Distributed Weighted Matching via Randomized Composable Coresets

Sepehr Assadi [1] **MohammadHossein Bateni** [2] **Vahab Mirrokni** [2]

Abstract

Maximum weight matching is one of the most fundamental combinatorial optimization problems with a wide range of applications in data mining and bioinformatics. Developing distributed weighted matching algorithms is challenging due to the sequential nature of efficient algorithms for this problem. In this paper, we develop a simple distributed algorithm for the problem on general graphs with approximation guarantee of $2+\epsilon$ that (nearly) *matches* that of the sequential *greedy* algorithm. A key advantage of this algorithm is that it can be easily implemented in only *two rounds* of computation in modern parallel computation frameworks such as MapReduce. We also demonstrate the efficiency of our algorithm in practice on various graphs (some with half a trillion edges) by achieving objective values always close to what is achievable in the centralized setting.

1. Introduction

A matching in a graph is defined as a collection of edges that do not share any vertices. The problem of finding a matching with a maximum weight in an edge-weighted graph—henceforth referred to as the *maximum weight matching* (MWM) problem—is one of the most fundamental combinatorial optimization problems with a wide range of applications in data mining and bioinformatics. For instance, maximum weight matchings can improve the quality of data clustering (Bateni et al., 2017) or partitioning (Karypis & Kumar, 1998), as well as discovery of subgraphs in networks in bioinformatics (Langmead & Donald, 2004; Berger et al., 2008). Other applications are in trading markets and computational advertising (Penn & Tennenholtz, 2000; Mehta et al., 2007; Charles et al., 2010), kidney exchange (Dickerson et al., 2012; Blum et al., 2015), online labor markets (Behnezhad & Reyhani, 2017), and semi-supervised learning (Jebara et al., 2009) (see (Manshadi et al., 2013) for other similar examples). Yet another application arise in numerical linear algebra, e.g., in sparse linear solvers (Duff & Koster, 1999; 2001), decomposition of sparse matrices (Pothen & Fan, 1990), and computing sparse bases for underdetermined matrices (Pinar et al., 2005).

The study of MWM dates back to the introduction of the complexity class $\mathbf{P}$ as the set of "tractable" problems by (Edmonds, 1965b;a) who designed a poly-time algorithm for this problem on general graphs. Since then, there have been numerous attempts in developing faster algorithms for MWM (see, e.g., (Gabow, 1976; Gabow et al., 1984; Gabow, 1985; 1990; Gabow & Tarjan, 1991; Cygan et al., 2012; Duan et al., 2017)) culminating in the $\widetilde{O}(m\sqrt{n})$-time[1] algorithm of (Gabow & Tarjan, 1991). For approximation algorithms, the greedy algorithm that repeatedly picks the heaviest edge possible in the matching achieves a two approximation and after a series of work (Preis, 1999; Vinkemeier & Hougardy, 2005; Pettie & Sanders, 2004; Duan & Pettie, 2010), an $\widetilde{O}(m/\varepsilon)$-time algorithm for $(1+\varepsilon)$-approximation was developed by (Duan & Pettie, 2014).

Nowadays, many applications that involve MWM require processing massive graphs that are typically being stored and processed in a distributed fashion. Classical algorithmic approaches to MWM are no longer viable options to cope with challenges that stem from processing massive graphs and one should now instead focus on algorithms that can be implemented *efficiently in distributed settings* even at the cost of a (slight) reduction in the quality of the solution.

In this paper, we design a simple and efficient greedy distributed algorithm for the maximum weight matching problem with an approximation ratio that nearly matches that of the sequential greedy algorithm for MWM. Our algorithm can also be easily implemented in two rounds of parallel computation in MapReduce-style computation frameworks, which is known to be the minimum number of rounds necessary for solving this problem.

A full version of the paper including the missing proofs and details is available on arXiv.

[1]Department of Computer Science, Princeton University, Princeton, NJ, US. Supported in part by the Simons Collaboration on Algorithms and Geometry. Majority of the work done while this author was a summer intern at Google Research, New York.
[2]Google Research, New York, NY, US.
Correspondence to: Sepehr Assadi <sassadi@princeton.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

[1]Throughout the paper, we use $\widetilde{O}(f) := O(f) \cdot \text{polylog}(f)$.

1.1. Background and Related Work

Maximum weight and maximum cardinality matchings have been studied extensively in different models of computation for processing massive graphs such as streaming and distributed settings; see, e.g., (McGregor, 2005; Epstein et al., 2011; Goel et al., 2012; Konrad et al., 2012; Ahn et al., 2012; Ahn & Guha, 2013; Kapralov, 2013; Manshadi et al., 2013; Crouch & Stubbs, 2014; Assadi et al., 2016; 2017; Paz & Schwartzman, 2017; Ahn & Guha, 2015; Lattanzi et al., 2011; Huang et al., 2015; Assadi & Khanna, 2017; Czumaj et al., 2018; Harvey et al., 2018; Assadi et al., 2019).

Closely related to our work, (Lattanzi et al., 2011) designed MapReduce algorithms with 2- and 8-approximation guarantees, respectively, for unweighted and weighted matchings in $O(1)$ rounds on machines with memory $n^{1+\Omega(1)}$. These results were subsequently improved to $(1+\varepsilon)$-approximation for both problems in $O(1/\varepsilon)$ rounds by (Ahn & Guha, 2015) using sophisticated primal-dual algorithms and multiplicative-weight-update method (for *unweighted bipartite* matching, a simpler algorithm with $(1+\varepsilon)$-approximation in $O(1/\varepsilon)$ rounds using $O(n\sqrt{n})$ space was recently proposed in (Behnezhad et al., 2017)). Very recently, (Harvey et al., 2018) designed a 2-approximation algorithm for weighted matchings in $O(1)$ rounds based on the local-ratio theorem of (Paz & Schwartzman, 2017) for MWM. Furthermore, (Assadi & Khanna, 2017) designed a MapReduce algorithm with $O(n\sqrt{n})$ memory—using the so-called randomized composable coreset method which we also exploit in this paper—that achieves an $O(1)$-approximation to both problems in only two rounds of computation which is the optimal number of rounds by a result of (Assadi et al., 2016). This result was very recently improved by (Assadi et al., 2019) to (almost) 1.5-approximation for unweighted matchings. Recent papers by (Czumaj et al., 2018; Assadi et al., 2019; Ghaffari et al., 2018) also considered these problems with smaller per-machine memory and achieved $(1+\varepsilon)$- and $(2+\varepsilon)$-approximation for unweighted and weighted matchings in $O(\log\log n)$ rounds and $\widetilde{O}(n)$ memory per-machine. The approximation ratio for weighted matchings in these results was recently improved to $(1+\varepsilon)$ (Gamlath et al., 2018).

