Faster All-Pairs Minimum Cut: Bypassing Exact Max-Flow

Robert Krauthgamer

Weizmann Institute of Science

Joint work with Yotam Kenneth-Mordoch (mostly in STOC'26, plus ESA'25 and SODA'26)

STOC workshop, June 2026

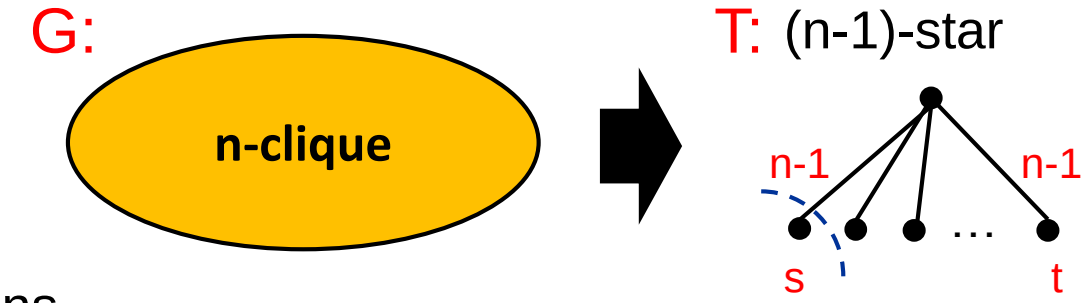
All-Pairs Minimum Cut (APMC)

- Classic problem:
  - Given graph $G = (V, E, w)$, find a minimum s,t-cut for every pair $s, t \in V$
- Classic solution: Gomory-Hu Tree (1961)
  - Theorem: Every graph G has a tree T on same vertex set V, and with same minimum s–t cut values, i.e.,

    $\forall s, t \in V, \quad \text{mincut}_T(s,t) = \text{mincut}_G(s,t).$

    n^2 constraints
  - Example:

    $n = |V|$



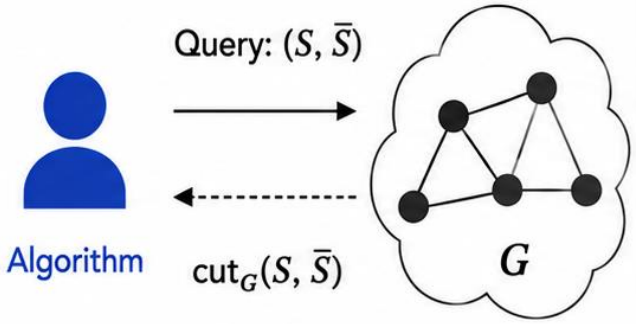
  - GH algorithm: uses $n - 1$ max-flow computations
  - Recent fast algorithms: use $\text{polylog}(n)$ max-flows [AKT20, AKT21a, LPS21, AKT21b, AKT22, AKL+22, AKL+25,…]

Take-away message: can reduce APMC ⟶ few exact max-flows

Bypass Max-Flow?

It's not easy to compute in some models (even for unweighted graphs):

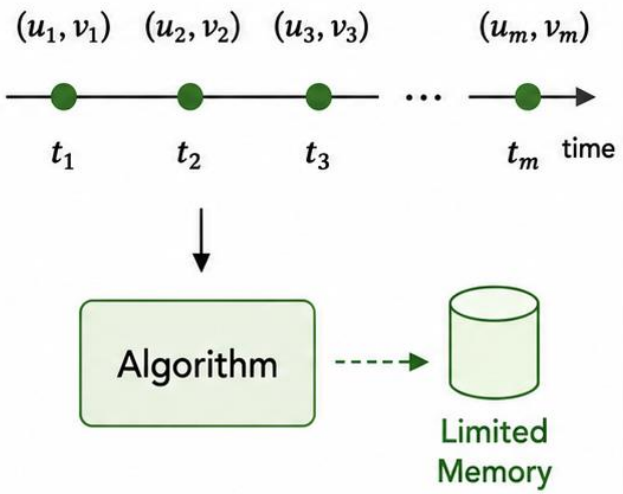
1. Cut-Query Model



- The algorithm has query access to the cuts of an unknown graph G.
- It can only ask for the value of cuts.

