Deterministic Minimum Steiner Cut in Maximum Flow Time

Matthew Ding[1] Jason Li[2]

[1]Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

[2]School of Computer Science
Carnegie Mellon University

European Symposium on Algorithms
September 2024

Motivation: Fast Graph Algorithms

Recent breakthroughs for fundamental deterministic graph problems:

Motivation: Fast Graph Algorithms

Recent breakthroughs for fundamental deterministic graph problems:

- (Global) Minimum cut: $m^{1+o(1)}$, "almost-linear" (Li, STOC 2021)
- $(s-t)$ Maximum flow: $m^{1+o(1)}$, "almost-linear" (vdBCKLPGSS, FOCS 2023)

Motivation: Fast Graph Algorithms

Recent breakthroughs for fundamental deterministic graph problems:

- (Global) Minimum cut: $m^{1+o(1)}$, “almost-linear” (Li, STOC 2021)
- $(s-t)$ Maximum flow: $m^{1+o(1)}$, “almost-linear” (vdBCKLPGSS, FOCS 2023)
- (Global) Minimum cut: $m \cdot \text{polylog}(n)$, “near-linear” (HLRW, SODA 2024)

Motivation: Fast Graph Algorithms

Recent breakthroughs for fundamental deterministic graph problems:

- (Global) Minimum cut: $m^{1+o(1)}$, "almost-linear" (Li, STOC 2021)
- $(s-t)$ Maximum flow: $m^{1+o(1)}$, "almost-linear" (vdBCKLPGSS, FOCS 2023)
- (Global) Minimum cut: $m \cdot \text{polylog}(n)$, **"near-linear"** (HLRW, SODA 2024)

Ultimate goal: near-linear time algorithms!

Problem Statement

Minimum Steiner Cut Problem

Given undirected, weighted graph $G(V, E)$ and a subset of vertices $T \subseteq V$ (called terminals), find the subset of edges of minimum total weight disconnecting *any* pair of vertices in T.

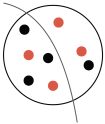

Figure: Minimum Steiner Cut. Terminals in red.

Problem Statement

Minimum Steiner Cut Problem

Given undirected, weighted graph $G(V, E)$ and a subset of vertices $T \subseteq V$ (called terminals), find the subset of edges of minimum total weight disconnecting *any* pair of vertices in T.

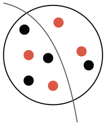

Figure: Minimum Steiner Cut. Terminals in red.

Generalizes two fundamental graph problems:

Problem Statement

Minimum Steiner Cut Problem

Given undirected, weighted graph $G(V, E)$ and a subset of vertices $T \subseteq V$ (called terminals), find the subset of edges of minimum total weight disconnecting *any* pair of vertices in T.

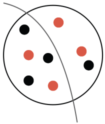

Figure: Minimum Steiner Cut. Terminals in red.

Generalizes two fundamental graph problems:

- Global min-cut: $T = V$

Problem Statement

Minimum Steiner Cut Problem

Given undirected, weighted graph $G(V, E)$ and a subset of vertices $T \subseteq V$ (called terminals), find the subset of edges of minimum total weight disconnecting *any* pair of vertices in T.

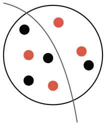

Figure: Minimum Steiner Cut. Terminals in red.

Generalizes two fundamental graph problems:

- Global min-cut: $T = V$
- $s - t$ min-cut: $T = \{s, t\}$

Problem Statement

Minimum Steiner Cut Problem

Given undirected, weighted graph $G(V, E)$ and a subset of vertices $T \subseteq V$ (called terminals), find the subset of edges of minimum total weight disconnecting *any* pair of vertices in T.

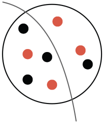

Figure: Minimum Steiner Cut. Terminals in red.

Generalizes two fundamental graph problems:

- Global min-cut: $T = V$
- $s - t$ min-cut: $T = \{s, t\}$

Deterministic near-linear minimum Steiner cut?

Previous Work: Minimum Steiner Cut

Folklore: $n - 1$ $s - t$ min-cuts (calls to any blackbox max-flow algorithm).

Previous Work: Minimum Steiner Cut

Folklore: $n-1$ $s-t$ min-cuts (calls to any blackbox max-flow algorithm).

Previous Result: Li, Panigrahi (FOCS 2020)

Minimum Steiner cut algorithms using only *polylogarithmic* maximum flow calls.

Previous Work: Minimum Steiner Cut

Folklore: $n-1$ $s-t$ min-cuts (calls to any blackbox max-flow algorithm).

