Space Optimal Vertex Cover in Dynamic Streams

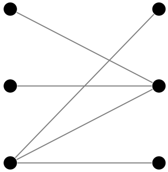
Kheeran K. Naidu & Vihan Shah

University of Bristol & Rutgers University

kheeran.naidu@bristol.ac.uk & vihan.shah98@rutgers.edu

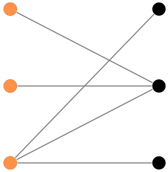
Vertex Cover

- Graph $G = (V, E)$



Vertex Cover

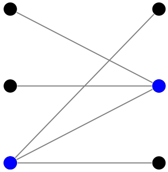
- Graph $G = (V, E)$
- Vertex Cover: $C \subseteq V$, $\forall e = (u, v) \in E$, $u \in C$ or $v \in C$



3

Vertex Cover

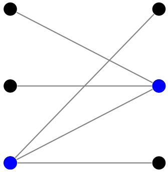
- Graph $G = (V, E)$
- Vertex Cover: $C \subseteq V$, $\forall e = (u, v) \in E$, $u \in C$ or $v \in C$
- Minimum Vertex Cover $\textsc{opt}$: Vertex Cover of the smallest size



2

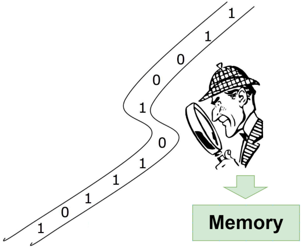
Vertex Cover

- Graph $G = (V, E)$
- Vertex Cover: $C \subseteq V$, $\forall e = (u, v) \in E$, $u \in C$ or $v \in C$
- Minimum Vertex Cover OPT: Vertex Cover of the smallest size



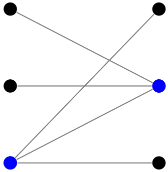
Graph Streaming:

- G arrives as a stream of edges



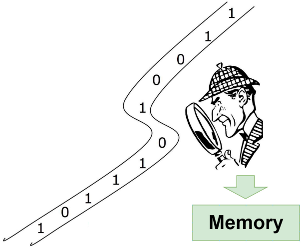
Vertex Cover

- Graph $G = (V, E)$
- Vertex Cover: $C \subseteq V$, $\forall e = (u, v) \in E$, $u \in C$ or $v \in C$
- Minimum Vertex Cover $\textsc{opt}$: Vertex Cover of the smallest size



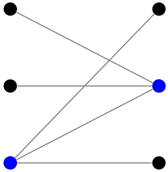
Graph Streaming:

- G arrives as a stream of edges
- Trivial: Store all edges ($\Omega(n^2)$ space)



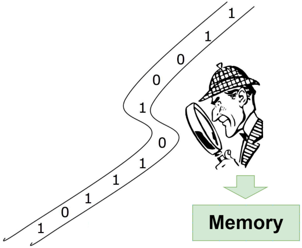
Vertex Cover

- Graph $G = (V, E)$
- Vertex Cover: $C \subseteq V$, $\forall e = (u, v) \in E$, $u \in C$ or $v \in C$
- Minimum Vertex Cover OPT: Vertex Cover of the smallest size



Graph Streaming:

- G arrives as a stream of edges
- Trivial: Store all edges ($\Omega(n^2)$ space)
- Goal: Minimize memory ($o(n^2)$ space)

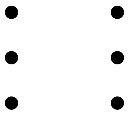


Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

Dynamic Graph Streams

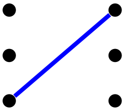
- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

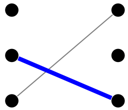
e_1



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

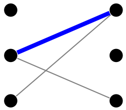
e_1	e_2



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

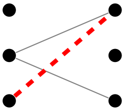
e_1	e_2	e_3



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

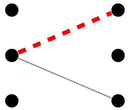
e_1	e_2	e_3	$\overline{e_1}$



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

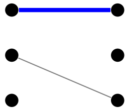
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

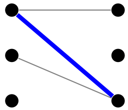
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

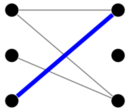
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

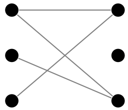
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5	e_1



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

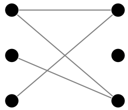
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5	e_1



Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions

e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5	e_1

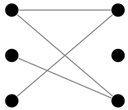


Minimum Vertex Cover:

- $O(1)$-approximation requires $\Omega(n^2)$ space
- α-approximation algorithms ($1 \leq \alpha \ll n$):

Dynamic Graph Streams

- $G = (V, E)$ arrives as a stream of edges
- Edge insertions and deletions



e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5	e_1

Minimum Vertex Cover:

- $O(1)$-approximation requires $\Omega(n^2)$ space
- α-approximation algorithms ($1 \leq \alpha \ll n$):
  - Lower bound: $\Omega(\frac{n^2}{\alpha^2})$ [DK20]
  - Upper bound: $O(\frac{n^2}{\alpha^2} \cdot \log \alpha)$ [DK20]

Understanding polylog factors

These type of polylog gaps appear frequently in the literature

- One main reason is storing counters or edges

Understanding polylog factors

These type of polylog gaps appear frequently in the literature

- One main reason is storing counters or edges

Are they inherent to the problem?