2. Streaming Model

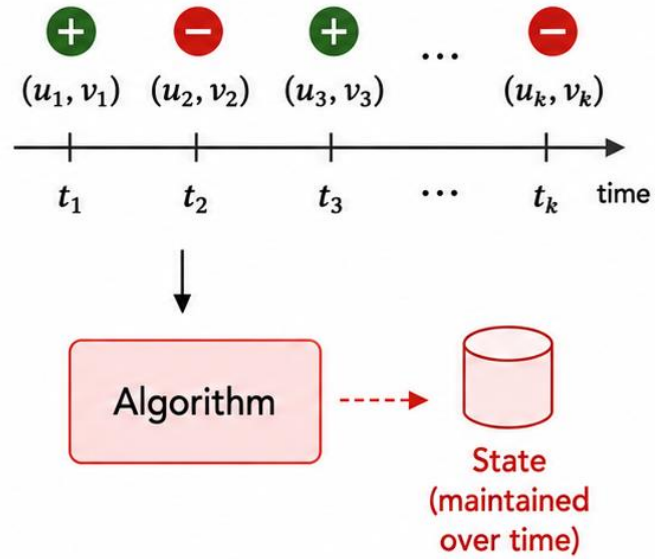
Edge stream (arrives one by one)



- Edges arrive one by one.
- The algorithm sees each edge once and uses limited memory.

3. Fully Dynamic Model

Update stream (over time)



- Edges can be inserted or deleted over time.
- The algorithm must maintain its state and be ready to answer at all times.

Bypass Max-Flow?

It's not easy to compute in some models (even for unweighted graphs):

- **Cut-query model**
  - Access G via queries for value of $\text{cut}_G(S)$ for $S \subseteq V$
  - Complexity: number of cut queries (and rounds/adaptivity)
- **Streaming model**
  - Edges of G arrive as an insertion-only stream
  - Complexity: storage (and number of passes)
- **Fully dynamic model**
  - G undergoes updates (edge insertions and deletions)
  - Complexity: worst-case update (and query) time

queries:

$\text{cut}_G(S_1)$, $\text{cut}_G(S_2)$, …

stream:

e_1, e_2, …

updates:

$+e_1$, $+e_2$, $-e_1$, …

But wait – APMC is only stronger than single-pair max-flow!!

A New Sparsifier

- Idea: Preserve every minimum s,t-cut in G=(V,E) but allow Steiner vertices
  - Weaker than Gomory-Hu tree
  - Example: For n-clique, the AMPC sparsifier can be a star with n leaves

- Formally: Weighted graph $H=(U,E_H,w)$ is APMC sparsifier of G if $U \supseteq V$ and

  $\forall s,t \in V,$ if S_H is min s,t-cut in H, then $S_H \cap V$ is min s,t-cut in G

- Key features:
  - Can solve APMC on H (e.g., build Gomory–Hu tree)
  - Can build small H efficiently (e.g., using a few approximate max-flow computations)
    - From here on, consider only unweighted inputs G

