able to find a balanced sparse cut in the cut-graph H. We then run a max-flow between the two terminal sets in G, which outputs either a large flow or (by duality) a small cut. In the former case, we add a corresponding large (fractional) matching to the cut graph H. In the latter case, we immediately find a balanced terminal-sparse cut in G, at which point we recursively decompose the two sides.

Before going through the formal procedure, we give high-level overviews of the cut and matching player strategies.

Cut Player Description

The cut player attempts to find a sparse, balanced cut $(U, T \setminus U)$ in the cut graph, which ensures that we make sufficient progress when the matching player creates a matching. If the cut player fails to find a cut, we terminate the cut-matching game and prove that the original graph G satisfies desirable properties.

Cut Player Strategy on current cut-graph H

- Find an (s, δ, γ)-strong decomposition on cut graph H
- If a cluster with size greater than $2|T|/3$ exists, we terminate the cut-matching game. We trim the cluster according to Algorithm 3 and certify the cluster U as (s, δ, γ)-terminal-strong. We then recursively apply Algorithm 4 on the smaller side.
- Otherwise, we merge the clusters into two groups that each contains between $1/3$ and $2/3$ fraction of all vertices of H (this is always possible, see Claim 12). Denote the bipartition as $(C, \overline{C})$.

Matching Player Description

The matching player's goal is to add edges in the cut graph corresponding to the maximum possible flow in G from one side of the bipartition to the other. They do this by running a maximum flow algorithm across the bipartition. The matching player adds edges to the cut graph if a large flow is successfully routed. Otherwise, a terminal-balanced cut is found, and we terminate the cut-matching game and recursively apply our terminal-strong-decomposition algorithm on both sides of the cut.

We formally define strategies for the cut and matching players in this game in Algorithm 2. Our guarantee given by our cut-matching game method is stated as the following:

Matching Player Strategy on bipartition C of cut-graph H

- We calculate a max-flow on graph G between the terminals in C and $T \setminus C$ using Algorithm 3. If the flow has value at least $|T|/6 \cdot \delta \cdot \psi$, we call the flow a "large flow". Otherwise, the flow has value less than $|T|/6 \cdot \delta \cdot \psi$, so we call the corresponding cut a "small cut".
- If we find a large flow, we add a large matching into cut graph H: we break down the flow into paths and add edges between vertices in graph H with the same corresponding weights as the flow paths.
- If we find a small cut, we certify the minimum cut found as a terminal-balanced, terminal-sparse cut. We stop the cut-matching game and recursively apply our terminal-decomposition algorithm on both sides.

▶ **Lemma 8.** *Given an undirected weighted graph G and parameters $\delta > 0$, $\psi = 1/\text{polylog}(n)$, Algorithm 2 runs in time $\tilde{O}(m)$ plus $\text{polylog}(n)$ calls to maximum flow, and outputs one of the following:*

1. *An $(O(\log^9 n/\psi^5), \delta, \Omega(\psi^5/\log^9 n), T)$-terminal-strong cluster U with $|U \cap T| \geq |T|/3$ such that $\overline{U}$ is either empty or $\psi \cdot \delta$-terminal-sparse, or*
2. *A $\psi \cdot \delta$-terminal-sparse cut $(U, \overline{U})$ with $|U \cap T|, |\overline{U} \cap T| \geq |T|/6$*

The correctness of the Cut Player strategy is shown in Section 4.1, matching player strategy in Section 4.2, and the termination within $L_{\max}$ rounds is proved in Section 4.3.

4.1 Cut Player

The lemma below for $\alpha = L_{\max}/\psi$ shows that Line 5 of Algorithm 2 can be computed efficiently. We apply the lemma on graph H and vertex set T.

▶ **Lemma 9.** *Given any parameters $\delta > 0$ and $\alpha \leq \text{polylog}(n)$ and a graph $G = (V, E)$ with total edge weight at most $\alpha\delta n$, there exists $s \leq O(\alpha^2 \log^2 n)$ and $\gamma = \Omega(1/s)$ and an algorithm in $\tilde{O}(m)$ time that outputs a decomposition of V into $(s, \alpha\delta, \gamma)$-strong clusters such that the total weight of inter-cluster edges is at most $n\delta/50$.*

The first step is to apply the following lemma to a slightly modified graph.

