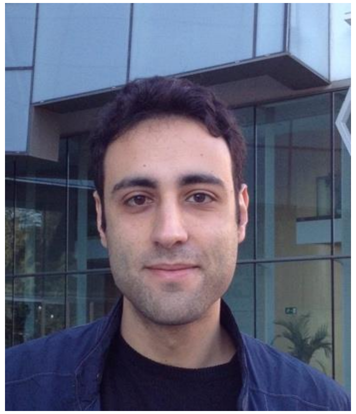
The Power of Graph Sparsification in the Continual Release Model

Quanquan C. Liu
Yale University

Joint work with



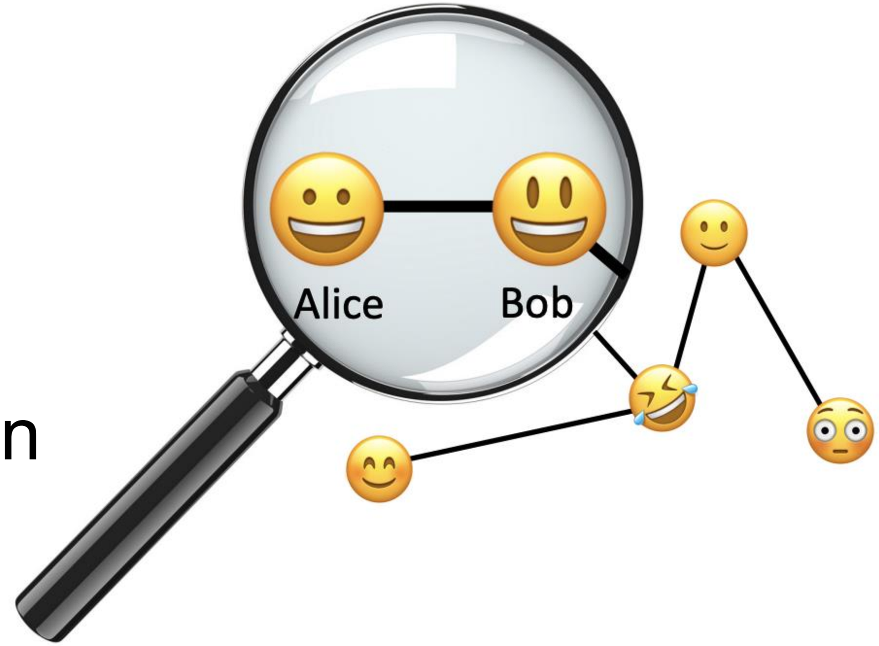
Alessandro Epasto
Google Research

Tamalika Mukherjee
Columbia University

Felix Zhou
Yale University

Publishing Sensitive Graph Information

- Potentially **sensitive connections between individuals** published as graphs
  - Social relationships
  - Financial transactions
  - Disease (e.g. COVID) transmission
  - Search data
  - Email and cell phone communication



Why do we want privacy on graphs?

- Privacy attacks can identify and deanonymize individuals and connections based on **external (e.g. public) information**
  - Re-identify nodes in **social networks** and **computer networks**

Why do we want privacy on graphs?

- Privacy attacks can identify and deanonymize individuals and connections based on **external (e.g. public) information**
  - Re-identify nodes in **social networks** and **computer networks**

Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography

Authors: Lars Backstrom, Cynthia Dwork, and Jon Kleinberg | Authors Info & Claims

WWW '07: Proceedings of the 16th international conference on World Wide Web • May 2007 • Pages 181 - 190

Playing Devil's Advocate:
Inferring Sensitive Information from Anonymized Network Traces

Scott E. Coull* Charles V. Wright* Fabian Monrose* Michael P. Collins† Michael K. Reiter‡

Graph Data Anonymization, De-Anonymization Attacks, and De-Anonymizability Quantification: A Survey

Publisher: IEEE

Shouling Ji ; Prateek Mittal ; Raheem Beyah

A Practical Attack to De-anonymize Social Network Users

Publisher: IEEE

Gilbert Wondracek ; Thorsten Holz ; Engin Kirda ; Christopher Kruegel

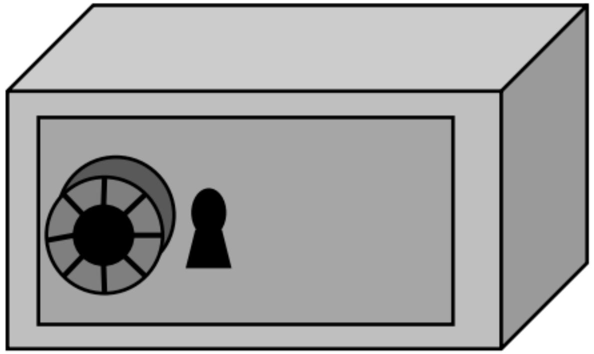
Link Prediction by De-anonymization: How We Won the Kaggle Social Network Challenge

Arvind Narayanan, Elaine Shi, Benjamin I. P. Rubinstein

Private Analysis of Graph Data

Graph G

Trusted Curator



Queries

Answers

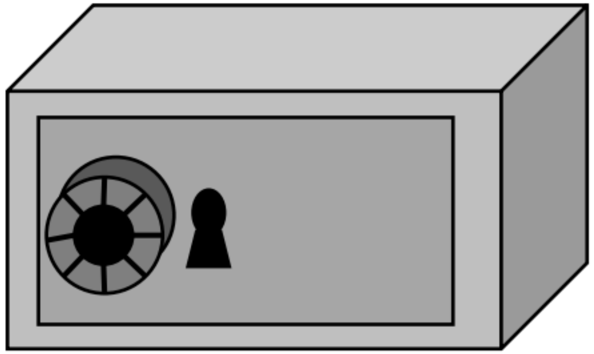
Users

Researchers
Government
Business
Malicious Adversaries

Private Analysis of Graph Data

Graph G

Trusted Curator



Queries

Answers

Users

Researchers
Government
Business
Malicious Adversaries

Answers satisfy **Differential Privacy**

Notation

Notation

- ε is **privacy parameter**, **not** approximation factor

Notation

- ε is **privacy parameter**, **not** approximation factor
- DP stands for **differential privacy**, **not** dynamic programming

Notation

- ε is **privacy parameter**, **not** approximation factor
- DP stands for **differential privacy**, **not** dynamic programming
- η is approximation factor

Notation

- ε is **privacy parameter**, **not** approximation factor
- DP stands for **differential privacy**, **not** dynamic programming
- η is approximation factor
- S represents a stream of edge updates

Notation

- ε is **privacy parameter**, **not** approximation factor
- DP stands for **differential privacy**, **not** dynamic programming
- η is approximation factor
- S represents a stream of edge updates
- n number of vertices, m number of edges in entire stream

Notation

- ε is **privacy parameter**, **not** approximation factor
- DP stands for **differential privacy**, **not** dynamic programming
- η is approximation factor
- S represents a stream of edge updates
- n number of vertices, m number of edges in entire stream
- UB = upper bound, LB = lower bound

Differential Privacy

Differential Privacy [Dwork-McSherry-Nissim-Smith '06]

A (randomized) algorithm $\mathcal{A}$ is **ε-differentially private** if for all pairs of neighbors G and G' and all sets of possible outputs Y:

$$e^{-\varepsilon} \leq \frac{\Pr[\mathcal{A}(G) \in Y]}{\Pr[\mathcal{A}(G') \in Y]} \leq e^{\varepsilon}$$

Neighboring Graphs

Edge-neighboring graphs
differ in **1 edge**

G

G'

Neighboring Graphs

Edge-neighboring graphs differ in **1 edge**

Node-neighboring graphs differ in **all edges adjacent to any 1 node**

G

G'

G

G'

Online Streaming Graphs

- Continuously release **accurate graph statistics** **after each update** while using sublinear space

Online Streaming Graphs

- Continuously release **accurate graph statistics after each update** while using sublinear space
  - Independent set [Halldórsson-Halldórsson-Losievskaja-Szegedy ’16, Cormode-Dark-Konrad ’18]

Online Streaming Graphs

- Continuously release **accurate graph statistics** **after each update** while using sublinear space
  - Independent set [Halldórsson-Halldórsson-Losievskaja-Szegedy ’16, Cormode-Dark-Konrad ’18]
  - Dominating set and matching [Chen-Chitnus-Eades-Wirth ‘23]

Online Streaming Graphs

- Continuously release **accurate graph statistics** **after each update** while using sublinear space
  - Independent set [Halldórsson-Halldórsson-Losievskaja-Szegedy ’16, Cormode-Dark-Konrad ’18]
  - Dominating set and matching [Chen-Chitnus-Eades-Wirth ‘23]
  - Edge coloring [Ghosh-Stoeckl ‘23]

Continual Release Model [DNPR10, CSS11]

- Given a stream of T **edge updates** (insertions and deletions), produce a vector of T **outputs**, one after every update

Continual Release Model [DNPR10, CSS11]

- Given a stream of T **edge updates** (insertions and deletions), produce a vector of T **outputs**, one after every update
  - **Vector of outputs** is **ε-differentially private** on **neighboring streams**

Continual Release Model [DNPR10, CSS11]

- Given a stream of T **edge updates** (insertions and deletions), produce a vector of T **outputs**, one after every update
  - **Vector of outputs** is **ε-differentially private** on **neighboring streams**
  - Edge-neighboring and node-neighboring streams

Continual Release Model [DNPR10, CSS11]

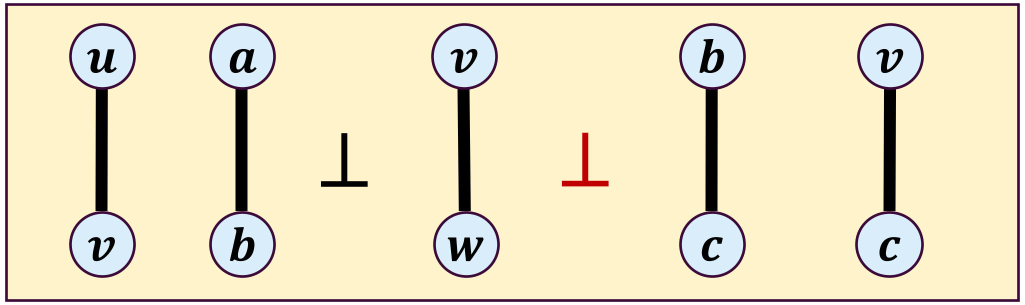
- Given a stream of T **edge updates** (insertions and deletions), produce a vector of T **outputs**, one after every update
  - **Vector of outputs** is **ε-differentially private** on **neighboring streams**
  - Edge-neighboring and node-neighboring streams
  - Focus on **insertion-only streams**

Continual Release Model [DNPR10, CSS11]

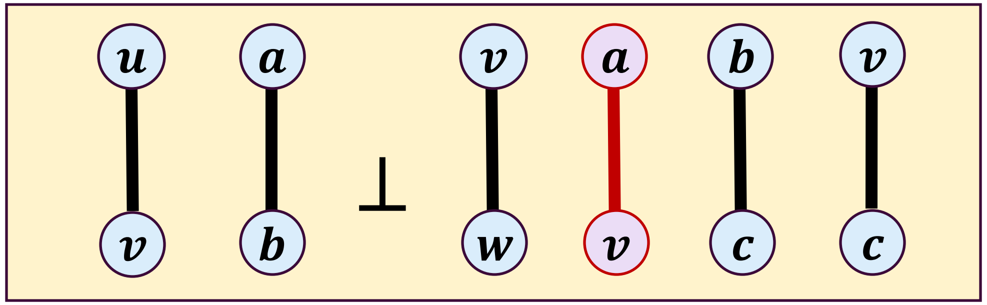
- Given a stream of T **edge updates** (insertions and deletions), produce a vector of T **outputs**, one after every update
  - **Vector of outputs** is **ε-differentially private** on **neighboring streams**
  - Edge-neighboring and node-neighboring streams
  - Focus on **insertion-only streams**
  - An update is either an **edge insertion** or $\perp$