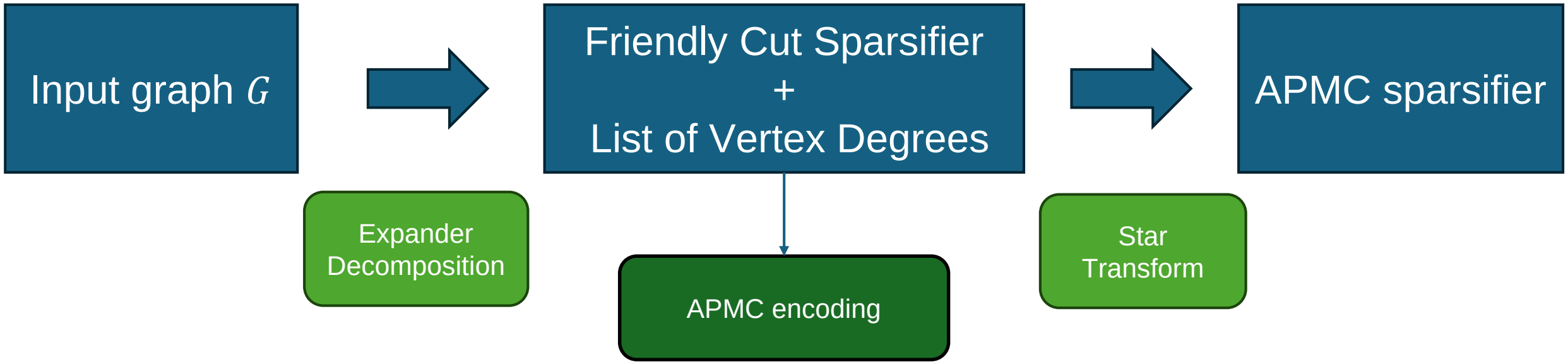
Our Results

- **Main Result:** "Efficient" algorithm (using approx. max-flows) that constructs for unweighted input G an APMC sparsifier with $\tilde{O}(n^{3/2})$ edges

Computational Model	Complexity Measure	Our APMC	Existing APMC	Existing Max-Flow
Cut query	Queries	$\tilde{O}(n^{3/2})$	$\tilde{O}(n^{7/4})$ [KK26]	$\tilde{O}(n^{8/5})$ [JNS26]
Fully dynamic	Worst-case update time	$n^{3/2+o(1)}$	$n^{2+o(1)}$	$n^{2+o(1)}$
Dynamic streaming	Storage, two passes	$\tilde{O}(n^{3/2})$	$\tilde{O}(n^2)$	$\tilde{O}(n^{5/3})$ [RSW18]
Two-party communication	bits	$\tilde{O}(n^{3/2})$	$\tilde{O}(n^2)$	$\tilde{O}(n^{3/2})$ [GJ25]

- Improve even single-pair max-flow in these models!

Overview



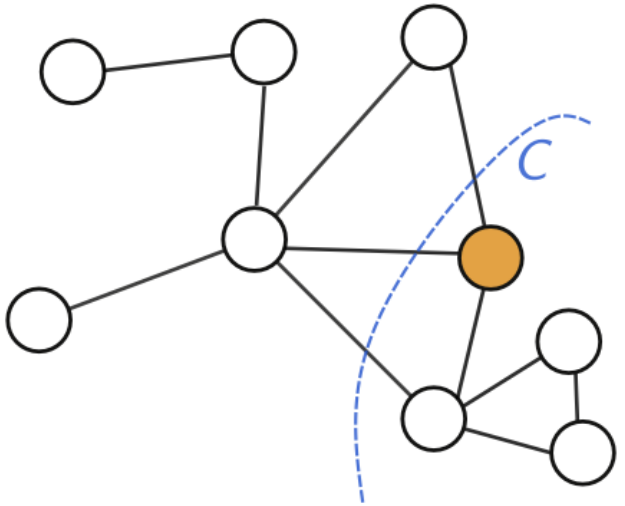
- The “magic”: expander decomposition finds dense clusters, but instead of contracting each one, we transform it into a star graph

Friendly Cuts

- Given a cut $C \subseteq V$, the cross degree of vertex v relative to C

$$\mathrm{cr}_C(v) = |E(v, V \setminus v) \cap E(C, V \setminus C)|$$

- Friendliness ratio
  - of vertex v is $\mathrm{fr}(v) = 1 - \frac{\mathrm{cr}_C(v)}{\deg(v)}$
  - of cut C is $\mathrm{fr}(C) = \min_{v \in V} \mathrm{fr}(v)$



$1 - \frac{\mathrm{cr}_C(\bigcirc)}{\deg(\bigcirc)} = 1/3$

$\Rightarrow C$ is a $\frac{1}{3}$-friendly cut

Friendly Cut Sparsifiers

- **Definition [AKT'21]:** H is (α, w)-friendly cut sparsifier of G if it has the same vertex set and for every α-friendly cut S in G,

