Space Optimal Vertex Cover in Dynamic Streams

Kheeran K. Naidu & Vihan Shah

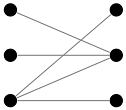
University of Bristol & Rutgers University

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

1. Introduction
2. Optimal Algorithm
3. Key Lemma
4. Conclusion

Vertex Cover

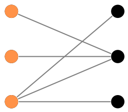
- Graph $G = (V, E)$
- Vertex Cover: $C \subseteq V$, $\forall e = (u, v) \in E$, $u \in C$ or $v \in C$



- Minimum Vertex Cover $\textsc{opt}$: Vertex Cover of the smallest size

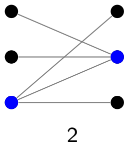
Vertex Cover

- 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



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



Classical Setting

Minimum Vertex Cover (NP-Complete)

The smallest set of vertices which includes at least a single endpoint of every edge.

Approximation in Poly-Time

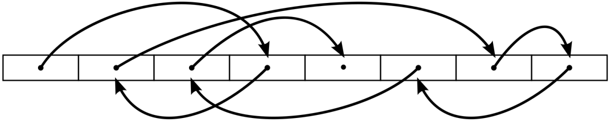
Return the vertices of a maximal GREEDY Matching algorithm to get a 2-approximation

Note: A 2-approximate vertex cover can have at most $2\ |\textsc{opt}|$ vertices

Classical Setting

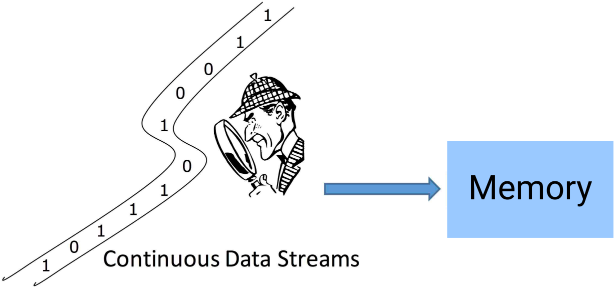
Assumption (Infeasible for massive graphs)

Classical algorithms rely on the assumption that they have a random access to the input of the algorithm



Graph Streaming

- $G = (V, E)$
- Edges of G appear in a stream
- Trivial Solution: Store all edges ($\Omega(n^2)$ space)
- Goal: Minimize Memory ($o(n^2)$ space)



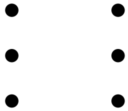
Streaming Models

Insertion-Only

Dynamic

Streaming Models

Insertion-Only

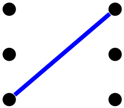


Dynamic

Streaming Models

Insertion-Only

$\boxed{e_1}$

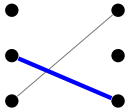


Dynamic

Streaming Models

Insertion-Only

e_1	e_2

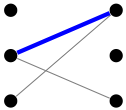


Dynamic

Streaming Models

Insertion-Only

e_1	e_2	e_3

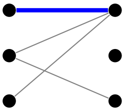


Dynamic

Streaming Models

Insertion-Only

e_1	e_2	e_3	e_4

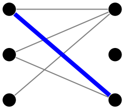


Dynamic

Streaming Models

Insertion-Only

e_1	e_2	e_3	e_4	e_5

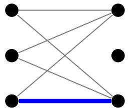


Dynamic

Streaming Models

Insertion-Only

e_1	e_2	e_3	e_4	e_5	e_6

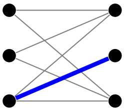


Dynamic

Streaming Models

Insertion-Only

e_1	e_2	e_3	e_4	e_5	e_6	e_7

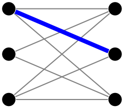


Dynamic

Streaming Models

Insertion-Only

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8

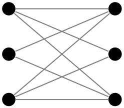


Dynamic

Streaming Models

Insertion-Only (finite stream)

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8

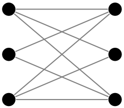


Dynamic

Streaming Models

Insertion-Only (finite stream)

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



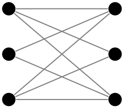
- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

Streaming Models

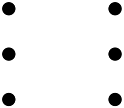
Insertion-Only (finite stream)

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

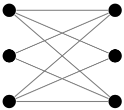
Dynamic



Streaming Models

Insertion-Only (finite stream)

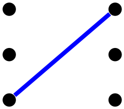
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- Greedy gives a 2-approximation

Dynamic

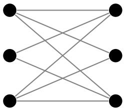
e_1



Streaming Models

Insertion-Only (finite stream)

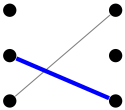
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

