Greedy and Local Ratio Algorithms in the MapReduce Model

Nicholas J. A. Harvey
University of British Columbia
Vancouver, Canada
nickhar@cs.ubc.ca

Christopher Liaw
University of British Columbia
Vancouver, Canada
cvliaw@cs.ubc.ca

Paul Liu
Stanford University
Stanford, California, USA
paulliu@stanford.edu

Abstract

MapReduce has become the *de facto* standard model for designing distributed algorithms to process big data on a cluster. There has been considerable research on designing efficient MapReduce algorithms for clustering, graph optimization, and submodular optimization problems. We develop new techniques for designing greedy and local ratio algorithms in this setting. Our randomized local ratio technique gives 2-approximations for weighted vertex cover and weighted matching, and an f-approximation for weighted set cover, all in a constant number of MapReduce rounds. Our randomized greedy technique gives algorithms for maximal independent set, maximal clique, and a $(1+\varepsilon)\ln\Delta$-approximation for weighted set cover. We also give greedy algorithms for vertex colouring with $(1+o(1))\Delta$ colours and edge colouring with $(1+o(1))\Delta$ colours.

CCS Concepts

• **Theory of computation → Packing and covering problems**; **MapReduce algorithms**; *Distributed computing models*; *Algorithm design techniques*;

Keywords

local ratio, vertex cover, weighted matching, mapreduce, graph colouring

1 Introduction

A wealth of algorithmic challenges arise in processing the large data sets common in social networks, machine learning, and data mining. The tasks to be performed on these data sets commonly involve graph optimization, clustering, selecting representative subsets, etc. — tasks whose theory is well-understood in a sequential model of computation.

The modern engineering reality is that these large data sets are distributed across clusters or data centers, for reasons including bandwidth and memory limitations, and fault tolerance. New programming models and infrastructure, such as MapReduce and Spark, have been developed to process this data efficiently. The MapReduce model is attractive to theoreticians as it is clean, simple, and has rigorous foundations in the work of Valiant [41] and Karloff et al. [25]. Designing MapReduce algorithms often involves concepts arising in parallel, distributed, and streaming algorithms, so it is necessary to understand classic optimization problems in a new light.

In the formalization of the MapReduce model [25], the data for a given problem is partitioned across many machines, where each machine has memory sublinear in the input size. Computation proceeds in a sequence of rounds: in each round, a machine performs a polynomial-time computation on its local data. Between rounds, each machine simultaneously sends data to all other machines, the size of which is restricted only by the sender's and recipients' memory capacity. The primary efficiency consideration in this model is the number of rounds, although it is also desirable to constrain other metrics, such as the memory overhead, the amount of data communicated, and the processing time.

Greedy approximation algorithms have been a popular choice for adapting to the MapReduce model, in the hopes that their simple structure suits the restrictions of the model. Unfortunately, many greedy algorithms also seem to be inherently sequential, a property which is rather incompatible with the parallel nature of MapReduce computations. This phenomenon is, of course, well known to parallel and distributed algorithm researchers, as it was the impetus for Valiant's [40] and Cook's [13] encouragement to study the maximal independent set problem in a parallel setting. Primal-dual approximation algorithms have received comparably less use in the MapReduce setting, perhaps because they seem even more sequential than greedy algorithms.

1.1 Techniques and Contributions

We develop two new techniques for designing MapReduce algorithms. The first is a "randomized local ratio" technique, which combines the local ratio method [6] with judicious random sampling in order to enable parallelization. At a high level, the technique choses an i.i.d. random sample of elements, then runs an ordinary local ratio algorithm on this sample. The crux is to show that the weight adjustments performed by the local ratio algorithm cause a significant fraction of the non-sampled elements to be eliminated. Repeating this several times allows us to execute the local ratio method with no loss in the approximation ratio. We use this technique to give a 2-approximation for min-weight vertex cover (Section 2) and a 2-approximation for max-weight matching (Section 5). The vertex cover result is a special case of our f-approximation for min-weight set cover, where f is the largest frequency of any element. Additional subtleties arise in applying this technique to the max-weight b-matching problem ($b \geq 2$), for which we obtain a $(3-\frac{2}{b}+\varepsilon)$-approximation (Appendix D). Adapting these randomized local ratio algorithms to the MapReduce setting is straightforward: all machines participate in the random sampling, and a single central machine executes the local ratio algorithm. It is worth emphasizing that these algorithms are very simple and could easily be implemented in practice.

Our second technique we call the "hungry-greedy" technique. Whereas randomized local ratio uses i.i.d. sampling, hungry-greedy first samples "heavy" elements, not to maximize an objective function, but to disqualify a large fraction of elements from the solution. Doing so allows us to rapidly shrink the problem size, and thereby execute the greedy method in just a few rounds. We emphasize that the hungry-greedy technique is applicable even to problems without an objective function or an *a priori* ordering on the greedy choices. We use this technique to give efficient algorithms for maximal independent set (Section 3), maximal clique (Appendix B), and a $(1+\varepsilon)\ln\Delta$ approximation to weighted set cover (Section 4).

At first glance one may think that the maximal clique result follows by a trivial reduction from the independent set result, but this is not the case. The formalization of the MapReduce model requires that computations be performed in linear space, so it is not possible to complement a sparse graph. Other problems that are widely studied in the distributed computing literature and can usually be solved by a trivial reduction to maximal independent set include vertex colouring with $\Delta+1$ colours and edge colouring with $2\Delta-1$ colours; see the monograph of Barenboim and Elkin [8]. Again, these reductions are not possible in the MapReduce model due to space restrictions. We make progress on these problems by giving algorithms for $(1+o(1))\Delta$ vertex colouring and $(1+o(1))\Delta$ edge colouring (Section 6).

1.2 Related work

Within the theory community, part of the appeal of the MapReduce model is the connection to several established topics, including distributed algorithms, PRAM algorithms, streaming algorithms, submodular optimization and composable coresets.

One of the earliest papers on MapReduce algorithms is due to Lattanzi et al. [27], in which the filtering technique is introduced. Filtering involves choosing a random sample, from which a partial solution is constructed and used to eliminate candidate elements from the final solution; the process is repeated until all elements are exhausted. This technique has been quite influential, and many subsequent papers can be viewed as employing a similar methodology. Indeed, our randomized local ratio technique can be viewed as a descendant of the filtering technique in which the local ratio weights are used to eliminate elements. The original filtering technique yielded 2-approximations for unweighted matching and vertex cover, and was combined with layering ideas to give an 8-approximation for weighted matching [27]. These layering ideas were improved by Crouch and Stubbs [14] and Grigorescu et al. [21] to give a $(3.5+\varepsilon)$-approximation for weighted matching in the semi-streaming and MapReduce models. In addition, it is also known that any c-approximation to maximum weighted matching can be automatically converted to a $(2+\varepsilon)$-approximation algorithm by using the former as a blockbox [30].

Recently Paz and Schwartzman [37] gave a $(2+\varepsilon)$-approximation for weighted matching in the semi-streaming model, using the local ratio method with clever pruning techniques to bound the recursion depth. This result was slightly improved and substantially simplified by Ghaffari [18]. Unfortunately, this work does not apply to the MapReduce setting: it is very space-efficient, but not distributed. Our randomized local ratio technique is inspired by this work, but different. Whereas [18, 37] are deterministic and process the input in a streaming fashion, our technique achieves its space efficiency through random sampling and requires several rounds to process the entire input.