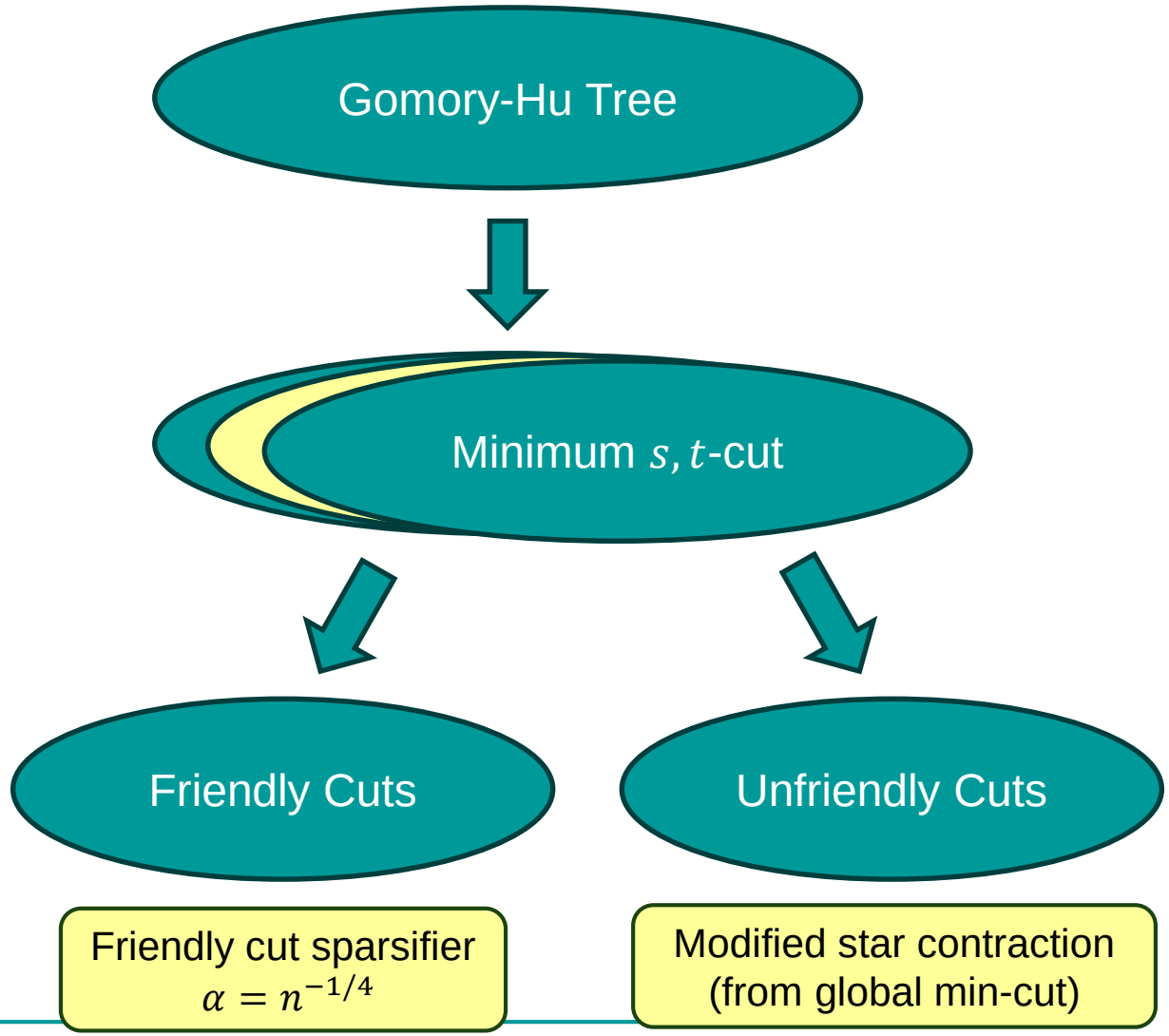
  if $\mathrm{cut}_G(S) \leq w$, then $\mathrm{cut}_G(S) = \mathrm{cut}_H(S)$

- **Lemma**: Can construct an (α, w)-friendly cut sparsifier using $\tilde{O}(\alpha^{-1} n\sqrt{w})$ complexity (cut-queries, streaming storage, update time)
  - H is actually a contraction of G (equivalent to adding infinite-capacity edges)
  - Thus, $\mathrm{cut}_H(S) \geq \mathrm{cut}_G(S)$ for all $S \subseteq V$