▶ **Lemma 10** ([5]). *Given a weighted graph $G = (V, E, w)$ and a parameter δ_0 such that $\delta_0 \leq \min_{v \in V} \deg(v)$ and a parameter $s_0 \leq \delta_0 \text{polylog}(n)$, there is an algorithm that runs in $\tilde{O}(m)$ time and partitions the vertex set V into components $V_1, \ldots, V_k$ such that*

1. *For any cluster V_i and any cut $(S, \overline{S})$ in G of weight at most δ_0, we have $\min\{\mathbf{vol}(S \cap V_i), \mathbf{vol}(\overline{S} \cap V_i)\} \leq s_0$. Here, $\mathbf{vol}(U)$ is the sum of weighted degrees of vertices in U.*
2. *The total weight of inter-cluster edges is at most an $O(\frac{\sqrt{\delta_0}\log n}{\sqrt{s_0}})$ fraction of the total weight of edges.*

Construct the graph G_0 from G as follows. For each vertex $v \in V$, add a new vertex v' with an edge to v of weight $\alpha\delta$. This new graph has minimum weighted degree $\alpha\delta$. Apply the lemma above to G_0 with parameters $\delta_0 = \alpha\delta$ and $s_0 = s\alpha\delta$. The total weight of inter-cluster edges is at most $O(\frac{\log n}{\sqrt{s}}) \cdot \alpha\delta n \leq n\delta/100$ for large enough $s = O(\alpha^2 \log^2 n)$. Since G_0 has minimum degree δ_0, the guarantee $\min\{\mathbf{vol}(S \cap V_i), \mathbf{vol}(\overline{S} \cap V_i)\} \leq s_0$ from property 1 implies that $\min\{|S \cap V_i|, |\overline{S} \cap V_i|\} \leq s_0/\delta_0 = s$. In other words, each V_i is $(s, \alpha\delta, 0)$-strong in G_0. Consider the partition in G obtained by removing all new vertices v'. It is straightforward to see that this partition is also $(s, \alpha\delta, 0)$-strong in G, and the total weight of inter-cluster edges is still at most $n\delta/100$.

Algorithm 2 Cut-Game(G, T, δ, ψ).

```
Input  : Undirected weighted graph $G$, terminal set $T \subseteq V$, decomposition
         parameters $\delta > 0$ and $\psi < 1$.
Output : Either (1) a $(\psi \cdot \delta)$-terminal-sparse cut $(U, \overline{U})$ with
         $|U \cap T|, |\overline{U} \cap T| \geq |T|/6$, or
         (2) an $(\tilde{O}(\psi^{-5}), \delta, \tilde{\Omega}(\psi^5), T)$-terminal-strong cluster $U$ with
         $|U \cap T| \geq |T|/3$ such that $\overline{U}$ is either empty or $(\psi \cdot \delta)$-terminal-sparse
1  Initialize cut graph: $H = (T, \emptyset)$
2  Initialize unmatched terminal set: $S \leftarrow \emptyset$
3  Set parameters $L_{\max} = O(\log |T|)$, $\alpha = L_{\max}/\psi$, $s = O((L_{\max}/\psi)^2 \log^2 n)$, $\gamma = \frac{1}{200\alpha s}$
4  for $t \leftarrow 1$ to $L_{\max}$ do
       // Cut Player Strategy
5      Partition $T$ into clusters that are $(s, \alpha\delta, \gamma)$-strong in $H$. (Lemma 9)
6      if there exists cluster $C$ of size $\geq 2|T|/3$ then
           // Trim Cluster
7          $(f, (U, \overline{U})) \leftarrow$ CutOrFlow$(G, C, \min\{\gamma/2s, \gamma/6\})$
8          return larger side $U$ with cut $(U, \overline{U})$ if $\overline{U}$ is non-empty
9      Combine clusters to create a bipartition $(C, \overline{C})$ with $|C| \geq |\overline{C}| \geq |T|/3$.
        (Claim 12)
       // Matching Player Strategy
10     $(f, (U, \overline{U})) \leftarrow$ CutOrFlow$(G, C, \psi)$
11     if flow $f$ has value $\geq |T|/6 \cdot \delta \cdot \psi$ then
           // Large Flow
12         Decompose flow into (implicit) paths $f_1, f_2, ..., f_k$ where $k \leq m$ and each path
            connects exactly two terminals $(t_1, t_2)$, one from each bipartition
13         For each $i \in [k]$, add an edge $(t_1, t_2)$ in $H$ whose weight is $1/\psi$ times the
            capacity of path $f_i$ in $G$
14     else
           // Terminal-Balanced Cut
15         return terminal-balanced cut $(U, \overline{U})$
   // Game must terminate within $L_{\max}$ rounds
```

