Distributed Weighted Matching via Randomized Composable Coresets

MohammadHossein Bateni
Google Research
New York, USA

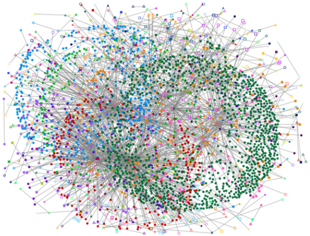
Sepehr Assadi

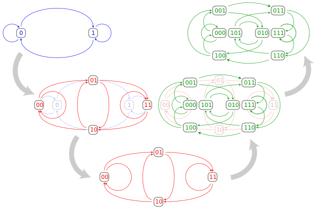
Vahab Mirrokni

36th International Conference on Machine Learning
12 June 2019 — Long Beach, California, USA
Poster #161, Pacific Ballroom

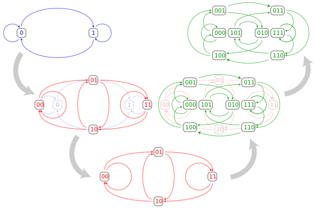
Massive graphs

Everywhere: web graph, social networks, biological networks, etc.



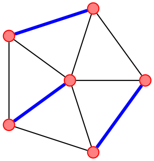


Everywhere: web graph, social networks, biological networks, etc.



Matching: a collection of vertex-disjoint edges

- Clustering, partitioning
- Finding motifs in bioinformatics
- Trade marketing, online advertisement
- Kidney exchange
- Linear algebra, matrix decomposition



Sequential results

Variants:

- Weighted vs unweighted
- Bipartite vs non-bipartite

Algorithmic results:

- First polytime algorithm: "Blossom" decomposition [Edm65a]
- Extended to weighted case [Edm65b]
- Fastest in time $\tilde{O}(m\sqrt{n})$ [GabTar91]

Approximation algorithms:

- Greedy algorithm: 2-approx in $O(m \log n)$ time
- $1 + \epsilon$ approx in $\tilde{O}(m/\epsilon)$ time [DuaPet14]

Sequential results

Variants:

- Weighted vs unweighted
- Bipartite vs non-bipartite

Algorithmic results:

- First polytime algorithm: "Blossom" decomposition [Edm65a]
- Extended to weighted case [Edm65b]
- Fastest in time $\tilde{O}(m\sqrt{n})$ [GabTar91]

Approximation algorithms:

- Greedy algorithm: 2-approx in $O(m \log n)$ time
- $1+\epsilon$ approx in $\tilde{O}(m/\epsilon)$ time [DuaPet14]

Sequential algorithms do not work

- $O(m)$ runtime is prohibitive
- $O(n)$ memory not available on a single machine
- Simply reading input data once may take too long!

Sequential algorithms do not work

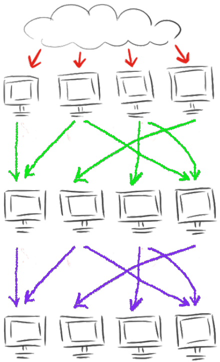
- $O(m)$ runtime is prohibitive
- $O(n)$ memory not available on a single machine
- Simply reading input data once may take too long!

Sequential algorithms do not work

- $O(m)$ runtime is prohibitive
- $O(n)$ memory not available on a single machine
- Simply reading input data once may take too long!

MapReduce: *de facto* industry standard

- Split data across many machines
- Split computation into several rounds
- Data is sent to machines based on keys
- Machines produce output for next round

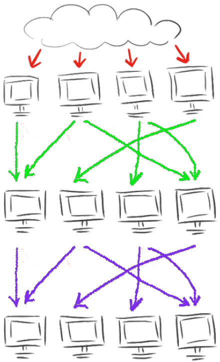


Sequential algorithms do not work

- $O(m)$ runtime is prohibitive
- $O(n)$ memory not available on a single machine
- Simply reading input data once may take too long!

MapReduce: *de facto* industry standard

- Split data across many machines
- Split computation into several rounds
- Data is sent to machines based on keys
- Machines produce output for next round



Important:

1. number of rounds
2. number of machines
3. memory on each machine
4. amount of computation on each machine

> "Greedy algorithms are practitioners' best friends—they are intuitive, simple to implement, and often lead to very good solutions. However, implementing greedy algorithms in a distributed setting is challenging since the greedy choice is inherently sequential, and it is not clear how to take advantage of the extra processing power."
>
> — [KumMosVasVat13]

Two typical types of results:

1. Relatively large number of rounds to "faithfully" simulate the greedy algorithm.
2. Very small number of rounds (one or two) for "weak" simulation, $O(1)$ worse than the greedy algorithm.