Neighboring Streams

Edge-neighboring streams differ in **1 edge insertion**



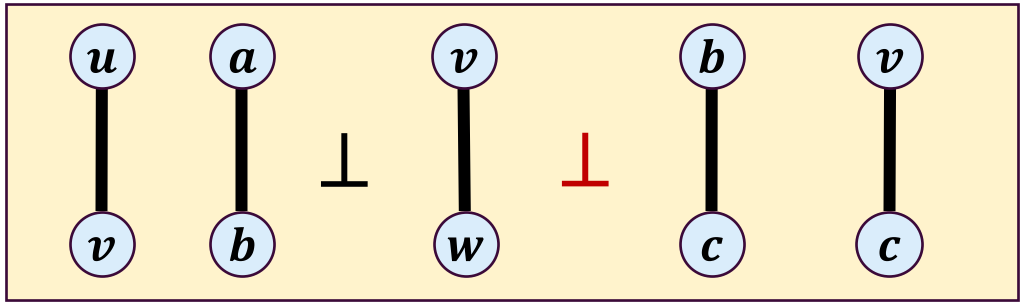
S



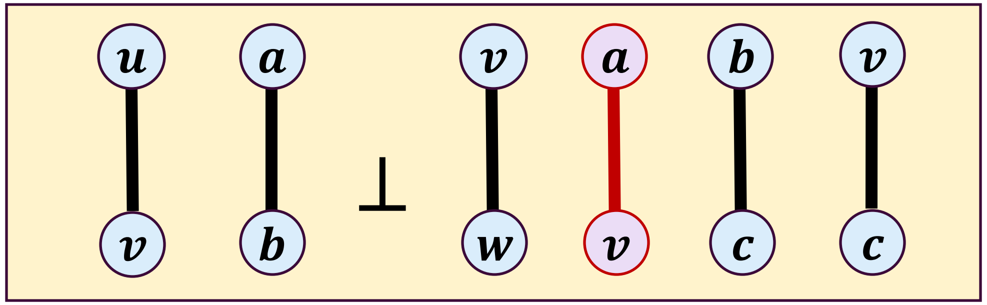
S'

Neighboring Streams

Edge-neighboring streams differ in **1 edge insertion**

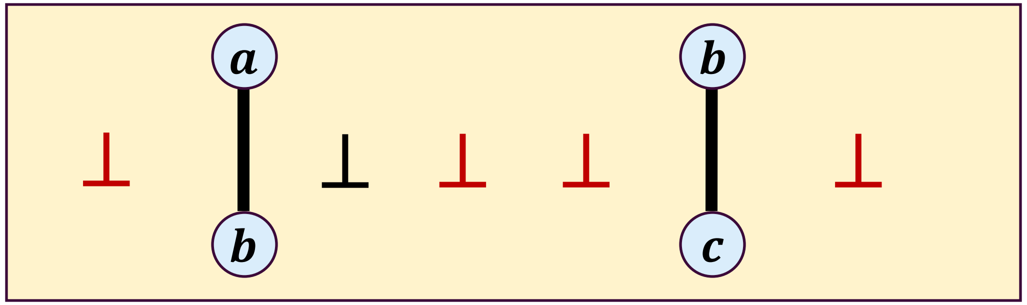


S

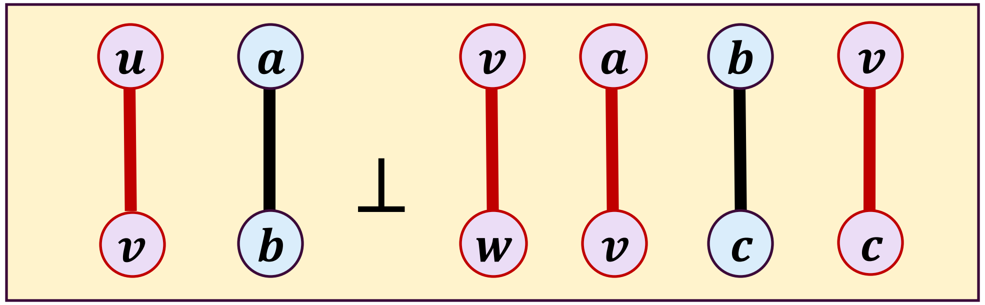


S'

Node-neighboring streams differ in **all edge insertions adjacent to 1 vertex**



S

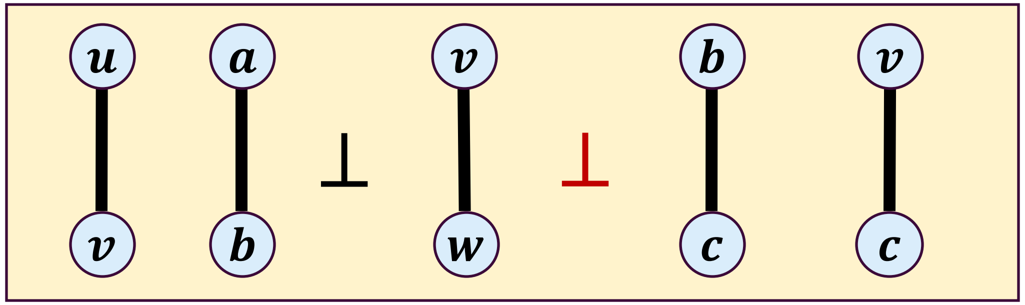


S'

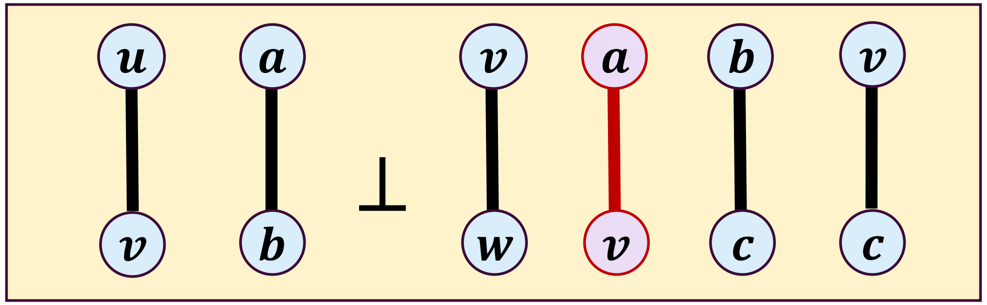
Neighboring Streams

Edge-DP

Edge-neighboring streams differ in **1 edge insertion**



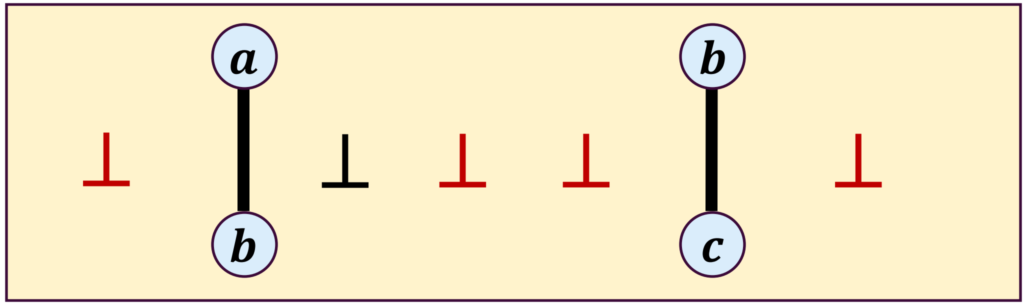
S



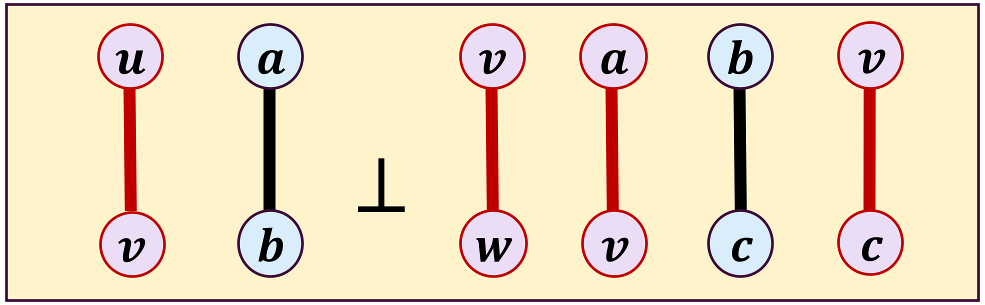
S'

Node-DP

Node-neighboring streams differ in **all edge insertions adjacent to 1 vertex**



S



S'

Challenges in the Continual Release Model

- In static setting, only **one release**

Challenges in the Continual Release Model

- In static setting, only **one release**
- Unlike static setting, T **releases in continual release**

Challenges in the Continual Release Model

- In static setting, only **one release**
- Unlike static setting, T **releases in continual release**
- If edge that differs **occurs early in the stream**, **each release loses privacy**

Challenges in the Continual Release Model

- In static setting, only **one release**
- Unlike static setting, $\boldsymbol{T}$ **releases in continual release**
- If edge that differs **occurs early in the stream**, **each release loses privacy**
- **Composition** over $\boldsymbol{T}$ **releases** could result in $O\left(\frac{T}{\varepsilon}\right)$ error

Previous Work in Graph Continual Release

- Release **numerical valued solutions** for many graph problems [Song-Little-Mehta-Vinterbo-Chaudhuri ‘18, Fichtenberger-Henzinger-Ost ‘21, Jain-Smith-Wagaman ’24]

Previous Work in Graph Continual Release

- Release **numerical valued solutions** for many graph problems [Song-Little-Mehta-Vinterbo-Chaudhuri ‘18, Fichtenberger-Henzinger-Ost ‘21, Jain-Smith-Wagaman ’24]
  - Minimum spanning tree size
  - Minimum cut size
  - Maximum matching size
  - Edge count
  - Degree histogram
  - Triangle count
  - k-star count

Previous Work in Graph Continual Release

- Requires function $f(G_t)$ computing the **exact value** of the counts

Previous Work in Graph Continual Release

- Requires function $f(G_t)$ computing the **exact value** of the counts
  - Function takes induced subgraph t prefix, G_t, and outputs value

Previous Work in Graph Continual Release

- Requires function $f(G_t)$ computing the **exact value** of the counts
  - Function takes induced subgraph t prefix, G_t, and outputs value
  - Compute **difference sequence** $f(G_t) - f(G_{t-1})$ and add Laplace noise with sensitivity of distance sequence [SLMVC18, FHO21]

Previous Work in Graph Continual Release

- Requires function $f(G_t)$ computing the **exact value** of the counts
  - Function takes induced subgraph t prefix, G_t, and outputs value
  - Compute **difference sequence** $f(G_t) - f(G_{t-1})$ and add Laplace noise with sensitivity of distance sequence [SLMVC18, FHO21]
  - **Prefix sum** of difference sequence then is **approximate solution**

Previous Work in Graph Continual Release

- Requires function $f(G_t)$ computing the **exact value** of the counts
  - Function takes induced subgraph t prefix, G_t, and outputs value
  - Compute **difference sequence** $f(G_t) - f(G_{t-1})$ and add Laplace noise with sensitivity of distance sequence [SLMVC18, FHO21]
  - **Prefix sum** of difference sequence then is **approximate solution**
  - Binary tree mechanism and SVT reduces additive error to $\frac{\text{poly}(\log n)}{\varepsilon}$ [FHO21]

Previous Work in Graph Continual Release

- DP on **node-neighboring** streams [SLMVC18, FHO21, JSW24]:

Previous Work in Graph Continual Release

- DP on **node-neighboring** streams [SLMVC18, FHO21, JSW24]:
  - Requires **bounded degree** graph streams for $\mathrm{poly}(\log n)$ additive error [SLMVC18, FHO21]

Previous Work in Graph Continual Release

- DP on **node-neighboring** streams [SLMVC18, FHO21, JSW24]:
  - Requires **bounded degree** graph streams for $\mathrm{poly}(\log n)$ additive error [SLMVC18, FHO21]
  - Or **nearly bounded degree** graph streams where number of nodes with unbounded degree is at most $\mathrm{poly}(\log n)$ [JSW24]

Previous Work in Graph Continual Release

- **Caveat 1:** exact values require storing all edges in stream
  - Cannot achieve **sublinear space guarantees** in the number of edges as in non-private streaming algorithms

Previous Work in Graph Continual Release

- **Caveat 1:** exact values require storing all edges in stream
  - Cannot achieve **sublinear space guarantees** in the number of edges as in non-private streaming algorithms

- **Caveat 2:** Can return only **value of solution** instead of vertex subsets

Previous Work in Graph Continual Release

- **Caveat 1**: exact values require storing all edges in stream
  - Cannot achieve **sublinear space guarantees** in the number of edges as in non-private streaming algorithms

- **Caveat 2**: Can return only **value of solution** instead of vertex subsets

- **Caveat 3**: Can return non-trivial node-privacy guarantees for (nearly) **bounded-degree streams**