We now modify the partition so that each cluster is $(s, \alpha\delta, \gamma)$-strong by applying the lemma below to each V_i. The total weight of additional inter-cluster edges guaranteed by the lemma is at most $\sum_i |V_i|\delta/100 \leq n\delta/100$. Together with the inter-cluster edges from the first step, the total weight is at most $n\delta/12$. It remains to prove the lemma below:

▶ **Lemma 11.** *Let C be an $(s, \alpha\delta, 0)$-strong cluster in G and let $\gamma = \frac{1}{200\alpha s}$. There is an algorithm in $\tilde{O}(|E(G[C])|)$ time that partitions C into $(s, \alpha\delta, \gamma)$-strong clusters such that the total weight of inter-cluster edges is at most $|C|\delta/100$.*

This proof uses a similar technique found in [5], and we leave the proof for the full version of the paper [4].

Let the *size* of a cluster be the number of vertices in the cluster. The following lemma shows that Line 9 of Algorithm 2 can be executed efficiently.

▷ Claim 12. Suppose no single cluster has size greater than $2|T|/3$. Then, there exists a bipartition of clusters such that each group of clusters has a total size in the range $[|T|/3, 2|T|/3]$, and this bipartition can be computed in nearly linear time.

Proof. We can split our proof into two cases:

1. There exists a cluster with size $\in [|T|/3, 2|T|/3]$:
   We make that cluster its own group and all remaining clusters the second group.
2. All clusters have size less than $|T|/3$:
   Enumerate the clusters in an arbitrary order, and consider the shortest prefix of clusters whose total size exceeds $|T|/3$. The prefix without its last cluster has total size less than $|T|/3$, and this last cluster of the prefix has size less than $|T|/3$, so this prefix has size in $[|T|/3, 2|T|/3]$. ◁

4.2 Matching Player

We introduce a key subroutine of the matching player, called CutOrFlow, which is used to find matchings over partitions and for trimming.

Algorithm 3 CutOrFlow(G, S, κ).

```
   Input : Undirected weighted graph $G = (V, E)$ and a set of terminals $S \subseteq T$
 1 $G' \leftarrow G$
 2 Add sink node $t$ to $G'$ with edges from all terminals in $T \setminus S$ with capacity $\delta \cdot \kappa$. Add
     source node $s$ to $G'$ with edges to all terminals in $S$ with capacity $\delta \cdot \kappa$.
 3 Compute $s - t$ max-flow $f'$ and $s - t$ min-cut $(U', \overline{U'})$ in $G'$
 4 Let $f$ be the flow $f'$ with vertices $s, t$ removed from every path
 5 Let cut $(U, \overline{U})$ in $G$ be $(U' \setminus s, \overline{U'} \setminus t)$
 6 return flow $f$ and cut $(U, \overline{U})$
```

▶ **Lemma 13.** *If Algorithm 3 returns a valid cut $(U, \overline{U})$, then it is $\delta \cdot \kappa$-terminal sparse in G.*