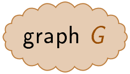
Credit	Approx	Space	Rounds
[LatMosSurVas11]	8	$\tilde{O}(n)$	$O(\log n)$
[CroStu14]	4	$\tilde{O}(n)$	$O(\log n)$
[AhnGuh15]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-1}\log n)$
[HarLiaLiu18]	2	$\tilde{O}(n)$	$O(\log n)$

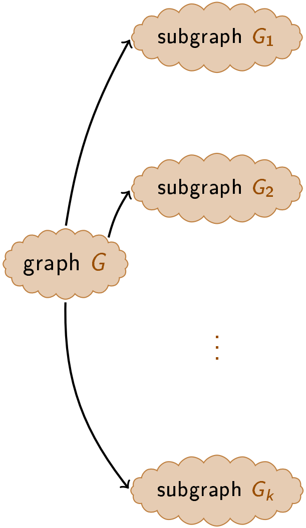
Credit	Approx	Space	Rounds
[LatMosSurVas11]	8	$\tilde{O}(n)$	$O(\log n)$
[CroStu14]	4	$\tilde{O}(n)$	$O(\log n)$
[AhnGuh15]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-1} \log n)$
[HarLiaLiu18]	2	$\tilde{O}(n)$	$O(\log n)$
[CzuLacMad+18]	$2+\varepsilon$	$\tilde{O}(n)$	$O\left(\varepsilon^{-\Theta(1/\varepsilon)} \cdot O(\log \log n)^2\right)$
[AssBatBer+19]	$2+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon)} \cdot \log \log n)$
[GamKalMitSve18]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon^2)} \cdot \log \log n)$

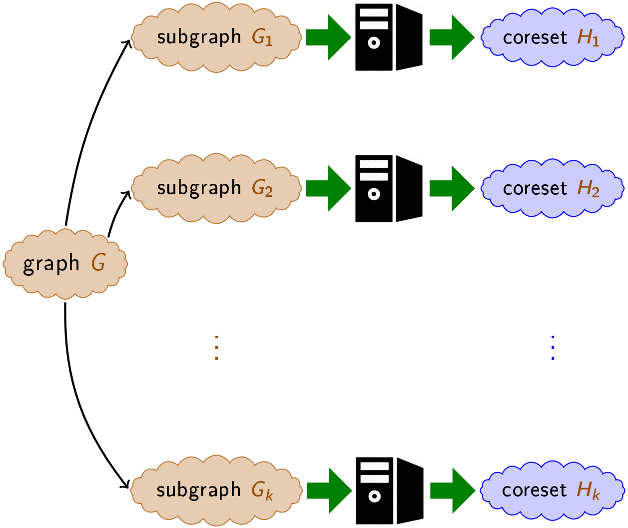
Credit	Approx	Space	Rounds
[LatMosSurVas11]	8	$\tilde{O}(n)$	$O(\log n)$
[CroStu14]	4	$\tilde{O}(n)$	$O(\log n)$
[AhnGuh15]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-1}\log n)$
[HarLiaLiu18]	2	$\tilde{O}(n)$	$O(\log n)$
[CzuLacMad+18]	$2+\varepsilon$	$\tilde{O}(n)$	$O\left(\varepsilon^{-\Theta(1/\varepsilon)} \cdot O(\log\log n)^2\right)$
[AssBatBer+19]	$2+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon)} \cdot \log\log n)$
[GamKalMitSve18]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon^2)} \cdot \log\log n)$
[LatMosSurVas11]	8	$n^{1+\Omega(1)}$	$O(1)$
[CroStu14]	4	$n^{1+\Omega(1)}$	$O(1)$
[AhnGuh15]	$1+\varepsilon$	$n^{1+\Omega(1)}$	$O(1/\varepsilon)$
[HarLiaLiu18]	2	$n^{1+\Omega(1)}$	$O(1)$