Previous Result: Li, Panigrahi (FOCS 2020)

Minimum Steiner cut algorithms using only *polylogarithmic* maximum flow calls.

- Randomized: $\log^{O(1)} n$ maximum flows, $\tilde{O}(m)$ overhead

Previous Work: Minimum Steiner Cut

Folklore: $n-1$ $s-t$ min-cuts (calls to any blackbox max-flow algorithm).

Previous Result: Li, Panigrahi (FOCS 2020)

Minimum Steiner cut algorithms using only *polylogarithmic* maximum flow calls.

- Randomized: $\log^{O(1)} n$ maximum flows, $\tilde{O}(m)$ overhead
- Deterministic: $\log^{O(1/\epsilon^4)} n$ maximum flows, $O(m^{1+\epsilon})$ overhead

Previous Work: Minimum Steiner Cut

Folklore: $n-1$ $s-t$ min-cuts (calls to any blackbox max-flow algorithm).

Previous Result: Li, Panigrahi (FOCS 2020)

Minimum Steiner cut algorithms using only *polylogarithmic* maximum flow calls.

- Randomized: $\log^{O(1)} n$ maximum flows, $\tilde{O}(m)$ overhead
- Deterministic: $\log^{O(1/\epsilon^4)} n$ maximum flows, $O(m^{1+\epsilon})$ overhead

Deterministic polylogarithmic maximum flows requires $m^{1+O(1)}$ overhead!

Previous Work: Minimum Steiner Cut

Folklore: $n-1$ $s-t$ min-cuts (calls to any blackbox max-flow algorithm).

Previous Result: Li, Panigrahi (FOCS 2020)

Minimum Steiner cut algorithms using only *polylogarithmic* maximum flow calls.

- Randomized: $\log^{O(1)} n$ maximum flows, $\tilde{O}(m)$ overhead
- Deterministic: $\log^{O(1/\epsilon^4)} n$ maximum flows, $O(m^{1+\epsilon})$ overhead

Deterministic polylogarithmic maximum flows requires $m^{1+O(1)}$ overhead!
Ideally we would like near-linear max-flow and **near-linear overhead.**

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

- Expander decomposition: separates graph into expander clusters with low intercluster edge weights between them

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

- Expander decomposition: separates graph into expander clusters with low intercluster edge weights between them

Key properties:

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

- Expander decomposition: separates graph into expander clusters with low intercluster edge weights between them

Key properties:

1. Minimum Steiner cut intersects small number of clusters

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

- Expander decomposition: separates graph into expander clusters with low intercluster edge weights between them

Key properties:

1. Minimum Steiner cut intersects small number of clusters
2. Any Steiner cut which intersects clusters cuts them "unbalanced"

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

- Expander decomposition: separates graph into expander clusters with low intercluster edge weights between them

Key properties:

1. Minimum Steiner cut intersects small number of clusters
2. Any Steiner cut which intersects clusters cuts them "unbalanced"

Bottleneck!

Previous Work: Minimum Steiner Cut

Deterministic algorithm: derandomization using expander decomposition

- Expander decomposition: separates graph into expander clusters with low intercluster edge weights between them

Key properties:

1. Minimum Steiner cut intersects small number of clusters
2. Any Steiner cut which intersects clusters cuts them "unbalanced"

Bottleneck! Necessary?

Our Work: Main Result

Answer: **No!**

Our Work: Main Result

Answer: **No!**

Theorem 1: Faster Deterministic Minimum Steiner Cut

There exists a deterministic minimum Steiner cut algorithm with polylogarithmic maximum flow calls and *near-linear* overhead

Our Work: Main Result

Answer: **No!**

Theorem 1: Faster Deterministic Minimum Steiner Cut

There exists a deterministic minimum Steiner cut algorithm with polylogarithmic maximum flow calls and *near-linear* overhead

1. Deterministic near-linear maximum flow algorithm is the *only* barrier towards deterministic near-linear minimum Steiner cut.

Our Work: Main Result

Answer: **No!**

Theorem 1: Faster Deterministic Minimum Steiner Cut

There exists a deterministic minimum Steiner cut algorithm with polylogarithmic maximum flow calls and *near-linear* overhead

1. Deterministic near-linear maximum flow algorithm is the *only* barrier towards deterministic near-linear minimum Steiner cut.
2. Runtime improvement by forgoing expander decomposition

s-strong Partitions

Definition: s-strong cluster (informal)

A cluster C is called s-strong if any cut of size $< \delta$ splits the cluster with at most s volume on one side.

s-strong Partitions

Definition: s-strong cluster (informal)

A cluster C is called s-strong if any cut of size $< \delta$ splits the cluster with at most s volume on one side.

