Randomized Composable Coreset for Matching and Vertex Cover

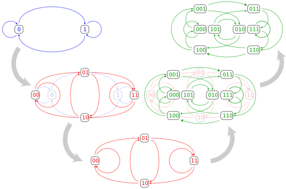
Sepehr Assadi

University of Pennsylvania

Joint work with Sanjeev Khanna (Penn)

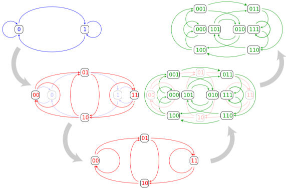
Massive Graphs

Massive graphs abound in variety of applications: web graph, social networks, biological networks, etc.



Massive Graphs

Massive graphs abound in variety of applications: web graph, social networks, biological networks, etc.



How to deal with computation over such massive graph inputs?

Distributed Computing

A common approach: distributed computing.

Distributed Computing

A common approach: distributed computing.

1. Distribute the edges of the graph between some machines.

Distributed Computing

A common approach: distributed computing.

1. Distribute the edges of the graph between some machines.
2. There is a communication network between the machines.

Distributed Computing

A common approach: distributed computing.

1. Distribute the edges of the graph between some machines.
2. There is a communication network between the machines.
3. The machines communicate with each other to compute the answer.

Distributed Computing

A common approach: distributed computing.

1. Distribute the edges of the graph between some machines.
2. There is a communication network between the machines.
3. The machines communicate with each other to compute the answer.

Main measures of efficiency: communication cost and round complexity.

The Simultaneous Communication Model

We are interested in the simultaneous communication model.

The Simultaneous Communication Model

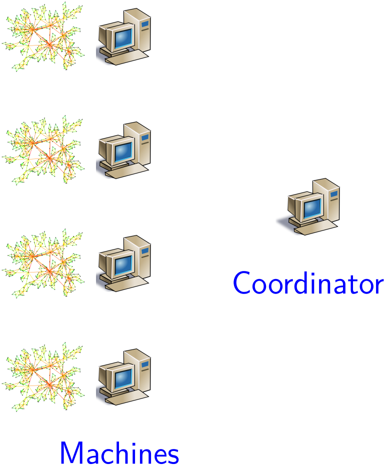
We are interested in the simultaneous communication model.

1. There are k machines plus an additional coordinator.

The Simultaneous Communication Model

We are interested in the simultaneous communication model.

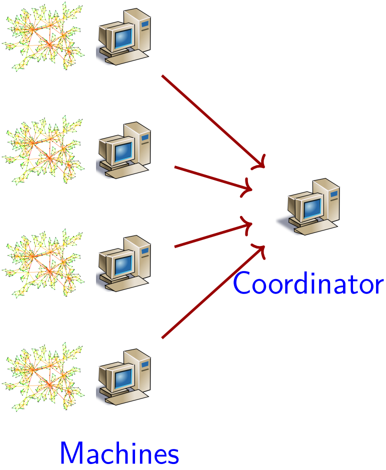
1. There are k machines plus an additional coordinator.
2. The input graph is edge-partitioned between the machines.



The Simultaneous Communication Model

We are interested in the simultaneous communication model.

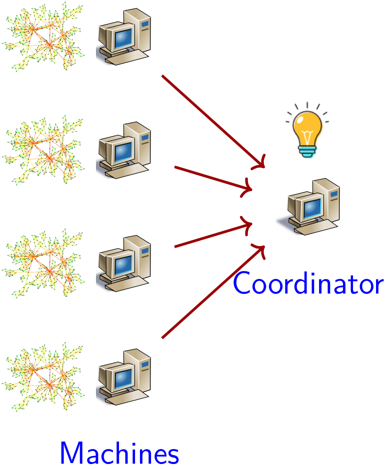
1. There are k machines plus an additional coordinator.
2. The input graph is edge-partitioned between the machines.
3. Each machine sends a summary of its input to the coordinator.



The Simultaneous Communication Model

We are interested in the simultaneous communication model.

1. There are k machines plus an additional coordinator.
2. The input graph is edge-partitioned between the machines.
3. Each machine sends a summary of its input to the coordinator.
4. The coordinator computes the answer based on the summaries.



Why Simultaneous Model?

1. Simultaneous protocols are inherently round-optimal.

Why Simultaneous Model?

1. Simultaneous protocols are inherently round-optimal.

2. Communication cost is simply determined by the size of the summary sent by each machine.

Why Simultaneous Model?

1. Simultaneous protocols are inherently round-optimal.
2. Communication cost is simply determined by the size of the summary sent by each machine.
3. Applications to other models of computation:
   - For example, lower bounds in dynamic streams.

Simultaneous Protocols

Many general techniques for designing simultaneous protocols, including:

- Linear sketches
- Composable coresets
- Mergable summaries
- Sampling

...

Linear Sketches

- We treat the input graph as a vector of edge multiplicities.

Linear Sketches

- We treat the input graph as a vector of edge multiplicities.
- Summary of each machine is a linear projection of its input subgraph.

Linear Sketches

- We treat the input graph as a vector of edge multiplicities.
- Summary of each machine is a linear projection of its input subgraph.
- Linearity of the sketches allows the coordinator to obtain a sketch of the combined input.

Linear Sketches

- We treat the input graph as a vector of edge multiplicities.
- Summary of each machine is a linear projection of its input subgraph.
- Linearity of the sketches allows the coordinator to obtain a sketch of the combined input.
- The coordinator runs an arbitrary function on the combined sketch to obtain the final answer.

Linear Sketches

- We treat the input graph as a vector of edge multiplicities.
- Summary of each machine is a linear projection of its input subgraph.
- Linearity of the sketches allows the coordinator to obtain a sketch of the combined input.
- The coordinator runs an arbitrary function on the combined sketch to obtain the final answer.

Introduced for graph problems by Ahn, Guha, and McGregor [Ahn et al., 2012a].

Composable Coresets

- Summary of each machine is a suitably chosen subgraph of its input.

Composable Coresets

- Summary of each machine is a suitably chosen subgraph of its input.
- Composability means that the union of the coresets for a collection of graphs yields a coreset for the union of the graphs.

Composable Coresets

- Summary of each machine is a suitably chosen subgraph of its input.
- Composability means that the union of the coresets for a collection of graphs yields a coreset for the union of the graphs.
- The coordinator solves the original problem over the combined coreset to obtain the final answer.

Composable Coresets

- Summary of each machine is a suitably chosen subgraph of its input.
- Composability means that the union of the coresets for a collection of graphs yields a coreset for the union of the graphs.
- The coordinator solves the original problem over the combined coreset to obtain the final answer.

Introduced by Indyk, Mahabadi, Mahdian, and Mirrokni [Indyk et al., 2014].