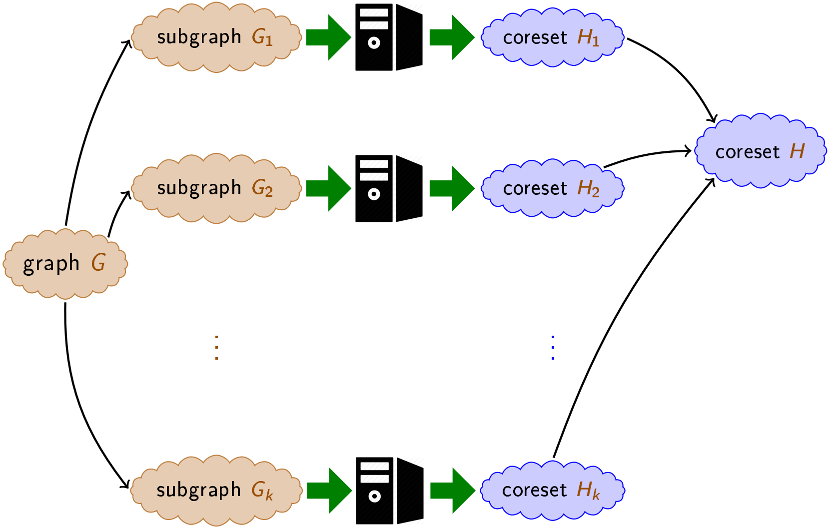
Credit	Approx	Space	Rounds
[LatMosSurVas11]	8	$\tilde{O}(n)$	$O(\log n)$
[CroStu14]	4	$\tilde{O}(n)$	$O(\log n)$
[AhnGuh15]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-1}\log n)$
[HarLiaLiu18]	2	$\tilde{O}(n)$	$O(\log n)$
[CzuLacMad+18]	$2+\varepsilon$	$\tilde{O}(n)$	$O\left(\varepsilon^{-\Theta(1/\varepsilon)} \cdot O(\log\log n)^2\right)$
[AssBatBer+19]	$2+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon)} \cdot \log\log n)$
[GamKalMitSve18]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon^2)} \cdot \log\log n)$
[LatMosSurVas11]	8	$n^{1+\Omega(1)}$	$O(1)$
[CroStu14]	4	$n^{1+\Omega(1)}$	$O(1)$
[AhnGuh15]	$1+\varepsilon$	$n^{1+\Omega(1)}$	$O(1/\varepsilon)$
[HarLiaLiu18]	2	$n^{1+\Omega(1)}$	$O(1)$
[AssKha17]	$O(1)$	$\tilde{O}(n\sqrt{n})$	2
[AssBatBer+19]	$3+\varepsilon$	$\tilde{O}(n\sqrt{n})$	2

Credit	Approx	Space	Rounds
[LatMosSurVas11]	8	$\tilde{O}(n)$	$O(\log n)$
[CroStu14]	4	$\tilde{O}(n)$	$O(\log n)$
[AhnGuh15]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-1}\log n)$
[HarLiaLiu18]	2	$\tilde{O}(n)$	$O(\log n)$
[CzuLacMad+18]	$2+\varepsilon$	$\tilde{O}(n)$	$O\left(\varepsilon^{-\Theta(1/\varepsilon)} \cdot O(\log\log n)^2\right)$
[AssBatBer+19]	$2+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon)} \cdot \log\log n)$
[GamKalMitSve18]	$1+\varepsilon$	$\tilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon^2)} \cdot \log\log n)$
[LatMosSurVas11]	8	$n^{1+\Omega(1)}$	$O(1)$
[CroStu14]	4	$n^{1+\Omega(1)}$	$O(1)$
[AhnGuh15]	$1+\varepsilon$	$n^{1+\Omega(1)}$	$O(1/\varepsilon)$
[HarLiaLiu18]	2	$n^{1+\Omega(1)}$	$O(1)$
[AssKha17]	$O(1)$	$\tilde{O}(n\sqrt{n})$	2
[AssBatBer+19]	$3+\varepsilon$	$\tilde{O}(n\sqrt{n})$	2
Our work	$2+\varepsilon$	$O(n\sqrt{n})$	2