Proof. Without loss of generality, assume that U contains at most as many terminals as $V \setminus U$. Consider the difference between the edges of $\partial U'$ and the edges of the cut $\partial(\{s\})$, which cuts all edges adjacent to s. The edges in $\partial U' \setminus \partial(\{s, t\})$ are precisely the edges of $\partial U'$ originally within G, and the edges in $\partial(\{s\}) \setminus \partial U'$ are precisely the edges between s and $U \cap T$. Since $\partial U'$ is an $s - t$ min-cut, we have $w(\partial U' \setminus \partial(\{s, t\})) \leq w(\partial(\{s\}) \setminus \partial U')$, which is equivalent to $w_G(\partial U) \leq |U \cap T| \cdot \delta \cdot \kappa$. A symmetric argument yields $w_G(\partial U) \leq |\overline{U} \cap T| \cdot \delta \cdot \kappa$, and combining the two proves the lemma. ◀

Lemma 13 proves the sparsity guarantees of Lemma 8, as the algorithm sets either sets $\kappa \leftarrow \psi$ or $\kappa \leftarrow \min\{\gamma/2s, \gamma/6\}) \ll \psi$ in every CutOrFlow call. Thus, every cut returned is always $\psi \cdot \delta$-terminal sparse in G.

▶ **Lemma 14.** *If $|S|, |T \setminus S| \geq |T|/3$ and flow f has value less than $|T|/6 \cdot \delta \cdot \kappa$, then the cut $(U, \overline{U})$ satisfies $|U|, |\overline{U}| \geq |T|/6$.*

Proof. In graph G', the value of flow f' and cut $(U', \overline{U'})$ are equal by flow-cut duality. In particular, cut $(U', \overline{U'})$ has weight less than $|T|/6 \cdot \delta \cdot \kappa$. Since s has edges to $S \cap \overline{U'}$ that cross the cut, and since t has edges from $(T \setminus S) \cap U'$ that cross the cut, we have $|S \cap \overline{U'}|, |(T \setminus S) \cap U'| \leq |T|/6$. Since $|S|, |T \setminus S| \geq |T|/3$, it follows that $|S \cap U'| = |S| - |S \cap \overline{U'}| \geq |T|/6$ and $|(T \setminus S) \cap \overline{U'}| = |T \setminus S| - |(T \setminus S) \cap U'| \geq |T|/6$. In particular, $|U|, |\overline{U}| \geq |T|/6$. ◀

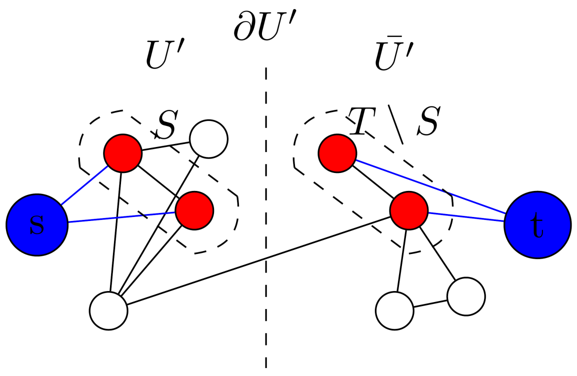

Figure 1 Construction of Algorithm 3. Blue vertices and edges are added to the original graph G and red vertices mark terminals.

4.2.1 Trimming

In Line 7 of Algorithm 2, we begin with a subset $C \subseteq T$ of size at least $2|T|/3$ such that V is (s, δ, γ, C)-terminal-strong in G. Our next goal is to find a cluster U that is a $(O(s/\gamma), \delta, \Omega(\gamma/s))$-terminal strong with $|U \cap T| \geq |T|/3$. This allows us to only recurse on $\overline{U}$, which satisfies $|\overline{U} \cap T| \leq 2|T|/3$, allowing for an efficient algorithm.

We used a modified form of the trimming method found in [13]. In their paper, the authors describe a simple "Slow Trimming" and an improved "Efficient Trimming" scheme, which is much more involved by circumventing the use of exact max-flow. However, the slow trimming scheme suffices for our purposes since we are fine with maximum flow time.