Our Contributions

Sublinear space continual release graph algorithms

Our Contributions

Sublinear space continual release graph algorithms
Returns **vertex subset solutions** in continual release

Our Contributions

Sublinear space continual release graph algorithms
Returns **vertex subset solutions** in continual release
Node-private algorithms for **bounded arboricity** graphs in continual release

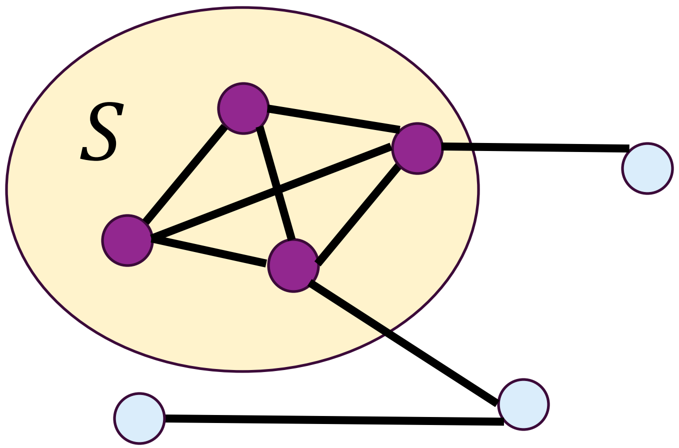
Our Contributions

Sublinear space continual release graph algorithms
Returns **vertex subset solutions** in continual release
Node-private algorithms for **bounded arboricity** graphs in continual release

- First continual release algorithm for k-core decomposition

Our Contributions: Densest Subgraph

- Find an induced subgraph $S \subseteq G$ with maximum induced density, $\max_{S \subseteq G} \left(\frac{E(S)}{V(S)} \right)$



Densest subgraph is S with density $\frac{3}{2}$

Our Contributions: Densest Subgraph

Edge-DP

Our Results

- **Vertex Subset**
- $\tilde{O}\left(\frac{n}{\varepsilon}\right)$ **space**
- **UB:** $\left(1+\eta, \frac{\log^5 n}{\varepsilon}\right)$

Continual Release

[FHO21, JSW24]

- Density value-only
- $\Theta(m)$ space
- **UB**: $\left(1+\eta, \frac{\log^2 n}{\varepsilon}\right)$

Non-Private

[MTVV15, EHW16]

- $(1+\eta, 0)$, $\tilde{O}(n)$

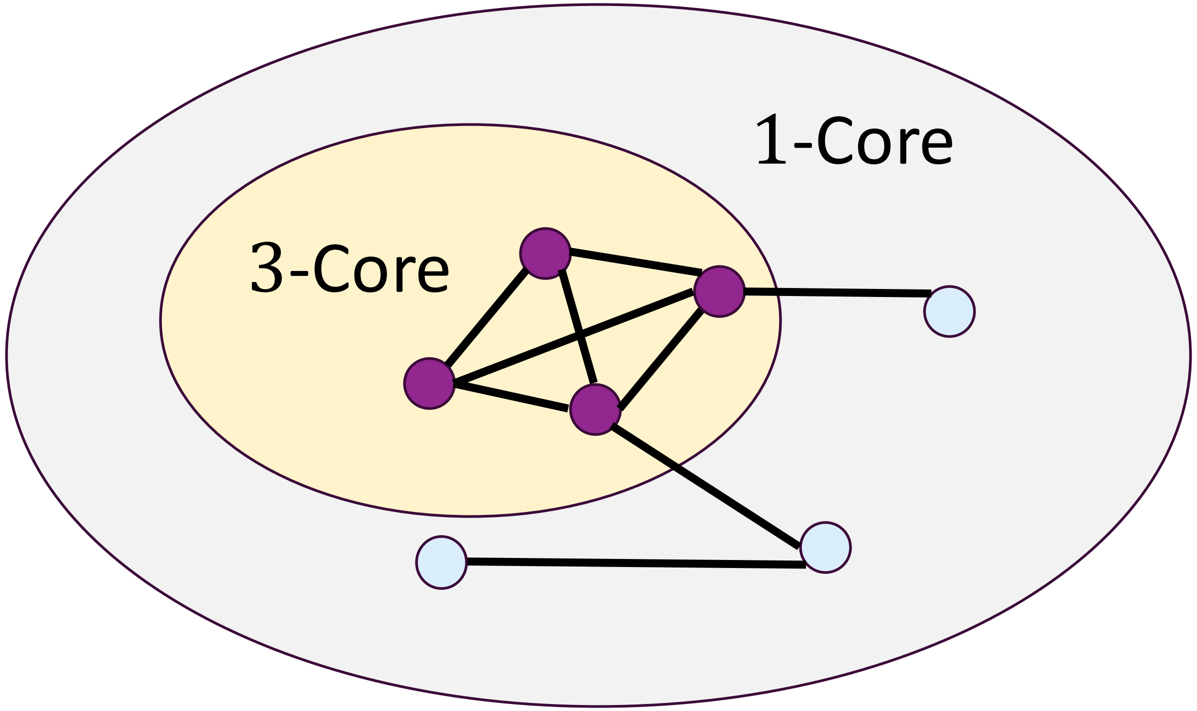
Static

[DLRSS22, DLL23, DKLV24]

- Vertex Subset
- **UB**: $\left(1+\eta, \frac{\log^4 n}{\varepsilon}\right)$
- **LB**: $\left(\beta, \Omega\left(\frac{1}{\beta}\sqrt{\frac{\log(n)}{\varepsilon}}\right)\right)$

Our Contributions: k-Core Decomposition

- Decomposition of nodes of G into cores where each k-core is a maximal induced subgraph with induced degree at least k



Graph contains a 1-core and 3-core

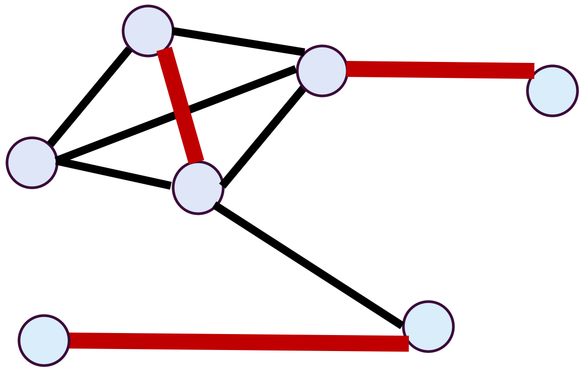
Our Contributions: k-Core Decomposition

Edge-DP

Our Results	Continual Release	Static [DLL23, HSZ24]
• $\tilde{O}\left(\frac{n}{\varepsilon}\right)$ space • UB: $\left(2+\eta, \frac{\log^3 n}{\varepsilon}\right)$	• None **Non-Private** [Esfandiari-Lattanzi-Mirrokni '18] • $(1+\eta, 0)$, $\tilde{O}(n)$ space	• **UB**: $\left(1, O\left(\frac{\log(n)}{\varepsilon}\right)\right)$ • **LB**: $\left(\beta, \Omega\left(\frac{\log(n)}{\varepsilon\beta}\right)\right)$

Our Contributions: Maximum Matching Size

- Find a matching (pairing of nodes where no node is paired with more than one other node) of maximum size

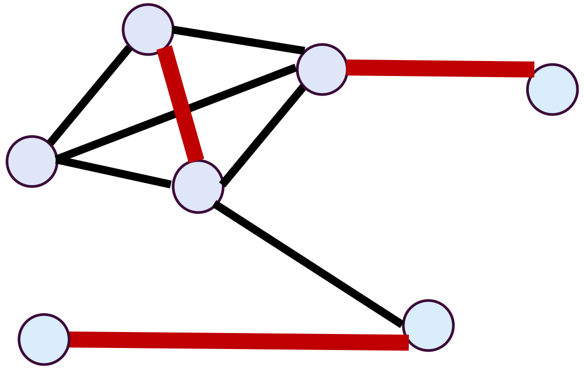


Maximum matching size: 3

Our Contributions: Maximum Matching Size

- Find a matching (pairing of nodes where no node is paired with more than one other node) of maximum size

Cannot differentially privately release set of edges in the matching



Maximum matching size: 3

Our Contributions: Maximum Matching Size

Edge-DP

Our Results

- $O\left(\frac{\text{poly}(\log n)}{\varepsilon}\right)$ **space**
- **UB:** $\left((1+\eta)(2+\tilde{\alpha}), \frac{\log^3 n}{\varepsilon}\right)$

Continual Release

[FHO21, JSW24]

- $\Theta(m)$ space
- **UB**: $\left(1+\eta, \frac{\log^2 n}{\varepsilon}\right)$
- **LB**: $\left(1, \Omega(\log n)\right)$

Non-Private

[McGregor-Voronikova ‘18]

- $(1+\eta)(2+\tilde{\alpha}), O(\log n)$

Our Contributions: Maximum Matching Size

Edge-DP

Our Results

- $O\left(\frac{\log^3 n}{\varepsilon}\right)$ **space**
- **UB:** $\left((1+\eta)(2+\tilde{\alpha}), \frac{\log^3 n}{\varepsilon}\right)$

$\tilde{\alpha}$ is a public bound on the arboricity

Continual Release

[FHO21, JSW24]

- $\Theta(m)$ space
- **UB**: $\left(1+\eta, \frac{\log^2 n}{\varepsilon}\right)$
- **LB**: $\left(1, \Omega(\log n)\right)$

Non-Private

[McGregor-Voronikova '18]

- $(1+\eta)(2+\tilde{\alpha}), O(\log n)$

Our Contributions: Maximum Matching Size

Node-DP

Our Results

- $O(n\widetilde{\alpha})$ **space**
- **UB:** $\left(1+\eta, \frac{\widetilde{\alpha}\log^2 n}{\varepsilon}\right)$

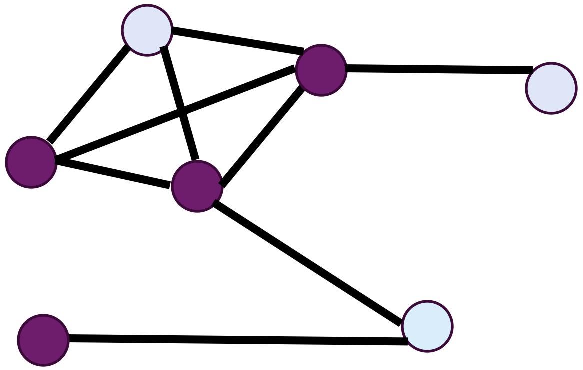
Continual Release

[FHO21, JSW24]

- $\Theta(m)$ space
- **UB**: $\left(1+\eta, \frac{\log^2 n}{\varepsilon}\right)$
- **LB**: $\left(1, \Omega(\log n)\right)$

Our Contributions: Implicit Vertex Cover

- Find a minimum sized set of vertices where every edge has at least one endpoint in the set

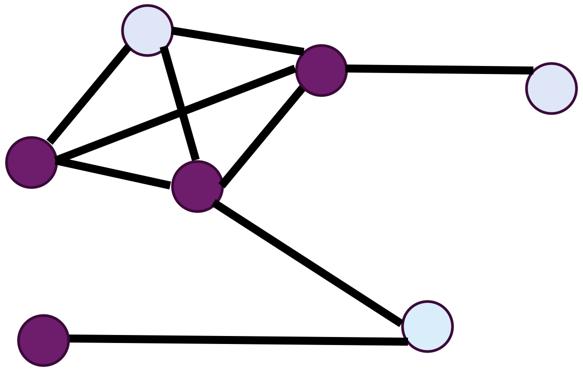


Minimum vertex cover size: 4

Our Contributions: Implicit Vertex Cover

- Find a minimum sized set of vertices where every edge has at least one endpoint in the set

Implicit Vertex Cover releases information such that every edge knows which vertex covers it



Minimum vertex cover size: 4

Our Contributions: Implicit Vertex Cover

Node-DP

Our Results (One Shot)	Continual Release	Static [GLMRT10]

• $O(n\tilde{\alpha})$ space
• **UB:** $\left(3+\eta + O\left(\frac{\tilde{\alpha}}{\varepsilon}\right), O\left(\frac{\tilde{\alpha}\log n}{\varepsilon}\right)\right)$
	• None	• **None** for Node-DP
		• Edge-DP:
		  • **UB:** $\left(2+\frac{16}{\varepsilon}, 0\right)$
		  • **LB:** $\left(\Omega\left(\frac{1}{\varepsilon}\right), 0\right)$