Submodular optimization has also been a fruitful avenue for MapReduce algorithms. Kumar et al. [26] gave a general framework for maximizing monotone submodular functions over hereditary constraints in the MapReduce setting while losing only ε in the approximation ratio. Their Sample-and-Prune and ε-Greedy techniques are vaguely related to our hungry-greedy technique, but nevertheless different as our technique does not maximize an objective. Barbosa et al. [16] develop a different framework that achieves the same approximation ratio while using fewer MapReduce rounds; it also supports the continuous greedy algorithm and non-monotone functions. A specific result relevant to our work is a $\frac{1}{3+\varepsilon}$-approximation for weighted matching in bipartite graphs [26] in $O(\log(\frac{w_{\max}}{w_{\min}})/\mu\varepsilon)$ rounds and $O(nm^{\mu})$ space per machine. The results of Barbosa et al. seem to improve this to $O(1/\varepsilon)$ rounds and $O(\sqrt{nm})$ space per machine.

Another problem related to submodular maximization is the submodular set cover problem [43], which generalizes the weighted set cover problem, and has been studied* in the MapReduce model by Mirzasoleiman et al. [34, 35]. Both of our techniques are applicable to weighted set cover. Our randomized local ratio technique matches the classic f-approximation [7], where f is the largest frequency of an element. Our hungry-greedy technique gives a $(1+\varepsilon)\ln\Delta$ approximation, nearly matching the optimal $\ln\Delta$-approximation for polynomial-time algorithms [12]. It is worth noting that our f-approximation is intended (as with vertex cover) for the scenario that $n \ll m$, whereas our second algorithm assumes $m \ll n$. Our latter result has several advantages over the work of Mirzasoleiman et al.: we handle the weighted case, improve the $\ln m$ in the approximation ratio to $\ln\Delta$, our space usage is controllable by an arbitrary parameter μ, and use significantly less space when $m \ll n$.

The classical PRAM model admits algorithms for many of the problems we consider. There are general simulations of EREW PRAMs [25] and CREW PRAMs [20] by MapReduce algorithms, although these lead to polylog(n)-rounds algorithms, which does not meet the MapReduce gold standard of $O(1)$ rounds.

Composable core-sets have recently emerged as a useful primitive for the design of distributed approximation algorithms; see, e.g., [5, 23, 33] and the references therein. The idea is to partition the data, compute on each part a representative subset called a core-set, then solve the problem on the union of the core-sets. Relevant to this is the work of Assadi et al. [3] (extending work by Assadi and Khanna [4]), which designs $3/2+\varepsilon$ (resp. $3+\varepsilon$) approximate core-sets for unweighted matching (resp. vertex cover). They obtain 2-round MapReduce algorithms as a corollary. Combining with the technique of Crouch and Stubbs [14], these can be extended to $O(1)$ (resp. $O(\log n)$) approximations for weighted variants of the problem.

Primal-dual algorithms have previously been studied in the MapReduce model, notably in the work of Ahn and Guha [1]. They develop sophisticated multiplicative-weight-type techniques to approximately solve the matching (and b-matching) LP in very few iterations. Their technique gives a $(1+\varepsilon)$-approximation in $O(1/\mu\varepsilon)$

* The set cover problem was studied earlier in the MapReduce setting by Stergiou and Tsioutsiouliklis [39], although their work appears not to have rigorous guarantees regarding concurrent access to shared state.

Figure 1: Results for MapReduce algorithms. For graph algorithms, the number of vertices is n, the number of edges is assumed to be n^{1+c}, and the maximum degree is Δ. For set cover, the number of sets is n, the size of the ground set is m, the size of the largest set is Δ, and the largest number of sets containing any element is f, the maximum (resp. minimum) weight is $w_{\max}$ (resp. $w_{\min}$). For Theorem 2.4 it is assumed that $m = n^{1+c}$. The space per machine is typically $n^{1+\mu}$, which for most results (except [1]) need not be constant; taking $\mu = 1/\log n$ one can often get $O(n)$ space and $O(\log n)$ rounds. For Theorem 4.6, n can depend arbitrarily on m.

Problem	Weighted?	Approximation	MapReduce Rounds	Space per machine	Reference
Vertex Cover		2	$O(c/\mu)$	$O(n^{1+\mu})$	[26]
	Y	$O(\log^2 n)$	2	$\tilde{O}(n^{1.5})$	[4]
	Y	2	$O(c/\mu)$	$O(n^{1+\mu})$	Theorem 2.4
Set Cover		$O(\log m)$	$O(\log(m)\|OPT\|)$	not analyzed	[34]
		$(1+\varepsilon)\ln m$	$O\big(\log(\frac{n}{\|OPT\|})\log(\Delta)/\varepsilon + \log m)$	$O(\sqrt{n\|OPT\|}\Delta)$	[35]
	Y	f	$O((c/\mu)^2)$	$O(f \cdot n^{1+\mu})$	Theorem 2.4
	Y	$(1+\varepsilon)\ln\Delta$	$O\left(\frac{\log\left(\frac{w_{\max}}{w_{\min}}\Delta\right)}{\mu^2\varepsilon}\right)$ if $n = \text{poly}(m)$	$O(m^{1+\mu})$	Theorem 4.6
Maximal Indep. Set			$O(c/\mu)$	$O(n^{1+\mu})$	Theorem A.3
Maximal Clique			$O(1/\mu)$	$O(n^{1+\mu})$	Corollary B.1
Matching		2	$O(c/\mu)$	$O(n^{1+\mu})$	[26]
	Y	8	$O(c/\mu)$	$O(n^{1+\mu})$	[26]
	Y	$4+\varepsilon$	$O(c/\mu)$	$O(n^{1+\mu})$	[14]
	Y	$3.5+\varepsilon$	$O(c/\mu)$	$O(n^{1+\mu})$	[21]
	Y	$2+\varepsilon$	$O((c/\mu)\log(1/\varepsilon))$	$O(n^{1+\mu})$	[30]
	Y	$1+\varepsilon$	$O(1/\mu\varepsilon)$	$O(n^{1+\mu})$	[1]
	Y	$O(1)$	2	$\tilde{O}(n^{1.5})$	[4]
		$2+\varepsilon$	$O((\log\log n)^2\log(1/\varepsilon))$	$O(n)$	[15]
		$1+\varepsilon$	$(1/\varepsilon)^{O(1/\varepsilon)}\log\log n$	$\tilde{O}(n)$	[3]
	Y	2	$O(c/\mu)$	$O(n^{1+\mu})$	Theorem 5.6
Vertex Colouring		$(1+o(1))\Delta$ colours	$O(1)$	$O(n^{1+\mu})$	Theorem 6.4
Edge Colouring		$(1+o(1))\Delta$ colours	$O(1)$	$O(n^{1+\mu})$	Theorem 6.6

MapReduce rounds while using only $O(n^{1+\mu})$ space per machine. While their algorithm achieves a superior approximation factor to ours, it is also highly technical. Our algorithm is concise, simple, and could plausibly be the end-of-the-road for filtering-type techniques for the matching problem.

Very recently there have been some exciting developments in MapReduce algorithms. Im et al. [22] have given a general dynamic programming framework that seems very promising for designing more approximation algorithms. Czumaj et al. [15] introduced a round compression technique for MapReduce and applied it to matching to obtain a linear space MapReduce algorithm which gives a $(2+\varepsilon)$-approximation algorithm for unweighted matching in only $O((\log\log n)^2\log(1/\varepsilon))$ rounds. For unweighted matching, Assadi et al. [3] built on this work and gave a $(2+\varepsilon)$-approximation algorithm in $O((\log\log n)\log(1/\varepsilon))$ rounds with $\tilde{O}(n)$ space and a $(1+\varepsilon)$-approximation algorithm in $(1/\varepsilon)^{O(1/\varepsilon)}\log\log n$ rounds with $\tilde{O}(n)$ space. Round compression has also been used to give a $O(\log n)$ approximation for unweighted vertex cover using $O(\log\log n)$ MapReduce rounds [2].