Composable Coresets

- Summary of each machine is a suitably chosen subgraph of its input.
- Composability means that the union of the coresets for a collection of graphs yields a coreset for the union of the graphs.
- The coordinator solves the original problem over the combined coreset to obtain the final answer.

Introduced by Indyk, Mahabadi, Mahdian, and Mirrokni [Indyk et al., 2014].

Many graph problems admit natural composable coresets; for instance, connectivity, sparsifiers, and spanners.

Previous Work

Successful applications of these two techniques have yielded $\widetilde{O}(n)$ size summaries for several graph problems:

Previous Work

Successful applications of these two techniques have yielded $\widetilde{O}(n)$ size summaries for several graph problems:

Connectivity, Minimum Spanning Tree, (Spectral) Sparsifiers, Spanners, Densest Subgraph, Subgraph Counting, . . .

Previous Work

Successful applications of these two techniques have yielded $\widetilde{O}(n)$ size summaries for several graph problems:

Connectivity, Minimum Spanning Tree, (Spectral) Sparsifiers, Spanners, Densest Subgraph, Subgraph Counting, ...

Two prominent problems are missing however:

Previous Work

Successful applications of these two techniques have yielded $\widetilde{O}(n)$ size summaries for several graph problems:

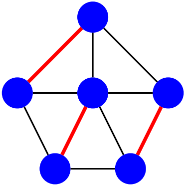
Connectivity, Minimum Spanning Tree, (Spectral) Sparsifiers, Spanners, Densest Subgraph, Subgraph Counting, ...

Two prominent problems are missing however:

Maximum Matching and Minimum Vertex Cover

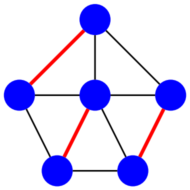
Matchings and Vertex Covers

- Matching: A collection of vertex-disjoint edges.



Matchings and Vertex Covers

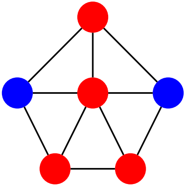
- Matching: A collection of vertex-disjoint edges.



- Maximum Matching problem: Find a matching with a largest number of edges.

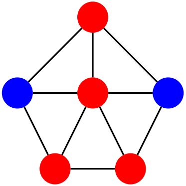
Matchings and Vertex Covers

- Vertex Cover: A collection of vertices containing at least one end point of every edge.



Matchings and Vertex Covers

- Vertex Cover: A collection of vertices containing at least one end point of every edge.



- Minimum Vertex Cover problem: Find a vertex cover with a smallest number of vertices.

Previous Work: Matching and Vertex

It turned out that matching and vertex cover do not admit efficient summaries!

Previous Work: Matching and Vertex

It turned out that matching and vertex cover do not admit efficient summaries!

[Assadi et al., 2016]:

> *Any simultaneous protocol that can compute an $n^{o(1)}$-approximation for these problems requires summaries of size $n^{2-o(1)}$.*

Previous Work: Matching and Vertex

It turned out that matching and vertex cover do not admit efficient summaries!

[Assadi et al., 2016]:

> *Any simultaneous protocol that can compute an $n^{o(1)}$-approximation for these problems requires summaries of size $n^{2-o(1)}$.*

As is traditional in this setting, this impossibility result is doubly worst case:

Previous Work: Matching and Vertex

It turned out that matching and vertex cover do not admit efficient summaries!

[Assadi et al., 2016]:

> *Any simultaneous protocol that can compute an $n^{o(1)}$-approximation for these problems requires summaries of size $n^{2-o(1)}$.*

As is traditional in this setting, this impossibility result is doubly worst case:

Both the underlying graph and the partitioning of the input are chosen adversarially!

Previous Work: Matching and Vertex

It turned out that matching and vertex cover do not admit efficient summaries!

[Assadi et al., 2016]:

> *Any simultaneous protocol that can compute an $n^{o(1)}$-approximation for these problems requires summaries of size $n^{2-o(1)}$.*

As is traditional in this setting, this impossibility result is doubly worst case:

Both the underlying graph and the partitioning of the input are chosen adversarially!

Can we distribute the original input in a better way?

Our Results in a Nutshell

A natural data oblivious partitioning scheme completely alters this landscape.

Our Results in a Nutshell

A natural data oblivious partitioning scheme completely alters this landscape.

Our work:

> *Both matching and vertex cover admit efficient simultaneous protocols provided that the edges of the graph are partitioned randomly across the machines.*

Our Results in a Nutshell

A natural data oblivious partitioning scheme completely alters this landscape.

Our work:

> *Both matching and vertex cover admit efficient simultaneous protocols provided that the edges of the graph are partitioned randomly across the machines.*

The idea that random partitioning can help was nicely illustrated by [Mirrokni and Zadimoghaddam, 2015] and [da Ponte Barbosa et al., 2015] on maximizing submodular functions.

Our Results in a Nutshell

A natural data oblivious partitioning scheme completely alters this landscape.

Our work:

> *Both matching and vertex cover admit efficient simultaneous protocols provided that the edges of the graph are partitioned randomly across the machines.*

The idea that random partitioning can help was nicely illustrated by [Mirrokni and Zadimoghaddam, 2015] and [da Ponte Barbosa et al., 2015] on maximizing submodular functions.

Our work is the first illustration in the domain of graph problems.

Randomized Composable Coresets

Define $G^{(1)}, \ldots, G^{(k)}$ as a random partitioning of a graph G: each edge $e \in G$ is sent to one of the graphs uniformly at random.

Randomized Composable Coresets

Define $G^{(1)}, \ldots, G^{(k)}$ as a random partitioning of a graph G: each edge $e \in G$ is sent to one of the graphs uniformly at random.

Consider an algorithm ALG that given any graph G computes a subgraph $\mathsf{ALG}(G) \subseteq G$ with at most s edges.

Randomized Composable Coresets

Define $G^{(1)}, \ldots, G^{(k)}$ as a random partitioning of a graph G: each edge $e \in G$ is sent to one of the graphs uniformly at random.

Consider an algorithm ALG that given any graph G computes a subgraph $\mathsf{ALG}(G) \subseteq G$ with at most s edges.

ALG outputs an α-approximation randomized composable coreset of size s for a problem P iff:

Randomized Composable Coresets

Define $G^{(1)}, \ldots, G^{(k)}$ as a random partitioning of a graph G: each edge $e \in G$ is sent to one of the graphs uniformly at random.

Consider an algorithm ALG that given any graph G computes a subgraph $\mathsf{ALG}(G) \subseteq G$ with at most s edges.

ALG outputs an α-approximation randomized composable coreset of size s for a problem P iff:

$P\left(\mathsf{ALG}(G^{(1)}) \cup \ldots \cup \mathsf{ALG}(G^{(k)})\right)$ is an α-approximation for $P\left(G\right)$ with high probability (over the randomness of the partitioning).