Fully Dynamic Lower Bounds

Edge-DP

Our Results

- **Matching size, triangle count, connected components**
- **LB:** $\left(\mathbf{1}, \min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}\right)\right)$

Continual Release

[FHO21]

- Matching size, triangle count
- **LB**: $(1, \Omega(\log T))$

Main Technique: Graph Sparsification

- Graph sparsification finds **a smaller subgraph** of the input graph

Main Technique: Graph Sparsification

- Graph sparsification finds **a smaller subgraph** of the input graph

- Determine property of the sparsified graph as an **approximate answer** of the original graph

Main Technique: Graph Sparsification

- Graph sparsification finds **a smaller subgraph** of the input graph

- Determine property of the sparsified graph as an **approximate answer** of the original graph

- Sparsification can be deterministic or randomized

Main Technique: Graph Sparsification

- Graph sparsification finds **a smaller subgraph** of the input graph

- Determine property of the sparsified graph as an **approximate answer** of the original graph

- Sparsification can be deterministic or randomized

  - Randomized approaches include various **edge sampling algorithms**

Main Technique: Graph Sparsification

- Previous work used sparsification in DP
  - Static DP setting [Upadhyay ’13, Arora-Upadhyay ‘19]
  - Sliding window DP model [Arora-Upadhyay-Upadhyay ‘21]

Main Technique: Graph Sparsification

- Previous work used sparsification in DP
  - Static DP setting [Upadhyay ’13, Arora-Upadhyay ‘19]
  - Sliding window DP model [Arora-Upadhyay-Upadhyay ‘21]
- Challenging in the continual release model
  - Error can compound over the stream

Main Technique: Graph Sparsification

- Previous work used sparsification in DP
  - Static DP setting [Upadhyay ’13, Arora-Upadhyay ‘19]
  - Sliding window DP model [Arora-Upadhyay-Upadhyay ‘21]
- Challenging in the continual release model
  - Error can compound over the stream
  - **Time-aware projections** [JSW24] takes an arbitrary stream and produces:

Main Technique: Graph Sparsification

- Previous work used sparsification in DP
  - Static DP setting [Upadhyay ’13, Arora-Upadhyay ‘19]
  - Sliding window DP model [Arora-Upadhyay-Upadhyay ‘21]
- Challenging in the continual release model
  - Error can compound over the stream
  - **Time-aware projections** [JSW24] takes an arbitrary stream and produces:
    - Stream satisfying degree bound $\widetilde{D}$

Main Technique: Graph Sparsification

- Previous work used sparsification in DP
  - Static DP setting [Upadhyay '13, Arora-Upadhyay '19]
  - Sliding window DP model [Arora-Upadhyay-Upadhyay '21]
- Challenging in the continual release model
  - Error can compound over the stream
  - **Time-aware projections** [JSW24] takes an arbitrary stream and produces:
    - Stream satisfying degree bound $\widetilde{D}$
    - Identical to every prefix of stream with vertices is $\widetilde{D}$-bounded

Challenges of Sparsification in Continual Release

- Error can compound over the stream

Challenges of Sparsification in Continual Release

- Error can compound over the stream
- Consider simple sparsification procedure:

Challenges of Sparsification in Continual Release

- Error can compound over the stream
- Consider simple sparsification procedure:
  - Remove all edges **adjacent to nodes with degree greater than $\tilde{D}$**, public bound

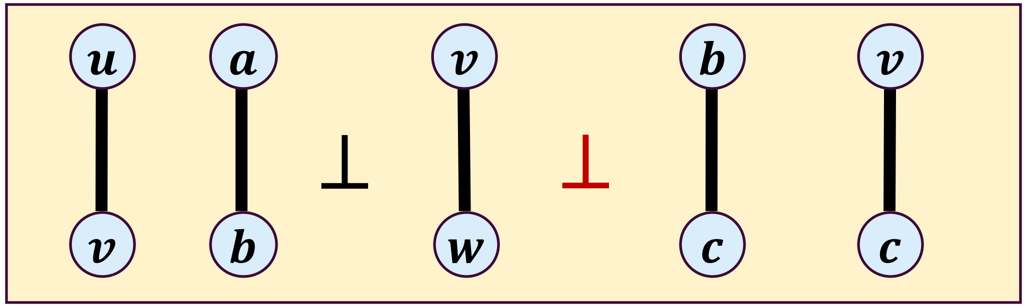
Challenges of Sparsification in Continual Release

- Error can compound over the stream
- Consider simple sparsification procedure:
  - Remove all edges **adjacent to nodes with degree greater than $\widetilde{D}$**, public bound
  - Results in graph with degree upper bounded by $\widetilde{D}$

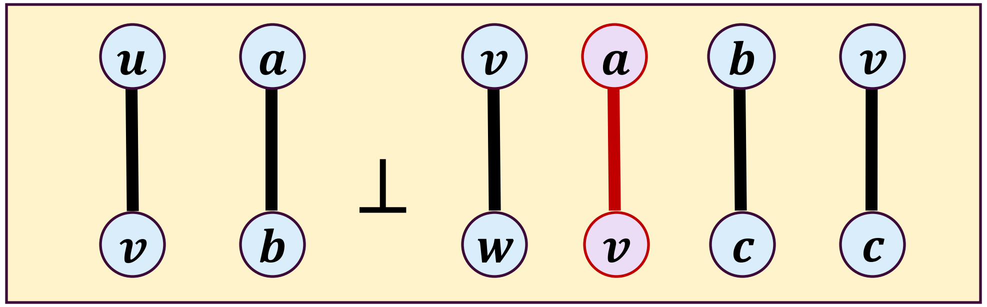
Challenges of Sparsification in Continual Release

- Error can compound over the stream
- Consider simple sparsification procedure:
  - Remove all edges **adjacent to nodes with degree greater than $\widetilde{D}$**, public bound
  - Results in graph with degree upper bounded by $\widetilde{D}$

Edge-neighboring with $\widetilde{D} = 3$



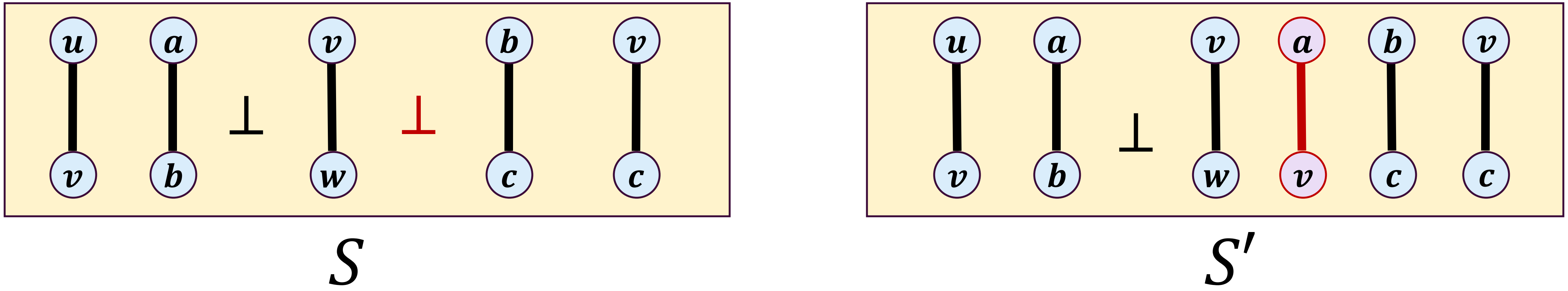
S



S'

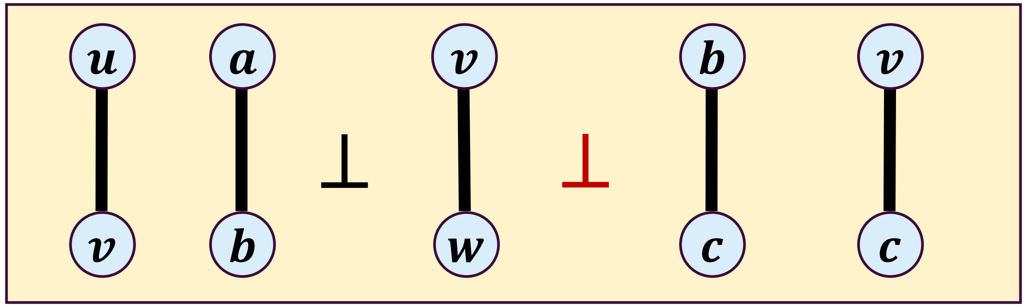
Challenges of Sparsification in Continual Release

Edge-neighboring with $\widetilde{D} = 3$

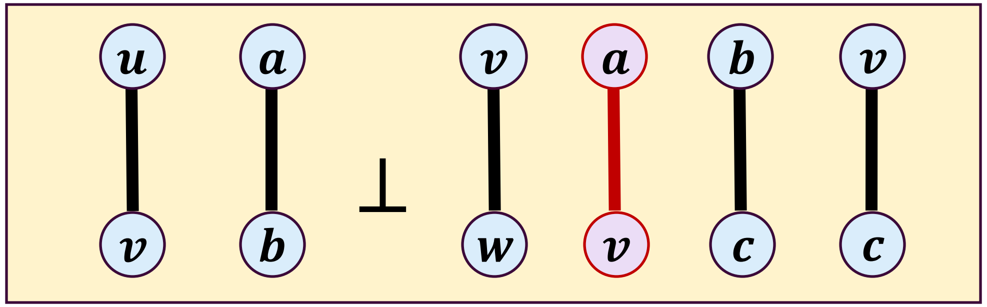


Challenges of Sparsification in Continual Release

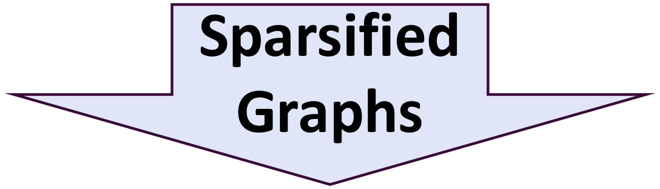
Edge-neighboring with $\widetilde{D} = 3$

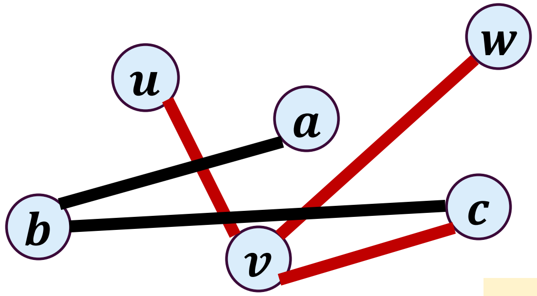


S



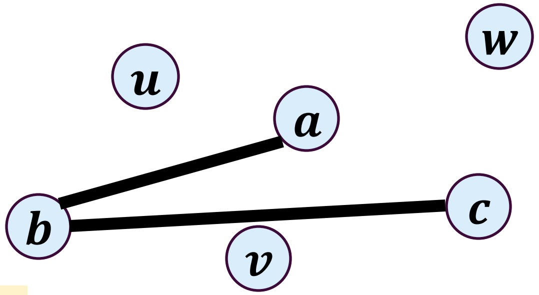
S'





G

Differs by $\widetilde{D}$ edges!