1.3 The MapReduce Model

In this paper we adopt the MapReduce model as formalized by Karloff et al. [25]; see also [19]. In their setting, there is an input of size N distributed across $O(N^{\delta})$ machines, each with $O(N^{1-\delta})$ memory. This models the property that modern big data sets must be stored across a cluster of machines. Computation proceeds in a sequence of rounds. In each round, each machine receives input of size $O(N^{1-\delta})$; it performs a computation on that input; then it produces a sequence of output messages, each of which is to be delivered to another machine at the start of the next round. The total size of these output messages must also be $O(N^{1-\delta})$. In between rounds, the output messages are delivered to their recipients as input.

The data format and computation performed is not completely arbitrary. Data is stored as a sequence of key-value pairs, and the computation is performed by the eponymous *map* and *reduce* functions that operate on such sequences. The requirements of these functions are not particularly relevant to our work, so we refer the interested reader to standard textbooks [29] or the work of Karloff et al. [25].

For graph algorithms, we will assume that the space per machine is bounded as a function of n, the number of vertices, whereas the input size is $O(N)$ where $N = m$ is the number of edges. As in Lattanzi et al. [27], we will typically assume that the space per machine is $O(n^{1+\mu})$ and the graph has $m = n^{1+c}$ edges, $c > \mu$. This conforms to the requirements of Karloff et al. with $\delta = \frac{c-\mu}{1+c}$.

The memory constraints of graph problems in the MapReduce model stem from the early work of Karloff et al. [25] and Leskovec et al. [28]. In these papers, graph problems with n nodes and n^{1+c} edges were examined. Specifically, Leskovec et al. [28] found that real world graph problems were often not sparse, but had n^{1+c} edges where c varied between 0.08 and larger than 0.5. In practice, MapReduce algorithms also require $\Omega(n)$ memory per machine. One such example is the diameter estimation algorithm of Kang et al. [24], in which $O((n+m)/M)$ memory is required per machine, where M is the number of machines in total.

Related models. There have been a series of refinements to the original MRC model of Karloff et al. [25], the most prominent of which is the massive parallel computation (MPC) model developed by Beame et al. [9]. The main difference with the MPC model is that the space requirements are more stringent than in MRC. Given an input of size N and M machines, each machine is only allowed to have at most $S = O(N/M)$ space. In each round, each machine can send $O(S)$ words to other machines. The majority of our algorithms apply in the MPC model as well as the MRC model. The only exceptions are our set cover and b-matching algorithms, where our space bound depends on structural parameters of the input.

1.4 Organization

In Sections 2 and 3, we develop our "randomized local ratio" and "hungry-greedy" techniques through the weighted set cover and maximal independent set problems respectively. We then apply these techniques to specific problems in the subsequent sections. Our randomized local ratio technique is applied to maximum weight matching (Section 5) and b-matching (Appendix D), and our hungry-greedy technique is applied to maximal clique (Appendix B) as well as an alternate approximation of weighted set cover (Section 4). Algorithms for vertex and edge colouring that may be of independent interest are given in Section 6.

2 f-approximation for weighted set cover

For notational convenience, let $[n] = \{1, \ldots, n\}$, $S(X) = \bigcup_{i \in X} S_i$ and $w(X) = \sum_{i \in X} w_i$ for all $X \subseteq [n]$. In the weighted set cover problem, we are given n sets $S_1, \ldots, S_n \subseteq [m]$ with weights $w_1, \ldots, w_n \in \mathbb{R}_{>0}$. The goal is to find a subset X such that $w(X)$ is minimal amongst all X with $S(X) = [m]$.

Let OPT denote a fixed set X achieving the minimum. The frequency of element $j \in U$ is defined to be $|\{\, i : j \in S_i \,\}|$. Let f denote the maximum frequency of any element.

Theorem 2.1 (Bar-Yehuda and Even [7]). The following algorithm gives an f-approximation to weighted set cover.

Sequential local ratio algorithm for minimum weight set cover

Arbitrarily select an element $j \in U$ such that the minimum weight of all sets containing j is strictly positive, then reduce all those weights by that minimum value. Remove all sets with zero weight, and add them to the cover. Repeat so long as uncovered elements remain.

More explicitly, the weight reduction step works as follows. If element $j \in U$ was selected, then we compute $\epsilon = \min_{i \in [n] : j \in S_i} w_i$ then perform the update $w_i \longleftarrow w_i - \epsilon$ for all i with $j \in S_i$.

2.1 Randomized local ratio

The local ratio method described above is not terribly parallelizable, as it processes elements sequentially. Nor is it terribly space-efficient, as it might require processing every element in $[m]$. However, it does have the virtue that elements can be processed in a fairly arbitrary order. Our randomized local ratio technique takes advantage of that flexibility and combines the local ratio algorithm with random sampling. There are essentially two components to our randomized local ratio technique: (1) we can cover a large fraction of the remaining sets with just a random sample of elements, and (2) this random sample will be comparable in weight to the optimal solution since the processing order of local ratio is fairly arbitrary. Intuitively, property (1) ensures that the algorithm runs in a few number of rounds, and property (2) ensures a good approximation ratio. These two ideas will come up frequently in the algorithms we present in this paper.

In Algorithm 1, the size of each sample U' is $O(\eta)$ w.h.p., and η will be taken to be $n^{1+\mu}$, the space available on each machine.

Algorithm 1 An f-approximation for minimum weight set cover. The lines highlighted in blue are run sequentially on a central machine, and all other lines are run in parallel across all machines.

```
1: procedure ApproxSC($S_1, \ldots, S_n \subseteq U = [m]$)
2:     $U_1 \longleftarrow U$, $r \longleftarrow 1$
3:     while $U_r \neq \emptyset$ do
4:         $\triangleright$: $U_r$ is the set of all $j \in [m]$ such that $\min_{i : j \in S_i} w_i > 0$
5:         Compute $U'$ by sampling each element of $U_r$ independently with probability $p = \min(1, \frac{2\eta}{|U_r|})$
6:         if $|U'| > 6\eta$ then fail
7:         Run the local ratio algorithm for set cover on the sets $S_1 \cap U', \ldots, S_n \cap U'$
8:         Let $C \subseteq [n]$ be the indices of all sets with zero weight
9:         $U_{r+1} \longleftarrow U_r \setminus S(C)$
10:        $r \longleftarrow r + 1$
11:    return the indices $C$ of all sets with zero weight
```

Our analysis uses the following useful fact, relating to the filtering technique of Lattanzi et al. [27].

Lemma 2.2. Let U_{r+1} be the set computed on line Line 9 of Algorithm 1. If $p = 1$ then $U_{r+1} = \emptyset$, otherwise $|U_{r+1}| < 2n/p$ with probability at least $1 - e^{-n}$.

Proof. If $p = 1$ then $U' = U_r$, so the local ratio method produces C covering all of U_r, and so $U_{r+1} = \emptyset$. So assume $p < 1$.