- We use $\alpha = 1/6,\ w = n$

Digression: Previous Algorithm [KK, SODA'26]

- All-Pairs Minimum Cut using $\tilde{O}(n^{7/4})$ cut queries
  - First non-trivial algorithm for this problem
  - Prior algorithms solve only single pair using $\tilde{O}(n^{8/5})$ cut queries [RSW18, ASW25, JNS26]
  - Uses friendly cut sparsifiers
  - Boils down to exact max-flows

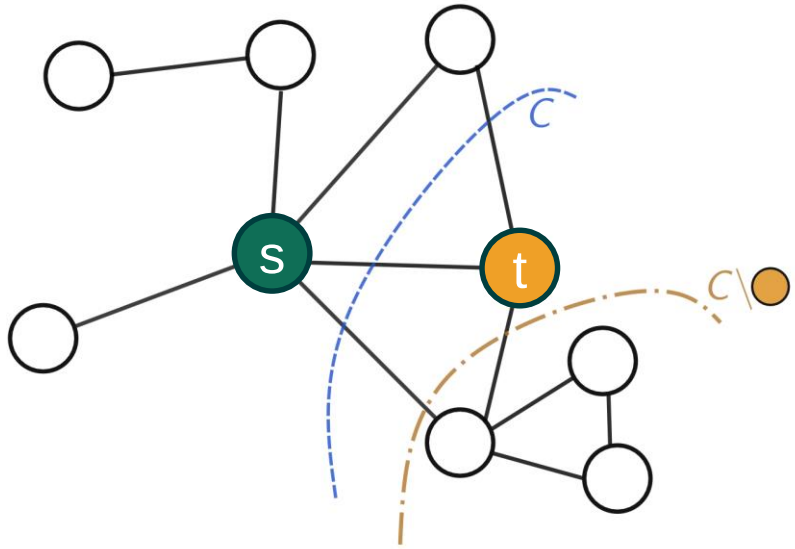


New Key Idea

- Minimum s,t-cuts are *"close"* (up to one vertex swap) to a friendly cut

- **Main Lemma:** If minimum s,t-cut $C \subset V$ is not 1/2-friendly, then moving one vertex (in fact, one of s,t) gives a 1/6-friendly cut
  - That is, either $X = C \setminus \{s\}$ or $X = \bar{C} \setminus \{t\}$ is 1/6-friendly

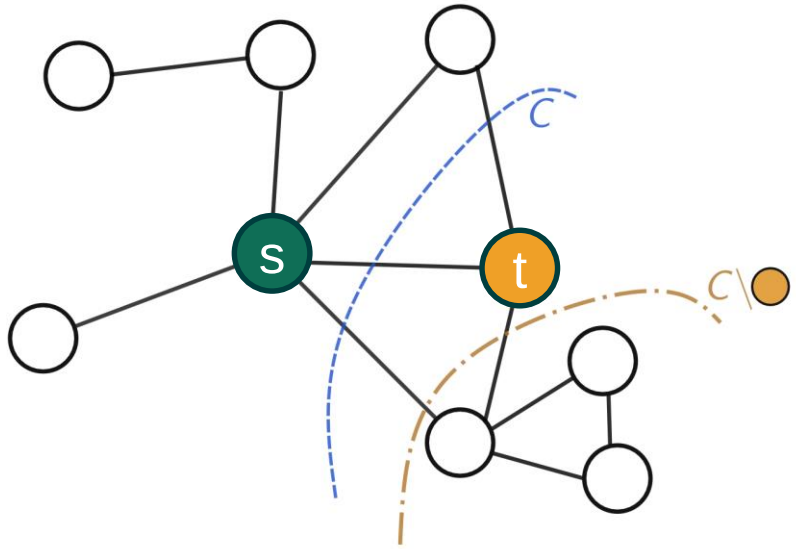
- Why helps?
  - A friendly cut sparsifier H preserves the cut X
  - We only need to know/handle edges incident to s,t



Proof of Main Lemma

Lemma: If min s, t-cut C is not 1/2-friendly, then $X = C \setminus \{s\}$ or $X = \overline{C} \setminus \{t\}$ is 1/6-friendly