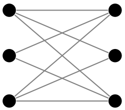
e_1	e_2



Streaming Models

Insertion-Only (finite stream)

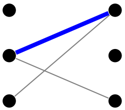
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

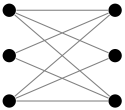
e_1	e_2	e_3



Streaming Models

Insertion-Only (finite stream)

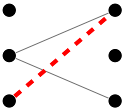
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- Greedy gives a 2-approximation

Dynamic

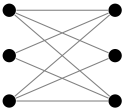
e_1	e_2	e_3	$\overline{e_1}$



Streaming Models

Insertion-Only (finite stream)

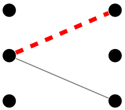
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

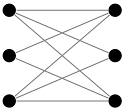
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$



Streaming Models

Insertion-Only (finite stream)

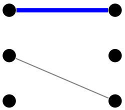
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

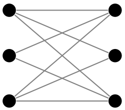
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4



Streaming Models

Insertion-Only (finite stream)

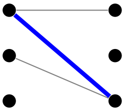
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

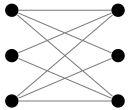
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5



Streaming Models

Insertion-Only (finite stream)

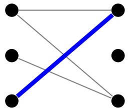
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic

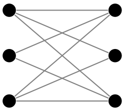
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5	e_1



Streaming Models

Insertion-Only (finite stream)

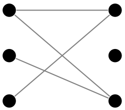
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic (finite stream)

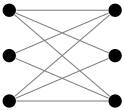
e_1	e_2	e_3	$\overline{e_1}$	$\overline{e_3}$	e_4	e_5	e_1



Streaming Models

Insertion-Only (finite stream)

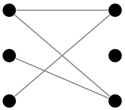
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic (finite stream)

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

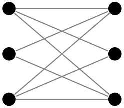


- $O(1)$-approximation requires $\Omega(n^2)$ space

Streaming Models

Insertion-Only (finite stream)

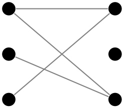
e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8



- Exact solution requires $\Omega(n^2)$ space
- GREEDY gives a 2-approximation

Dynamic (finite stream)

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



- $O(1)$-approximation requires $\Omega(n^2)$ space
- α-approximation algorithms ($1 \leq \alpha \ll n$):
  - LB: $\Omega(\frac{n^2}{\alpha^2})$ and UB: $O(\frac{n^2}{\alpha^2} \log \alpha)$ [DK20]

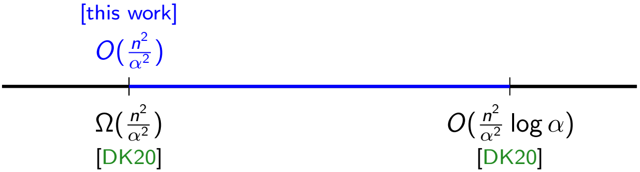
Understanding polylog factors

- These types of $\mathrm{polylog}(n)$ gaps appear frequently in the literature
- One main reason is storing counters or edges
- [SW15] showed that for many problems the lower bounds can be improved to include the log factors (Bipartiteness, Approximate Minimum Cut etc)
- Connectivity has a lower bound of $\Omega(n \log^3 n)$ ([NY19])
- [AS22] was the first result that showed $\mathrm{polylog}(n)$ factors can be removed in the upper bound by giving an algorithm for approximate matching using $O(n^2/\alpha^3)$ bits

Our Results

Theorem

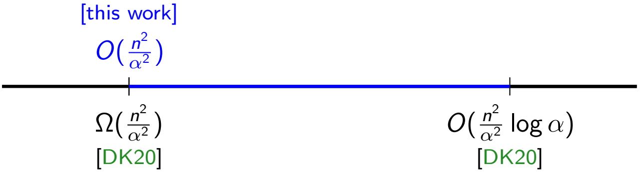
There exists a randomised dynamic graph streaming algorithm for α-approximate minimum vertex cover that succeeds with high probability and uses $O(\frac{n^2}{\alpha^2})$ bits of space for any $\alpha \leq n^{1-\delta}$ where $\delta > 0$.