For each $X \subseteq [n]$, let $\overline{S(X)}$ be the elements left uncovered by X. Let $\mathcal{E}_X$ be the event that $U' \cap \overline{S(X)} = \emptyset$. Say that X is large if $|\overline{S(X)}| \geq 2n/p =: \tau$. For large X, $\Pr[\mathcal{E}_X] \leq (1-p)^{\tau} \leq e^{-p\tau} = e^{-2n}$. By a union bound, the probability that all large $\mathcal{E}_X$ fail to occur is at least $1 - e^{-n}$. Since the local ratio method produces a set C for which $U' \cap \overline{S(C)} = \emptyset$, with high probability it must be that $|C|$ is not large. □

Theorem 2.3. Suppose that $m \leq n^{1+c}$. Let $\eta = n^{1+\mu}$, for any $\mu > 0$. Then Algorithm 1 terminates within $\lceil c/\mu \rceil$ iterations and returns an f-approximate set cover, w.h.p.

Proof. *Correctness:* Observe that the sequential local ratio algorithm for set cover can pick elements in an arbitrary order. Algorithm 1 is an instantiation of that algorithm that uses random sampling to partially determine the order in which elements are picked. It immediately follows that Algorithm 1 outputs an f-approximation to the minimum set cover, assuming it does not fail. By a standard Chernoff bound, an iteration fails with probability at most $\exp(-n^{1+\mu}) \le \exp(-n)$.

Efficiency: By Lemma 2.2, while $p < 1$ we have $|U_{r+1}| \le 2n/p = 2n|U_r|/\eta = 2|U_r|/n^{\mu} \le 2n^{1+c-r\mu}$, whp. By iteration $r = \lceil c/\mu \rceil - 1$, we will have $|U_r| \le 2n^{1+c} = 2\eta$ and so $p = 1$. After one more iteration the algorithm will terminate, again by Lemma 2.2. □

2.2 MapReduce implementation

Theorem 2.4. There is a MapReduce algorithm that computes an f-approximation for the minimum weight set cover where each machine has memory $O(f \cdot n^{1+\mu})$. The number of rounds is $O(c/\mu)$ in the case $f = 2$ (i.e., vertex cover) and $O((c/\mu)^2)$ in the case $f > 2$.

Proof. Instead of representing the input as the sets $S_1, \dots, S_n$, we instead assume that it is represented as the "dual" set system $T_1, \dots, T_m$ where $T_j = \{ i : j \in S_i \}$. By definition of frequency, we have $|T_j| \le f$.

Each element $j \in [m]$ will be assigned arbitrarily to one of the machines, with $n^{1+\mu}$ elements per machine, so $M := m/n^{1+\mu} = n^{c-\mu}$ machines are required. Element j will store on its machine the set T_j and a bit indicating if $j \in U_r$. Thus, the space per machine is $O(f \cdot n^{1+\mu})$.

The main centralized step is the local ratio computation (lines 7-8). The centralized machine receives as input the sets T_j for all $j \in U'$. As $|U'| \le 6\eta$, the input received by the central machine is $O(f\eta) = O(fn^{1+\mu})$ words. After executing the local ratio algorithm, the centralized machine then computes C.

There are two key steps requiring parallel computation: the sampling step (line 5) and the computation of U_{r+1} (line 9). After the central machine computes C, it must send C to all other machines. Sending this directly could require $|C| \cdot M = \Omega(n^{1+c-\mu})$ bits of output, which could exceed the machine's space of $O(n^{1+\mu})$. Instead, we can form a broadcast tree over all machines with degree n^{μ} and depth c/μ, allowing us to send C to all machines in $O(c/\mu)$ MapReduce rounds. Since each machine knows C, each element j can determine if $j \in U_{r+1}$ by determining if $T_j \cap C \ne \emptyset$. The same broadcast tree can be used to compute $|U_{r+1}|$ and send that to all machines. Since each element j knows if $j \in U_r$ and knows $|U_r|$, it can independently perform the sampling step.

In the case $f = 2$ (i.e., vertex cover), this can be improved. Each set S_i (i.e., vertex) will also be assigned to one of the M machines, randomly chosen. By a Chernoff bound (using that $|S_i| \le n$) the space required per machine is $O(f \cdot n^{1+\mu})$ w.h.p. After computing C, the central machine sends a bit to each set S_i indicating whether $i \in C$ or not. Each set S_i then forwards that bit to each element $j \in S_i$. Thus j can determine whether $j \in U_{r+1}$. Each machine can send to a central location the number of edges on that machine that lie in U_{r+1}, so the central machine can compute $|U_{r+1}|$ and send it back to all machines. □

3 Maximal independent set

As a warm-up to the "hungry-greedy" technique, we first present an algorithm to find a maximal independent set (MIS) in a constant number of MapReduce rounds. In order to clearly explain the concepts, this section presents a simple algorithm using $O(1/\mu^2)$ rounds. In Appendix A, we show how to further parallelize this algorithm so that it takes $O(c/\mu)$ rounds.

The algorithm, shown in Algorithm 2, proceeds in phases where each phase takes $O(1/\mu)$ rounds. In phase $i \ge 1$, we reduce the maximum degree from $n^{1-(i-1)\alpha}$ to $n^{1-i\alpha}$. We will choose $\alpha = \mu/2$ so that after $O(1/\mu)$ phases, the maximum degree will be at most n^{μ}. Following this, we can finish the algorithm in one more round by placing the entire graph onto a central machine.

For a vertex v, we define $N(v)$ as the neighbours of v and $N^+(v) = N(v) \cup \{v\}$. For a set $I \subseteq V$, we define $N^+(I) = \cup_{v \in I} N^+(v)$ and $N_I(v) = N(v) \setminus N^+(I)$. In other words, $N_I(v)$ is the set of neighbours of v which are not in I and not adjacent to a vertex in I. Finally let $d_I(v) = |N_I(v)|$ if $v \notin N^+(I)$, otherwise $d_I(v) = 0$.

Algorithm 2 A simple algorithm for maximal independent set. The lines highlighted in blue are run sequentially on a central machine, and all other lines are run in parallel across all machines.

```
1:  procedure MIS1($G = (V, E)$)
2:      $I \leftarrow \emptyset$                                ▷ $I$ is the current independent set
3:      for $i = 1, \dots, 1/\alpha$ do
4:          ▷: $d_I(v) \le n^{1-(i-1)\alpha}$ for all $v \in V$
5:          $V_H \leftarrow \{v : d_I(v) \ge n^{1-i\alpha}\}$                ▷ Heavy vertices
6:          while $|V_H| \ge n^{i\alpha}$ do
7:              Draw $n^{i\alpha}$ groups of $n^{\mu/2}$ vertices from $V_H$, say $X_1, \dots, X_{n^{i\alpha}}$
8:              for $j = 1, \dots, n^{i\alpha}$ do
9:                  if $\exists v_j \in X_j$ such that $d_I(v_j) \ge n^{1-i\alpha}$ then
10:                     $I \leftarrow I \cup \{v_j\}$
11:             $V_H \leftarrow \{v : d_I(v) \ge n^{1-i\alpha}\}$              ▷ Update heavy vertices
12:         Find maximal independent set in $V_H$ and add it to $I$   ▷ $|V_H| < n^{i\alpha}$
13:     return $I$
```

Remark 3.1. In the algorithm, a vertex can lose its label as a heavy vertex during the for loop. This is intentional as when we add a vertex to I, we want to make sure we make substantial progress. Moreover, since a vertex sends all its neighbours, it is easy to tell whether a vertex is heavy and to update $N^+(I)$.