G'

Sparsification Need to Preserve Edge Edit Distance

- **Edge edit distance**: number of update events that differ between two streams

Sparsification Need to Preserve Edge Edit Distance

- **Edge edit distance**: number of update events that differ between two streams
- Sparsified streams should differ by **bounded number of events**:

Sparsification Need to Preserve Edge Edit Distance

- **Edge edit distance**: number of update events that differ between two streams
- Sparsified streams should differ by **bounded number of events**:
  - Deterministic algorithms

Sparsification Need to Preserve Edge Edit Distance

- **Edge edit distance**: number of update events that differ between two streams
- Sparsified streams should differ by **bounded number of events**:
  - Deterministic algorithms
  - **Randomized** sparsification algorithms

Sparsification Need to Preserve Edge Edit Distance

- **Edge edit distance**: number of update events that differ between two streams
- Sparsified streams should differ by **bounded number of events**:
  - Deterministic algorithms
  - **Randomized** sparsification algorithms
    - Exists coupling of randomness where output streams differ by bounded number of events

Sublinear Space Densest Subgraph

- **Our results**:
  - Vertex Subset
  - $\tilde{O}\left(\frac{n}{\varepsilon}\right)$ space
  - UB: $\left(1+\eta, \frac{\text{poly}(\log n)}{\varepsilon}\right)$-approximation

Sublinear Space Densest Subgraph

- Uses uniform sampling idea of [McGregor-Tench-Vorotnikova-Vu ‘15] and [Esfandiari-Hajiaghayi-Woodruff ‘16]

Sublinear Space Densest Subgraph

- Uses uniform sampling idea of [McGregor-Tench-Vorotnikova-Vu ‘15] and [Esfandiari-Hajiaghayi-Woodruff ‘16]

  - Uniformly sample edges from stream

Sublinear Space Densest Subgraph

- Uses uniform sampling idea of [McGregor-Tench-Vorotnikova-Vu ‘15] and [Esfandiari-Hajiaghayi-Woodruff ‘16]
  - Uniformly sample edges from stream
  - Obtain $O(n \log n)$ sized sample via appropriate p

Sublinear Space Densest Subgraph

- Uses uniform sampling idea of [McGregor-Tench-Vorotnikova-Vu ‘15] and [Esfandiari-Hajiaghayi-Woodruff ‘16]
  - Uniformly sample edges from stream
  - Obtain $O(n \log n)$ sized sample via appropriate p
  - Find densest subgraph in sample, return vertex set as densest subgraph in original, scale by $1/p$ for size

Sublinear Space Densest Subgraph

- **Several challenges in continual release**:

Sublinear Space Densest Subgraph

- **Several challenges in continual release**:
  - [MTVV15] and [EHW16] are release solution at **end of the stream**
    - Each edge used ``once” in solution output

Sublinear Space Densest Subgraph

- **Several challenges in continual release**:
  - [MTVV15] and [EHW16] are release solution at **end of the stream**
    - Each edge used ``once" in solution output
  - We need to release **answer at every step**

Sublinear Space Densest Subgraph

- **Several challenges in continual release**:
  - [MTVV15] and [EHW16] are release solution at **end of the stream**
    - Each edge used ``once” in solution output
  - We need to release **answer at every step**
    - **Adaptively** choose sampling probability

Sublinear Space Densest Subgraph

- **Several challenges in continual release**:
  - [MTVV15] and [EHW16] are release solution at **end of the stream**
    - Each edge used ``once" in solution output
  - We need to release **answer at every step**
    - **Adaptively** choose sampling probability
    - Ensure adaptive sampling probability is **edge edit distance preserving**

Sublinear Space Densest Subgraph

- Progressively **decrease edge sampling probability**

Sublinear Space Densest Subgraph

- Progressively **decrease edge sampling probability**
- When number edges exceeds $(1 + \eta) \cdot m_{prev}$

Sublinear Space Densest Subgraph

- Progressively **decrease edge sampling probability**
- When number edges exceeds $(1+\eta)\cdot m_{prev}$
  - Rescale sampling probability to $\Theta\left(\frac{n\log n}{m_{now}}\right)$ where $m_{now} = (1+\eta)\cdot m_{prev}$

Sublinear Space Densest Subgraph

- Progressively **decrease edge sampling probability**
- When number edges exceeds $(1+\eta)\cdot m_{prev}$
  - Rescale sampling probability to $\Theta\left(\frac{n\log n}{m_{now}}\right)$ where $m_{now} = (1+\eta)\cdot m_{prev}$
  - Cannot do this naively: naive scaling could result in **different probabilities** for different edges; not distance preserving

Sublinear Space Densest Subgraph

- Progressively **decrease edge sampling probability**
- When number edges exceeds $(1 + \eta) \cdot m_{prev}$
  - Rescale sampling probability

    $(1 + \eta) \cdot m_{prev}$

Compounding Errors in Continual Release

  - Cannot do this naively: naive scaling could result in **different probabilities** for different edges; not distance preserving

Sublinear Space Densest Subgraph

- Use **sparse vector technique** to determine when to reduce sampling probability

Sublinear Space Densest Subgraph

- Use **sparse vector technique** to determine when to reduce sampling probability
  - **Sparse vector technique:** DP technique for determining when a query **exceeds a threshold**

Sublinear Space Densest Subgraph

- Use **sparse vector technique** to determine when to reduce sampling probability
  - **Sparse vector technique:** DP technique for determining when a query **exceeds a threshold**
  - **Loses privacy proportional to number of times threshold exceeded**

Sublinear Space Densest Subgraph

- Use **sparse vector technique** to determine when to reduce sampling probability
  - **Sparse vector technique:** DP technique for determining when a query **exceeds a threshold**
  - **Loses privacy proportional to number of times threshold exceeded**

- Hence, coupling exists between sampling probabilities

Sublinear Space Densest Subgraph

- Use **sparse vector technique** to determine when to reduce sampling probability
  - **Sparse vector technique:** DP technique for determining when a query **exceeds a threshold**
  - **Loses privacy proportional to number of times threshold exceeded**
- Hence, coupling exists between sampling probabilities
  - Sampled edges preserve edge edit distance

Sublinear Space Densest Subgraph

- Finally, additional challenge:
  - Additive noise of $O\left(\frac{\log n}{\varepsilon}\right)$

Sublinear Space Densest Subgraph

- Finally, additional challenge:
  - Additive noise of $O\left(\frac{\log n}{\varepsilon}\right)$
  - Need to ensure noise **does not increase multiplicative approximation**

Sublinear Space Densest Subgraph

- Finally, additional challenge:
  - Additive noise of $O\left(\frac{\log n}{\varepsilon}\right)$
  - Need to ensure noise **does not increase multiplicative approximation**
  - Scaling up the additive error by $\left(\frac{1}{p}\right)$-factor

Sublinear Space Densest Subgraph

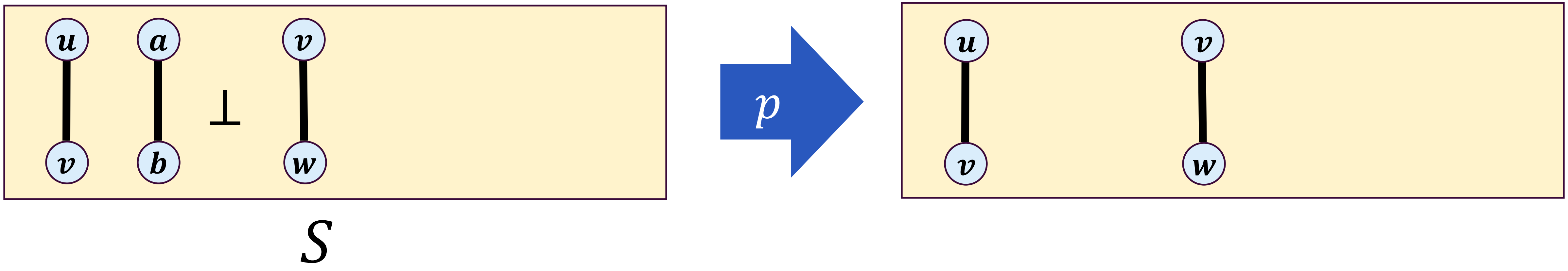
- Finally, additional challenge:
  - Additive noise of $O\left(\frac{\log n}{\varepsilon}\right)$
  - Need to ensure noise **does not increase multiplicative approximation**
  - Scaling up the additive error by $\left(\frac{1}{p}\right)$-factor
  - **Additive error becomes $O\left(\frac{\log n}{p \cdot \varepsilon}\right)$**

Sublinear Space Densest Subgraph

- Finally, additional challenge:
  - Additive noise of $O\left(\frac{\log n}{\varepsilon}\right)$
  - Need to ensure noise **does not increase multiplicative approximation**
  - Scaling up the additive error by $\left(\frac{1}{p}\right)$-factor
  - **Additive error becomes $O\left(\frac{\log n}{p \cdot \varepsilon}\right)$**
  - **Solution**: Ensure returned densest subgraph has large enough size

Sublinear Space Densest Subgraph

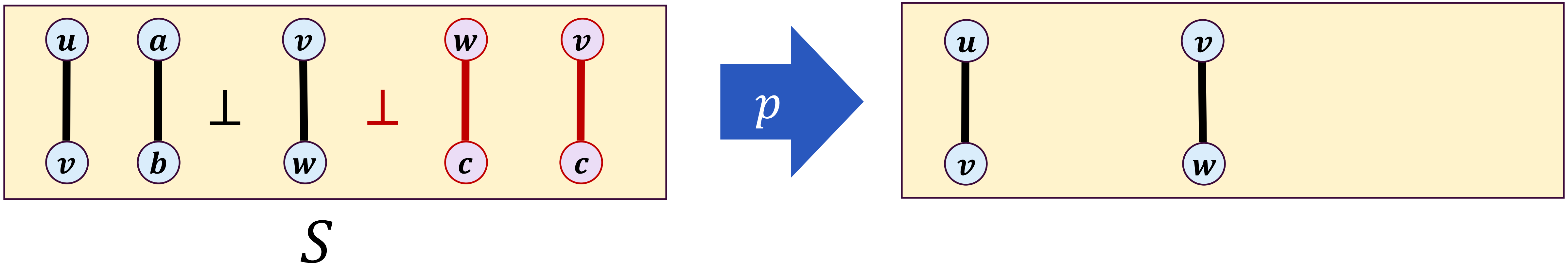
Putting it Together



Sample each edge update with probability $p = \Theta\left(\frac{n \log n}{m'}\right)$

Sublinear Space Densest Subgraph

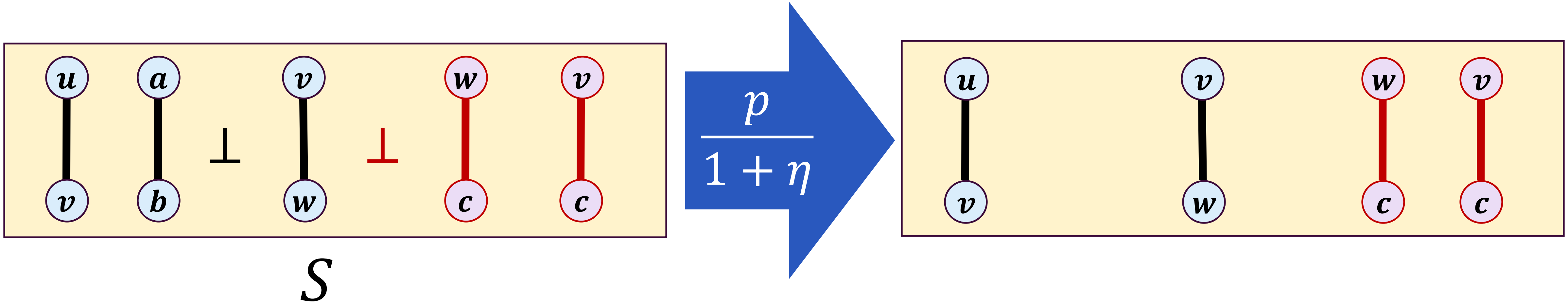
Putting it Together



Use SVT to determine when threshold of number of edges seen exceeds $(1 + \eta)m'$