Understanding polylog factors

These type of polylog gaps appear frequently in the literature

- One main reason is storing counters or edges

Are they inherent to the problem?

- [SW15] showed that for several problems (Bipartiteness, Approximate Minimum Cut, etc.) the lower bounds can be improved to $\Omega(n \log n)$
- [NY19] showed that Connectivity has a lower bound of $\Omega(n \log^3 n)$

Understanding polylog factors

These type of polylog gaps appear frequently in the literature

- One main reason is storing counters or edges

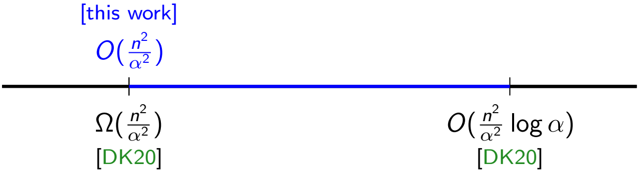
Are they inherent to the problem?

- [SW15] showed that for several problems (Bipartiteness, Approximate Minimum Cut, etc.) the lower bounds can be improved to $\Omega(n \log n)$
- [NY19] showed that Connectivity has a lower bound of $\Omega(n \log^3 n)$
- [AS22] gave the first result showing that the polylog factors can be removed by giving an algorithm for α-approximate Maximum Matching using $O(n^2/\alpha^3)$ bits, matching the lower bound [DK20]

Our Work

Theorem

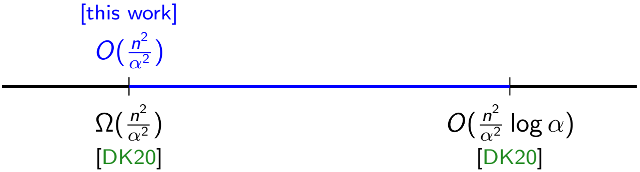
There exists a randomised dynamic graph streaming algorithm for α-approximate minimum vertex cover that succeeds with high probability and uses $O(\frac{n^2}{\alpha^2})$ bits of space for any $\alpha \leq n^{1-\delta}$ where $\delta > 0$.



Our Work

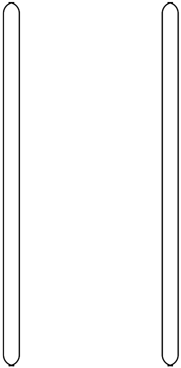
Theorem

There exists a randomised dynamic graph streaming algorithm for α-approximate minimum vertex cover that succeeds with high probability and uses $O(\frac{n^2}{\alpha^2})$ bits of space for any $\alpha \leq n^{1-\delta}$ where $\delta > 0$.



An algorithm that uses optimal space up to constant factors!

α-Approx Det. Dynamic Vertex Cover [DK20]

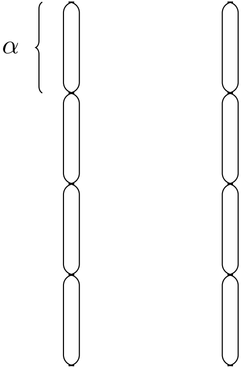


Simplifying Assumption (for the talk):

- The input graph is bipartite

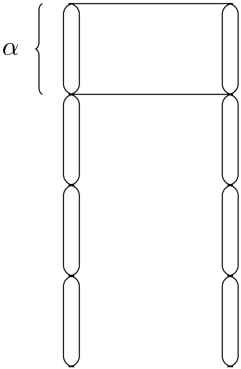
It is easily lifted!