Our Results

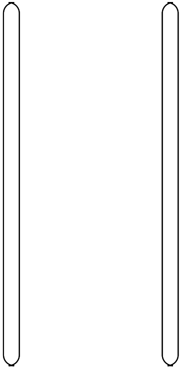
Theorem

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]

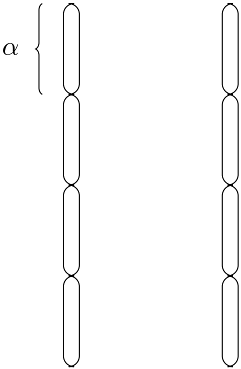


Simplifying Assumption (for the talk):

- The input graph is bipartite

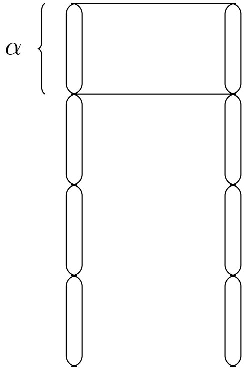
It is easily lifted!

α-Approx Det. Dynamic Vertex Cover [DK20]



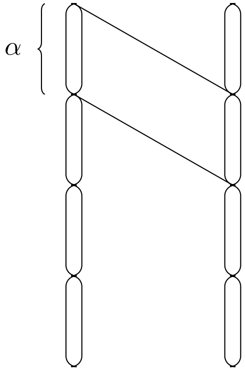
1. Vertex groups of size α
    - about $\frac{n}{\alpha}$ groups

α-Approx Det. Dynamic Vertex Cover [DK20]



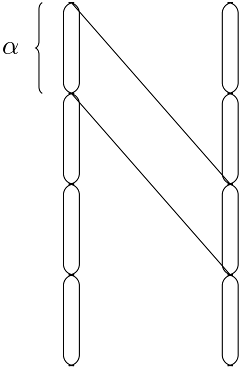
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



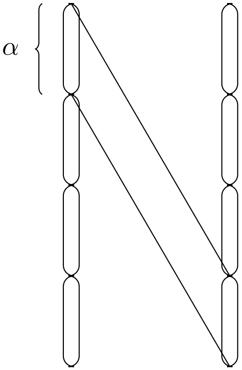
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



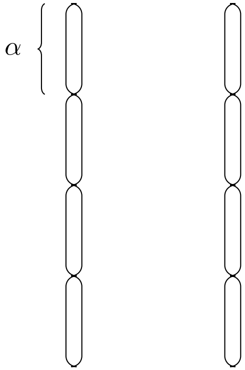
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



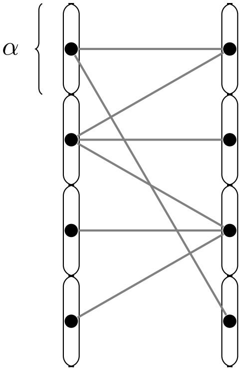
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



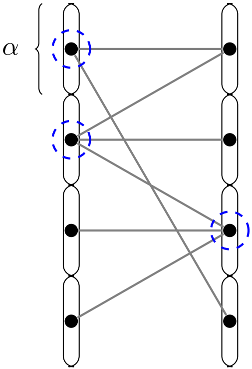
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs

α-Approx Det. Dynamic Vertex Cover [DK20]



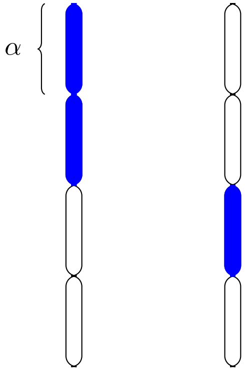
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph

α-Approx Det. Dynamic Vertex Cover [DK20]



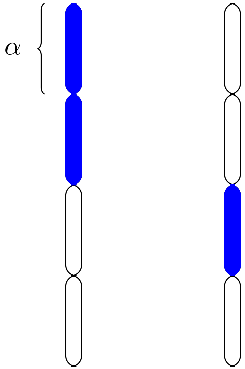
1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph
4. Compute a group-level vertex cover

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

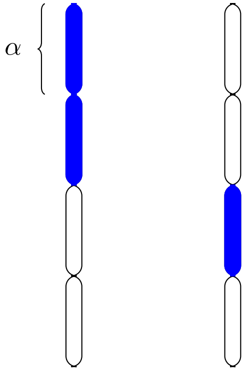
α-Approx Det. Dynamic Vertex Cover [DK20]



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]

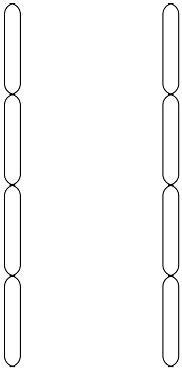


1. Vertex groups of size α
   - about $\frac{n}{\alpha}$ groups
2. Use counters to check if there is at least one edge between each pair of groups
   - about $\frac{n^2}{\alpha^2}$ pairs
3. Construct the group-level graph
4. Compute a group-level vertex cover
5. Return vertices of the covering groups

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.