Sublinear Space Densest Subgraph

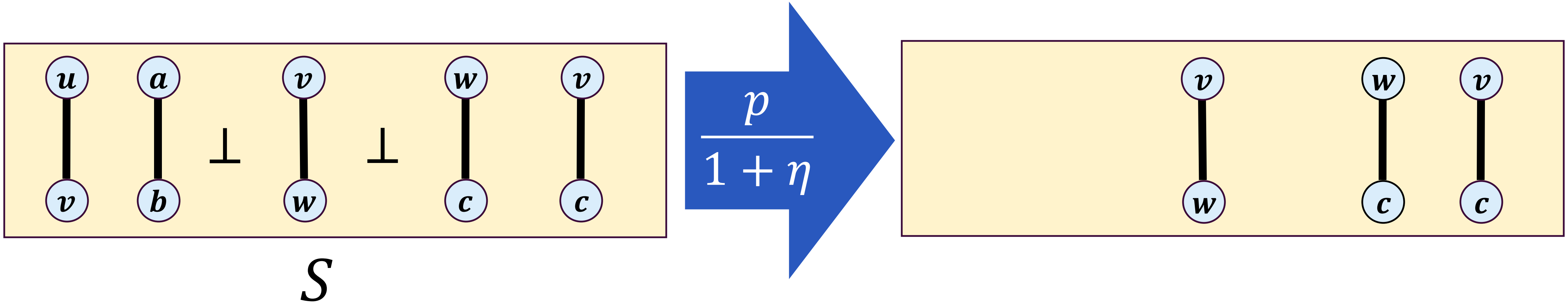
Putting it Together



If SVT is satisfied decrease probability by $(1+\eta)$ factor

Sublinear Space Densest Subgraph

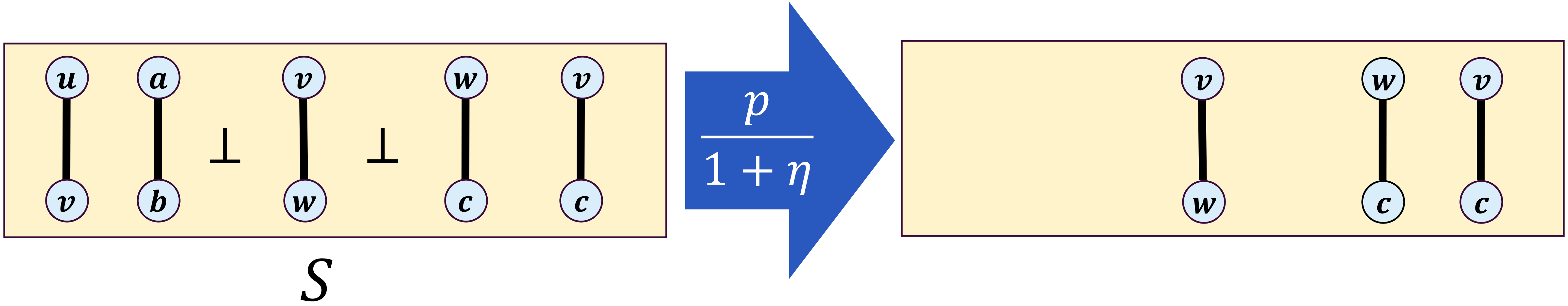
Putting it Together



Resample existing sampled edges with probability $\frac{1}{1+\eta}$

Sublinear Space Densest Subgraph

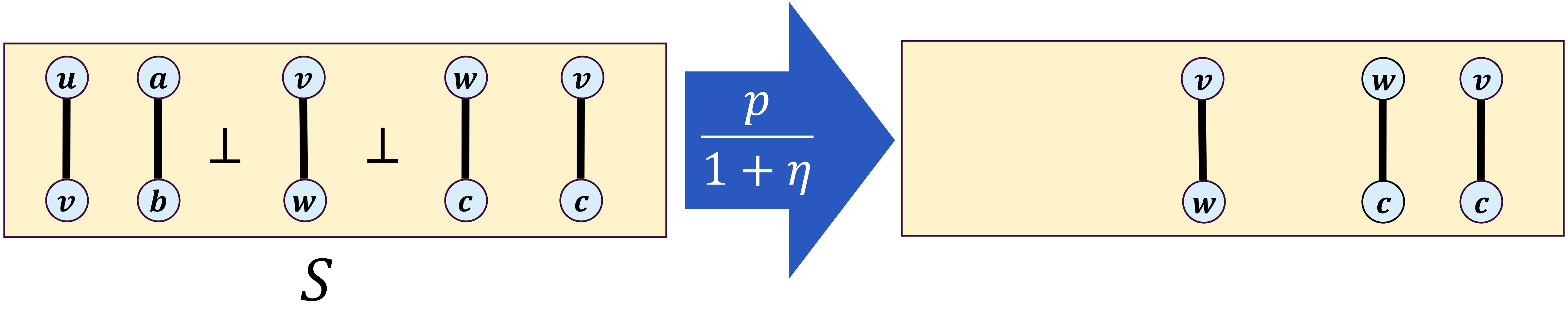
Putting it Together



Use ε-DP algorithm for each sample to determine released solution at **appropriate timestamps**

Sublinear Space Densest Subgraph

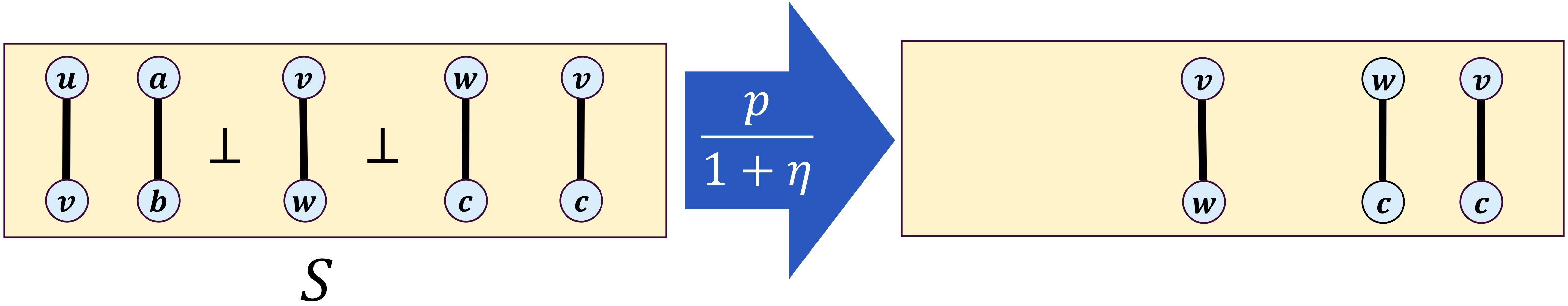
Putting it Together



Use SVT to determine if densest subgraph **increased in value by $(1+\eta)$-factor**

Sublinear Space Densest Subgraph

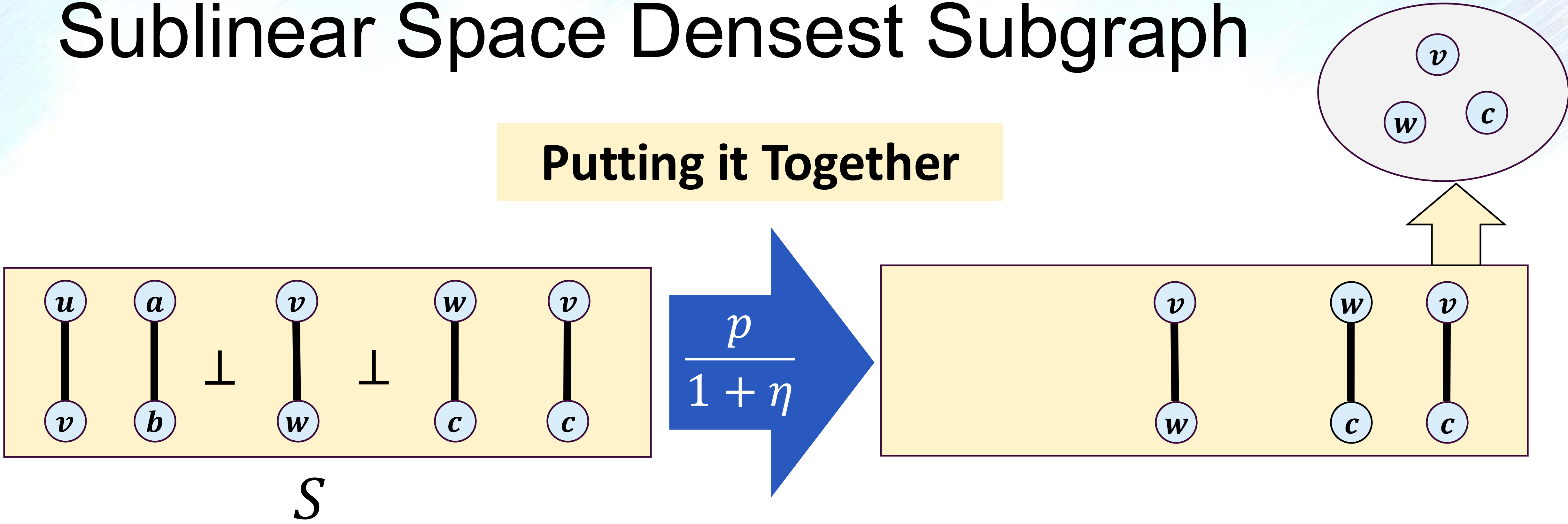
Putting it Together



Only release when densest subgraph **increased in value by $(1+\eta)$-factor**

Sublinear Space Densest Subgraph

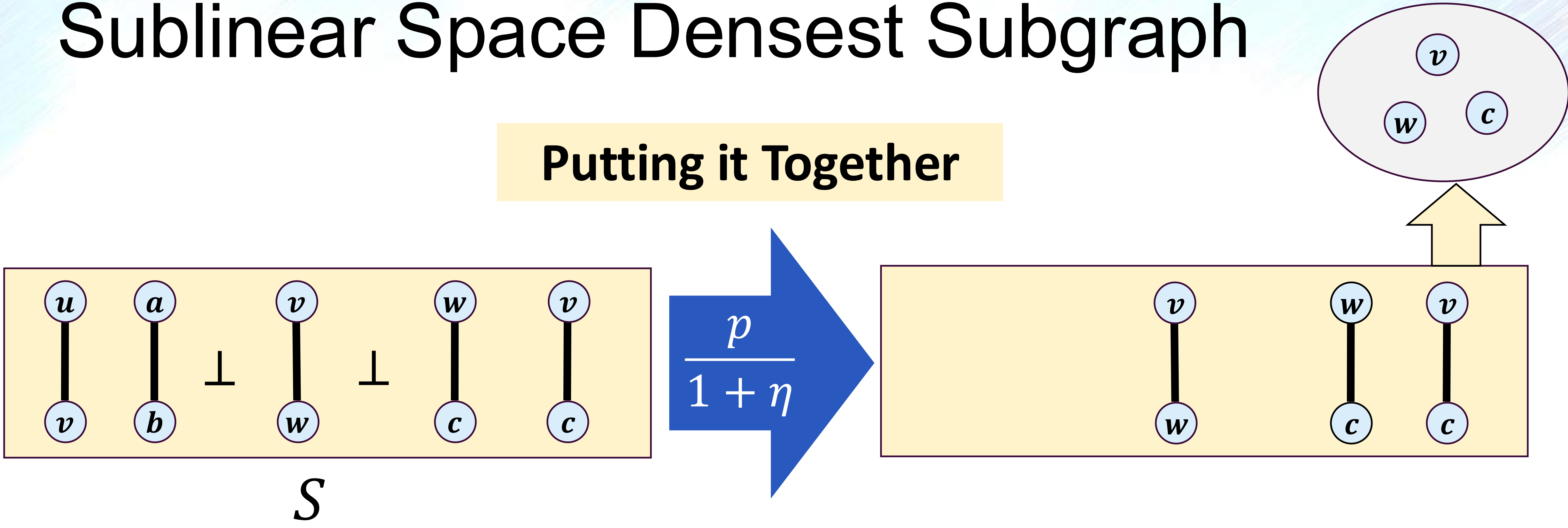
Putting it Together



Only release subset of vertices when densest subgraph **increased in value by $(1+\eta)$-factor**