α-Approx Det. Dynamic Vertex Cover [DK20]



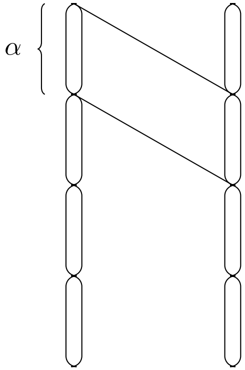
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups

α-Approx Det. Dynamic Vertex Cover [DK20]



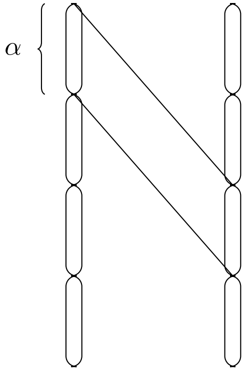
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



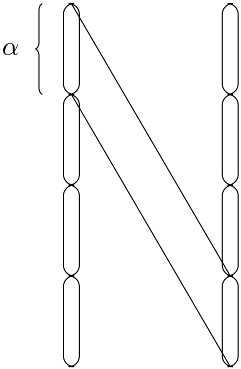
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



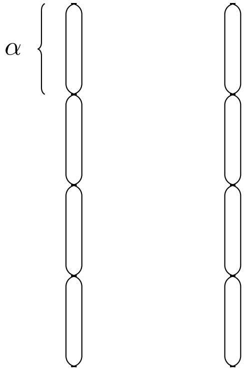
1. Vertex groups of size α
    - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
    - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



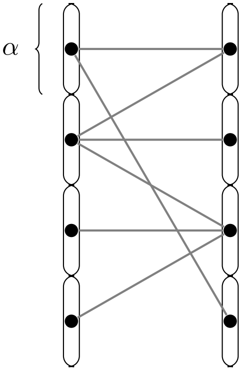
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



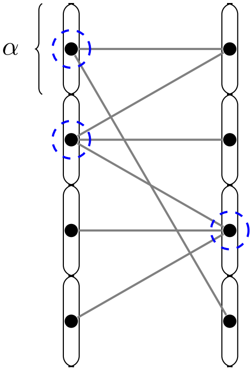
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



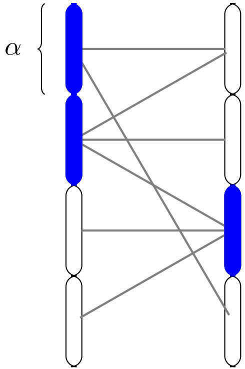
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph

α-Approx Det. Dynamic Vertex Cover [DK20]



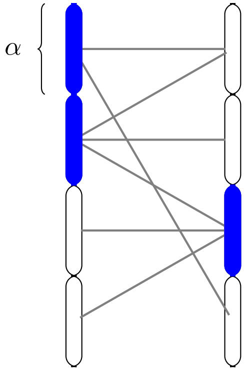
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph
4. Compute a group-level vertex cover

α-Approx Det. Dynamic Vertex Cover [DK20]



1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph
4. Compute a group-level vertex cover
5. Return vertices of the covering groups

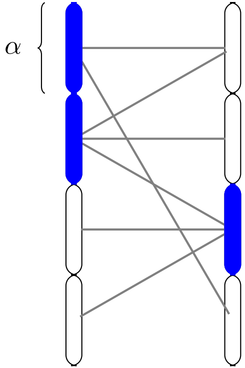
α-Approx Det. Dynamic Vertex Cover [DK20]



1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph
4. Compute a group-level vertex cover
5. Return vertices of the covering groups

This is an α-approximation.

α-Approx Det. Dynamic Vertex Cover [DK20]

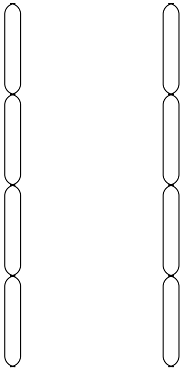


1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph
4. Compute a group-level vertex cover
5. Return vertices of the covering groups

This is an α-approximation.

Space: $O(\frac{n^2}{\alpha^2})$ counters, each using $O(\log \alpha)$ bits. Hence, $O(\frac{n^2}{\alpha^2} \log \alpha)$ bits.

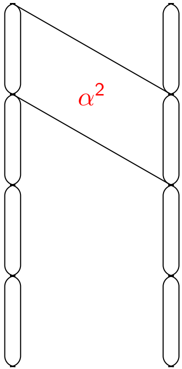
What's the issue?



What's the issue?

Problem:

- Each counter counts upto α^2 edges.
- Counters use $O(\log \alpha)$ bits.



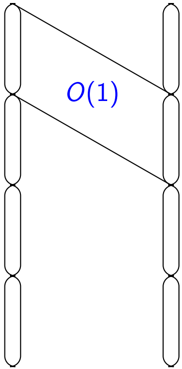
What's the issue?

Problem:

- Each counter counts upto α^2 edges.
- Counters use $O(\log \alpha)$ bits.

Goal:

- Counters to count upto $O(1)$ edges
- Counters to use $O(1)$ bits.