We begin with the following lemma, which we use to prove the correctness of trimming:

▶ **Lemma 15.** *If a cluster S is $(s, (L_{\max}/\psi)\delta, \gamma)$-strong in the cut graph H, then V is (s, δ, γ, S)-terminal-strong in G.*

Proof. By construction, each edge (u, v) of weight w in the cut graph H certifies the existence of a flow of capacity $1/\psi \cdot w$ in the original graph G. Since we run the cut-game for at most $L_{\max}$ rounds, we can simultaneously route flows between terminals u and v of weight $1/\psi \cdot w(u, v)$ for all edges $(u, v) \in E_H$ with capacities scaled by at most $L_{\max}$ in graph G. Equivalently, scaling everything by $1/\psi$, we can simultaneously route flows between terminals u and v of weight $w(u, v)$ for all edges $(u, v) \in E_H$ with capacities scaled by at most $L_{\max} \cdot 1/\psi$ in graph G.

We proceed with two cases. First, assume for contradiction that there exists a Steiner cut $(C, \overline{C})$ of graph G with at most weight δ which satisfies $\min\{|C \cap S|, |\overline{C} \cap S|\} > s$. Consider cut $(C \cap T, \overline{C} \cap T)$ in cut graph H. Since $C \cap S \subseteq C \cap T$ and $\overline{C} \cap S \subseteq \overline{C} \cap T$, we have

$$\min\{|(C \cap T) \cap S|, |(\overline{C} \cap T) \cap S|\} \geq \min\{|C \cap S|, |\overline{C} \cap S|\} > s. \tag{3}$$

The weight of cut $(C \cap T, \overline{C} \cap T)$ in H is at most a $L_{\max} \cdot 1/\psi$ factor greater than the amount of (scaled) flow able to be routed over cut $(C, \overline{C})$ in graph G. In other words, $w_H(C \cap T, \overline{C} \cap T) \leq (L_{\max} \cdot 1/\psi)\delta$. This contradicts the assumption that S is an $(s, (L_{\max}/\psi)\delta, \gamma)$-strong cluster in the cut graph.

For the second case, assume for contradiction that there exists a cut $(C, \overline{C})$ of graph G with at most weight δ which satisfies $w(\partial_G C) < \gamma \cdot \delta$ and $\min\{|C \cap S|, |\overline{C} \cap S|\} > 0$. Similar to above, the weight of cut $(C \cap T, \overline{C} \cap T)$ in H is at most a $L_{\max} \cdot 1/\psi$ factor larger than cut $(C, \overline{C})$ in graph G. Since $\min\{|C \cap S|, |\overline{C} \cap S|\} > 0$, $\partial_{H[S]} C$ is an actual cut of $H[S]$, and $\partial_{H[S]} C \leq \partial_H C < \gamma \cdot (L_{\max}/\psi)\delta$, contradicting the assumption that S is $(s, (L_{\max}/\psi)\delta, \gamma)$-strong in H. ◀

Now, we introduce the section's main theorem, which shows that the cluster $\overline{U}$ is terminal-strong and contains a large fraction of terminals.

▶ **Theorem 16.** *If V is (s, δ, γ, S)-terminal-strong in G and $|S| \geq 2|T|/3$, the cut $(U, \overline{U})$ returned by Algorithm 3 with parameter $\kappa = \min\{\gamma/(2s), \gamma/6\}$ satisfies the property that U is $(\max\{2/\kappa + s, 3s\}, \delta, \kappa, U \cap T)$-terminal strong in G and $|U \cap T| \geq |T|/3$.*

We prove this theorem in the full version of the paper [4].

4.2.2 Final Parameters

Finally, we plug in our parameters $L_{\max} = O(\log |T|)$, $\alpha = L_{\max}/\psi$, $s = O((L_{\max}/\psi)^2 \log^2 n) = O(\log^4 n/\psi^2)$, and $\gamma = \frac{1}{200\alpha s} = \Omega(\psi^3/\log^5 n)$ in Algorithm 2. We have $\kappa = \Omega(\gamma/s) = \Omega(\psi^5/\log^9 n)$, so the $(\max\{2/\kappa + s, 3s\}, \delta, \kappa, U \cap T)$-terminal strong cluster U output by Algorithm 2 is $(O(\log^9 n/\psi^5), \delta, \Omega(\psi^5/\log^9 n))$-terminal-strong, fulfilling the output guarantee of Algorithm 2.

4.3 Termination

To show that the cut-matching game terminates within $L_{\max}$ rounds, we introduce the following guarantee of the cut-matching game analysis.