Lemma 3.2. Let V_H be the set of heavy vertices at the beginning of the inner for loop (line 9) and V'_H be the set of heavy vertices after the for loop. Then $|V'_H| \le |V_H|/n^{\mu/4}$ w.h.p.

Proof. Observe that the first group X_1 of vertices will definitely contain a heavy vertex so it will remove at least $n^{1-i\alpha}$ vertices from the graph.[*] Now suppose we are currently processing group X_j. If the number of heavy vertices is at most $|V_H|/n^{\mu/4}$ at this point then we are done. So suppose the number of heavy vertices is at least $|V_H|/n^{\mu/4}$. Since we sample the vertices uniformly at random, the group X_j contains a heavy vertex with probability at least $1 - \left(1 - 1/n^{\mu/4}\right)^{n^{\mu/2}} \ge 1 - \exp\left(-n^{\mu/4}\right)$. At this point, we add another heavy vertex to I and remove it and all its neighbours.

[*]We will say v is removed from the graph if it is added to $N^+(I)$.

The above process can happen at most $n^{i\alpha}$ times before the number of heavy vertices is at most $|V_H|/n^{\mu/4}$. This is because if it happens $n^{i\alpha}$ times then no vertices remain. Hence, by taking a union bound over all groups, we have that $|V_H'| \leq |V_H|/n^{\mu/4}$ with probability at least $1 - n \cdot \exp\left(-n^{\mu/4}\right)$. □

Theorem 3.3. There is a MapReduce algorithm to find a maximal independent set in $O(1/\mu^2)$ rounds and $O(n^{1+\mu})$ space per machine, w.h.p.

Proof sketch. As before, there are $M := n^{c-\mu}$ machines. Each vertex and its adjacency list is assigned to one of the M machines, randomly chosen. By a Chernoff bound, the space required per machine is $O(n^{1+\mu})$ whp. Each vertex v will maintain its value of $d_I(v)$, allowing it to determine if $v \in V_H$. Thus, the sampling step (line 7) can be performed in parallel.

A central machine maintains and updates the sets I and $N(I)$. It receives as input the sets of vertices $\mathcal{X}_1, \ldots, \mathcal{X}_{n^{i\alpha}}$ and their lists of alive neighbours, so the total input size is proportional to

$$\sum_{j=1}^{n^{i\alpha}} \sum_{v \in \mathcal{X}_j} d_I(v) \;\leq\; n^{i\alpha} \cdot n^{\mu/2} \cdot n^{1-(i-1)\alpha} \;=\; n^{1+\mu/2+\alpha}.$$

After executing the for-loop, the central machine can use these lists of neighbours to also compute $N^+(I)$.

The next step requiring parallelization is line 11. To execute this, the central machine sends a bit to each vertex v indicating if $v \in N^+(I)$. Then, every $v \notin N^+(I)$ asks each neighbour $w \in N(v)$ if $w \in N^+(I)$. The results of these queries allow v to compute $d_I(v)$.

By Lemma 3.2, the while loop takes $O(1/\mu)$ rounds, so the entire algorithm takes $O(1/\mu^2)$ rounds. □

Maximal clique. One might initially assume that an algorithm for maximal independent set immediately implies an algorithm for maximal clique. But, as mentioned in Section 1, it is not clear how to reduce maximal clique to maximal independent set in $O(m)$ space: complementing the graph might require $\Omega(n^2)$ space. Nevertheless, it is possible to adapt Algorithm 2 so that it will compute a maximal clique. A description of the necessary modifications appears in Appendix B.

4 $(1+\varepsilon)\ln\Delta$-approximation for weighted set cover

For convenience, we reuse the same notation from Section 2. The standard greedy algorithm for weighted set cover is as follows. We maintain a set C of *covered* elements (initially $C = \emptyset$). In each iteration, we find the set S_i which maximizes $|S_i \setminus C|/w_i$. We add S_i to our solution and add the elements of S_i to C. It is known [13] that this algorithm has approximation ratio H_Δ where $\Delta = \max_\ell |S_\ell|$ and $H_k = \sum_{i=1}^k \frac{1}{k}$.

Unfortunately, implementing this greedy algorithm is tricky in MapReduce but, following Kumar et al. [26], we can implement the *ε-greedy algorithm*, which differs from the standard greedy method as follows. Instead of choosing the set S_i to maximize $|S_i \setminus C|/w_i$, we choose S_i such that $|S_i \setminus C|/w_i \geq \frac{1}{1+\varepsilon}\max_j\{|S_j \setminus C|/w_j\}$. The standard dual fitting argument [13, 42] can be easily modified to prove that this gives a $(1+\varepsilon)H_\Delta$-approximate solution.

Our algorithm for set cover is also inspired by some of the PRAM algorithms for set cover (see, for example, [10, 11, 17, 38]). Indeed, we use a similar bucketing approach as [10, 11, 17, 38] which we now describe. Let $L = \max_\ell\{|S_\ell|/w_\ell\}$. We first consider only the "bucket" of sets that have a cost ratio of at least $L/(1+\varepsilon)$ and continue to add sets from this bucket to the cover until the bucket is empty (after which we decrease L by $1+\varepsilon$ and repeat). However, note that once we add a set from the bucket to the cover, some of the sets in the bucket may no longer have a sufficiently cost ratio to remain in the bucket; in this case, we need to remove them from the bucket.

It is shown in [11] (improving on [10]) that exhausting a bucket can be done in $O(\log^2 n)$ time in the PRAM model; this can be easily translated to a $O(\log^2(n)/(\mu\log(m)))$-round algorithm in the MapReduce model. Our main contribution is to show that, in MapReduce, one can exhaust a bucket in $O(\log^2(n)/(\mu^2\log^2(m)))$ rounds. In particular, when $n = \mathrm{poly}(m)$ and μ is a constant, the number of rounds decreases from $O(\log n)$ to $O(1)$.

The high level idea of our parallel algorithm for approximate minimum set cover is as follows. Let C denote the current elements in the partial solution constructed so far (initially $C = \emptyset$). At this point, the algorithm considers only considers sets, S_ℓ, which are almost optimal, i.e. $|S_\ell \setminus C|/w_\ell \geq L(1+\varepsilon)$, where initially, $L = \max_\ell\{|S_\ell|/w_\ell\}$ but is decreased as the algorithm runs. We then partition these sets into $1/\alpha$ groups where group i consists of sets whose cardinatliy is in $[m^{1-(i+1)\alpha}, m^{1-i\alpha})$. From each group i, we then sample $m^{1-(i+1)\alpha}$ collections of $m^{\mu/2}$ sets. Starting from the group 1, we will see that either every collection contains an almost optimal set or we have made a large amount of progress. It will also turn out that we can decrease a certain potential function by a large amount in each iteration so within a few iterations, there will be no more profitable sets. If we have not covered the universe at this point then we decrease L by $(1+\varepsilon)$ and repeat this process.

We now begin with a more formal treatment of the algorithm for minimum weight set cover. We will assume that each machine has $O(m^{1+\mu}\log(n))$ space and $\sum_{i=1}^n |S_i| \geq m^{1+c}$ so that the memory is sublinear in the input size. The pseudocode for the algorithm is given in Algorithm 3.

In Section 4.1, we will describe the MapReduce implementation of this algorithm. Of particular note is that the second while loop can be implemented in a small number of MapReduce rounds. Our goal in this section is to show that the number of times the while loop is executed is small.