Our work is also closely aligned with the trend on "parallelizing" sequential greedy algorithms in distributed settings, e.g., (Kumar et al., 2013; Mirrokni & Zadimoghaddam, 2015; da Ponte Barbosa et al., 2015; 2016; Harvey et al., 2018). As noted elegantly by (Kumar et al., 2013): "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." As such there have been extensive efforts in recent years to carry over the greedy algorithms in the sequential setting to distributed models as well. These results are typically of two types: they either use a relatively large number of rounds to "faithfully" simulate the greedy algorithm, i.e., to obtain approximation guarantees that (almost) match that of the greedy algorithm (Kumar et al., 2013; da Ponte Barbosa et al., 2016), or use a very small number of rounds, say one or two, for "weak" simulation, resulting in approximation guarantees that are within some constant factor of the corresponding greedy algorithm (Mirrokni & Zadimoghaddam, 2015; da Ponte Barbosa et al., 2015). Table 1 provides a summary of previous work.

1.2. Our Contribution

In this paper, we "faithfully" parallelize the sequential greedy algorithm for MWM in two rounds of parallel computation. In particular, we present an algorithm in the MapReduce framework (defined formally in Section 2) that for any constant $\varepsilon > 0$, outputs a $(2+\varepsilon)$-approximation to maximum weight matching in expectation using $O(\sqrt{\frac{m}{n}})$ machines each with $O(\sqrt{mn})$ memory and in only two rounds of computation; here, m and n denote the number of edges and vertices in the graph, respectively. See Theorem 2 for the formal statement of this result.

Our distributed algorithm works as follows: send each edge of the graph to $O(1)$ machines *randomly*, run the greedy algorithm—the one that repeatedly picks the heaviest available edge in the matching—on each part separately, combine the output of the greedy algorithms on a single machine, and find a near-optimal weighted matching among these edges using any standard offline algorithm, say algorithm of (Duan & Pettie, 2014) (see also Section 3.1 for details on when one can simply run the greedy algorithm at the end). We prove that this simple algorithm leads to an almost two approximate matching of the original graph. This technique of partitioning the input randomly and computing a subgraph of each piece (here, a matching output by the greedy algorithm) is called the *randomized composable coreset* technique and has been used previously in context of unweighted matchings (Assadi & Khanna, 2017; Assadi et al., 2019) and constrained submodular maximization (Mirrokni & Zadimoghaddam, 2015; da Ponte Barbosa et al., 2015) (see Section 2.1). Finally, the number of rounds of our algorithm is *optimal* by a result of (Assadi et al., 2016).

Comparison with prior work. We conclude this section by making the following two comparisons between our result and previous results in the literature:

- Number of rounds used by our algorithm is an **absolute constant two**, independent of the approximation of the algorithm. This significantly improves upon the previ-

Table 1: A summary of previous work on MapReduce algorithms for MWM and our result.

Reference	Approximation	Memory Per-Machine	Rounds
(Lattanzi et al., 2011)	8	$O(n)$	$O(\log n)$
(Crouch & Stubbs, 2014)	4	$O(n)$	$O(\log n)$
(Ahn & Guha, 2015)	$1+\varepsilon$	$\widetilde{O}(n)$	$O(\varepsilon^{-1}\log n)$
(Harvey et al., 2018)	2	$O(n)$	$O(\log n)$
(Czumaj et al., 2018)	$2+\varepsilon$	$O(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon)}\cdot(\log\log n)^2)$
(Assadi et al., 2019)	$2+\varepsilon$	$\widetilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon)}\cdot\log\log n)$
(Gamlath et al., 2018)	$1+\varepsilon$	$\widetilde{O}(n)$	$O(\varepsilon^{-\Theta(1/\varepsilon^2)}\cdot\log\log n)$
(Lattanzi et al., 2011)	8	$n^{1+\Omega(1)}$	$O(1)$
(Crouch & Stubbs, 2014)	4	$n^{1+\Omega(1)}$	$O(1)$
(Ahn & Guha, 2015)	$1+\varepsilon$	$n^{1+\Omega(1)}$	$O(1/\varepsilon)$
(Harvey et al., 2018)	2	$n^{1+\Omega(1)}$	$O(1)$
(Assadi & Khanna, 2017)	$O(1)$	$\widetilde{O}(n\sqrt{n})$	2
(Assadi et al., 2019)	$3+\varepsilon$	$\widetilde{O}(n\sqrt{n})$	2
This paper	$2+\varepsilon$	$O(n\sqrt{n})$	2

ously best MapReduce algorithms of (Ahn & Guha, 2015; Harvey et al., 2018) that require a **large unspecified constant** number of rounds to achieve a similar guarantee on the approximation ratio. Other algorithms for weighted matching with similar guarantee on the number of rounds as ours are that of (Lattanzi et al., 2011) that achieves 8-approximation (improvable to (almost) 4-approximation using the Crouch-Stubbs technique (Crouch & Stubbs, 2014)) in *six* rounds when using the same per-machine memory as ours (and *at least three* rounds by allowing even more memory), and (almost) 3-approximation of (Assadi et al., 2019) which is based on a considerably complicated algorithm (as it first finds an approximation to *unweighted* matching with better than 2-approximation which is well-known to be a "hard task" for matchings[2]). We emphasize that the main bottleneck in MapReduce computation is the transition between different rounds (see, e.g., (Lattanzi et al., 2011)) and hence minimizing the number of rounds is the primary goal in this setting.