Randomized Composable Coresets

Define $G^{(1)}, \ldots, G^{(k)}$ as a random partitioning of a graph G: each edge $e \in G$ is sent to one of the graphs uniformly at random.

Consider an algorithm ALG that given any graph G computes a subgraph $\mathsf{ALG}(G) \subseteq G$ with at most s edges.

ALG outputs an α-approximation randomized composable coreset of size s for a problem P iff:

$P\left(\mathsf{ALG}(G^{(1)}) \cup \ldots \cup \mathsf{ALG}(G^{(k)})\right)$ is an α-approximation for $P(G)$ with high probability (over the randomness of the partitioning).

Defined originally by [Mirrokni and Zadimoghaddam, 2015] in the context of distributed submodular maximization.

Upper Bound Results: Maximum Matching

Greedy and local search are typical choices for composable coresets.

Upper Bound Results: Maximum Matching

Greedy and local search are typical choices for composable coresets.

However, one can show that the greedy algorithm for matching, i.e., picking a maximal matching, performs poorly in general.

Upper Bound Results: Maximum Matching

Greedy and local search are typical choices for composable coresets.

However, one can show that the greedy algorithm for matching, i.e., picking a maximal matching, performs poorly in general.

Our approach: pick a maximum matching!

Upper Bound Results: Maximum Matching

Greedy and local search are typical choices for composable coresets.

However, one can show that the greedy algorithm for matching, i.e., picking a maximal matching, performs poorly in general.

Our approach: pick a maximum matching!

Theorem

Any maximum matching is an $O(1)$-randomized composable coreset of size $n/2$ for the matching problem.

Upper Bound Results: Vertex Cover

Can a minimum vertex cover also be used as a randomized composable coreset for this problem?

Upper Bound Results: Vertex Cover

Can a minimum vertex cover also be used as a randomized composable coreset for this problem? Not really; consider a star with k petals for example.

Upper Bound Results: Vertex Cover

Can a minimum vertex cover also be used as a randomized composable coreset for this problem? Not really; consider a star with k petals for example.

Unlike most problems that admit a composable coreset, the vertex cover problem has a hard to verify feasibility constraint.

Upper Bound Results: Vertex Cover

Can a minimum vertex cover also be used as a randomized composable coreset for this problem? Not really; consider a star with k petals for example.

Unlike most problems that admit a composable coreset, the vertex cover problem has a hard to verify feasibility constraint.

This motivates a slightly more general notion of composable coresets.

Composable Coresets for Vertex Cover

A (randomized) composable coreset for the vertex cover problem contains both:

Composable Coresets for Vertex Cover

A (randomized) composable coreset for the vertex cover problem contains both:

1. A subset of edges of the input graph to guide the coordinator on the choice of the vertex cover.

Composable Coresets for Vertex Cover

A (randomized) composable coreset for the vertex cover problem contains both:

1. A subset of edges of the input graph to guide the coordinator on the choice of the vertex cover.
2. An explicitly specified subset of vertices to be always included in the final vertex cover

Composable Coresets for Vertex Cover

A (randomized) composable coreset for the vertex cover problem contains both:

1. A subset of edges of the input graph to guide the coordinator on the choice of the vertex cover.
2. An explicitly specified subset of vertices to be always included in the final vertex cover

Size of a coreset: number of edges + number of specified vertices.

Upper Bound Results: Vertex Cover

The vertex cover problem admits an efficient randomized composable coreset.

Upper Bound Results: Vertex Cover

The vertex cover problem admits an efficient randomized composable coreset.

Theorem

There exists an $O(\log n)$*-approximation randomized composable coreset of size* $O(n \cdot \log n)$ *for the vertex cover problem.*

Lower Bound Results: Randomized Coresets

Why coresets of size $\widetilde{O}(n)$?

Lower Bound Results: Randomized Coresets

Why coresets of size $\widetilde{O}(n)$?

- $\widetilde{O}(n)$ space is a "sweet spot" for graph streaming algorithms: typically the space needed to even store the answer.

Lower Bound Results: Randomized Coresets

Why coresets of size $\widetilde{O}(n)$?

- $\widetilde{O}(n)$ space is a "sweet spot" for graph streaming algorithms: typically the space needed to even store the answer.
- However, such considrations only imply that size of all coresets together need to be $\Omega(n)$.

Lower Bound Results: Randomized Coresets

Why coresets of size $\widetilde{O}(n)$?

- $\widetilde{O}(n)$ space is a "sweet spot" for graph streaming algorithms: typically the space needed to even store the answer.
- However, such considrations only imply that size of all coresets together need to be $\Omega(n)$.

Can we achieve coresets of size, say, $\Theta(n/k)$?

Lower Bound Results: Randomized Coresets

Why coresets of size $\widetilde{O}(n)$?

- $\widetilde{O}(n)$ space is a "sweet spot" for graph streaming algorithms: typically the space needed to even store the answer.
- However, such considrations only imply that size of all coresets together need to be $\Omega(n)$.

Can we achieve coresets of size, say, $\Theta(n/k)$? No!

Theorem

Any α-approximation randomized composable coreset requires,

- $\Omega(n/\alpha^2)$ *space for the matching problem, and,*
- $\Omega(n/\alpha)$ *space for the vertex cover problem.*

Lower Bound Results: Randomized Coresets

Why coresets of size $\widetilde{O}(n)$?

- $\widetilde{O}(n)$ space is a "sweet spot" for graph streaming algorithms: typically the space needed to even store the answer.
- However, such considrations only imply that size of all coresets together need to be $\Omega(n)$.

Can we achieve coresets of size, say, $\Theta(n/k)$? No!

Theorem

Any α-approximation randomized composable coreset requires,

- $\Omega(n/\alpha^2)$ *space for the matching problem, and,*
- $\Omega(n/\alpha)$ *space for the vertex cover problem.*

Remark. These bounds are tight for all values of α.

Upper Bound Results: Distributed Computing

Our randomized composable coresets immediately imply simultaneous distributed protocols:

Upper Bound Results: Distributed Computing

Our randomized composable coresets immediately imply simultaneous distributed protocols:

Theorem

There exists simultaneous protocol with approximation guarantee

1. $O(1)$ *for the matching problem, and,*
2. $O(\log n)$ *for the vertex cover problem,*

that require only $\widetilde{O}(k \cdot n)$ *total communication when the input is randomly partitioned between* k *machines.*

Upper Bound Results: Distributed Computing

Remark. These result also imply MapReduce algorithms for matching and vertex cover with the same approximation guarantee in at most 2 rounds of computation and $O(n\sqrt{n})$ space per each machine.