▶ **Lemma 17.** *For large enough $L_{\max} = O(\log |T|)$, Algorithm 2 proceeds for at most $L_{\max}$ iterations.*

The proof is a direct adaptation of the cut-matching game analysis of [8, 9], and thus we leave the details to the full version [4].

4.4 Terminal Decomposition

Finally, we introduce the complete algorithm for terminal decomposition, which uses the cut-matching game algorithm as a key subroutine.

Algorithm 4 TERMINAL-DECOMP(C, T, δ, ψ).

```
   Input  : Cluster $C \subseteq V$, terminal set $T \subseteq V$, decomposition parameters $\delta, \psi$
1  Run CUT-GAME$(G[C], T, \delta, \psi)$
2  if $G[C]$ is certified as a terminal-strong cluster then
3      return $G[C]$
4  else if Cut-Game returns balanced cut $(U, \overline{U})$ with terminal-sparsity $\psi \cdot \delta$ then
5      return
        TERMINAL-DECOMP$(U, U \cap T, \delta, \psi)$ ∪ TERMINAL-DECOMP$(\overline{U}, \overline{U} \cap T, \delta, \psi)$
6  else
       // Larger side $G[U]$ is certified as an $(s, \delta, \gamma)$-terminal-strong
          cluster
7      return $G[U]$ ∪ TERMINAL-DECOMP$(\overline{U}, \overline{U} \cap T, \delta, \psi)$
```

The algorithm uses CUT-GAME as a subroutine and Lemma 8 as its guarantee. First, we note that since we recurse on both sides of a cut in Line 5 only if they both have at least $\Omega(1)$ terminals (from Lemma 8), we have at most $O(\log n)$ recursive levels. We now prove the formal theorems for our terminal decomposition.

▶ **Theorem 18.** *Algorithm 4 runs with $O(\log^2 n)$ max-flows and $\tilde{O}(m)$ additional time.*

Proof. First, in each recursive level, each TERMINAL-DECOMP call is on a mutually disjoint portion of the graph. Therefore, all maximum flows on a single recursive level can be done in parallel with a single maximum flow call on a graph of size $O(m)$ edges. Additionally, each round of the cut-matching game uses at most a single max-flow call (in CUTORFLOW). With a total of $L_{\max} = O(\log n)$ cut-matching game rounds and $O(\log n)$ recursive levels, the entire terminal decomposition runs in $O(\log^2 n)$ max-flows.

All other cut-matching game procedures (specifically the (s, δ, γ)-strong decomposition of Lemma 9) run in near-linear time. With $O(\log n)$ recursive levels, the entire algorithm runs in near-linear time, excluding max-flows. ◀

▶ **Theorem 19.** *Algorithm 4 returns a $(\tilde{O}(1/\psi^5), \delta, \tilde{\Omega}(\psi^5), T)$-terminal-strong decomposition of G.*

Proof. We begin by proving the upper-bound on intercluster edges:

▶ **Lemma 20.** *The total weight of intercluster edges from the decomposition outputted by Algorithm 4 is at most $O(\psi \cdot \delta \cdot |T| \log |T|)$.*

Proof. Every cut made by Algorithm 4 is $\psi \cdot \delta$ terminal-sparse due to Lemma 13. We can charge $\psi \cdot \delta$ weight to each terminal on the smaller side of the cut. Since there are at most $\log |T|$ recursive levels and each cluster gets only one cut per recursive level, each terminal gets charged at most $\log |T|$ times. Summing up the weights charged to each terminal gives us a total edge weight of $O(\psi \cdot \delta \cdot |T| \log |T|)$. ◀

From Lemma 8, every cluster returned is certified to be $(O(\log^9 n/\psi^5), \delta, \Omega(\psi^5/\log^9 n), T)$-terminal strong, completing the proof. ◀

5 Minimum Steiner Cut Using Sparsification

We complete our algorithm by showing a polylogarithmic maximum flow algorithm for minimum Steiner cut on a graph by using its terminal-strong decomposition. We use the minimum isolating cuts method and terminology described by [11]. The critical difference is that we use a terminal-strong decomposition instead of an expander decomposition. However, we prove that the same guarantees apply.