- Previous work on parallelizing greedy algorithms in MapReduce framework suffered from one of the following two drawbacks: either a **suboptimal approximation guarantee** compared to the greedy algorithm even by allowing unbounded computation time on each machine (da Ponte Barbosa et al., 2015; Mirrokni & Zadimoghaddam, 2015), or a **relatively large number of rounds** to match the performance of the greedy algorithm exactly or to within a factor of $(1+\varepsilon)$ (da Ponte Barbosa et al., 2016; Kumar et al., 2013; Harvey et al., 2018). Obtaining MapReduce algorithms that can (almost) match the performance of the greedy algorithm without blowing up the number of rounds (in the context of submodular maximization) has been posed as an open question very recently (Liu & Vondrák, 2019). To our knowledge, ours is the *first* parallel implementation of the greedy algorithm in a minimal number of rounds with (almost) no blow-up in the approximation ratio. It is a fascinating open question if our improvement for MWM can be extended to other greedy algorithms, in particular, for constrained submodular maximization.

In addition to aforementioned theoretical improvements over previous works, our algorithm has the benefit of being extremely simple (with nearly all the details "pushed" to the analysis), making it easily implementable in the MapReduce model (the only other MapReduce algorithm for matching that we know of to be implemented previously is (Behnezhad et al., 2017) which is *limited to bipartite graphs in a crucial way*). We believe this additional feature of our algorithm is an important contribution of this paper. We discuss this further in Section 4 where we present our

[2]For instance, getting efficient algorithms with better than 2-approximation in both streaming and dynamic graphs models are longstanding open problems, and very recently proven to be impossible in the (general) online model (Gamlath et al., 2019).

experimental results and in more details in the full version of the paper in which we further compare our algorithm with prior algorithms in practice.

2. Preliminaries

Throughout, $[t] := \{1, \ldots, t\}$. For a graph $G(V, E)$, $\text{opt}(G)$ denotes the weight of a maximum weight matching in G.

Greedy algorithm. Let $G(V, E)$ be a graph and $\pi := \pi(E)$ denote any permutation of E. $\textsf{Greedy}(G, \pi)$ denotes the standard greedy algorithm which iterates over edges according to π and add $e = (u, v)$ to the matching iff both u and v are unmatched. It is a standard fact that when π is sorted in non-increasing order of weights, $\textsf{Greedy}(G, \pi)$ outputs a 2-approximation to $\text{opt}(G)$.

MapReduce framework. We adopt the MapReduce model as formalized by Karloff et al. (Karloff et al., 2010); see also (Goodrich et al., 2011; Beame et al., 2013). Let $G(V, E)$ with $n := |V|$ and $m := |E|$ be the input graph. In this model, there are p machines, each with a memory of s such that $p \cdot s = O(m)$, i.e., at most a constant factor larger than the input size, and both $p, s = m^{1-\Omega(1)}$, i.e., sublinear in the input size. The motivation behind these constraints is that the number of machines, and local memory of each machine should be much smaller than the input size to the problem since these frameworks are used to process massive datasets. Computation in this model proceeds in synchronous rounds: in each round, each machine performs some local computation and at the end of the round machines exchange messages to guide the computation for the next round. All messages received by each machine in one round have to fit into the memory of the machine.

2.1. Randomized Composable Coresets

We briefly review the notion of randomized composable coresets originally introduced by (Mirrokni & Zadimoghaddam, 2015) in the context of submodular maximization, and further refined in (Assadi & Khanna, 2017) for graph problems. Our definition slightly deviates from previous works as we will remark below.

Let E be an edge-set of a weighted graph $G(V, E)$. Let $c \geq 1$ be a parameter. A collection of edges $\left\{E^{(1)}, \ldots, E^{(k)}\right\}$ is a *random k-clustering* of E with *expected multiplicity* c iff each edge e in E is sent to c_e different sets $E^{(i_1)}, \ldots, E^{(i_{c_e})}$ chosen uniformly at random, where c_e is chosen independently for each edge from the *binomial distribution* with k trials and expected value c. A random clustering of E naturally defines clustering the graph G into k subgraphs $G^{(1)}, \ldots, G^{(k)}$ where $G^{(i)} := G(V, E^{(i)})$ for all $i \in [k]$; as a result, we use random clustering for both the edge-set and the input graph interchangeably.

Definition 1 (cf. (Mirrokni & Zadimoghaddam, 2015; Assadi & Khanna, 2017))**.** *Consider an algorithm* $\textsf{ALG}$ *that given a graph $G(V, E)$ outputs a subgraph $\textsf{ALG}(G) \subseteq G$ with at most s edges. Let $k, c \geq 1$ be integers and $G^{(1)}, \ldots, G^{(k)}$ denote a random k-clustering with expected multiplicity c of a graph G. We say that* $\textsf{ALG}$ *outputs an α-approximate (k, c)-randomized composable coreset of size s for the weighted matching problem iff*

$$\alpha \cdot \mathbb{E}\left[\text{opt}\left(\textsf{ALG}(G^{(1)}) \cup \ldots \cup \textsf{ALG}(G^{(k)})\right)\right] \geq \text{opt}(G),$$

where $\text{opt}(\cdot)$ *denotes the weight of a maximum weight matching in the given graph. Here, the expectation is taken over the random choice of the random clustering.*

We remark that our definition is somewhat different from (Mirrokni & Zadimoghaddam, 2015; Assadi & Khanna, 2017) in the following sense: Previous works considered the case where each edge is sent to *exactly* c different subgraphs (only $c = 1$ was considered in (Assadi & Khanna, 2017)), while we send each edge to c subgraphs in *expectation*. This way of partitioning has the simple yet quite helpful property that makes the distribution of graphs $G^{(1)}, \ldots, G^{(k)}$ a *product* distribution, i.e., each graph $G^{(i)}$ is chosen *independently* even conditioned on all other graphs in the random k-clustering. At the same time, size of each subgraph and the total number of edges across all subgraphs, are still respectively $O(m \cdot c/k)$ and $c \cdot m \pm \Theta(\sqrt{c \cdot m \log n})$ with high probability.