Upper Bound Results: Distributed Computing

Remark. These result also imply MapReduce algorithms for matching and vertex cover with the same approximation guarantee in at most 2 rounds of computation and $O(n\sqrt{n})$ space per each machine.

Our MapReduce algorithms outperform the previous algorithms for these problems [Lattanzi et al., 2011, Ahn and Guha, 2015] in terms of number of rounds, albeit with a larger approximation guarantee.

Upper Bound Results: Distributed Computing

Remark. These result also imply MapReduce algorithms for matching and vertex cover with the same approximation guarantee in at most 2 rounds of computation and $O(n\sqrt{n})$ space per each machine.

Our MapReduce algorithms outperform the previous algorithms for these problems [Lattanzi et al., 2011, Ahn and Guha, 2015] in terms of number of rounds, albeit with a larger approximation guarantee.

The number of rounds of a MapReduce algorithm usually determines the dominant cost of the computation.

Lower Bound Results: Distributed Computing

Our lower bound on size of randomized composable coresets implies that our distributed protocols are optimal among all coreset-based protocols.

Lower Bound Results: Distributed Computing

Our lower bound on size of randomized composable coresets implies that our distributed protocols are optimal among all coreset-based protocols.

What about general protocols?

Lower Bound Results: Distributed Computing

Our lower bound on size of randomized composable coresets implies that our distributed protocols are optimal among all coreset-based protocols.

What about general protocols?

Theorem

Any α-approximation simultaneous protocol (not necessarily a coreset) requires

- $\Omega(nk/\alpha^2)$ *communication for the matching problem, and,*
- $\Omega(nk/\alpha)$ *communication for the vertex cover problem,*

even when the input is randomly partitioned across the k machines.

Lower Bound Results: Distributed Computing

Our lower bound on size of randomized composable coresets implies that our distributed protocols are optimal among all coreset-based protocols.

What about general protocols?

Theorem

Any α-approximation simultaneous protocol (not necessarily a coreset) requires

- $\Omega(nk/\alpha^2)$ *communication for the matching problem, and,*
- $\Omega(nk/\alpha)$ *communication for the vertex cover problem,*

even when the input is randomly partitioned across the k machines.

For adversarial partitions, an $\Omega(nk/\alpha^2)$ lower bound for matching was known previously even for protocols that are allowed multiple rounds of communication [Huang et al., 2015].

A Randomized Composable Coreset for Matching

A Randomized Coreset for Matching

Theorem

Any maximum matching is an $O(1)$*-randomized composable coreset of size* $n/2$ *for the matching problem.*

A Randomized Coreset for Matching

Theorem

Any maximum matching is an $O(1)$-randomized composable coreset of size $n/2$ for the matching problem.

Let M_i be the maximum matching computed by machine $i \in [k]$.

A Randomized Coreset for Matching

Theorem

Any maximum matching is an $O(1)$-randomized composable coreset of size $n/2$ for the matching problem.

Let M_i be the maximum matching computed by machine $i \in [k]$.

Consider running the greedy algorithm over the edges in $M_1, \ldots, M_k$ in this order to obtain a matching M.

A Randomized Coreset for Matching

Theorem

Any maximum matching is an $O(1)$-randomized composable coreset of size $n/2$ for the matching problem.

Let M_i be the maximum matching computed by machine $i \in [k]$.

Consider running the greedy algorithm over the edges in $M_1, \ldots, M_k$ in this order to obtain a matching M.

We prove that $|M| = \Omega(\text{opt})$, where opt is the size of a maximum matching in G.

A Randomized Coreset for Matching

Theorem

Any maximum matching is an $O(1)$-randomized composable coreset of size $n/2$ for the matching problem.

Let M_i be the maximum matching computed by machine $i \in [k]$.

Consider running the greedy algorithm over the edges in $M_1, \ldots, M_k$ in this order to obtain a matching M.

We prove that $|M| = \Omega(\text{opt})$, where opt is the size of a maximum matching in G.

This implies that there exists an $O(1)$-approximate matching in $M_1 \cup \ldots \cup M_k$.

Analysis Sketch: A Key Lemma

Lemma

At any step $i \in [k]$, either the greedy matching is already of size $\Omega(\text{opt})$, or w.h.p., we can increase the size of the current matching by adding $\Omega(\text{opt}/k)$ edges from M_i greedily.

Analysis Sketch: A Key Lemma

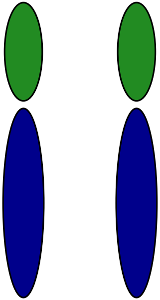
Lemma

At any step $i \in [k]$, either the greedy matching is already of size $\Omega(\mathsf{opt})$, or w.h.p., we can increase the size of the current matching by adding $\Omega(\mathsf{opt}/k)$ edges from M_i greedily.

This immediately implies that the matching output by the greedy algorithm has size $\Omega(\mathsf{opt})$ w.h.p.

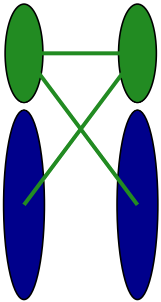
Proof Sketch

- Consider the set of $o(\text{opt})$ already matched vertices by the greedy algorithm.



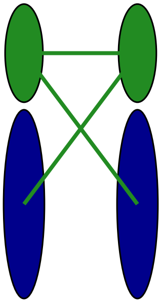
Proof Sketch

- Consider the set of $o(\mathsf{opt})$ already matched vertices by the greedy algorithm.
- Define E_{old} as the set of edges in $G^{(i)}$ incident on these already matched vertices.



Proof Sketch

- Consider the set of $o(\text{opt})$ already matched vertices by the greedy algorithm.
- Define E_{old} as the set of edges in $G^{(i)}$ incident on these already matched vertices.
- Define μ_{old} as size of a maximum matching in $G^{(i)}$ using only edges in E_{old}.



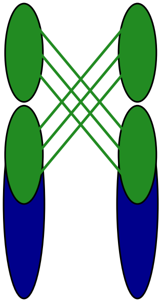
Proof Sketch

Claim. W.h.p. there is a matching of size $\geq \mu_{\mathsf{old}} + \Omega(\mathsf{opt}/k)$ in $G^{(i)}$.

Proof Sketch

Claim. W.h.p. there is a matching of size $\geq \mu_{\mathsf{old}} + \Omega(\mathsf{opt}/k)$ in $G^{(i)}$.

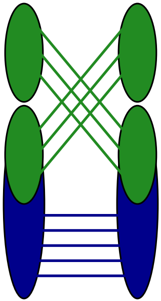
- Fix a maximum matching in E_{old}: at most $o(\mathsf{opt})$ vertices that were previously unmatched are in the matching.



Proof Sketch

Claim. W.h.p. there is a matching of size $\geq \mu_{\mathsf{old}} + \Omega(\mathsf{opt}/k)$ in $G^{(i)}$.