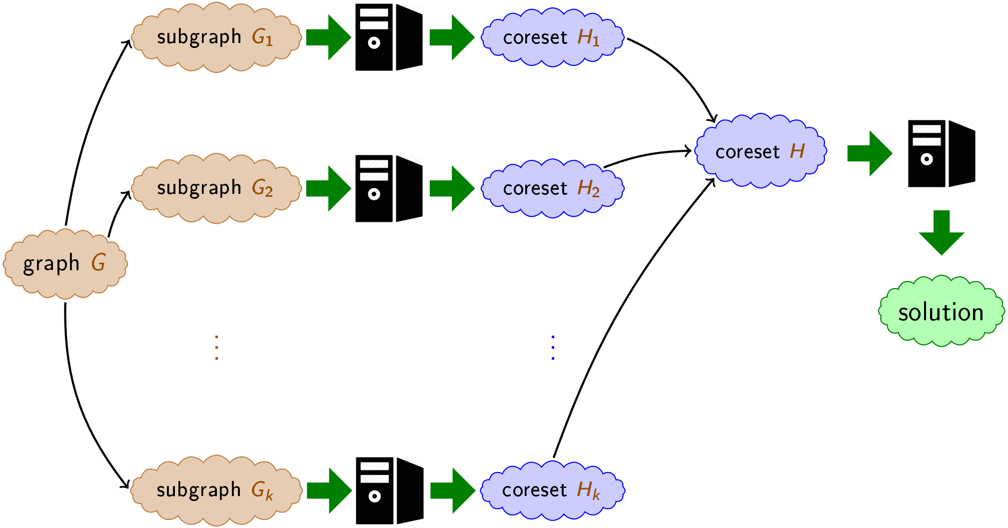

graph G


subgraph G_1
subgraph G_2
graph G
$\vdots$
subgraph G_k


graph G
subgraph G_1
subgraph G_2
subgraph G_k
coreset H_1
coreset H_2
coreset H_k



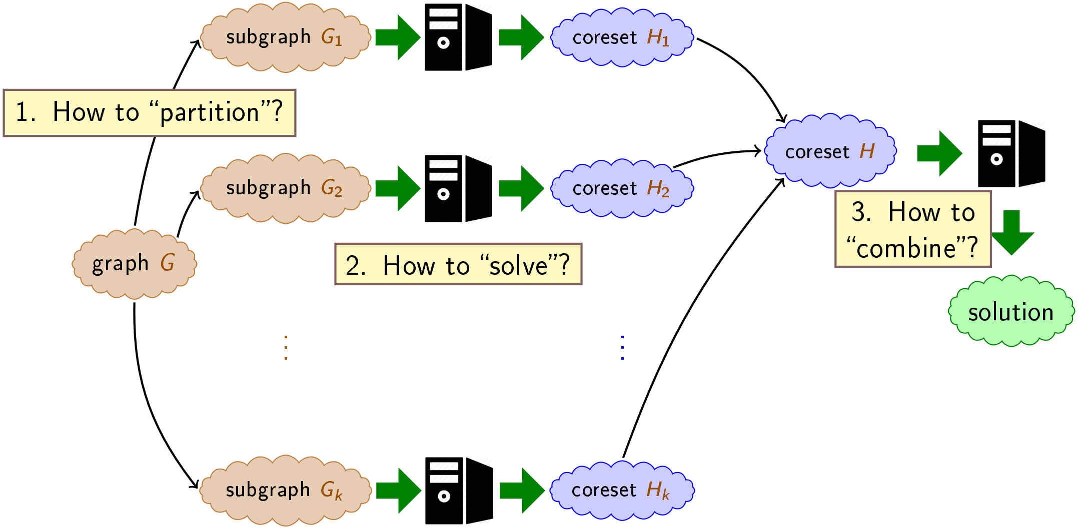
subgraph G_1
coreset H_1
subgraph G_2
coreset H_2
coreset H
graph G
⋮
⋮
subgraph G_k
coreset H_k



graph G
subgraph G_1
subgraph G_2
subgraph G_k
coreset H_1
coreset H_2
coreset H_k
coreset H
solution

Composable coresets

Basic idea



Definition

- Let $G_1, \ldots, G_k$ be a partitioning of G; send each edge $e \in G$ to a subgraph G_i arbitrarily.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx composable coreset of size s for a problem P iff $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

Definition

- Let $G_1, \ldots, G_k$ be a partitioning of G; send each edge $e \in G$ to a subgraph G_i arbitrarily.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx composable coreset of size s for a problem P iff $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

- Design ALG for small α (quality) and s (size).