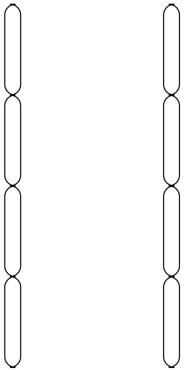
Solving the issue for sparse graphs

For G with $\approx \frac{n^2}{\alpha^2}$ edges

Solving the issue for sparse graphs

For G with $\approx \frac{n^2}{\alpha^2}$ edges

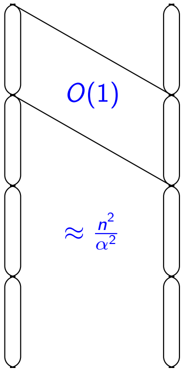
- Randomly partition into groups of size α



Solving the issue for sparse graphs

For G with $\approx \frac{n^2}{\alpha^2}$ edges

- Randomly partition into groups of size α
- $\frac{n^2}{\alpha^2}$ pairs of groups
- Counters use $O(1)$ bits *(in expectation)*



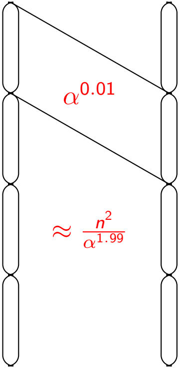
Solving the issue for sparse graphs

For G with $\approx \frac{n^2}{\alpha^2}$ edges

- Randomly partition into groups of size α
- $\frac{n^2}{\alpha^2}$ pairs of groups
- Counters use $O(1)$ bits *(in expectation)*

For G with $\approx \frac{n^2}{\alpha^{1.99}}$ edges or more:

- Counters use $\Theta(\log \alpha)$ bits



Solving the issue (in general)

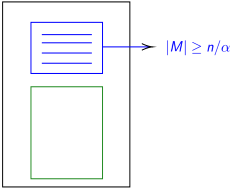
G may not be sparse

Solving the issue (in general)

G may not be sparse

Match-or-Sparsify Lemma:

- **either** $|M| \geq \frac{n}{\alpha}$ then $|\mathrm{OPT}| \geq \frac{n}{\alpha}$
  $\implies$ V is an α-approx

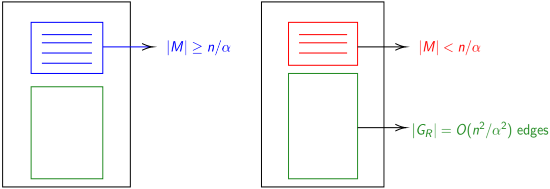


Solving the issue (in general)

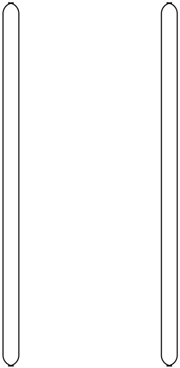
G may not be sparse

Match-or-Sparsify Lemma:

- **either** $|M| \geq \frac{n}{\alpha}$ then $|\mathrm{OPT}| \geq \frac{n}{\alpha}$
  $\implies$ V is an α-approx
- **or** $|G_R| = O(\frac{n^2}{\alpha^2})$
  $\implies$ counters use $O(1)$ bits (in expectation)

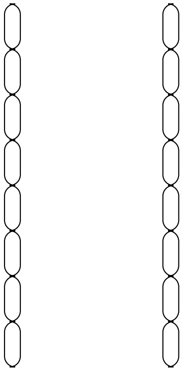


Space Optimal Algorithm



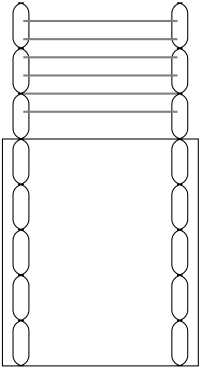
Space Optimal Algorithm

1. Randomly partition vertices ($\frac{n}{\alpha}$ groups)



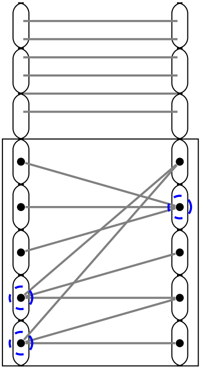
Space Optimal Algorithm

1. Randomly partition vertices ($\frac{n}{\alpha}$ groups)
2. Run Match-or-Sparsify lemma
   - if $|M|$ is large, return V