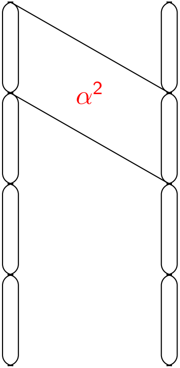
What’s the issue?



What's the issue?

Problem:

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



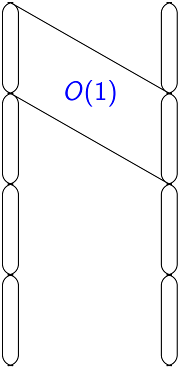
What's the issue?

Problem:

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

Goal:

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



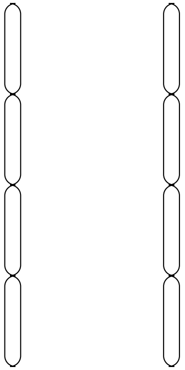
Solving the issue for sparse graphs

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

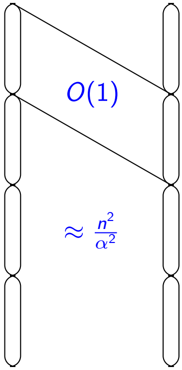
- Randomly partition into groups of size α



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)*



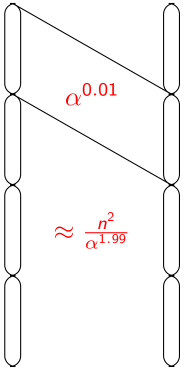
Solving the issue for sparse graphs

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)

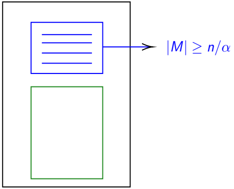
G may not be sparse

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

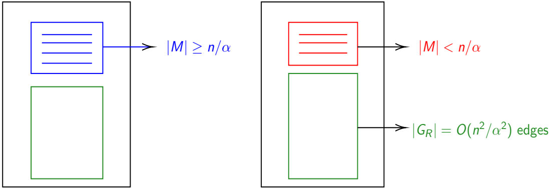


Solving the issue (in general)

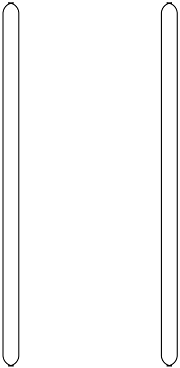
G may not be sparse

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)

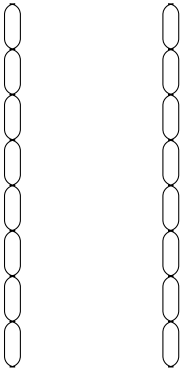


Space Optimal Algorithm



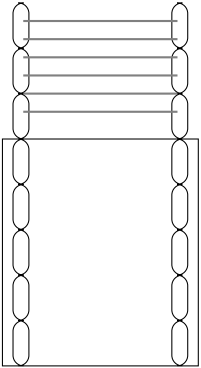
Space Optimal Algorithm

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



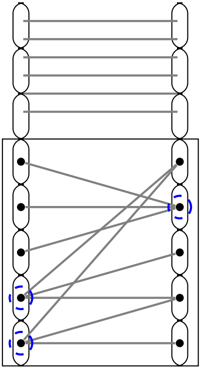
Space Optimal Algorithm

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



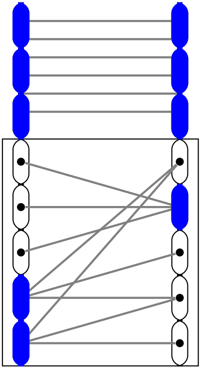
Space Optimal Algorithm

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



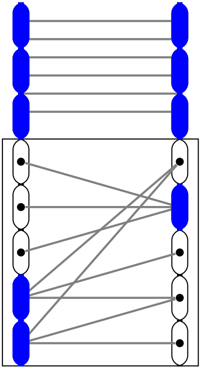
Space Optimal Algorithm

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



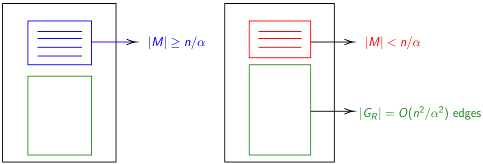
How to prove the Match-or-Sparsify lemma?