Randomized Coresets in MapReduce Framework. Suppose $G(V, E)$ is the input and let $k := \sqrt{m \cdot c/s}$. We use a randomized coreset to obtain a MapReduce algorithm:

1. **Random clustering:** Create a random k-clustering $G^{(1)}, \ldots, G^{(k)}$ of expected multiplicity c and allocate each graph $G^{(i)}$ to the machine $i \in [k]$.
2. **Coreset:** Each machine $i \in [k]$ creates a randomized composable coreset $C_i \leftarrow \textsf{ALG}(G^{(i)})$.
3. **Post-processing:** Collect the union of coresets to create $H := H(V, C_1, \ldots, C_k)$ on one machine and return a β-approximation to MWM on H using *any* offline algorithm.

It is straightforward to verify that the algorithm requires $O(k) = O(\sqrt{m \cdot c/s})$ machines with $O(\sqrt{m \cdot c \cdot s} + n)$ memory and only *two* rounds of computation. Moreover, by Definition 1, the output of this algorithm is an $(\alpha \cdot \beta)$-approximation to maximum weight matching of G.

3. A Randomized Coreset for Maximum Weight Matching

We present a simple randomized composable coreset for MWM in this section and then use it to design an efficient MapReduce algorithm for this problem. In the full version, we prove the optimality of the size of our coreset using an adaptation of the argument in (Assadi & Khanna, 2017).

Theorem 1. *For $\varepsilon > 0$, there exists a $(2+\varepsilon)$-randomized composable coreset of size $O(n)$ with expected multiplicity $O\left(\frac{\log(1/\varepsilon)}{\varepsilon}\right)$ for the maximum weight matching problem.*

Our coreset in Theorem 1 is simply the greedy algorithm for MWM (with consistent tie-breaking). Let $G(V,E)$ be a graph and $G^{(1)},\ldots,G^{(k)}$ be a random k-clustering of G with expected multiplicity c. We propose the GreedyCoreset for approximating MWM on G: on each subgraph $G^{(i)}$, simply return $M_i := \mathsf{Greedy}(G^{(i)},\pi_i)$ as the coreset, where π_i sorts the edges in $G^{(i)}$ in non-increasing order of their weights (breaking the ties *consistently* across all $i \in [k]$). In the following lemma, we analyze the performance of this coreset.

Lemma 3.1. *Suppose $G^{(1)},\ldots,G^{(k)}$ is a random k-clustering of G with expected multiplicity c and $M_i := \mathsf{Greedy}(G^{(i)},\pi_i)$. Define the graph $H(V,E(H))$ with $E(H) := \bigcup_{i=1}^{k} M_i$; then,*

$$\mathbb{E}\left[\mathrm{opt}(H)\right] \geq \left(\frac{1}{2} - O\left(\frac{\log c}{c}\right)\right)\cdot \mathrm{opt}(G).$$

Theorem 1 follows immediately from Lemma 3.1 (by setting $c = \Theta\left(\frac{\log(1/\varepsilon)}{\varepsilon}\right)$). The rest of this section is devoted to the proof sketch of Lemma 3.1. Missing proofs are all deferred to the full version of the paper.

Notation. Let π be a permutation of all edges in G in non-increasing order of their weights (*consistent* with orderings π_i for $i \in [k]$). Notice that for each $i \in [k]$, $M_i = \mathsf{Greedy}(G^{(i)},\pi)$ as well. Hence, in the following, we use the permutation π instead of each π_i. Throughout the proof, we fix an arbitrary maximum weight matching M^* in G and denote by $w(M^*) = \mathrm{opt}(G)$ the weight of M^*. For any edge $e \in M^*$, we use $\pi^{<e}$ to refer to the set of edges in π that appear before e. We slightly abuse the notation and use $\mathsf{Greedy}(G,\pi^{<e})$ to mean that we run $\mathsf{Greedy}(G,\pi)$ and stop exactly before processing the edge e, i.e., we only consider the edges in $\pi^{<e}$.

Definition 2 (Free/Blocked Edges)**.** *We say that an edge $e \in M^*$ is* free *for machine $i \in [k]$ iff no end point of e is matched by $\mathsf{Greedy}(G^{(i)},\pi^{<e})$; otherwise we call $e \in M^*$* blocked*. We use F_i to denote the set of free edges in $G^{(i)}$ and B_i to denote the blocked edges.*

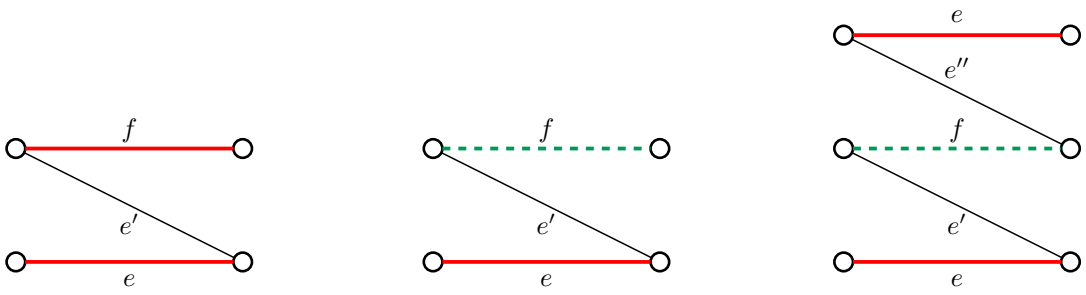
We emphasize that in Definition 2, an edge $e \in M^*$ can be free or blocked on some machine $i \in [k]$, without necessarily even appearing in $G^{(i)}$. In other words, this definition is independent of whether e belongs to $G^{(i)}$ or not. Notice that if an edge e is free on machine i and it also appears in $G^{(i)}$, then e would definitely belong to the matching M_i (i.e., the coreset on machine i). On the other hand, if an edge e is blocked in machine i, then necessarily some edge e' exists in M_i such that e' is incident on e and $w(e') \geq w(e)$. We refer to e' as the *certificate* of e in machine i.

Overview. The idea behind the proof of Lemma 3.1 is as follows. Recall that the distribution of each graph $G^{(i)}$ in the random clustering is the same, and is independent of other graphs. Hence, we can focus on each machine $i \in [k]$, say machine 1, separately. Consider blocked edges B_1 in machine 1: for any such edge, we have already picked another edge with at least the same weight in the matching M_1 of machine 1 (by definition of an edge being blocked). We can hence use a simple charging argument here to argue that the matching M_1 of machine 1 already has enough edges to "compensate" for blocked edges of M^* that were not picked by machine 1.