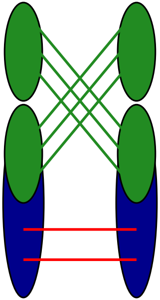
- Fix a maximum matching in E_{old}: at most $o(\mathsf{opt})$ vertices that were previously unmatched are in the matching.
- Hence, G contains a matching of size $\Omega(\mathsf{opt})$ outside the set of vertices matched by μ_{old}.



Proof Sketch

Claim. W.h.p. there is a matching of size $\geq \mu_{\text{old}} + \Omega(\text{opt}/k)$ in $G^{(i)}$.

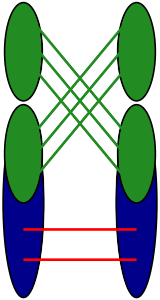
- Fix a maximum matching in E_{old}: at most $o(\text{opt})$ vertices that were previously unmatched are in the matching.
- Hence, G contains a matching of size $\Omega(\text{opt})$ outside the set of vertices matched by μ_{old}.
- By random partitioning, w.h.p., $\Omega(\text{opt}/k)$ such edges appear in $G^{(i)}$.



Proof Sketch

Claim. W.h.p. there is a matching of size $\geq \mu_{\text{old}} + \Omega(\text{opt}/k)$ in $G^{(i)}$.

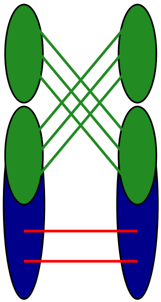
- Fix a maximum matching in E_{old}: at most $o(\text{opt})$ vertices that were previously unmatched are in the matching.
- Hence, G contains a matching of size $\Omega(\text{opt})$ outside the set of vertices matched by μ_{old}.
- By random partitioning, w.h.p., $\Omega(\text{opt}/k)$ such edges appear in $G^{(i)}$.
- $\mu_{\text{old}} + \Omega(\text{opt}/k)$ forms the desired matching.



Proof Sketch

Claim. W.h.p. there is a matching of size $\geq \mu_{\text{old}} + \Omega(\text{opt}/k)$ in $G^{(i)}$.

- Fix a maximum matching in E_{old}: at most $o(\text{opt})$ vertices that were previously unmatched are in the matching.
- Hence, G contains a matching of size $\Omega(\text{opt})$ outside the set of vertices matched by μ_{old}.
- By random partitioning, w.h.p., $\Omega(\text{opt}/k)$ such edges appear in $G^{(i)}$.
- $\mu_{\text{old}} + \Omega(\text{opt}/k)$ forms the desired matching.



Corollary. Any maximum matching of $G^{(i)}$ contains $\Omega(\text{opt}/k)$ edges that can be added to the greedy matching.

Randomized Composable Coreset for Matching

We showed that,

Theorem

Any maximum matching is an $O(1)$-randomized composable coreset of size at most $n/2$ for the matching problem.

A Randomized Composable Coreset for Vertex Cover

A Randomized Coreset for Vertex Cover

Theorem

There exists an $O(\log n)$*-approximation randomized composable coreset of size* $O(n \cdot \log n)$ *for the vertex cover problem.*

A Randomized Coreset for Vertex Cover

Theorem

There exists an $O(\log n)$-approximation randomized composable coreset of size $O(n \cdot \log n)$ for the vertex cover problem.

Each machine computes a coreset using the following peeling process.

A Randomized Coreset for Vertex Cover

Theorem

There exists an $O(\log n)$-approximation randomized composable coreset of size $O(n \cdot \log n)$ for the vertex cover problem.

Each machine computes a coreset using the following peeling process.

Iteratively remove high degree vertices and their neighboring edges; specify any removed vertex to be added to the final vertex cover.

A Randomized Coreset for Vertex Cover

Theorem

There exists an $O(\log n)$-approximation randomized composable coreset of size $O(n \cdot \log n)$ for the vertex cover problem.

Each machine computes a coreset using the following peeling process.

Iteratively remove high degree vertices and their neighboring edges; specify any removed vertex to be added to the final vertex cover.

When the remaining graph is sufficiently sparse, send it as the coreset.

A Randomized Coreset for Vertex Cover

Theorem

There exists an $O(\log n)$-approximation randomized composable coreset of size $O(n \cdot \log n)$ for the vertex cover problem.

Each machine computes a coreset using the following peeling process.

Iteratively remove high degree vertices and their neighboring edges; specify any removed vertex to be added to the final vertex cover.

When the remaining graph is sufficiently sparse, send it as the coreset.

This peeling process was introduced originally by [Parnas and Ron, 2007] in the context of sublinear time algorithms.

A Randomized Coreset for Vertex Cover

The algorithm to compute the coreset on each machine $i \in [k]$:

1. Pick all vertices in $G^{(i)}$ with degree more than $n/2k$ and add them to the final vertex cover.

A Randomized Coreset for Vertex Cover

The algorithm to compute the coreset on each machine $i \in [k]$:

1. Pick all vertices in $G^{(i)}$ with degree more than $n/2k$ and add them to the final vertex cover.
2. Remove these vertices from $G^{(i)}$ together with all their edges.

A Randomized Coreset for Vertex Cover

The algorithm to compute the coreset on each machine $i \in [k]$:

1. Pick all vertices in $G^{(i)}$ with degree more than $n/2k$ and add them to the final vertex cover.
2. Remove these vertices from $G^{(i)}$ together with all their edges.
3. Continue with degree threshold $n/4k$ and so on; stop when the degree of each vertex is $O(\log n)$.

A Randomized Coreset for Vertex Cover

The algorithm to compute the coreset on each machine $i \in [k]$:

1. Pick all vertices in $G^{(i)}$ with degree more than $n/2k$ and add them to the final vertex cover.
2. Remove these vertices from $G^{(i)}$ together with all their edges.
3. Continue with degree threshold $n/4k$ and so on; stop when the degree of each vertex is $O(\log n)$.
4. Return all edges in the remaining graph as the coreset.

A Randomized Coreset for Vertex Cover

The algorithm to compute the coreset on each machine $i \in [k]$:

1. Pick all vertices in $G^{(i)}$ with degree more than $n/2k$ and add them to the final vertex cover.
2. Remove these vertices from $G^{(i)}$ together with all their edges.
3. Continue with degree threshold $n/4k$ and so on; stop when the degree of each vertex is $O(\log n)$.
4. Return all edges in the remaining graph as the coreset.

Size of the coreset is clearly $O(n \cdot \log n)$.

Analysis Sketch

Define opt as size of a minimum vertex cover in G.

Analysis Sketch

Define opt as size of a minimum vertex cover in G.

It follows from the known results that each coreset only specifies $O(\mathsf{opt} \cdot \log n)$ vertices to be added to the final vertex cover

Analysis Sketch

Define opt as size of a minimum vertex cover in G.