- Consider minimum s, t -cut
  - Every v has friendliness ratio $\geq 1/2$
  - WLOG only t is not 1/2-friendly
- Swapping t yields a $1/6$-friendly cut:
  - Friendliness of t increases to $> 1/2$
  - Friendliness of $v \in V \setminus C$ and s only increases
  - Friednliness of $v \in C$ cannot decrease a lot



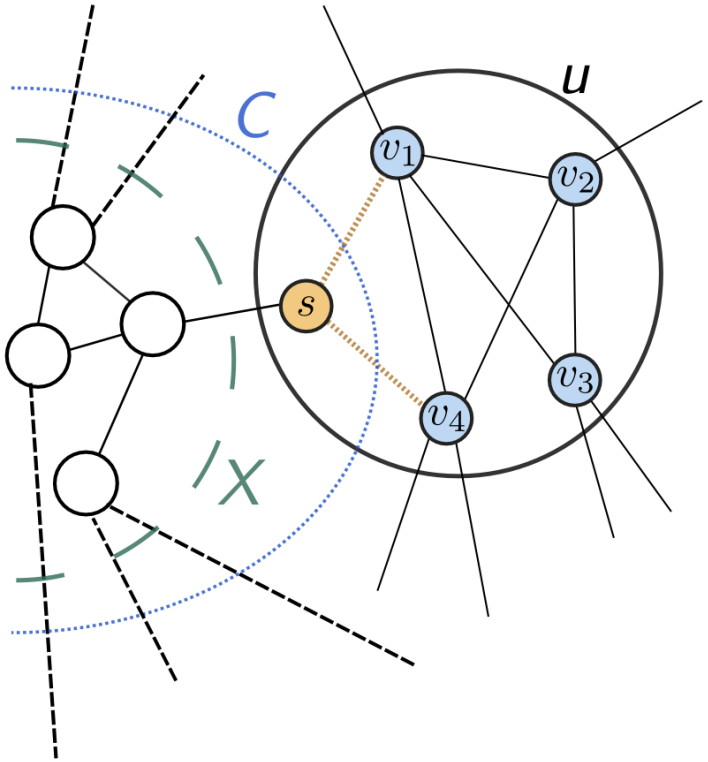
Encoding all Minimum s,t-Cuts

- **Theorem:** Given a 1/6-friendly cut sparsifier H and the list of vertex degrees $\{\deg(v)\}_{v\in V}$, one can find a minimum s,t-cut for all $s,t \in V$

- Proof: Examine a minimum s,t-cut C
- If C is friendly, then it is captured in H
- If C not friendly, then:
  - wlog $X = C \setminus \{s\}$ is friendly hence $\mathrm{cut}_G(X)$ is preserved in H
  - $\deg(s)$ is stored explicitly
  - Can evaluate $E(X,s)$ since these appear in H

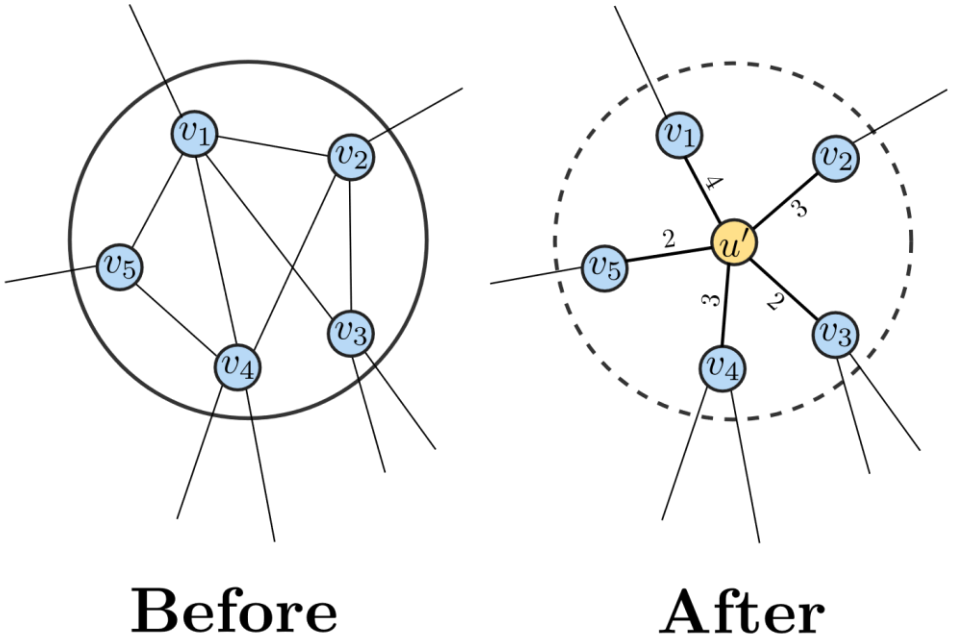
  (edges outgoing from X are not contracted)