Sublinear Space Densest Subgraph

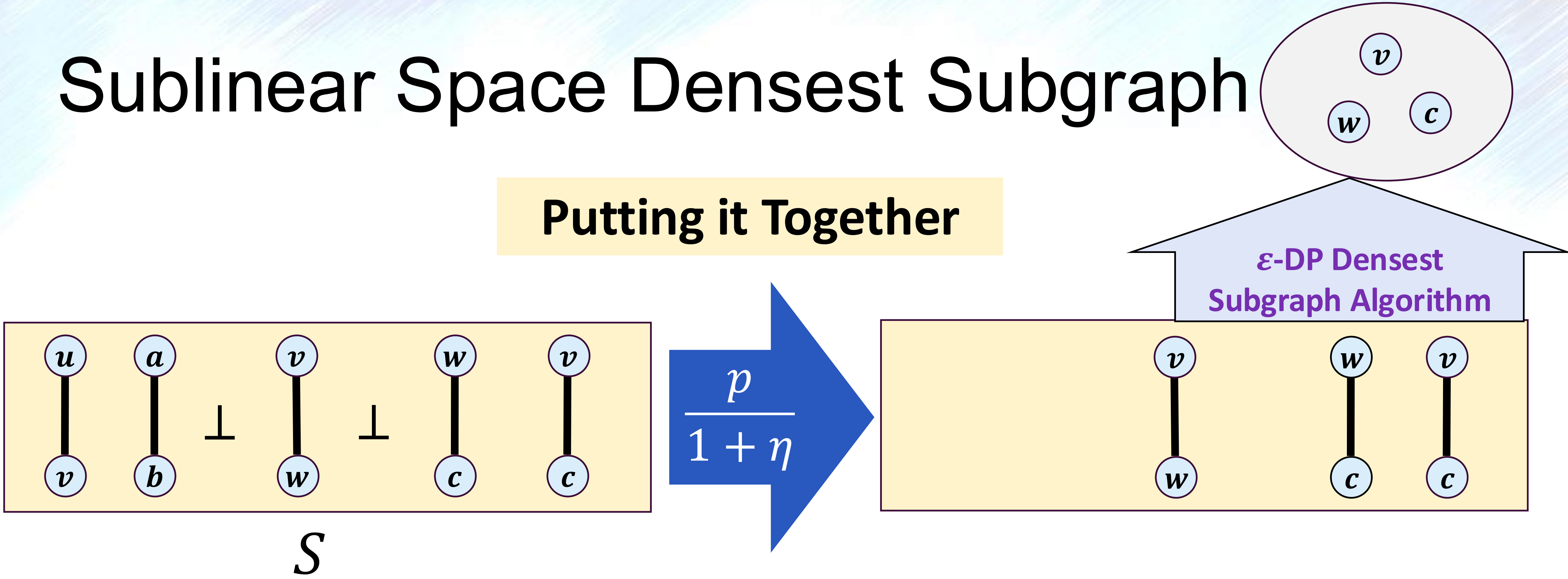
Putting it Together



Scale value of densest subgraph by **inverse of current sampling probability**

Sublinear Space Densest Subgraph

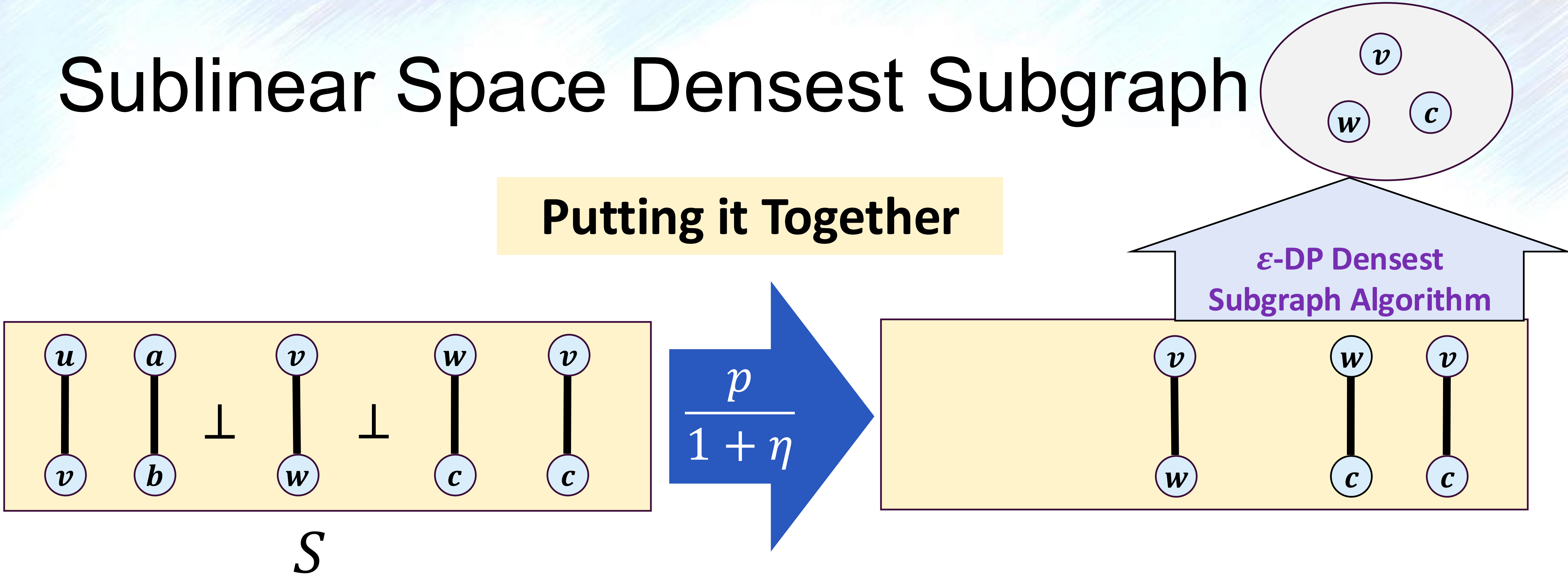
Putting it Together



Use any ε-DP (static) Densest Subgraph algorithm for determining subset of vertices to release

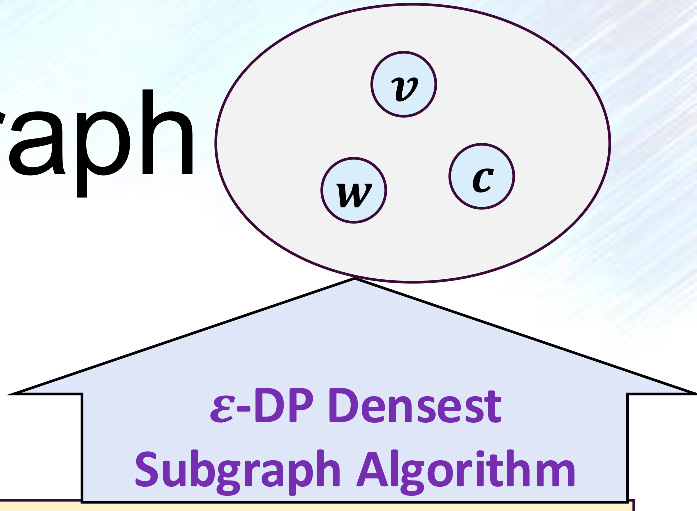
Sublinear Space Densest Subgraph

Putting it Together

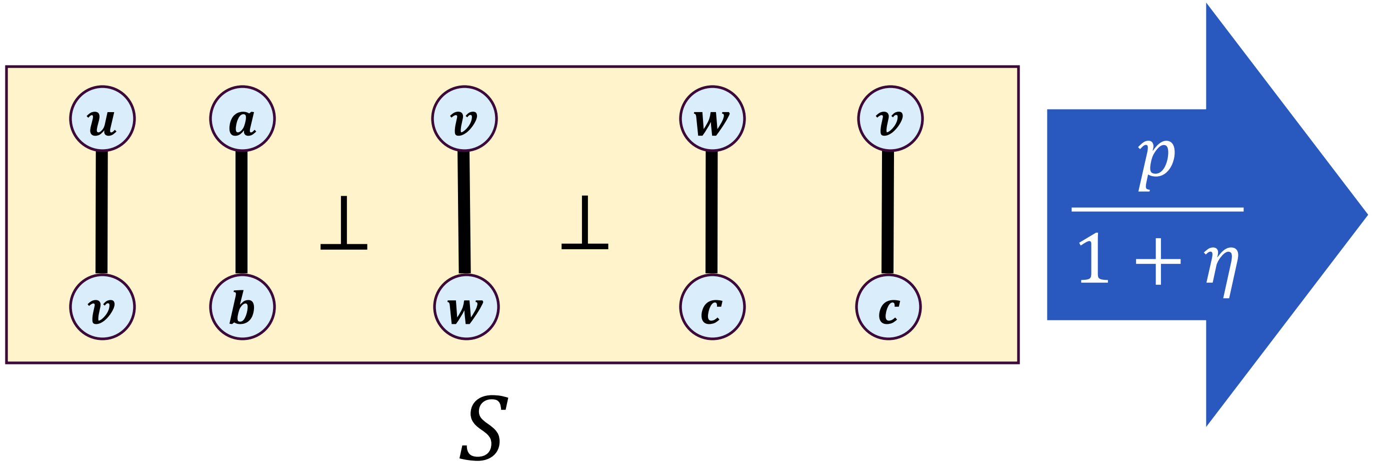


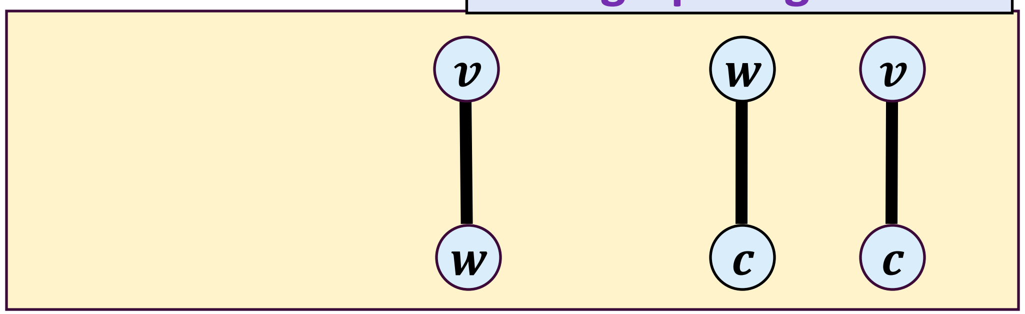
Set first non-trivial initial release for density to be $\Omega\left(\frac{\log^2 n}{\varepsilon}\right)$ to account for DP alg additive error

Sublinear Space Densest Subgraph



Putting it Together

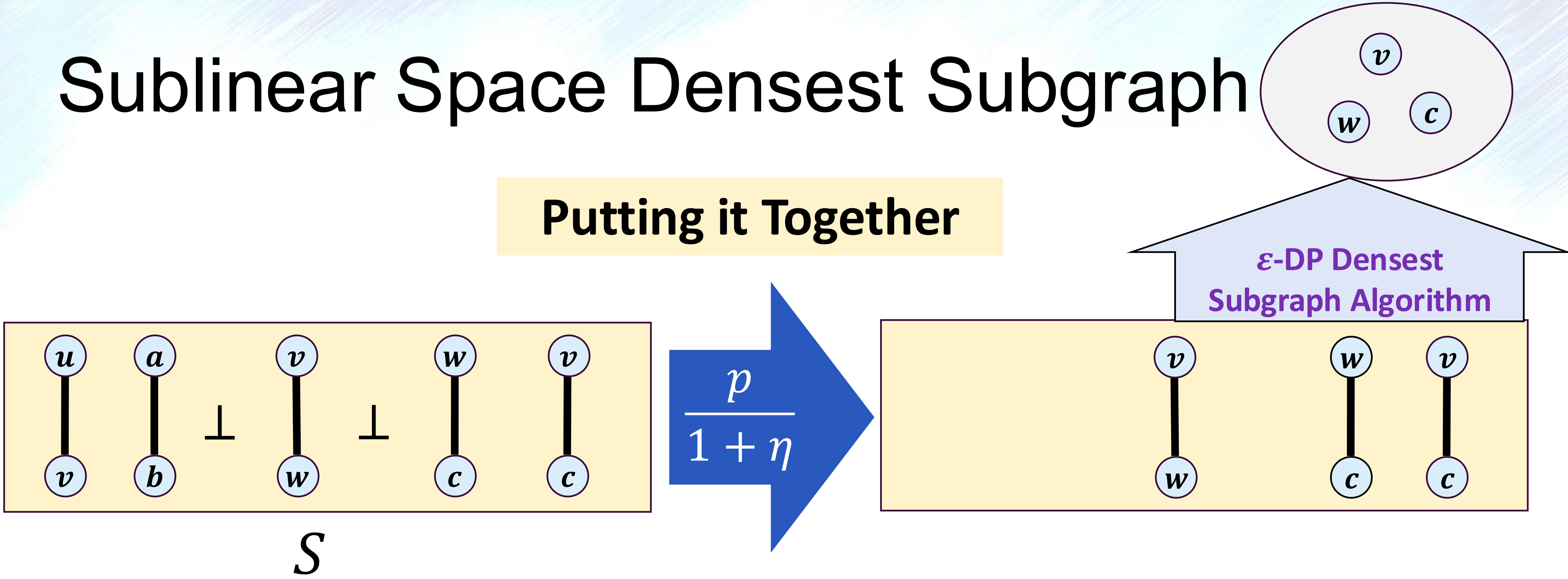




ε-DP Guarantee from DP of SVT, Edge Edit Distance is preserved, and Composition

Sublinear Space Densest Subgraph

Putting it Together



Approximation guarantee from very intricate **Chernoff Bound argument accounting for errors from SVT and DP algorithm**