- Relaxation of expander decomposition

s-strong Partitions

Definition: s-strong cluster (informal)

A cluster C is called s-strong if any cut of size $< \delta$ splits the cluster with at most s volume on one side.

- Relaxation of expander decomposition
- Used to construct deterministic near-linear time global min-cut algorithms

s-strong Partitions

Definition: s-strong cluster (informal)

A cluster C is called s-strong if any cut of size $< \delta$ splits the cluster with at most s volume on one side.

- Relaxation of expander decomposition
- Used to construct deterministic near-linear time global min-cut algorithms
  1. Unweighted: Kawabarayashi, Thorup (J. ACM 2018)

s-strong Partitions

Definition: s-strong cluster (informal)

A cluster C is called s-strong if any cut of size $< \delta$ splits the cluster with at most s volume on one side.

- Relaxation of expander decomposition
- Used to construct deterministic near-linear time global min-cut algorithms
  1. Unweighted: Kawabarayashi, Thorup (J. ACM 2018)
  2. Weighted: HLRW (SODA 2024)

Our Results: Terminal-Based Partitioning

We want a “terminal-versions” of previous methods!

Our Results: Terminal-Based Partitioning

We want a "terminal-versions" of previous methods!

Definition: s-terminal-strong cluster (informal)

A cluster C is called s-terminal-strong if any cut of size $< \delta$ splits the cluster with at most s terminals on one side.

Our Results: Terminal-Based Partitioning

We want a "terminal-versions" of previous methods!

Definition: s-terminal-strong cluster (informal)

A cluster C is called s-terminal-strong if any cut of size $< \delta$ splits the cluster with at most s terminals on one side.

Definition: ψ-terminal-sparsity

A cut S is called ψ-terminal-sparse if

$$\frac{w(S, \bar{S})}{\min(|S \cap T|, |\bar{S} \cap T|)} < \psi \tag{1}$$

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

1. Any cut of size $< \delta$ splits all clusters with at most polylog(n) terminals on one side

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

1. Any cut of size $< \delta$ splits all clusters with at most polylog(n) terminals on one side
2. The total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

1. Any cut of size $< \delta$ splits all clusters with at most polylog(n) terminals on one side
2. The total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$
3. For any cluster, any cut of size $< \delta$ either does not split the cluster or splits edges within the cluster with total weight $> \delta/\text{polylog}(n)$

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

1. Any cut of size $< \delta$ splits all clusters with at most polylog(n) terminals on one side
2. The total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$
3. For any cluster, any cut of size $< \delta$ either does not split the cluster or splits edges within the cluster with total weight $> \delta/\text{polylog}(n)$

Uses $\log^2 n$ maximum flows and near-linear overhead.

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

1. **Any cut of size $< \delta$ splits all clusters with at most polylog(n) terminals on one side**
2. **The total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$**
3. For any cluster, any cut of size $< \delta$ either does not split the cluster or splits edges within the cluster with total weight $> \delta/\text{polylog}(n)$

Uses $\log^2 n$ maximum flows and near-linear overhead.

- (1) and (2) directly generalize s-strong partitions to terminals

Our Results: Terminal-Based Partitioning

Theorem 2: Deterministic Terminal-Strong Partition

Algorithm partitioning weighted graph into clusters such that

1. **Any cut of size $< \delta$ splits all clusters with at most polylog(n) terminals on one side**
2. The total weight of edges between clusters is at most $\tilde{O}(\delta \cdot |T|)$
3. **For any cluster, any cut of size $< \delta$ either does not split the cluster or splits edges within the cluster with total weight $> \delta/\mathbf{polylog}(n)$**

Uses $\log^2 n$ maximum flows and near-linear overhead.

- (1) and (2) directly generalize s-strong partitions to terminals
- (3) ensures few clusters are split, (1) ensures clusters are split unbalanced!

Review: Cut-Matching Game

- Khandekar, Rao, Vazirani (J. ACM 2009)

Review: Cut-Matching Game

- Khandekar, Rao, Vazirani (J. ACM 2009)
- Goal: Find sparse cut or certify none exist

Review: Cut-Matching Game

- Khandekar, Rao, Vazirani (J. ACM 2009)
- Goal: Find sparse cut or certify none exist
- Key idea: Two players, iteratively embed low-congestion expander into graph $G(V, E)$

Review: Cut-Matching Game

- Khandekar, Rao, Vazirani (J. ACM 2009)
- Goal: Find sparse cut or certify none exist
- Key idea: Two players, iteratively embed low-congestion expander into graph $G(V, E)$
- Begin by instantiating an empty graph H with vertices V, called the "cut graph"