The main part of the argument is however to show that we can find enough edges in $M_2,\ldots,M_k$ chosen by other machines that can be added to M_1 to also compensate for free edges in machine 1. The idea here is that since the distribution of input to all machines is the same and is independent across, if an edge e is free in machine 1, it is "most likely" free in *many* other machines as well, in particular, in a machine $j \in [k]\setminus\{1\}$ which also *contains* this edge. By definition, this edge then would be chosen in matching M_j. We then use another careful charging scheme to argue that we can indeed "augment" the matching M_1 by free edges in F_1 that appear in $M_2,\ldots,M_k$ to obtain an almost-two approximation. The main difficulty here is that even though edges in F_1 were free in machine 1, they may still be incident on edges in M_1 with equal or smaller weight (as being free only implies that these edges were not incident on edges with *higher* weight) and hence they cannot be readily added to M_1; this is why we need to find "short augmenting paths" in $F_1 \cup M_1$ which require switching some edges out of M_1.

We now start with the formal proof. Throughout, define $F_1' := F_1 \cap E(H)$, i.e., the set of free edges in machine 1 that are present in H. We refer to these edges as *available* free edges (we prove later that essentially any free edge is also available with a large probability). To perform the charging, we need to partition the edges of F_1', B_1 and M_1 as follows (see Figure 1 for an illustration):

1. Let e be a max-weight edge in B_1 and e' be its certificate in M_1. Also, let f be the *other* edge incident on e' in $F_1' \cup B_1$ ($f = \perp$ if no such edge exists).



(a) A set of type 1 edges. (b) A set of type 2 edges. (c) A set of type 3 edges.

Figure 1: Illustration of the partitioning used in Lemma 3.1. Thick solid edges (red) are blocked edges, thick dashed edges (green) are available free edges, and normal edges (black) are certificate edges.

2. **Type 1 edges:** If f belongs to B_1 or is $\perp$, then add e' to $M_{1,1}$ and e, f to $B_{1,1}$. Remove *both* e and f from B_1. We refer to (e, f, e') as a set of type 1 edges.
3. **Type 2 edges:** If $f \in F'_1$ and f is *not* incident on any certificate edge other than e', then add f to $F'_{1,2}$, e to $B_{1,2}$, and e' to $M_{1,2}$. Remove f from F'_1 and e from B_1. We refer to (e, f, e') as a set of type 2 edges.
4. **Type 3 edges:** If $f \in F'_1$ and f is incident on another certificate edge e'' which is a certificate for some edge $z \in B_1$, then add f to $F'_{1,3}$, e, z to $B_{1,3}$, and e', e'' to $M_{1,3}$. Remove f from $F'_{1,3}$ and e, z from $B_{1,3}$. We refer to (e, f, z, e', e'') as a set of type 3 edges.
5. Continue the process from first line until no edge remains in B_1. Add the remaining edges in F'_1 after this step the set to $F'_{1,0}$.

It is immediate to verify that $F'_1 = F'_{1,0} \cup F'_{1,2} \cup F'_{1,3}$, $B_1 = B_{1,1} \cup B_{1,2} \cup B_{1,3}$ and $M_{1,1} \cup M_{1,2} \cup M_{1,3} \subseteq M_1$, and all these sets are pairwise disjoint.

Claim 1. *In the partitioning scheme, for any set of edges:*

- *type* 1 (e, f, e')*:* $w(e') \geq \frac{1}{2} \cdot (w(e) + w(f))$.
- *type* 2 (e, f, e')*:* $w(f) \geq w(e') \geq w(e)$.
- *type* 3 (e, f, z, e', e'')*:* $\max\{w(f), w(e') + w(e'')\} \geq \frac{1}{2} \cdot (w(e) + w(f) + w(z))$.

Claim 1 can then be used to lower bound the weight of the maximum weight matching in the graph H using a careful charging scheme whose proof (along with the proof of Claim 1) appear in the full version.

Lemma 3.2. $\text{opt}(H) \geq \frac{1}{2} \cdot (w(F'_1) + w(B_1))$.

Finally, we use the randomness in the clustering to argue that nearly all free edges on machine 1 are also available.

Lemma 3.3. $\mathbb{E}\left[w(F'_1)\right] \geq \mathbb{E}\left[w(F_1)\right] - O\left(\frac{\log c}{c}\right) \cdot \text{opt}$.

Proof. Firstly, $\mathbb{E}\left[w(F'_1)\right]$ is:

$$= \sum_{e \in M^*} w(e) \cdot \Pr\left(e \in F_1 \wedge e \in M_j \text{ for some } j \in [k]\right)$$
$$\geq \sum_{e \in M^*} w(e) \cdot \Pr\left(e \in F_1\right) \cdot \Pr\left(e \in M_j \text{ for some } j \in [k]\right)$$

This is because random clustering induces a product distribution on inputs across machines and hence the fact that e is free in $G^{(1)}$ is independent of whether e is picked in some other matching M_j for $j \neq i$. We can now calculate the probability that an edge e belongs to a matching M_j for a fixed $j \in [k]$.

$$\begin{aligned}\Pr\left(e \in M_j\right) &= \Pr\left(e \in F_j \text{ and } e \text{ is sampled in } G^{(j)}\right)\\ &= \Pr\left(e \in F_j\right) \cdot \Pr\left(e \in G^{(j)}\right)\\ &= \Pr\left(e \in F_j\right) \cdot \frac{c}{k}, \end{aligned} \tag{1}$$

since each edge is in $G^{(j)}$ with probability c/k. Now notice that the marginal distribution of the graph $G^{(1)}$ and $G^{(j)}$ under random clustering is the same; as a result the probability that an edge is good in $G^{(j)}$ is equal to this probability for the graph $G^{(1)}$ as well. Using this, plus the fact that the event that e belongs to M_j is independent of all other graphs $G^{(\ell)}$ for $\ell \neq j$, we have, $\Pr\left(e \in M_j \text{ for } j \in [k] \setminus \{1\}\right)$

$$\begin{aligned}&= 1 - \prod_{j \in [k] \setminus \{1\}} \Pr\left(e \notin M_j\right)\\ &= 1 - \prod_{j \in [k] \setminus \{1\}} \left(1 - \Pr\left(e \in F_j\right) \cdot \frac{c}{k}\right) \qquad \text{(by Eq (1))}\\ &= 1 - \left(1 - \Pr\left(e \in F_1\right) \cdot \frac{c}{k}\right)^{k-1}.\end{aligned}$$