Definition

- Let $G_1, \ldots, G_k$ be a partitioning of G; send each edge $e \in G$ to a subgraph G_i arbitrarily.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx composable coreset of size s for a problem P iff $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

- Design ALG for small α (quality) and s (size).
- $n^{o(1)}$ approx requires $n^{2-o(1)}$ space [AssKhaLiYar16].

Definition

- Let $G_1, \ldots, G_k$ be a random partitioning of G; send each edge $e \in G$ to a subgraph G_i uniformly at random.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx randomized composable coreset of size s for a problem P iff
  $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

- Design ALG for small α (quality) and s (size).
- $n^{o(1)}$ approx requires $n^{2-o(1)}$ space [AssKhaLiYar16].
- Randomized coresets were introduced by [MirZad15] for submodular maximization.

Definition

- Let $G_1, \ldots, G_k$ be a random μ-partitioning of G; send each edge $e \in G$ to μ subgraphs G_i uniformly at random.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx randomized composable coreset of size s and multiplicity μ for a problem P iff $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

- Design ALG for small α (quality) and s (size).
- $n^{o(1)}$ approx requires $n^{2-o(1)}$ space [AssKhaLiYar16].
- Randomized coresets were introduced by [MirZad15] for submodular maximization.

Definition

- Let $G_1, \ldots, G_k$ be a random μ-partitioning of G; send each edge $e \in G$ to μ subgraphs G_i uniformly at random.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx randomized composable coreset of size s and multiplicity μ for a problem P iff $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

- Maximum-matching coresets do not give $\alpha < 2$, but sparsification (EDCS) + randomized coresets give $\alpha = 1.5 + \epsilon$ approx [AssBatBer$^+$19].

Definition

- Let $G_1, \ldots, G_k$ be a random μ-partitioning of G; send each edge $e \in G$ to μ subgraphs G_i uniformly at random.
- Consider an algorithm ALG that given G_i outputs a subgraph H_i of G_i with s edges.
- ALG outputs an α-approx randomized composable coreset of size s and multiplicity μ for a problem P iff $P(\mathrm{ALG}(G_1) \cup \ldots \cup \mathrm{ALG}(G_k))$ is an α-approx to $P(G)$.

- Maximum-matching coresets do not give $\alpha < 2$, but sparsification (EDCS) + randomized coresets give $\alpha = 1.5 + \epsilon$ approx [AssBatBer+19].

We present a simple $(2 + \epsilon)$-approximation randomized coreset with mulitplicity $\mu = O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ for maximum-weight matching.

1. Send each edge to $\mu = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ machines.
2. Sort edges according to weight on each machine (with consistent tie-breaking), and greedily find a maximal matching (the coresets).
3. Find a maximum matching M of the union of coresets H.

- This gives a $2 + \epsilon$ approx.
- Simple, scalable implementation except for Step 3.
  - Replacing Step 3 with a greedy maximal matching algorithm is provably a $3 + \epsilon$ approx (not a $4 + \epsilon$ approx).
- Better analysis uses "consistency" of maximal (vs. maximum) matching coresets.

1. Send each edge to $\mu = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ machines.
2. Sort edges according to weight on each machine (with consistent tie-breaking), and greedily find a maximal matching (the coresets).
3. Find a maximum matching M of the union of coresets H.

- This gives a $2+\epsilon$ approx.
- Simple, scalable implementation except for Step 3.
  - Replacing Step 3 with a greedy maximal matching algorithm is provably a $3+\epsilon$ approx (not a $4+\epsilon$ approx).
- Better analysis uses "consistency" of maximal (vs. maximum) matching coresets.

1. Send each edge to $\mu = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ machines.
2. Sort edges according to weight on each machine (with consistent tie-breaking), and greedily find a maximal matching (the coresets).
3. Find a maximum matching M of the union of coresets H.

- This gives a $2+\epsilon$ approx.
- Simple, scalable implementation except for Step 3.
  - Replacing Step 3 with a greedy maximal matching algorithm is provably a $3+\epsilon$ approx (not a $4+\epsilon$ approx).
- Better analysis uses "consistency" of maximal (vs. maximum) matching coresets.

1. Send each edge to $\mu = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ machines.
2. Sort edges according to weight on each machine (with consistent tie-breaking), and greedily find a maximal matching (the coresets).
3. Find a maximum matching M of the union of coresets H.