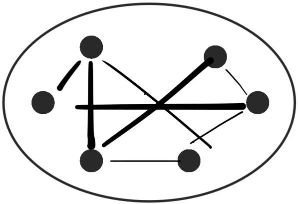

Figure: Graph G (black)

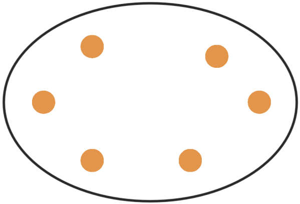

Figure: Cut Graph H (orange)

Review: Cut-Matching Game

Cut-Matching Game Iteration

Review: Cut-Matching Game

Cut-Matching Game Iteration

1. **Cut player finds bipartition of cut graph H containing sparse cut**

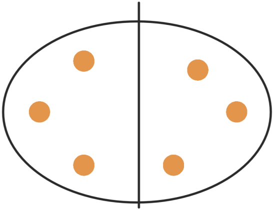

Figure: Cut Player: Bipartition

Review: Cut-Matching Game

Cut-Matching Game Iteration

1. Cut player finds bipartition of cut graph H containing sparse cut
2. **Matching player computes maximum flow between bipartition on original graph G and embeds the flow as a matching in H**
    - If flow cannot be successfully routed, we get a sparse cut

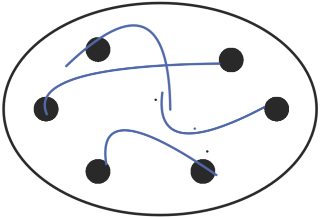

Figure: Matching Player: Flow

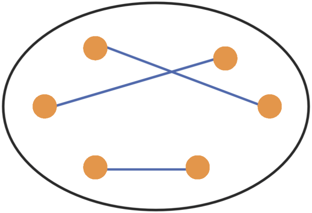

Figure: Matching Player: Matching

Review: Cut-Matching Game

Cut-Matching Game Iteration

1. Cut player finds bipartition of cut graph H containing sparse cut
2. Matching player computes maximum flow between bipartition on original graph G and embeds the flow as a matching in H
   - If flow cannot be successfully routed, we get a sparse cut
3. **After $\log^2 n$ rounds, G is guaranteed to be an edge-expander.**

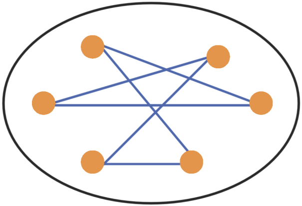

Figure: Certify Expander!

Terminal Cut-Matching Game

- Goal: Find terminal-sparse cut or certify none exist

Terminal Cut-Matching Game

- Goal: Find terminal-sparse cut or certify none exist
- Key idea: Embed low-congestion s-strong graph on terminals

Terminal Cut-Matching Game

- Goal: Find terminal-sparse cut or certify none exist
- Key idea: Embed low-congestion s-strong graph on terminals
- Begin by instantiating an empty cut-graph of terminals T, instead of all vertices

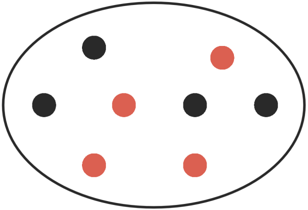

Figure: Graph G (black), terminals (red)

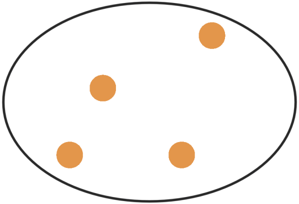

Figure: Cut Graph H (orange)

Terminal Cut-Matching Game

Terminal Cut-Matching Game Iteration

Terminal Cut-Matching Game

Terminal Cut-Matching Game Iteration

1. **Cut player finds bipartition of cut graph H containing sparse cut.**

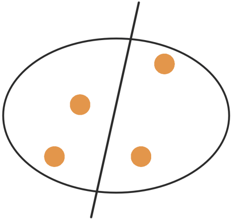

Figure: Cut Player: Bipartition

Terminal Cut-Matching Game

Terminal Cut-Matching Game Iteration

1. Cut player finds bipartition of cut graph H containing sparse cut.
2. **Matching player computes maximum flow between terminal partition on original graph G and embeds flow as matching edges in cut graph.**