Define $S \subseteq M^*$ as the set of all edge $e \in M^*$ such that $\Pr\left(e \text{ is free in } G^{(i)}\right) \geq \frac{4 \log c}{c}$. By above equation, for any edge $e \in S$, we have that, $\Pr\left(e \in M_j \text{ for } j \in [k] \setminus \{1\}\right)$ is:

$$\geq 1 - \left(1 - \frac{4 \log c}{c} \cdot \frac{c}{k}\right)^{k-1} = 1 - O(1/c).$$

Consequently, $\mathbb{E}\left[w(F_1')\right]$ is

$$\geq \sum_{e \in S} w(e) \cdot \Pr\left(e \in F_1\right) \cdot \Pr\left(e \in M_j \text{ for } j \in [k] \setminus \{1\}\right)$$
$$\geq \sum_{e \in S} w(e) \cdot \Pr\left(e \in F_1\right) \cdot (1 - O(1/c))$$
$$= (1 - O(1/c)) \cdot$$
$$\left(\sum_{e \in M^*} \Pr\left(e \in F_1\right) \cdot w(e) - \sum_{e \in M^* \setminus S} \Pr\left(e \in G_1\right) \cdot w(e)\right)$$
$$\geq (1 - O(1/c)) \cdot \left(\mathbb{E}\left[w(F_i)\right] - \text{opt} \cdot O(\frac{\log c}{c})\right)$$
$$\geq \mathbb{E}\left[w(F_i)\right] - O\left(\frac{\log c}{c}\right) \cdot \text{opt},$$

which finalizes the proof. □

Lemma 3.1 now follows as $\mathbb{E}\left[\text{opt}(H)\right]$ is:

$$\underset{\text{Lemma 3.2}}{\geq} \frac{1}{2} \cdot \mathbb{E}\left[w(F_1') + w(B_1)\right]$$
$$\underset{\text{Lemma 3.3}}{\geq} \frac{1}{2} \cdot \mathbb{E}\left[w(F_1) + w(B_1)\right] - O\left(\frac{\log c}{c}\right) \cdot \text{opt},$$

since $F_1 \cup B_1 = M^*$ and $w(M^*) = \text{opt}$.

3.1. MapReduce Algorithms for Maximum Weight Matching

We now present our MapReduce algorithm based on Theorem 1 using the connection outlined in Section 2.1.

Theorem 2. *There is a MapReduce algorithm that for any $\varepsilon > 0$, outputs a $(2+\varepsilon)$-approximation to maximum weight matching in expectation using $O(\sqrt{\frac{m \cdot \log(1/\varepsilon)}{\varepsilon \cdot n}})$ machines each with $O(\sqrt{mn \cdot (1/\varepsilon) \cdot \log(1/\varepsilon)} + n)$ memory and in only two rounds of parallel computation. The local computation on each machine requires $O((1/\varepsilon) \cdot \log(1/\varepsilon) \cdot \sqrt{mn \cdot (1/\varepsilon) \cdot \log(1/\varepsilon)} + n)$ time. Here, m and n denote the number of edges and vertices, respectively.*

The algorithm in Theorem 2 implies a theoretically efficient (almost) 2-approximation for MWM in the MapReduce model. Implementing this algorithm in practice however can be slightly challenging, simply due to the post-processing step which requires computing an (almost) maximum weight matching (such algorithms tend to be tricky on general graphs, mainly to handle "blossoms" (see, e.g., (Gabow & Tarjan, 1991; Duan & Pettie, 2014)) although one can use *any* readily available algorithm for MWM in this step). It is thus natural to consider simpler post-processing steps also that are easier to implement in practice. An obvious candidate here is the greedy algorithm itself. It follows already from the guarantee of Greedy and Theorem 1 (by the same exact argument as in Theorem 2) that this would lead to an (almost) 4-approximation. We next prove that this algorithm in fact already achieves an improved approximation of 3.

Theorem 3. *The MapReduce algorithm for maximum weight matching obtained by applying* Greedy *as the post-processing step to* GreedyCoreset *always outputs a $(3+\varepsilon)$-approximation in expectation.*

Proof of Theorem 3 appears in the full version.

4. Empirical Study

In this section, we report the results of evaluating our algorithm on a number of publicly available datasets.

Datasets. We use different datasets with varying sizes, ranging from about six million to half a trillion edges. Table 2 provides the statistics. We remark that the number of edges is half the sum of vertex degrees, whereas some previous work (e.g., (Bateni et al., 2017)) report the latter as the number of edges. Three of our datasets (namely Friendster, Orkut, and LiveJournal, all taken from SNAP) are the *public* datasets used in evaluating a hierarchical clustering algorithm in (Bateni et al., 2017) (they also have a fourth *private* dataset). Maximum weight matching is important in the context of hierarchical clustering, because it can be used to generate a more balanced hierarchy: Iteratively find a maximum weight matching and contract the edges of the matching. Then the size of clusters at each level k will be nearly the same and equal to 2^k. We also add a fourth dataset of our own based on *public* DBLP co-authorship (Demaine & Hajiaghayi, 2014): vertices denote authors and edge weights denote the number of papers two researchers co-authored.

Implementation Details. We implement our algorithm on a real proprietary MapReduce-like cluster, i.e., no simulations, with tens to hundreds of machines, depending on the size of the dataset. In our set-up, as in any standard MapReduce system, data is initially stored in distributed storage (each edge on an arbitrary machine), and is then fed into the system and the output matching will be stored in a single file on one machine. To be more precise, our implementation follows the approach in Theorem 3: we first compute a coreset on each machine using GreedyCoreset and then after combining the edges, we simply run the greedy algorithm again to compute the final solution. As such, our algorithm can be seen as the distributed version of the *sequential* greedy algorithm for the weighted matching problem. Our experiments verify that our coreset approach provides significant speed-ups with little quality loss compared to the sequential greedy algorithm, complementing our theoretical results.