To that end, let us fix L and analyze the number of iterations of the second while loop starting at Line 7. To do this, it will be convenient to introduce the potential function

$$\Phi_k := \sum_{\ell \in [n] : \frac{|S_\ell \setminus C_k|}{w_\ell} \geq \frac{L}{1+\varepsilon}} |S_\ell \setminus C_k|.$$

Observe that we have the trivial upper bound $\Phi_k \leq nm$. Moreover, $\Phi_k = 0$ if and only if we finished with the second while loop by iteration k.

Using a straightforward Chernoff bound (Theorem E.1), we can show that each iteration has a small proabilility of failure, i.e. in each iteration, we make it into line 16 with very small probability.

Algorithm 3 A $(1+\varepsilon)\ln\Delta$-approximation for minimum weight set cover. Blue lines are centralized.

```
1: procedure ApproxSC($S_1, \ldots, S_n \subseteq U = [m]$, $w_1, \ldots, w_n \in \mathbb{R}_{>0}$)
2:     $L \leftarrow \max_\ell \left\{\frac{|S_\ell|}{w_\ell}\right\}$
3:     $\mathcal{S} \leftarrow \emptyset$
4:     $C \leftarrow \emptyset$        ▷ $C \subseteq [m]$ maintains set of covered elements
5:     while $C \neq [m]$ do
6:         $k \leftarrow 1$
7:         while $\exists \ell$ s.t. $|S_\ell \setminus C_k|/w_\ell \geq L/(1+\varepsilon)$ do
8:             $C_k \leftarrow C$        ▷ Maintain temporary set of covered elements; used only for proof.
9:             Let $\mathcal{S}_{k,i} = \{S_\ell : m^{1-i\alpha} \leq |S_\ell \setminus C_k| < m^{1-(i-1)\alpha}$ and $|S_\ell \setminus C_k|/w_k \geq L/(1+\varepsilon)\}$
10:            for $i = 1, \ldots, 1/\alpha$ do
11:                for $j = 1, \ldots, 2m^{(i+1)\alpha}$ do
12:                    Include each $S_\ell \in \mathcal{S}_{k,i}$ into group $\mathcal{X}_{i,j}$ with probability $\min\{1, m^{\mu/2}/|\mathcal{S}_{k,i}|\}$
13:            if $|\mathcal{X}_{i,j}| \leq 4m^{\mu/2}$ then
14:                Send each $\mathcal{X}_{i,j}$ to the central machine.
15:            else        ▷ At least one $\mathcal{X}_{i,j}$ is too big so fail and continue to next iteration.
16:                $k \leftarrow k+1$
17:                continue
18:            for $i = 1, \ldots, 1/\alpha$ do
19:                for $j = 1, \ldots, 2m^{(i+1)\alpha}$ do
20:                    if $\exists S_\ell \in \mathcal{X}_{i,j}$ such that $|S_\ell \setminus C| \geq m^{1-(i+1)\alpha}/2$ then
21:                        $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_\ell\}$
22:                        $C \leftarrow C \cup S_\ell$
23:            $k \leftarrow k+1$
24:        $L \leftarrow L/(1+\varepsilon)$
25:    return $\mathcal{S}$
```

Claim 4.1. Fix an iteration k. We have $|\mathcal{X}_{i,j}| \leq 4m^{\mu/2}$ for all i,j with probability at least $1 - \frac{m}{\alpha}\exp\left(-m^{\mu/2}\right)$.

The first lemma states that for a good fraction of the sets, a good fraction of their remaining elements has been covered. This will be useful for showing that Φ_k decreases significantly after each iteration.

Lemma 4.2. Fix i and an iteration k. Let $\mathcal{S}_{k,i}$ be as in Algorithm 3 and

$$\mathcal{S}'_{k,i} = \{S_\ell \in \mathcal{S}_{k,i} : |S_\ell \setminus C_{k+1}| \geq m^{1-(i+1)\alpha}/2 \text{ and } |S_\ell \setminus C_{k+1}|/w_k \geq L/(1+\varepsilon)\}.$$

Then $|\mathcal{S}'_{k,i}| \leq |\mathcal{S}_{k,i}|/2m^{\mu/4}$ with probability at least $1-m\exp\left(-m^{\mu/4}/2\right)$.

Proof. Suppose we are currently processing group $\mathcal{X}_{i,j}$. If at this point, the number of sets $S_\ell \in \mathcal{S}_{k,i}$ with both $|S_\ell \setminus C| \geq m^{1-(i+1)\alpha}/2$ and $|S_\ell \setminus C|/w_k \geq L/(1+\varepsilon)$ is at most $|\mathcal{S}_{i,k}|/2m^{\mu/4}$ then we are done (since $C \subseteq C_{k+1}$). Otherwise, with probability at least $1 - \left(1 - m^{\mu/2}/|\mathcal{S}_{k,i}|\right)^{|\mathcal{S}_{k,i}|/2m^{\mu/4}} \geq 1 - \exp\left(-m^{\mu/4}/2\right)$, the group $\mathcal{X}_{i,j}$ contains a set S_ℓ with both $|S_\ell \setminus C| \geq m^{1-(i+1)\alpha}/2$ and $|S_\ell \setminus C|/w_k \geq L/(1+\varepsilon)$. The algorithm then adds S_ℓ to the solution. Note that if this happens $2m^{(i+1)\alpha}$ times then we have covered the ground set and so the conclusion of the lemma holds trivially. Hence, we conclude that with probability at least $1-m\exp\left(-m^{\mu/4}/2\right)$, we have $|\mathcal{S}'_{k,i}| \leq |\mathcal{S}_{k,i}|/2m^{\mu/4}$. □

Let us now fix $\alpha = \mu/8$. The following lemma shows that the potential function decreases geometrically after every iteration.

Lemma 4.3. Fix an iteration k. Then $\Phi_{k+1} \leq \Phi_k/m^{\mu/8}$ with probability at least $1 - \frac{16m}{\mu}\exp\left(-m^{\mu/4}/2\right)$.

Proof. By Claim 4.1, an iteration fails with probability at most $\frac{m}{\alpha}\exp\left(-m^{\mu/2}\right) = \frac{8m}{\mu}\exp\left(-m^{\mu/2}\right)$.

For $i \in [1/\alpha]$, let $\mathcal{A}_i = \left\{S_\ell \in \mathcal{S}_{k,i} \cap \mathcal{S}'_{k,i} : \frac{|S_\ell \setminus C_{k+1}|}{w_\ell} \geq \frac{L}{1+\varepsilon}\right\}$ and $\mathcal{B}_i = \left\{S_\ell \in \mathcal{S}_{k,i} \setminus \mathcal{S}'_{k,i} : \frac{|S_\ell \setminus C_{k+1}|}{w_\ell} \geq \frac{L}{1+\varepsilon}\right\}$. Then

$$\Phi_{k+1} = \sum_{i=1}^{1/\alpha}\Big(\sum_{S_\ell \in \mathcal{A}_i}|S_\ell \setminus C_{k+1}| + \sum_{S_\ell \in \mathcal{B}_i}|S_\ell \setminus C_{k+1}|\Big).$$

By Lemma 4.2 and the definition of $\mathcal{S}'_{k,i}$, we have

$$\sum_{S_\ell \in \mathcal{A}_i}|S_\ell \setminus C_{k+1}| \leq |\mathcal{S}'_{k,i}|m^{1-(i-1)\alpha} \leq |\mathcal{S}_{k,i}|m^{1-(i-1)\alpha-\mu/4}/2.$$

On the other hand, we have $\sum_{S_\ell \in \mathcal{B}_i}|S_\ell \setminus C_{k+1}| \leq |\mathcal{S}_{k,i}|m^{1-(i+1)\alpha}/2$. Plugging in $\alpha = \mu/8$, we have

$$\Phi_{k+1} \leq \sum_{i=1}^{1/\alpha}|\mathcal{S}_{k,i}|m^{1-i\mu/8}/m^{\mu/8}.$$

Finally, we can also write

$$\Phi_k = \sum_{i=1}^{1/\alpha}\sum_{S_\ell \in \mathcal{S}_{k,i}}|S_\ell \setminus C_k| \geq \sum_{i=1}^{1/\alpha}|\mathcal{S}_{k,i}|m^{1-i\mu/8}.$$

Hence, we conclude that $\Phi_{k+1} \leq \Phi_k/m^{\mu/8}$. □

We now show that the number of iterations of the inner while loop is quite small with high probability.