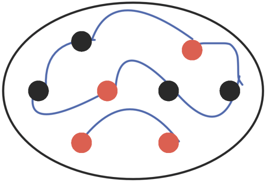

Figure: Matching Player: Flow

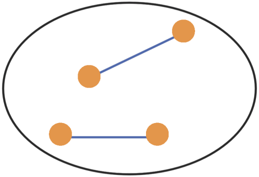

Figure: Matching Player: Matching

Terminal Cut-Matching Game

Terminal Cut-Matching Game Iteration

1. Cut player finds bipartition of cut graph H containing sparse cut.
2. Matching player computes maximum flow between terminal partition on original graph G and embeds flow as matching edges in cut graph.
3. **After** $\log n$ **rounds, we certify** G **as a terminal-strong cluster.**

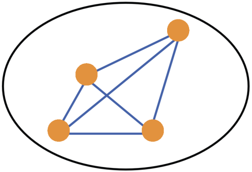

Figure: Certify Cut-Graph s-strong

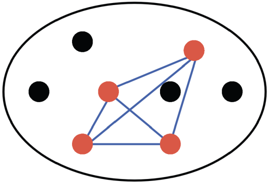

Figure: Certify Graph s-terminal-strong!

Terminal Cut-Matching Game: No Flow?

- What happens when we can't make a complete flow?

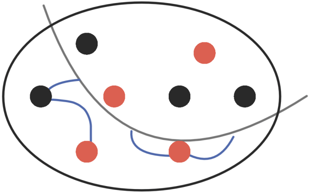

Figure: No Flow?

Terminal Cut-Matching Game: No Flow?

- What happens when we can't make a complete flow?

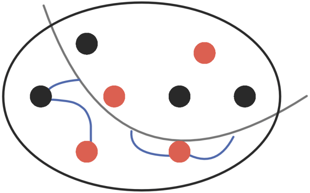

Figure: No Flow?

- Original cut-matching game: sparse cut

Terminal Cut-Matching Game: No Flow?

- What happens when we can't make a complete flow?

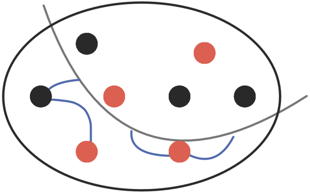

Figure: No Flow?

- Original cut-matching game: sparse cut
- Terminal cut-matching game: terminal-sparse cut, since terminals are sources/sinks

Terminal Cut-Matching Game: No Flow?

Two cases to consider:

Terminal Cut-Matching Game: No Flow?

Two cases to consider:

1. Balanced Cut: Recurse on both sides

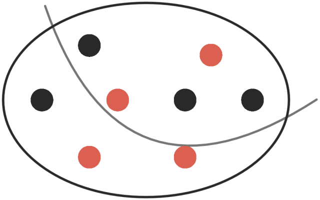

Figure: Small flow, balanced cut

Terminal Cut-Matching Game: No Flow?

Two cases to consider:

1. Balanced Cut: Recurse on both sides
2. Unbalanced Cut: Larger side is terminal-strong, recurse on smaller side

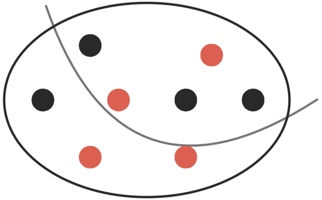

Figure: Small flow, balanced cut

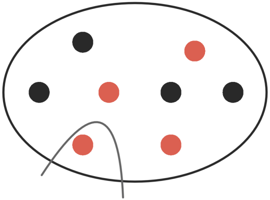

Figure: Small flow, unbalanced cut

Conclusion

Main takeaway:

Conclusion

Main takeaway:

- **We reduce the problem of finding terminal-sparse cuts in a graph into finding sparse cuts in the cut-graph**

Conclusion

Main takeaway:

- **We reduce the problem of finding terminal-sparse cuts in a graph into finding sparse cuts in the cut-graph**
- Allows us to deal with terminals for minimum Steiner cut

Conclusion

Main takeaway:

- **We reduce the problem of finding terminal-sparse cuts in a graph into finding sparse cuts in the cut-graph**
- Allows us to deal with terminals for minimum Steiner cut

Future Directions:

Conclusion

Main takeaway:

- **We reduce the problem of finding terminal-sparse cuts in a graph into finding sparse cuts in the cut-graph**
- Allows us to deal with terminals for minimum Steiner cut

Future Directions:

1. Near-linear maximum flow algorithm (randomized or deterministic)

Conclusion

Main takeaway:

- **We reduce the problem of finding terminal-sparse cuts in a graph into finding sparse cuts in the cut-graph**
- Allows us to deal with terminals for minimum Steiner cut

Future Directions:

1. Near-linear maximum flow algorithm (randomized or deterministic)
2. Terminal-based partitioning algorithm as subroutine?

Thanks!

Questions?