It follows from the known results that each coreset only specifies $O(\mathsf{opt} \cdot \log n)$ vertices to be added to the final vertex cover

Using this directly only implies an approximation ratio of $O(k \cdot \log n)$, i.e., a factor k worse than our goal.

Analysis Sketch

Define opt as size of a minimum vertex cover in G.

It follows from the known results that each coreset only specifies $O(\mathsf{opt} \cdot \log n)$ vertices to be added to the final vertex cover

Using this directly only implies an approximation ratio of $O(k \cdot \log n)$, i.e., a factor k worse than our goal.

We show that the set of all specified vertices across all coresets is of size $O(\mathsf{opt} \cdot \log n)$.

Analysis Sketch

Define opt as size of a minimum vertex cover in G.

It follows from the known results that each coreset only specifies $O(\mathsf{opt} \cdot \log n)$ vertices to be added to the final vertex cover

Using this directly only implies an approximation ratio of $O(k \cdot \log n)$, i.e., a factor k worse than our goal.

We show that the set of all specified vertices across all coresets is of size $O(\mathsf{opt} \cdot \log n)$.

This finalizes the proof as any edge not covered by any of specified vertices is communicated in some coreset.

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ *vertices are specified to be added to the final vertex cover in total.*

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ *vertices are specified to be added to the final vertex cover in total.*

Intuitively:

1. By random partitioning, degree of vertices is almost the same across the coresets.

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ *vertices are specified to be added to the final vertex cover in total.*

Intuitively:

1. By random partitioning, degree of vertices is almost the same across the coresets.
2. Hence, the same set of vertices should be peeled across in each iteration.

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ vertices are specified to be added to the final vertex cover in total.

Intuitively:

1. By random partitioning, degree of vertices is almost the same across the coresets.
2. Hence, the same set of vertices should be peeled across in each iteration.

Any problem?

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ vertices are specified to be added to the final vertex cover in total.

Intuitively:

1. By random partitioning, degree of vertices is almost the same across the coresets.
2. Hence, the same set of vertices should be peeled across in each iteration.

Any problem?

1. The peeling process is quite sensitive to the exact degrees.

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ vertices are specified to be added to the final vertex cover in total.

Intuitively:

1. By random partitioning, degree of vertices is almost the same across the coresets.
2. Hence, the same set of vertices should be peeled across in each iteration.

Any problem?

1. The peeling process is quite sensitive to the exact degrees.
2. Slight changes in the degree can move vertices across iterations, potentially leading to a cascading effect.

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ *vertices are specified to be added to the final vertex cover in total.*

Our approach:

1. Define a hypothetical peeling process that is aware of a minimum vertex cover in G.

Analysis Sketch: A Key Lemma

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ vertices are specified to be added to the final vertex cover in total.

Our approach:

1. Define a hypothetical peeling process that is aware of a minimum vertex cover in G.
2. Prove that this peeling process never picks more than $O(\mathsf{opt} \cdot \log n)$ vertices.

Analysis Sketch: A Key Lemma

Lemma

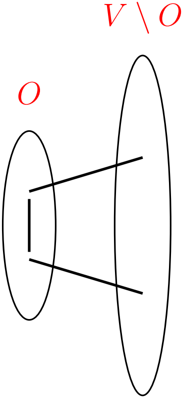
W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ vertices are specified to be added to the final vertex cover in total.

Our approach:

1. Define a hypothetical peeling process that is aware of a minimum vertex cover in G.
2. Prove that this peeling process never picks more than $O(\mathsf{opt} \cdot \log n)$ vertices.
3. Show that the actual peeling process on each machine "faithfully" mimics this hypothetical process.

Proof Sketch

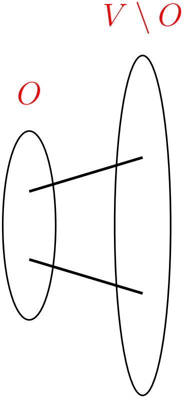
Define O as a minimum vertex cover of G. The hypothetical peeling process is as follows:



Proof Sketch

Define O as a minimum vertex cover of G. The hypothetical peeling process is as follows:

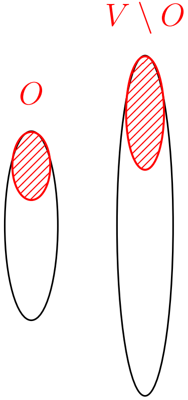
- Remove all edges inside O.



Proof Sketch

Define O as a minimum vertex cover of G. The hypothetical peeling process is as follows:

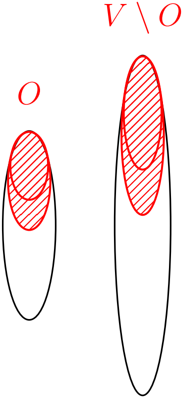
- Remove all edges inside O.
- Remove vertices with degree $\frac{n}{1.5}$ from O and degree $\frac{n}{2.5}$ from $V \setminus O$.



Proof Sketch

Define O as a minimum vertex cover of G. The hypothetical peeling process is as follows:

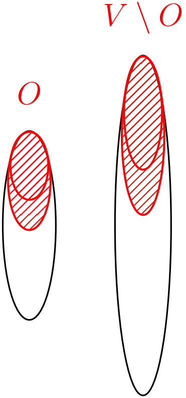
- Remove all edges inside O.
- Remove vertices with degree $\frac{n}{1.5}$ from O and degree $\frac{n}{2.5}$ from $V \setminus O$.
- Remove all incident edges on these vertices; continue with degree threshold $\frac{n}{2\cdot(1.5)}$ from O and $\frac{n}{2\cdot(2.5)}$ from $V \setminus O$.



Proof Sketch

Define O as a minimum vertex cover of G. The hypothetical peeling process is as follows:

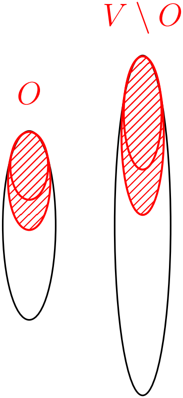
- Remove all edges inside O.
- Remove vertices with degree $\frac{n}{1.5}$ from O and degree $\frac{n}{2.5}$ from $V \setminus O$.
- Remove all incident edges on these vertices; continue with degree threshold $\frac{n}{2 \cdot (1.5)}$ from O and $\frac{n}{2 \cdot (2.5)}$ from $V \setminus O$.
- Repeat the above process until the degree threshold reaches $\Theta(k \log n)$.