- This gives a $2+\epsilon$ approx.
- Simple, scalable implementation except for Step 3.
  - Replacing Step 3 with a greedy maximal matching algorithm is provably a $3+\epsilon$ approx (not a $4+\epsilon$ approx).
- Better analysis uses "consistency" of maximal (vs. maximum) matching coresets.

1. Send each edge to $\mu = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ machines.
2. Sort edges according to weight on each machine (with consistent tie-breaking), and greedily find a maximal matching (the coresets).
3. Find a maximum matching M of the union of coresets H.

- This gives a $2 + \epsilon$ approx.
- Simple, scalable implementation except for Step 3.
  - Replacing Step 3 with a greedy maximal matching algorithm is provably a $3 + \epsilon$ approx (not a $4 + \epsilon$ approx).
- Better analysis uses "consistency" of maximal (vs. maximum) matching coresets.

Proof idea

- **Machine** i works on subgraph G_i and contributes maximal matching M_i to coreset $H = \cup_i M_i$.
- Compare to a reference maximum-weight matching M^*.
- Let permutation π of edges be the consistent sorting order.
- Edge $e \in M^*$ is free on machine i iff its endpoints are available when e "arrives."
  - Otherwise $e \in M^*$ is blocked on machine i.
  - Maybe e is missing on machine i.

1. Subgraphs G_i have the same distribution; focus on G_1 and M_1.
2. A blocked edge has a heavier edge in M_i as its "certificate."
   - Use it in the charging argument for M_1.
3. Free edge part of G_i will make it to M_i and H.
   - Free edges in $M_2 \cup M_3 \cup \dots$ compensate the free edges in M_1.
   - Subtle argument for "augmentation."

Proof idea

- **Machine** i works on subgraph G_i and contributes maximal matching M_i to coreset $H = \cup_i M_i$.
- Compare to a reference maximum-weight matching M^*.
- Let permutation π of edges be the consistent sorting order.
- Edge $e \in M^*$ is free on machine i iff its endpoints are available when e "arrives."
  - Otherwise $e \in M^*$ is blocked on machine i.
  - Maybe e is missing on machine i.

1. Subgraphs G_i have the same distribution; focus on G_1 and M_1.
2. A blocked edge has a heavier edge in M_i as its "certificate."
   - Use it in the charging argument for M_1.
3. Free edge part of G_i will make it to M_i and H.
   - Free edges in $M_2 \cup M_3 \cup \ldots$ compensate the free edges in M_1.
   - Subtle argument for "augmentation."

Proof idea

- **Machine** i works on subgraph G_i and contributes maximal matching M_i to coreset $H = \cup_i M_i$.
- Compare to a reference maximum-weight matching M^*.
- Let permutation π of edges be the consistent sorting order.
- Edge $e \in M^*$ is free on machine i iff its endpoints are available when e "arrives."
  - Otherwise $e \in M^*$ is blocked on machine i.
  - Maybe e is missing on machine i.

1. Subgraphs G_i have the same distribution; focus on G_1 and M_1.
2. A blocked edge has a heavier edge in M_i as its "certificate."
   - Use it in the charging argument for M_1.
3. Free edge part of G_i will make it to M_i and H.
   - Free edges in $M_2 \cup M_3 \cup \ldots$ compensate the free edges in M_1.
   - Subtle argument for "augmentation."

Proof idea

- **Machine** i works on subgraph G_i and contributes maximal matching M_i to coreset $H = \cup_i M_i$.
- Compare to a reference maximum-weight matching M^*.
- Let permutation π of edges be the consistent sorting order.
- Edge $e \in M^*$ is free on machine i iff its endpoints are available when e "arrives."
  - Otherwise $e \in M^*$ is blocked on machine i.
  - Maybe e is missing on machine i.

1. Subgraphs G_i have the same distribution; focus on G_1 and M_1.
2. A blocked edge has a heavier edge in M_i as its "certificate."
   - Use it in the charging argument for M_1.
3. Free edge part of G_i will make it to M_i and H.
   - Free edges in $M_2 \cup M_3 \cup \ldots$ compensate the free edges in M_1.
   - Subtle argument for "augmentation."

Proof idea

- **Machine** i works on subgraph G_i and contributes maximal matching M_i to coreset $H = \cup_i M_i$.
- Compare to a reference maximum-weight matching M^*.

- Let permutation π of edges be the consistent sorting order.
- Edge $e \in M^*$ is free on machine i iff its endpoints are available when e "arrives."
  - Otherwise $e \in M^*$ is blocked on machine i.
  - Maybe e is missing on machine i.