Lemma 4.4. Let $K = \inf\{k > 0 : \Phi_k = 0\}$. Then $K \leq \frac{18\log\Phi_0}{\mu\log m}$ with probability $\geq 1 - \frac{32m}{\mu}\exp\left(-m^{\mu/4}/2\right)$.

Proof. Let us define a new random process Φ'_k, which is coupled to Φ_k as follows. First, $\Phi'_0 = \Phi_0$. For $k \geq 1$ we have two cases. If $\Phi_k \geq 1$ then we set

$$\Phi'_{k+1} = \begin{cases}\Phi'_k/m^{\mu/8} & \text{if } \Phi_{k+1} \leq \Phi_k/m^{\mu/8} \\ 0 & \text{otherwise}\end{cases}.$$

On the other hand if $\Phi_k = 0$ then we set $\Phi'_{k+1} = \Phi'_k/m^{\mu/8}$ with probability $1-\frac{16m}{\mu}\exp\left(-m^{\mu/4}/2\right)$. With the remainder probability, we set $\Phi'_{k+1} = \Phi_k$.

Observe that with this coupling, we have $\Phi_k \le \Phi'_k$ for all $k \ge 0$. Set $K' = \frac{18 \ln \Phi_0}{\mu \ln m}$. We will show that $\Phi'_{K'} < 1$ with probability at least $1 - \frac{32m}{\mu} \exp\left(-m^{\mu/4}/2\right)$. This then implies that $K \le K'$ with the same probability since $\Phi_{K'}$ must be a nonnegative integer.

Let us say iteration k is *good* if $\Phi'_{k+1} \le \Phi'_k / m^{\mu/8}$. Otherwise, we say it is *bad*. By Lemma 4.3 and the definition of $\{\Phi'_k\}_k$, it follows that iteration k is good with probability at least $1 - \frac{16m}{\mu} \exp\left(-m^{\mu/4}/2\right)$. After $\frac{8 \ln \Phi_0}{\mu \ln m} + 1 \le \frac{9 \ln \Phi_0}{\mu \ln m}$ good iterations, we have $\Phi'_k < 1$ so it suffices to show that after K' iterations there are at most $\frac{9 \ln \Phi_0}{\mu \ln m} = K'/2$ bad iterations. Indeed, after K' iterations, the expected number of bad iterations is at most $K' \cdot \frac{16m}{\mu} \exp\left(-m^{\mu/4}/2\right)$ so by Markov's Inequality, there are more than $K'/2$ bad iterations with probability at most $\frac{32m}{\mu} \exp\left(-m^{\mu/4}/2\right)$. This completes the proof. □

We are now ready to prove the main theorem in this section. Define $\Phi = \sum_{\ell \in [n]} |S_\ell|$.

Theorem 4.5. Algorithm 3 returns $(1+\varepsilon)H_\Delta$-approximate minimum set cover. Moreover, the inner loop is executed at most

$$O\left(\frac{\ln(\Phi) \log_{1+\varepsilon}(\Delta w_{\max}/w_{\min})}{\mu \ln m}\right)$$

times with probability at least

$$1 - \frac{64m}{\mu} \exp\left(-m^{\mu/4}/2\right).$$

Proof. The correctness follows because the algorithm implements the ε-greedy algorithm for set cover.

Let $M = \max_\ell \left\{\frac{|S_\ell|}{w_\ell}\right\}$. We can prove the running time as follows. Let $K = \frac{18 \log \Phi}{\mu \log m}$ and consider splitting up the iterations of the inner while loop into blocks of size K. By block t, we will refer to iterations $tK, \ldots, (t+1)K - 1$ of the inner while loop. We say that a block t is good if L has decreased at least once during that block. If the algorithm has already completed by that point, we will instead just flip a coin which comes up heads with probability at least $1 - \frac{32m}{\mu} \exp\left(-m^{\mu/4}/2\right)$. If it comes up heads we will call that block good. Otherwise, we call the block bad.

Note that after $\log_{1+\varepsilon}(M w_{\max})$ good blocks, we are guaranteed that the algorithm has terminated. If we consider $2 \log_{1+\varepsilon}(M w_{\max})$ blocks then it suffices that at most $\log_{1+\varepsilon}(M w_{\max})$ blocks are bad. The expected number of bad blocks is $\frac{32m}{\alpha} \exp\left(-m^{\mu/4}/2\right)$ so applying Markov's Inequality shows that after $O\left(\frac{\ln(\Phi) \log_{1+\varepsilon}(M w_{\max})}{\mu \ln m}\right)$ iterations, the algorithm has terminated with probability at least $1 - \frac{64m}{\alpha} \exp\left(-m^{\mu/4}/2\right)$. Using the trivial bound $M \le \Delta / w_{\min}$ and replacing α with $\mu/8$ completes the proof. □

4.1 MapReduce Implementation

It is straightforward to implement most steps in Algorithm 3 in MapReduce. However, we will highlight two nontrivial step here. The first is how to propagate information such as the set of covered elements C to all the machines. To do this, it is convenient to imagine all the machines as arranged in an $O(m^\mu)$-ary tree where the root of the tree is the central machine. Then the central machine can pass C down to its children. These machines then pass down to their children and so on. By doing this, all machines will know C in $O\left(\frac{\ln n}{\mu \ln m}\right)$ MapReduce rounds.

The machines can also determine $|\mathcal{S}_{k,i}|$ in a similar manner but starting at the leaves of tree. Here, each machine will compute the number of sets that are in $\mathcal{S}_{k,i}$ and send that quantity to its parents. The parents then sum up the input and their own contribution to $|\mathcal{S}_{k,i}|$ which they send to their own parents. Eventually the root is able to compute $|\mathcal{S}_{k,i}|$ and then propagates the number back down to the leaves as done above. This again takes $O\left(\frac{\ln n}{\mu \ln m}\right)$ rounds in MapReduce.

We can also use a similar strategy to check that $|\mathcal{X}_{i,j}|$ is small in Line 16 of Algorithm 3. We thus have the following theorem.

Theorem 4.6. There is a MapReduce algorithm that returns a $(1+\varepsilon)H_\Delta$-approximate minimum set cover which uses memory $O(m^{1+\mu} \log n)$ and runs in $O\left(\frac{\ln(\Phi) \log_{1+\varepsilon}(\Delta w_{\max}/w_{\min}) \ln(n)}{\mu^2 \ln^2(m)}\right)$ with probability at least $1 - O\left(\frac{m}{\mu}\right) \exp\left(-m^{\mu/4}/2\right)$.

Using the bound $\Phi \le nm$ yields the bound given in Figure 1.