Proof Sketch

Define O as a minimum vertex cover of G. The hypothetical peeling process is as follows:

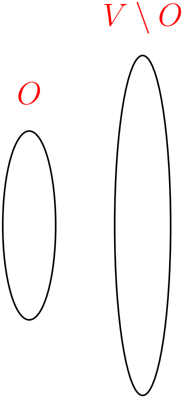
- Remove all edges inside O.
- Remove vertices with degree $\frac{n}{1.5}$ from O and degree $\frac{n}{2.5}$ from $V \setminus O$.
- Remove all incident edges on these vertices; continue with degree threshold $\frac{n}{2\cdot(1.5)}$ from O and $\frac{n}{2\cdot(2.5)}$ from $V \setminus O$.
- Repeat the above process until the degree threshold reaches $\Theta(k \log n)$.



Claim. The number of peeled vertices from $V \setminus O$ in each iteration is at most $2\,|O|$.

Proof Sketch

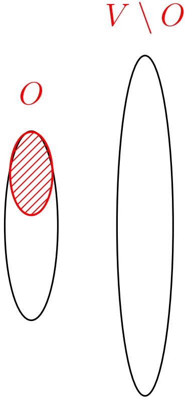
Main Claim. For any machine $i \in [k]$ and any iteration of the peeling process:



Proof Sketch

Main Claim. For any machine $i \in [k]$ and any iteration of the peeling process:

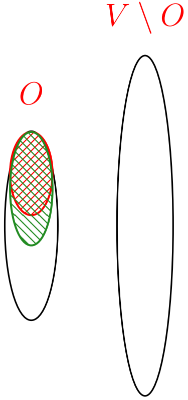
- For vertices in O: the set of peeled vertices in the hypothetical process is a subset of vertices peeled in the actual coreset.



Proof Sketch

Main Claim. For any machine $i \in [k]$ and any iteration of the peeling process:

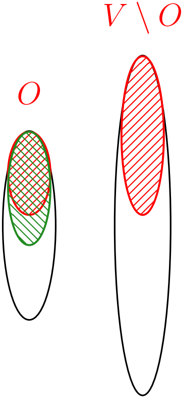
- For vertices in O: the set of peeled vertices in the hypothetical process is a subset of vertices peeled in the actual coreset.



Proof Sketch

Main Claim. For any machine $i \in [k]$ and any iteration of the peeling process:

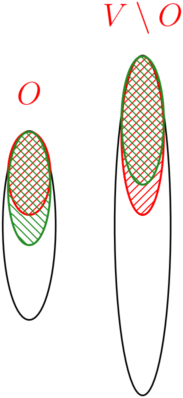
- For vertices in O: the set of peeled vertices in the hypothetical process is a subset of vertices peeled in the actual coreset.
- For vertices in $V \setminus O$: the set of peeled vertices in the hypothetical process is a superset of vertices peeled in the actual coreset.



Proof Sketch

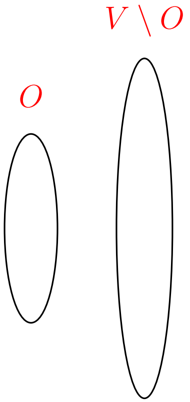
Main Claim. For any machine $i \in [k]$ and any iteration of the peeling process:

- For vertices in O: the set of peeled vertices in the hypothetical process is a subset of vertices peeled in the actual coreset.
- For vertices in $V \setminus O$: the set of peeled vertices in the hypothetical process is a superset of vertices peeled in the actual coreset.



Proof Sketch

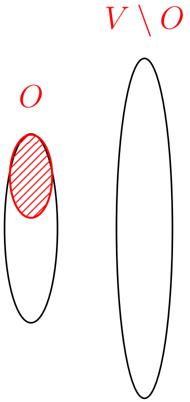
In the first iteration, by random partitioning:



Proof Sketch

In the first iteration, by random partitioning:

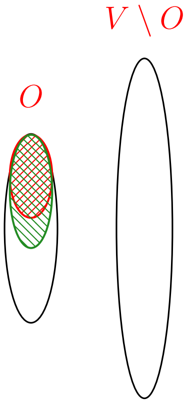
- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).



Proof Sketch

In the first iteration, by random partitioning:

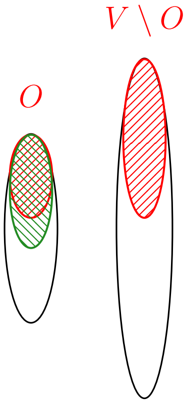
- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).



Proof Sketch

In the first iteration, by random partitioning:

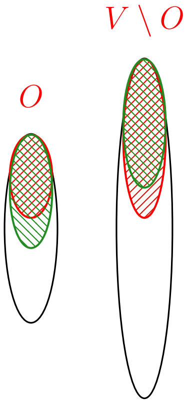
- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).
- For vertices in $V \setminus O$: the exact opposite.



Proof Sketch

In the first iteration, by random partitioning:

- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).
- For vertices in $V \setminus O$: the exact opposite.

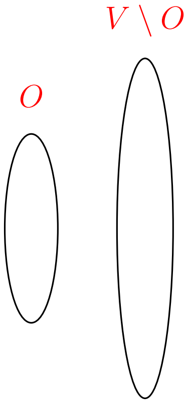


Proof Sketch

In the first iteration, by random partitioning:

- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).
- For vertices in $V \setminus O$: the exact opposite.

In the next iterations:



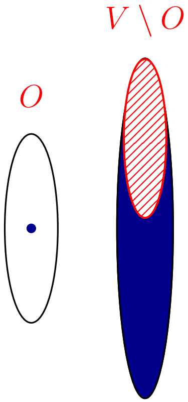
Proof Sketch

In the first iteration, by random partitioning:

- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).
- For vertices in $V \setminus O$: the exact opposite.

In the next iterations:

- For vertices in O: the degree of remaining vertices after peeling is smaller in the hypothetical process compared to the actual coreset.



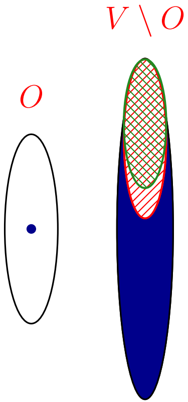
Proof Sketch

In the first iteration, by random partitioning:

- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).
- For vertices in $V \setminus O$: the exact opposite.

In the next iterations:

- For vertices in O: the degree of remaining vertices after peeling is smaller in the hypothetical process compared to the actual coreset.



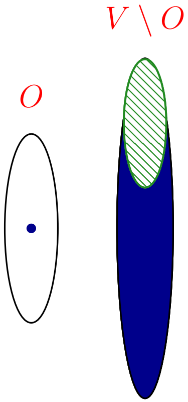
Proof Sketch

In the first iteration, by random partitioning:

- For vertices in O: the degree threshold of peeling vertices in the hypothetical process is larger than the actual coreset (after scaling by k).
- For vertices in $V \setminus O$: the exact opposite.