Table 2: Statistics about datasets used for empirical evaluation.

Dataset	Number of vertices	Number of edges	Maximum degree
Friendster	65,608,366	546,396,770,507	2,151,462
Orkut	3,072,441	21,343,527,822	893,056
LiveJournal	4,846,609	3,930,691,845	444,522
DBLP	1,482,070	5,946,953	1,961

The two main measures we are interested in are the *quality* of the returned solution and the *speed-up* of the algorithms compared to the sequential greedy algorithm:

- *Quality.* Our coreset-based approach typically secures close to 99.5% of the weight of the solution found by the sequential greedy algorithm. Table 3 shows the quality of the solution obtained on each dataset separately. We emphasize that even though our theoretical proof in Theorem 3 guarantees a 66% performance (for the coreset-based algorithm with greedy as post-processing compared to the sequential greedy algorithm), we never lose more than 1–2% in terms of the solution quality (see full version for a theoretical explanation of this phenomenon).
- *Speed-up.* The speed-up achieved by our coreset-based approach varies significantly between the datasets—ranging from 3x to 130x—compared to the sequential greedy algorithm. The speed-up is larger for bigger graphs, where the coreset computation can really take advantage of the parallelism. Table 3 provides the speed-up obtained on each dataset separately.

Table 3: Speed-up and relative solution quality of our coreset-based algorithm compared to sequential greedy.

Dataset	Speed-up	Weight	Cardinality
Friendster	130x	99.83%	99.74%
Orkut	17x	98.90%	99.42%
LiveJournal	6x	99.86%	99.79%
DBLP	3x	99.55%	99.27%

Several remarks are in order. Firstly, since the graphs in our datasets are too big, we could not compute the value of optimal weighted matching on these graphs. Therefore we only compare the quality of our solution to the quality of the sequential greedy algorithm that does not use a coreset. Additionally, in Table 3, the actual running times and number of machines are withheld due to company's policy so as to not reveal specifics of hardware and architecture in our company's clusters. Instead we focus on speed-up (as did several previous work) which is, unlike absolute running time, mostly independent of computing infrastructure, and thus a better performance measure for the algorithms. Indeed, we expect that any MapReduce-based system to see similar speed-up to what we report. However, we provide the following back-of-the-envelope calculation in order to help the reader estimate the general whereabouts of the running times. In the biggest dataset we consider, namely Friendster, storing the graph in the standard edge format used by the Stanford Network Analysis Project (SNAP) requires 20–30TB. Merely reading this data into memory from a typical 7200 RPM HDD (with maximum read speed of 160MB/s) takes at least 35–50 hours. On this dataset, we obtain about 130x speed-up by building a coreset and running the greedy algorithm on it. Our solution secures 99.83% of the weight of the solution found by sequential greedy, and its cardinality is 99.74% that of the greedy's.

Role of Multiplicity. We also verify empirically that increasing the multiplicity parameter in coreset computation improves the overall quality of the final matching solution. We demonstrate this for two values of k, which is the number of subgraphs we partition the original graph into in the coreset approach (which is also the number of machines). Our results in this part are summarized in Figure 2.

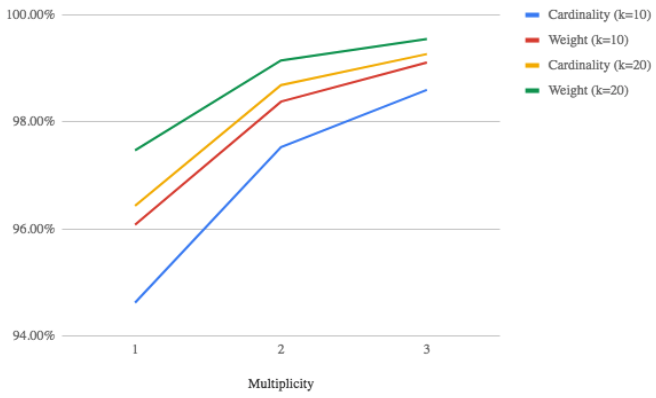


Figure 2: The effect of increasing multiplicity on the quality of the solution. The y-axis numbers are relative to the weight and cardinality of the sequential greedy solution and k denotes the number of subgraphs we partition the original graph into in the coreset approach.

Comparison with Prior Works. An experimental comparison with prior works appear in the full version.

References

Ahn, K. J. and Guha, S. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013.

Ahn, K. J. and Guha, S. 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*, pp. 202–211, 2015.

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

Assadi, S. and Khanna, S. Randomized composable coresets for matching and vertex cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pp. 3–12, 2017.

Assadi, S., Khanna, S., Li, Y., and Yaroslavtsev, G. 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*, pp. 1345–1364, 2016.

Assadi, S., Khanna, S., and Li, Y. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pp. 1723–1742, 2017.

Assadi, S., Bateni, M., Bernstein, A., Mirrokni, V. S., and Stein, C. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 1616–1635, 2019.

Bateni, M., Behnezhad, S., Derakhshan, M., Hajiaghayi, M., Kiveris, R., Lattanzi, S., and Mirrokni, V. S. Affinity clustering: Hierarchical clustering at scale. In *30th Annual Conference on Neural Information Processing Systems*, pp. 6867–6877, 2017.

Beame, P., Koutris, P., and Suciu, D. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pp. 273–284, 2013.

Behnezhad, S. and Reyhani, N. Almost optimal stochastic weighted matching with few queries. *CoRR*, abs/1710.10592. To appear in EC 2018, 2017.

Behnezhad, S., Derakhshan, M., Esfandiari, H., Tan, E., and Yami, H. Brief announcement: Graph matching in massive datasets. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pp. 133–136, 2017.

Berger, B., Singh, R., and Xu, J. Graph algorithms for biological systems analysis. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, pp. 142–151, 2008.

Blum, A., Dickerson, J. P., Haghtalab, N., Procaccia, A. D., Sandholm, T., and Sharma, A. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC '15, Portland, OR, USA, June 15-19, 2015*, pp. 325–342, 2015.