1. Subgraphs G_i have the same distribution; focus on G_1 and M_1.
2. A blocked edge has a heavier edge in M_i as its "certificate."
   - Use it in the charging argument for M_1.
3. Free edge part of G_i will make it to M_i and H.
   - Free edges in $M_2 \cup M_3 \cup \dots$ compensate the free edges in M_1.
   - Subtle argument for "augmentation."

- Focus on 2-round algorithms: [AssBatBer+19] too complicated
- [AssKha17] without [CroStu14] not viable for *weighted* matching

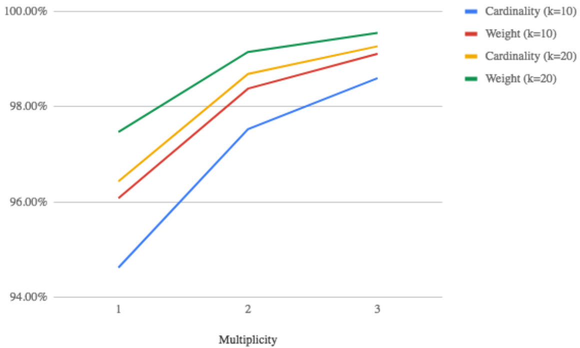
- Focus on 2-round algorithms: [AssBatBer+19] too complicated
- [AssKha17] without [CroStu14] not viable for *weighted* matching
- [AssKha17] with [CroStu14] not as good as the new algorithm

Dataset	$\lvert V \rvert$	$\lvert E \rvert$	Δ	Speed-up [AssKha17]	Speed-up This work
Friendster	66M	550B	2.1M	4.7x	130x
Orkut	3M	21B	900K	1.4x	17x
LiveJournal	4.8M	3.9B	444K	2.5x	6x
DBLP	1.5M	5.9M	1961	1.2x	3x

- Focus on 2-round algorithms: [AssBatBer+19] too complicated
- [AssKha17] without [CroStu14] not viable for *weighted* matching
- [AssKha17] with [CroStu14] not as good as the new algorithm

Dataset				Quality	
	$\lvert V\rvert$	$\lvert E\rvert$	Δ	[AssKha17]	This work
Friendster	66M	550B	2.1M	94.4%	99.8%
Orkut	3M	21B	900K	92.4%	98.9%
LiveJournal	4.8M	3.9B	444K	96.5%	99.9%
DBLP	1.5M	5.9M	1961	92.4%	99.6%

- Focus on 2-round algorithms: [AssBatBer+19] too complicated
- [AssKha17] without [CroStu14] not viable for *weighted* matching
- [AssKha17] with [CroStu14] not as good as the new algorithm
- Multiplicity 2 or 3 is sufficient



Summary

- A simple, scalable algorithm for maximum-weight matching with $2 + \epsilon$ approximation.
    - Uses greedy algorithm for coreset construction.
    - Uses multiplicity.
- Good performance in practice.
    - Small multiplicity suffices.
    - Beats prior algorithms.
    - Huge speed-up of sequential algorithm, but faithful simulation.

Summary

- A simple, scalable algorithm for maximum-weight matching with $2 + \epsilon$ approximation.
  - Uses greedy algorithm for coreset construction.
  - Uses multiplicity.
- Good performance in practice.
  - Small multiplicity suffices.
  - Beats prior algorithms.
  - Huge speed-up of sequential algorithm, but faithful simulation.

Thanks!

Poster #161
Pacific Ballroom

- Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *SPAA*, pages 202–211, 2015.
- Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *SODA*, pages 1616–1635, 2019.
- Sepehr Assadi and Sanjeev Khanna. Randomized composable coresets for matching and vertex cover. In *SPAA*, pages 3–12, 2017.
- Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *SODA*, pages 1345–1364, 2016.
- Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *APPROX*, pages 96–104, 2014.
- Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *STOC*, pages 471–484, 2018.

Biblios II

- Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.
- Jack Edmonds. Maximum matching and a polyhedron with $0, 1$-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.
- Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.
- Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.
- Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. *CoRR*, abs/1811.02760. To appear in PODC 2019., 2018.
- Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the mapreduce model. In *SPAA*, pages 43–52, 2018.
- Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *SPAA*, pages 1–10, 2013.

Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, pages 85–94, 2011.

Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *STOC*, pages 153–162, 2015.