We begin by introducing some definitions from [11].

▶ **Definition 21** (k-unbalanced, k-balanced)**.** *A subset of vertices $U \subseteq V$ is considered k-unbalanced if there exists a minimum Steiner cut S such that $\min\{|S \cap U|, |\bar{S} \cap U|\} \leq k$. U is considered k-balanced with witness $(S, \bar{S})$ if there exists a minimum Steiner cut S such that $\min\{|S \cap U|, |\bar{S} \cap U|\} > k$.*

Our main result of the section is as follows:

▶ **Theorem 22.** *There exists a deterministic algorithm which given an undirected weighted graph $G = (V, E)$, an (s, δ, γ, T)-terminal-strong decomposition $G' = \{V_1, V_2, ..., V_\ell\}$, a parameter $k \leftarrow C \log^C n$ for some large enough constant $C > 0$, and a subset of terminals $U \subseteq T$, does the following:*

1. *If U is k-unbalanced, we return the minimum Steiner cut of G with polylogarithmic maximum flow calls and near-linear additional runtime.*
2. *If U is k-balanced with witness (S_1, S_2), we return a subset $U' \subset U$ such that $|U'| \leq |U|/2$ and $S_i \cap U' \neq \emptyset$ for both $i = 1, 2$.*

We leave the full proof details to the full paper [4].

After computing the sparsified set $U \leftarrow U'$ in the balanced case, we can recursively run our minimum Steiner cut algorithm on graph G and terminal set $T \leftarrow U$. Since the size of U at least halves each time we sparsify, this only needs to be done at most $\log n$ times.

Also, since we never know which case we are specifically in (balanced or unbalanced) in Theorem 22, we run both cases until U must be guaranteed to be k-unbalanced. At this point we take the minimum over all Steiner cuts found, and we are guaranteed to have found a minimum one (see Algorithm 1).

6 Conclusion

Our algorithm solves deterministic minimum Steiner cut with polylogarithmic max flow calls and near-linear additional processing time. We thus show minimum Steiner cut reduces to maximum flow up to polylogarithmic factors in runtime. Specifically, the existence of a deterministic near-linear time $s-t$ max-flow algorithm would imply a deterministic near-linear time algorithm for minimum Steiner cut.

Our main contribution is the (s, δ, γ)-terminal-strong decomposition. We are able to do this deterministically in polylogarithmic max flows and near-linear additional time for small δ, which is not yet known for standard expander decompositions. We also believe that (s, δ, γ)-strong and terminal-strong decompositions may have additional future applications in faster algorithms for graph problems.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M.R. Reddy. Chapter 1 applications of network optimization. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 1–83. Elsevier, 1995.

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

3 Alexander F. Brust, Eric J. Payton, Toren J. Hobbs, and Stephen R. Niezgoda. Application of the maximum flow–minimum cut algorithm to segmentation and clustering of materials datasets. *Microscopy and Microanalysis*, 25(4):924–941, 2019. `doi:10.1017/S1431927619014569`.

4 Matthew Ding and Jason Li. Deterministic minimum steiner cut in maximum flow time, 2023. `arXiv:2312.16415`.

5 Monika Henzinger, Jason Li, Satish Rao, and Di Wang. Deterministic near-linear time minimum cut in weighted graphs. In *2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3089–3139, 2024. `doi:10.1137/1.9781611977912.111`.

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

7 Ken-Ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1), December 2018. `doi:10.1145/3274663`.

8 Rohit Khandekar, Subhash A. Khot, Lorenzo Orecchia, and Nisheeth K. Vishnoi. On a cut-matching game for the sparsest cut problem. Technical Report UCB/EECS-2007-177, EECS Department, University of California, Berkeley, December 2007. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-177.html`.

9 Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4), July 2009. `doi:10.1145/1538902.1538903`.

10 Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 384–395, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451114.

11 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, 2020. doi:10.1109/FOCS46700.2020.00017.

12 Jason Li and Thatchaphol Saranurak. Deterministic weighted expander decomposition in almost-linear time, 2021. arXiv:2106.01567.

13 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler, 2021. arXiv:1812.08958.