Charles, D. X., Chickering, M., Devanur, N. R., Jain, K., and Sanghi, M. Fast algorithms for finding matchings in lopsided bipartite graphs with applications to display ads. In *Proceedings 11th ACM Conference on Electronic Commerce (EC-2010), Cambridge, Massachusetts, USA, June 7-11, 2010*, pp. 121–128, 2010.

Crouch, M. and Stubbs, D. S. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pp. 96–104, 2014. doi: 10.4230/LIPIcs.APPROX-RANDOM.2014.96.

Cygan, M., Gabow, H. N., and Sankowski, P. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter and matchings. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pp. 531–540, 2012.

Czumaj, A., Lacki, J., Madry, A., Mitrovic, S., Onak, K., and Sankowski, P. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pp. 471–484, 2018.

da Ponte Barbosa, R., Ene, A., Nguyen, H. L., and Ward, J. 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*, pp. 1236–1244, 2015.

da Ponte Barbosa, R., Ene, A., Nguyen, H. L., and Ward, J. A new framework for distributed submodular maximization. In *IEEE 57th Annual Symposium on Foundations*

of Computer Science, FOCS 2016, New Brunswick, New Jersey, USA, pp. 645–654, 2016.

Demaine, E. and Hajiaghayi, M., 2014. URL http://projects.csail.mit.edu/dnd/DBLP/.

Dickerson, J. P., Procaccia, A. D., and Sandholm, T. Optimizing kidney exchange with transplant chains: theory and reality. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pp. 711–718, 2012.

Duan, R. and Pettie, S. Approximating maximum weight matching in near-linear time. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pp. 673–682, 2010.

Duan, R. and Pettie, S. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.

Duan, R., Pettie, S., and Su, H. Scaling algorithms for weighted matching in general graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pp. 781–800, 2017.

Duff, I. S. and Koster, J. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis Applications*, 20(4):889–901, 1999.

Duff, I. S. and Koster, J. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis Applications*, 22(4):973–996, 2001.

Edmonds, J. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965a.

Edmonds, J. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965b.

Epstein, L., Levin, A., Mestre, J., and Segev, D. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM J. Discrete Math.*, 25(3): 1251–1265, 2011.

Gabow, H. N. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23 (2):221–234, 1976.

Gabow, H. N. A scaling algorithm for weighted matching on general graphs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pp. 90–100, 1985.

Gabow, H. N. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California.*, pp. 434–443, 1990.

Gabow, H. N. and Tarjan, R. E. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4): 815–853, 1991.

Gabow, H. N., Galil, Z., and Spencer, T. H. Efficient implementation of graph algorithms using contraction. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pp. 347–357, 1984.

Gamlath, B., Kale, S., Mitrovic, S., and Svensson, O. Weighted matchings via unweighted augmentations. *CoRR*, abs/1811.02760. To appear in PODC 2019., 2018.

Gamlath, B., Kapralov, M., Maggiori, A., Svensson, O., and Wajc, D. Online matching with general arrivals. *CoRR*, abs/1904.08255, 2019.

Ghaffari, M., Gouleakis, T., Konrad, C., Mitrovic, S., and Rubinfeld, R. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pp. 129–138, 2018.

Goel, A., Kapralov, M., and Khanna, S. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pp. 468–485, 2012.

Goodrich, M. T., Sitchinava, N., and Zhang, Q. Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, pp. 374–383, 2011.

Harvey, N. J. A., Liaw, C., and Liu, P. Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pp. 43–52, 2018.

Huang, Z., Radunovic, B., Vojnovic, M., and Zhang, Q. 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*, pp. 460–473, 2015.

Jebara, T., Wang, J., and Chang, S. Graph construction and *b*-matching for semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pp. 441–448, 2009.

Kapralov, M. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pp. 1679–1697, 2013.

Karloff, H. J., Suri, S., and Vassilvitskii, S. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pp. 938–948, 2010.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

Konrad, C., Magniez, F., and Mathieu, C. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pp. 231–242, 2012.

Kumar, R., Moseley, B., Vassilvitskii, S., and Vattani, A. Fast greedy algorithms in mapreduce and streaming. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pp. 1–10, 2013.

Langmead, C. J. and Donald, B. R. High-throughput 3D structural homology detection via NMR resonance assignment. In *3rd International IEEE Computer Society Computational Systems Bioinformatics Conference, CSB 2004*, pp. 278–289, 2004.

Lattanzi, S., Moseley, B., Suri, S., and Vassilvitskii, S. 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)*, pp. 85–94, 2011.

Liu, P. and Vondrák, J. Submodular optimization in the mapreduce model. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pp. 18:1–18:10, 2019.

Manshadi, F. M., Awerbuch, B., Gemulla, R., Khandekar, R., Mestre, J., and Sozio, M. A distributed algorithm for large-scale generalized matching. *PVLDB*, 6(9):613–624, 2013.

McGregor, A. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, pp. 170–181, 2005.

Mehta, A., Saberi, A., Vazirani, U. V., and Vazirani, V. V. Adwords and generalized online matching. *J. ACM*, 54 (5):22, 2007.

Mirrokni, V. S. and Zadimoghaddam, M. 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*, pp. 153–162, 2015.

Paz, A. and Schwartzman, G. A $(2 + \varepsilon)$-approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pp. 2153–2161, 2017.

Penn, M. and Tennenholtz, M. Constrained multi-object auctions and b-matching. *Inf. Process. Lett.*, 75(1-2): 29–34, 2000.

Pettie, S. and Sanders, P. A simpler linear time 2/3-epsilon approximation for maximum weight matching. *Inf. Process. Lett.*, 91(6):271–276, 2004.

Pinar, A., Chow, E., and Pothen, A. Combinatorial algorithms for computing column space bases that have sparse inverses. *Electronic Transactions on Numerical Analysis*, 22:122–145, 2005.

Pothen, A. and Fan, C. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Softw.*, 16(4): 303–324, 1990.

Preis, R. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, pp. 259–269, 1999.

Vinkemeier, D. E. D. and Hougardy, S. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Trans. Algorithms*, 1(1):107–122, 2005.