- Find it by iterating over all possible $C \subset V$



Star Transform

- Recall friendly cut sparsifier $H = (V_H, E_H)$ is a contraction
  - View it as adding to G infinite-capacity edges
- Replace every supervertex $u \in V_H$ by a star:
  - Add a proxy vertex u'
  - Connect it to each inner-vertex $v \in u$ with weight $|E(v, u \setminus v)|$, i.e., the degree of v in $G[u]$
- Keep edges going "outside" of u
  - Connected to their original inner-vertex $v \in u$
- Easy to verify it preservers all minimum s, t-cuts



Digression: Round Complexity [KK, ESA'25]

- **Definition**: An algorithm has round complexity r if its queries are arranged into r batches, each depending only on answers of prior rounds
  - Measures adaptivity

- **Goal:** find tradeoff between query and round complexity
  - Every problem is trivial in $O(n^2)$ queries in 1 round
  - Prior literature did not consider round complexity, but typically use $O(\log^2 n)$ rounds

- Our results are for global min-cut problem

Digression: Round Complexity [KK, ESA'25]

Problem	Graph Type	Query Complexity	Round Complexity	Reference
Connectivity	Weighted	$\tilde{O}(n)$	1	[ACK21]
½-Approximate Max Cut	Weighted	$O(\log n)$	1	[PRW24]
$(1-\epsilon)$-Approximate Max Cut	Weighted	$\tilde{O}(\epsilon^{-2}n^2)$	$O(\log^2 n)$	
Minimum s,t-cut	Unweighted	$\tilde{O}(n^{3/2})$	3	[here]
Global Minimum Cut	Unweighted	$\tilde{O}(n)$	$O(\log^2 n)$	[RSW18,AEG+22,ASW25]
	Weighted			[MN20]
Global Minimum Cut	Unweighted	$\tilde{O}(n^{4/3})$	2	[KK25]
	Unweighted	$\tilde{O}(\mathrm{r}n^{1+\frac{1}{\mathrm{r}}}\delta_G^{-\frac{1}{r}})$	$2r+1$	
	Weighted	$\tilde{O}(\mathrm{r}n^{1+(1+\log \mathrm{W})/\mathrm{r}})$	$4r+3$	

W is the ratio between the heaviest and lightest edge of G, and δ_G is the minimum degree

Round Complexity: Key Technique

- **Weighted Edge Sampling:** Given a vertex $v \in V$, sample one edge incident to v proportionally to its weight
  - i.e., return each $e \in E(v, V \setminus v)$ with probability $\frac{w(e)}{w(E(v,V\setminus v))}$

- **Lemma:** There is a **two-round** algorithm for weighted edge sampling that uses $O(\log^4 nW)$ cut queries
  - where W is ratio of heaviest and lightest edge
- Relies on heavy-hitters (streaming technique) to recover heavy edges

Future Directions (Cut Queries)

- Optimal adaptivity (tradeoff between queries and rounds) for global min-cut
  - Even for symmetric submodular function minimization

- Unweighted min s,t-cut: break the barrier of $O(n^{3/2})$ queries (seen here)
- Weighted min s,t-cut: beat trivial $O(n^2)$ queries (recover the graph)
  - There is a lower bound of $\Omega(n^2/r^5)$ queries when using r rounds

- Global min-cut in hypergraphs: beat $O(n^2)$ queries [MN21]
- Reachability in directed graphs [Nanongkai]: beat trivial $O(n^2)$ queries

Thank You!