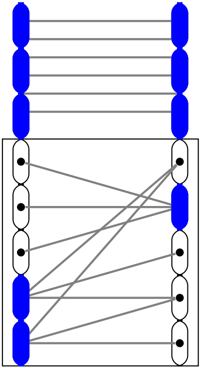
Space Optimal Algorithm

1. Randomly partition vertices ($\frac{n}{\alpha}$ groups)
2. Run Match-or-Sparsify lemma
   - if $|M|$ is large, return V
3. Check if an edge is present between pairs and compute group-level vertex cover



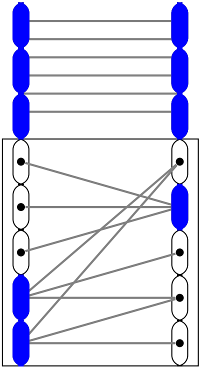
Space Optimal Algorithm

1. Randomly partition vertices ($\frac{n}{\alpha}$ groups)
2. Run Match-or-Sparsify lemma
   - if $|M|$ is large, return V
3. Check if an edge is present between pairs and compute group-level vertex cover
4. Return vertices of the covering groups including those with matched vertices



Space Optimal Algorithm

1. Randomly partition vertices ($\frac{n}{\alpha}$ groups)
2. Run Match-or-Sparsify lemma
   - if $|M|$ is large, return V
3. Check if an edge is present between pairs and compute group-level vertex cover
4. Return vertices of the covering groups including those with matched vertices



How to prove the Match-or-Sparsify lemma? Checkout the long talk!

Summary

1. There is a dynamic streaming algorithm that whp outputs an α-approximation to minimum vertex cover using $O(n^2/\alpha^2)$ bits of space
   - Match or Sparsify in $O(n^2/\alpha^2)$ bits of space
   - The ideas from [DK20] along with random partitioning solve the sparse case in $O(n^2/\alpha^2)$ bits of space
   - We run both algorithms in parallel and get the final algorithm

Summary

1. There is a dynamic streaming algorithm that whp outputs an α-approximation to minimum vertex cover using $O(n^2/\alpha^2)$ bits of space
   - Match or Sparsify in $O(n^2/\alpha^2)$ bits of space
   - The ideas from [DK20] along with random partitioning solve the sparse case in $O(n^2/\alpha^2)$ bits of space
   - We run both algorithms in parallel and get the final algorithm
2. The lower bound of $\Omega(n^2/\alpha^2)$ [DK20] makes our algorithm optimal

Summary

1. There is a dynamic streaming algorithm that whp outputs an α-approximation to minimum vertex cover using $O(n^2/\alpha^2)$ bits of space
    - Match or Sparsify in $O(n^2/\alpha^2)$ bits of space
    - The ideas from [DK20] along with random partitioning solve the sparse case in $O(n^2/\alpha^2)$ bits of space
    - We run both algorithms in parallel and get the final algorithm
2. The lower bound of $\Omega(n^2/\alpha^2)$ [DK20] makes our algorithm optimal
3. The $\mathrm{polylog}(n)$ overhead is not always necessary (Like [AS22])

Open Problems

- Could similar techniques to this work and [AS22] be used to bypass the $\mathrm{polylog}(n)$ overheads of other problems?
    - E.g. Dominating Set, Spectral Sparsification

- Can we get a deterministic algorithm for this problem that uses only $O(\frac{n^2}{\alpha^2})$ bits of space or improve the lower bound?
    - The current best deterministic algorithm is that of [DK20] which uses $O(\frac{n^2}{\alpha^2}\log\alpha)$ bits of space

Open Problems

- Could similar techniques to this work and [AS22] be used to bypass the $\text{polylog}(n)$ overheads of other problems?
  - E.g. Dominating Set, Spectral Sparsification
- Can we get a deterministic algorithm for this problem that uses only $O(\frac{n^2}{\alpha^2})$ bits of space or improve the lower bound?
  - The current best deterministic algorithm is that of [DK20] which uses $O(\frac{n^2}{\alpha^2} \log \alpha)$ bits of space

Thank you!

References I

- Sepehr Assadi and Vihan Shah, *An asymptotically optimal algorithm for maximum matching in dynamic streams*, 13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA (Mark Braverman, ed.), LIPIcs, vol. 215, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 9:1–9:23.
- Jacques Dark and Christian Konrad, *Optimal lower bounds for matching and vertex cover in dynamic graph streams*, 35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference) (Shubhangi Saraf, ed.), LIPIcs, vol. 169, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 30:1–30:14.

References II

Jelani Nelson and Huacheng Yu, *Optimal lower bounds for distributed and streaming spanning forest computation*, Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2019, pp. 1844–1860.

Xiaoming Sun and David P Woodruff, *Tight bounds for graph problems in insertion streams*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.