In the next iterations:

- For vertices in O: the degree of remaining vertices after peeling is smaller in the hypothetical process compared to the actual coreset.
- For vertices in $V \setminus O$: the exact opposite.



Proof Sketch

To wrap-up:

Proof Sketch

To wrap-up:

1. Across the machines, the set of peeled vertices in $V \setminus O$ by the coresets is a subset of peeled vertices by the hypothetical process.

Proof Sketch

To wrap-up:

1. Across the machines, the set of peeled vertices in $V \setminus O$ by the coresets is a subset of peeled vertices by the hypothetical process.
2. The set of peeled vertices in $V \setminus O$ by the hypothetical process is of size $O(\mathsf{opt} \cdot \log n)$.

Proof Sketch

To wrap-up:

1. Across the machines, the set of peeled vertices in $V \setminus O$ by the coresets is a subset of peeled vertices by the hypothetical process.
2. The set of peeled vertices in $V \setminus O$ by the hypothetical process is of size $O(\mathsf{opt} \cdot \log n)$.
3. The remaining peeled vertices across the machines belong to O and hence are of size $O(\mathsf{opt})$.

Proof Sketch

To wrap-up:

1. Across the machines, the set of peeled vertices in $V \setminus O$ by the coresets is a subset of peeled vertices by the hypothetical process.
2. The set of peeled vertices in $V \setminus O$ by the hypothetical process is of size $O(\mathsf{opt} \cdot \log n)$.
3. The remaining peeled vertices across the machines belong to O and hence are of size $O(\mathsf{opt})$.

Lemma

W.h.p. at most $O(\mathsf{opt} \cdot \log n)$ vertices are specified to be added to the final vertex cover in total.

Randomized Composable Coreset for Vertex Cover

We showed that,

Theorem

There exists an $O(\log n)$*-approximation randomized composable coreset of size* $O(n \cdot \log n)$ *for the vertex cover problem.*

Concluding Remarks

- We provided efficient simultaneous protocols for matching and vertex cover when the edges of the graph are partitioned randomly across the machines.

Concluding Remarks

- We provided efficient simultaneous protocols for matching and vertex cover when the edges of the graph are partitioned randomly across the machines.
- Our protocols bypass the strong impossibility results known for these problems under adversarially partitioned inputs.

Concluding Remarks

- We provided efficient simultaneous protocols for matching and vertex cover when the edges of the graph are partitioned randomly across the machines.
- Our protocols bypass the strong impossibility results known for these problems under adversarially partitioned inputs.

Open problems:

- Better approximation factors for matching and vertex cover?

Concluding Remarks

- We provided efficient simultaneous protocols for matching and vertex cover when the edges of the graph are partitioned randomly across the machines.
- Our protocols bypass the strong impossibility results known for these problems under adversarially partitioned inputs.

Open problems:

- Better approximation factors for matching and vertex cover?
- Any super-linear (in n) lower bound for $(1+\varepsilon)$-approximation of matching under random partitions?

Concluding Remarks

- We provided efficient simultaneous protocols for matching and vertex cover when the edges of the graph are partitioned randomly across the machines.
- Our protocols bypass the strong impossibility results known for these problems under adversarially partitioned inputs.

Open problems:

- Better approximation factors for matching and vertex cover?
- Any super-linear (in n) lower bound for $(1+\varepsilon)$-approximation of matching under random partitions?
- Randomized composable coresets for other problems?

Concluding Remarks

- We provided efficient simultaneous protocols for matching and vertex cover when the edges of the graph are partitioned randomly across the machines.
- Our protocols bypass the strong impossibility results known for these problems under adversarially partitioned inputs.

Open problems:

- Better approximation factors for matching and vertex cover?
- Any super-linear (in n) lower bound for $(1+\varepsilon)$-approximation of matching under random partitions?
- Randomized composable coresets for other problems?
  - In particular, for obtaining a maximal matching?

Ahn, K. J. and Guha, S. (2015).
Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints.
In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 202–211.

Ahn, K. J., Guha, S., and McGregor, A. (2012a).
Analyzing graph structure via linear measurements.
In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 459–467. SIAM.

Ahn, K. J., Guha, S., and McGregor, A. (2012b).
Graph sketches: sparsification, spanners, and subgraphs.
In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14.

Assadi, S., Khanna, S., Li, Y., and Yaroslavtsev, G. (2016).
Maximum matchings in dynamic graph streams and the simultaneous communication model.
In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. (2014).
Streaming submodular maximization: massive data summarization on the fly.
In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680.

Balcan, M., Ehrlich, S., and Liang, Y. (2013).
Distributed k-means and k-median clustering on general communication topologies.

In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1995–2003.

Bateni, M., Bhaskara, A., Lattanzi, S., and Mirrokni, V. S. (2014).
Distributed balanced clustering via mapping coresets.
In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2591–2599.

Bhattacharya, S., Henzinger, M., Nanongkai, D., and Tsourakakis, C. E. (2015).
Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams.

In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182.

Bulteau, L., Froese, V., Kutzkov, K., and Pagh, R. (2016).
Triangle counting in dynamic graph streams.
Algorithmica, 76(1):259–278.

Chitnis, R., Cormode, G., Esfandiari, H., Hajiaghayi, M., McGregor, A., Monemizadeh, M., and Vorotnikova, S. (2016).
Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams.
In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344.

da Ponte Barbosa, R., Ene, A., Nguyen, H. L., and Ward, J. (2015).

The power of randomization: Distributed submodular maximization on massive datasets.
In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1236–1244.

Huang, Z., Radunovic, B., Vojnovic, M., and Zhang, Q. (2015).
Communication complexity of approximate matching in distributed graphs.
In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 460–473.

Indyk, P., Mahabadi, S., Mahdian, M., and Mirrokni, V. S. (2014).
Composable core-sets for diversity and coverage maximization.
In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 100–108.

Kapralov, M., Lee, Y. T., Musco, C., Musco, C., and Sidford, A. (2014).
Single pass spectral sparsification in dynamic streams.
In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 561–570.

Kapralov, M. and Woodruff, D. (2014).
Spanners and sparsifiers in dynamic streams.
PODC.

Lattanzi, S., Moseley, B., Suri, S., and Vassilvitskii, S. (2011).
Filtering: a method for solving graph problems in mapreduce.
In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94.

McGregor, A., Tench, D., Vorotnikova, S., and Vu, H. T. (2015).
Densest subgraph in dynamic graph streams.

In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 472–482.

Mirrokni, V. S. and Zadimoghaddam, M. (2015).
Randomized composable core-sets for distributed submodular maximization.
In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 153–162.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. (2013).
Distributed submodular maximization: Identifying representative elements in massive data.
In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2049–2057.

Parnas, M. and Ron, D. (2007).
Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms.
Theor. Comput. Sci., 381(1-3):183–196.