Sublinear Space Densest Subgraph

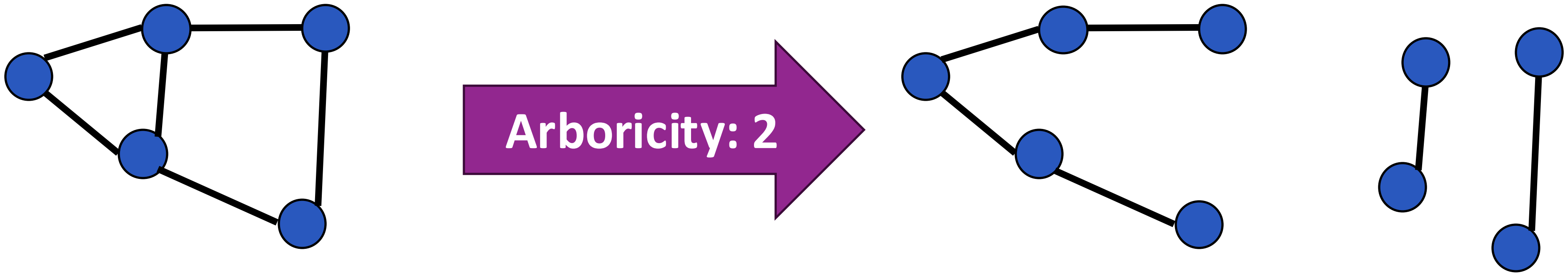
ε-DP Densest
graph Algorithm

One Takeaway: adaptive uniform sampling with SVT is a sublinear simplification in DP that is edge distance preserving

Approximation guarantee from very intricate **Chernoff Bound argument accounting for errors from SVT and DP algorithm**

Node-DP Maximum Matching

- **Arboricity sparsification**: sparsification using upper bound based on the arboricity α
  - **Arboricity:** minimum number of forests to decompose a graph
    - Measure of local sparsity



Node-DP Maximum Matching

- **Arboricity sparsification**: sparsification using upper bound based on the arboricity α
- Solomon '16 obtained $O(n\alpha)$ **sparsifier for maximum matching**:

Node-DP Maximum Matching

- **Arboricity sparsification**: sparsification using upper bound based on the arboricity α
- Solomon '16 obtained $O(n\alpha)$ **sparsifier for maximum matching**:
    - Mark $\Lambda = O\left(\frac{\alpha}{\eta}\right)$ arbitrary edges adjacent to every vertex

Node-DP Maximum Matching

- **Arboricity sparsification**: sparsification using upper bound based on the arboricity α
- Solomon '16 obtained $O(n\alpha)$ **sparsifier for maximum matching**:
  - Mark $\Lambda = O\left(\frac{\alpha}{\eta}\right)$ arbitrary edges adjacent to every vertex
  - Keep edges marked by both endpoints

Node-DP Maximum Matching

- **Arboricity sparsification**: sparsification using upper bound based on the arboricity α
- Solomon '16 obtained $O(n\alpha)$ **sparsifier for maximum matching**:
  - Mark $\Lambda = O\left(\frac{\alpha}{\eta}\right)$ arbitrary edges adjacent to every vertex
  - Keep edges marked by both endpoints
  - Matching in sparsified graph is a $(1+\eta)$-approximation of the maximum matching in the original graph

Node-DP Maximum Matching

- In the streaming model, mark the **first $\tilde{\alpha}$ edges** incident to every vertex where **$\tilde{\alpha}$ is public bound on max arboricity**

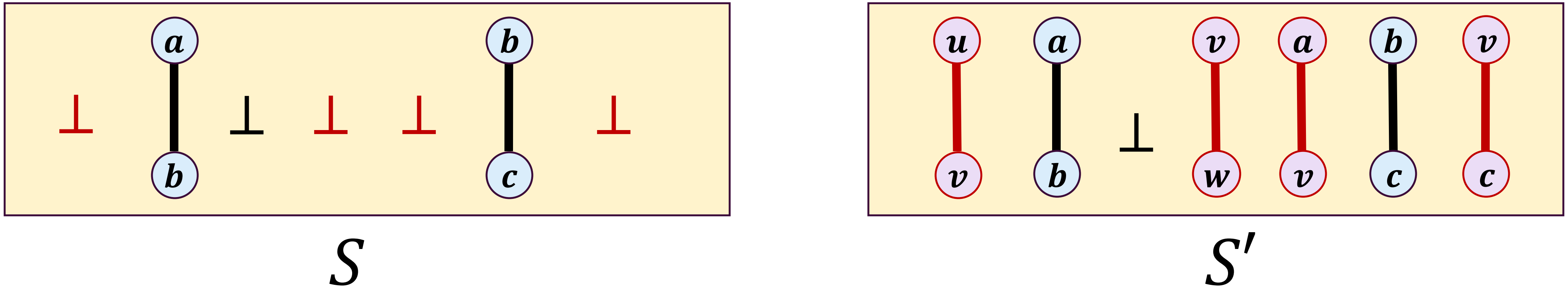
Node-DP Maximum Matching

- In the streaming model, mark the **first $\tilde{\alpha}$ edges** incident to every vertex where $\tilde{\alpha}$ **is public bound on max arboricity**
  - **Use SVT** to determine when to release a new matching size

Node-DP Maximum Matching

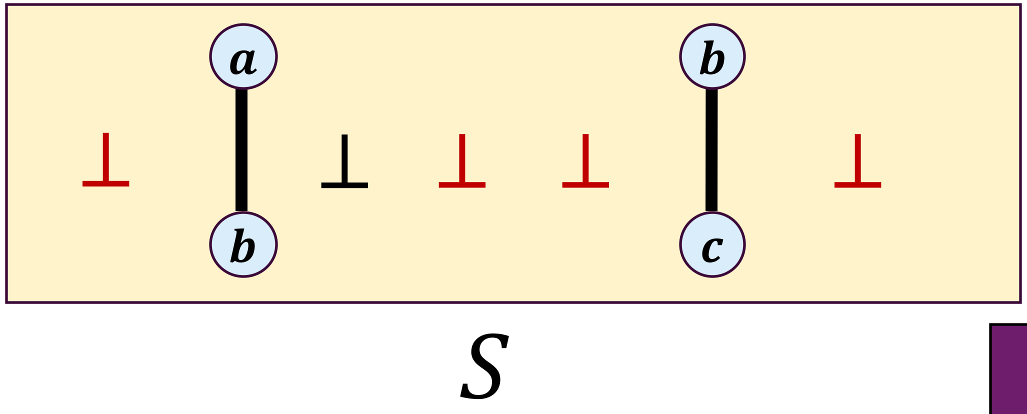
- In the streaming model, mark the **first $\tilde{\alpha}$ edges** incident to every vertex where **$\tilde{\alpha}$ is public bound on max arboricity**
  - **Use SVT** to determine when to release a new matching size

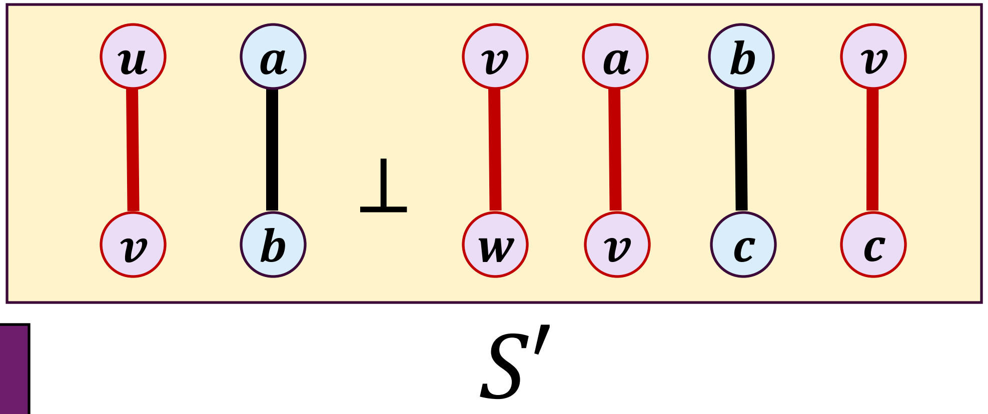
Node-neighboring streams and given $\tilde{\alpha} = 2$

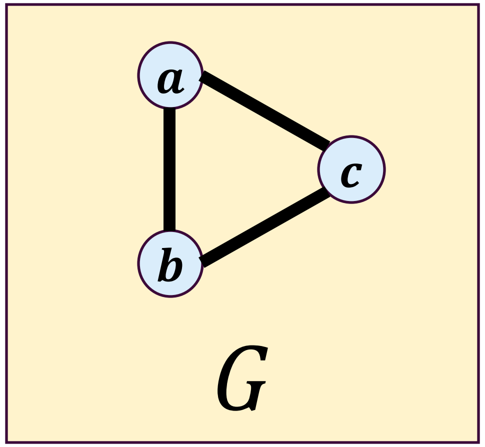


Node-DP Maximum Matching

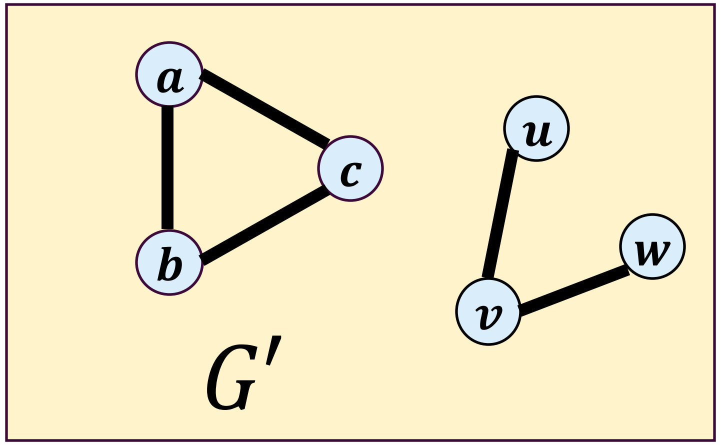
Node-neighboring streams and given $\tilde{\alpha} = 2$







Edge edit distance is $\tilde{\alpha}$



Node-DP Maximum Matching

- In the streaming model, mark the **first $\tilde{\alpha}$ edges** incident to every vertex where **$\tilde{\alpha}$ is public bound on max arboricity**
  - **Use SVT** with **sensitivity $O(\tilde{\alpha})$** to determine when to release a new matching size

Node-DP Maximum Matching

- In the streaming model, mark the **first $\tilde{\alpha}$ edges** incident to every vertex where **$\tilde{\alpha}$ is public bound on max arboricity**
  - **Use SVT** with **sensitivity $O(\tilde{\alpha})$** to determine when to release a new matching size
  - Sensitivity of Λ in SVT translates to $\frac{\Lambda \operatorname{poly}(\log n)}{\varepsilon}$ additive error

Node-DP Maximum Matching

- In the streaming model, mark the **first $\tilde{\alpha}$ edges** incident to every vertex where **$\tilde{\alpha}$ is public bound on max arboricity**
  - **Use SVT** with **sensitivity $O(\tilde{\alpha})$** to determine when to release a new matching size
  - Sensitivity of Λ in SVT translates to $\frac{\Lambda \operatorname{poly}(\log n)}{\varepsilon}$ additive error
  - **Our result:** $O(n\tilde{\alpha})$ space with $\left(1+\eta, \frac{\tilde{\alpha} \operatorname{poly}(\log n)}{\varepsilon}\right)$-approximation

Node-DP Maximum Matching

- In the stre … nt to every vertex whe …

  - **Use SV** … o release a new ma …

  - Sensitiv … ive error

  - **Our result:** $O(n\tilde{\alpha})$ space with $\left(1+\eta, \frac{\tilde{\alpha}\,\mathrm{poly}(\log n)}{\varepsilon}\right)$-approximation

One Takeaway: arboricity sparsification for node-DP applies to vertex cover and DP in the static setting

Open Questions

- Closing the gap for fully dynamic streams

Open Questions

- Closing the gap for fully dynamic streams
- **Space lower bounds** in addition to error lower bounds

Open Questions

- Closing the gap for fully dynamic streams
- **Space lower bounds** in addition to error lower bounds
  - Many of our space guarantees include a factor of $O\left(\frac{1}{\varepsilon}\right)$

Open Questions

- Closing the gap for fully dynamic streams
- **Space lower bounds** in addition to error lower bounds
  - Many of our space guarantees include a factor of $O\left(\frac{1}{\varepsilon}\right)$
  - This factor is not present in non-private streaming algorithms

Open Questions

- Closing the gap for fully dynamic streams
- **Space lower bounds** in addition to error lower bounds
  - Many of our space guarantees include a factor of $O\left(\frac{1}{\varepsilon}\right)$
  - This factor is not present in non-private streaming algorithms
  - Is this factor necessary?