How to prove Match-or-Sparsify Lemma

Lemma

There is an algorithm that uses $O(\frac{n^2}{\alpha^2})$ bits of space and with high probability outputs a matching M that satisfies <u>at least one</u> of the following conditions:

- Match-case: $|M| \geq \frac{n}{\alpha}$.
- Sparsify-case: G_R, *has* $O(\frac{n^2}{\alpha^2})$ *edges.*



Attempt 1: Uniform Sampling

Algorithm:

- Sample $\tilde{\Theta}(\frac{n^2}{\alpha^2})$ edges uniformly at random
- Let M be a greedy matching over the sampled edges

Attempt 1: Uniform Sampling

Algorithm:

- Sample $\tilde{\Theta}(\frac{n^2}{\alpha^2})$ edges uniformly at random

- Let M be a greedy matching over the sampled edges

Space: $\tilde{\Theta}(\frac{n^2}{\alpha^2}) \cdot \mathrm{polylog}(n) = O(\frac{n^2}{\alpha^2})$ bits

Attempt 1: Uniform Sampling

Algorithm:

- Sample $\tilde{\Theta}(\frac{n^2}{\alpha^2})$ edges uniformly at random

- Let M be a greedy matching over the sampled edges

Space: $\tilde{\Theta}(\frac{n^2}{\alpha^2}) \cdot \mathrm{polylog}(n) = O(\frac{n^2}{\alpha^2})$ bits

The residual graph is sparse!

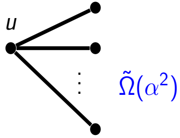
G_R has maximum degree $\tilde{O}(\alpha^2)$

Attempt 1: Uniform Sampling

G_R has maximum degree $\tilde{O}(\alpha^2)$

The probability that a vertex u survives with $\tilde{\Omega}(\alpha^2)$ neighbors:

- None of these $\tilde{\Omega}(\alpha^2)$ edges are sampled
- There are at most n^2 total edges



$$\left(1 - \frac{\tilde{\Omega}(\alpha^2)}{n^2}\right)^{\tilde{\Theta}\left(\frac{n^2}{\alpha^2}\right)} \leq \exp(-\tilde{\Omega}(1)) \leq \frac{1}{\mathrm{poly}(n)}$$

Limitations

This algorithm only works for small α

- We want a sparse graph with at most $O(n^2/\alpha^2)$ edges
- The max degree bound is $\widetilde{O}(\alpha^2)$
- $n \cdot \widetilde{O}(\alpha^2) \leq O(\frac{n^2}{\alpha^2})$ implies $\alpha \ll n^{1/4}$

Drawbacks

Drawback 1

- Do not need sparse graph when we find large matching
- This algorithm is more like Match <u>and</u> Sparsify

Fix 1: Match <u>or</u> Sparsify

Drawback 2

- There is a hard instance showing that uniform sampling does not work for large α
- Uniform sampling is biased towards high degree vertices

Fix 2: Non-Uniform Sampling

Attempt 2: Addressing both the drawbacks gives us the Lemma.

Space

The main algorithm works for any $\alpha \leq n^{1-\delta}$ for any constant $\delta > 0$.

- When $\alpha = n^{1-\delta}$, space used is $O(n^2/\alpha^2) = O(n^{2\delta})$ bits
- But the minimum vertex cover can be of size $\Omega(n)$

Space

The main algorithm works for any $\alpha \leq n^{1-\delta}$ for any constant $\delta > 0$.

- When $\alpha = n^{1-\delta}$, space used is $O(n^2/\alpha^2) = O(n^{2\delta})$ bits
- But the minimum vertex cover can be of size $\Omega(n)$
- Recall we either pick an entire group or no vertex from the group
- We output the indices of groups that are picked (space: $\widetilde{O}(1)$)
- Space: $\frac{n}{\alpha} \cdot \widetilde{O}(1) = O(n^2/\alpha^2)$

Summary

1. Match or Sparsify: In $O(n^2/\alpha^2)$ bits of space
   - We either get a large matching (which implies a large vertex cover)
   - Or get a sparse graph
2. The ideas from [DK20] along with random partitioning solve the sparse case in $O(n^2/\alpha^2)$ bits of space
3. We run both algorithms in parallel and get the final algorithm

Summary

- 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

Summary

- 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
- The lower bound of [DK20] is $\Omega(n^2/\alpha^2)$ bits making our algorithm optimal

Summary

- 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
- The lower bound of [DK20] is $\Omega(n^2/\alpha^2)$ bits making our algorithm optimal
- $\mathrm{polylog}(n)$ overhead is not always necessary (Like [AS22])

Open Problems

- Could similar techniques to this work and [AS22] be used to bypass $\mathrm{polylog}(n)$ overheads for 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 $\mathrm{polylog}(n)$ overheads for 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.