Remark 4.7. Lemma 2.5 in [10] gives a simple way to preprocess the input so that $w_{\max}/w_{\min} \le mn/\varepsilon$ as follows. Let $\gamma = \max_{i \in [m]} \min_{S \ni i} w(S)$. This is a lower bound on the cost of any cover. First, we add every set with cost at most $\gamma\varepsilon/n$ to the cover; this constributes a cost of at most $\gamma\varepsilon \le \varepsilon \cdot OPT$. Next, we remove any set with cost more than $m\gamma$ since $OPT \le m\gamma$. All these steps can be done using a broadcast tree in $O(\log(n)/(\mu \log(m)))$ rounds of MapReduce.

5 Maximum weight matching

We have a graph $G = (V, E, w)$ but now $w \colon E \longrightarrow \mathbb{R}$ is a weight function on the edges. A matching in a graph is a subset $M \subseteq E$ such that $e_1 \cap e_2 = \emptyset$ for any distinct $e_1, e_2 \in M$. The maximum weight matching in a graph is a matching M that maximizes $\sum_{e \in M} w_e$.

5.1 The local ratio method

Sequential local ratio algorithm for maximum weight matching

Arbitrarily select an edge e with positive weight and reduce its weight from itself and its neighboring edges. Push e onto a stack and repeat this procedure until there are no positive weight edges remaining. At the end, unwind the stack adding edges greedily to the matching.

If the edge $e = (u, v)$ was selected, then the weight reduction makes the updates $w_{e'} \leftarrow w_{e'} - w_e$ for any edge e' such that $e' \cap e \ne \emptyset$. In contrast to the minimum vertex cover algorithm, weights in the graph can be reduced to negative weights.

Theorem 5.1 (Paz-Schwartzman [37]). The above algorithm returns a matching which is at least half the optimum value.

5.2 Randomized local ratio

As in Section 2.1, we apply our randomized local ratio technique to make the algorithm above amenable to parallelization. For intuition, consider a fixed a vertex v and suppose we sample approximately n^μ of its edges uniformly at random. If e is the heaviest sampled edge then there are only about $d(v)/n^\mu$ edges incident to v that are heavier than e. Hence, in the local ratio algorithm, if we choose e as the edge to reduce then this effectively decreases the degree of v by a factor of $n^{-\mu}$.

Algorithm 4 2-approximation for maximum weight matching. Blue lines are centralized.

```
procedure ApproxMaxMatching($G = (V, E)$)
    $E_1 \leftarrow E$, $d_1(v) \leftarrow d(v)$, $i \leftarrow 1$
    $S \leftarrow \emptyset$                          ▷ Initialize an empty stack.
    while $E_i \neq \emptyset$ do
        for each vertex $v \in V$ do
            if $|E_i| < 4\eta$ then
                Let $E'_v$ be all edges in $E_i$ incident to $v$
            else
                Construct $E'_v \subseteq E_i \cap \delta(v)$ by sampling i.i.d. with probability $p = \min\left\{\frac{\eta}{|E_i|}, 1\right\}$
        if $\sum_v |E'_v| > 8\eta$ then
            Fail
        for each vertex $v \in V$ do
            Let $e \in E'_v$ be the heaviest edge and apply weight reduction to $e$
            Push $e$ onto the stack $S$
        Let $E_{i+1}$ be the subset of $E_i$ with positive weights
        ▷: Let $d_{i+1}(v)$ denote $|\{e \in E_{i+1} : v \in e\}|$
        $i \leftarrow i + 1$
    Unwind $S$, adding edges greedily to the matching $M$
    return $M$
```

The following claim follows by a simple Chernoff bound.

Claim 5.2. For $i \geq 1$ and conditioned on iteration $i-1$ not failing, iteration i fails with probability at most $\exp(-\eta)$.

Lemma 5.3. Suppose $\eta = n^{1+\mu}$ for some constant $\mu > 0$ and $|E| = n^{1+c}$ for some $c > \mu$. Then, with probability at least $1 - (n + 1) \cdot \exp(-n^\mu)$:

- the first iteration does not fail; and
- $d_2(v) \leq n^c$ for all $v \in V$, where d_2 is as defined in Line 16 of Algorithm 4.

Proof. Let k_v be the number of edges incident to v with positive weight when we reach v in the for loop in Line 13. If $k_v \leq n^c$ then we are done so suppose $k_v > n^c$. The probability that we do not sample any of the heaviest n^c edges that are currently incident to v is at most $\left(1 - \frac{n^{1+\mu}}{|E|}\right)^{n^c} \leq \exp\left(-\frac{n^{1+\mu+c}}{|E|}\right) \leq \exp(-n^\mu)$. Combining with Claim 5.2 and taking a union bound completes the proof. □

In the subsequent analysis, we will assume that μ is a positive constant. (Actually, it suffices to take $\mu = \omega(\log\log n/\log n)$.)

Lemma 5.4. Suppose $\eta = n^{1+\mu}$ for any constant $\mu > 0$. Let $\Delta_i = \max_v d_i(v)$. For $i > 2$ and conditioning on the past $i-1$ iterations not failing, with probability at least $1 - (n + 1) \cdot \exp(-n^{\mu/2})$:

- iteration i does not fail; and
- $\Delta_{i+1} \leq \Delta_i/n^{\mu/4}$.

Proof. Let k_v be the number of edges incident to v with positive weight when we reach v in the for loop in Line 13. If $k_v \leq \Delta_i/n^{\mu/4}$ then we are done so suppose $k_v > \Delta_i/n^{\mu/4}$. The probability that we do not sample any of the heaviest $k_v/n^{\mu/4}$ edges that are currently incident to v is at most $\left(1 - \frac{n^{1+\mu}}{|E|}\right)^{k_v/n^{\mu/4}} \leq \exp\left(-\frac{n^{1+\mu/2}\Delta_i}{|E|}\right) \leq \exp(-n^{\mu/2})$. Hence, we have $d_{i+1}(v) \leq k_v/n^{\mu/4} \leq d_i(v)/n^{\mu/4}$ with probability at least $1-\exp(-n^{\mu/2}) = 1-\exp(-\omega(\log n))$. Combining with Claim 5.2 and taking a union bound completes the proof. □

Theorem 5.5. Suppose $\eta = O(n^{1+\mu})$ for any constant $\mu > 0$. With probability $1-O(cn/\mu)\exp(-n^\mu)$, Algorithm 7 terminates in $O(c/\mu)$ iterations and returns a 2-approximate maximum matching.

Proof. By Lemma 5.3, the first iteration does not fail and $d_2(v) \leq n^c$ for all $v \in V$. By Lemma 5.4, after at most $O(c/\mu)$ iterations, the algorithm has not failed and the total number of edges with positive weight remaining is at most $8n^{1+\mu}$. At this point, the algorithm completes its last iteration of weight reduction then unwinds the stack and returns a matching. The correctness of the algorithm follows from Theorem 5.1. □

The preceding analysis assumes that $\mu = \omega(\log\log n/\log n)$. In Appendix C we additionally handle the case $\mu = 0$ (or equivalently, $\mu = \Theta(1/\log n)$).

5.3 MapReduce implementation

Theorem 5.6. There is a MapReduce algorithm that computes a 2-approximation to the maximum weight matching using $O(n^{1+\mu})$ space per machine and $O(c/\mu)$ rounds (when $\mu = \omega(\log\log n/\log n)$) or $O(\log n)$ rounds (when $\mu = 0$).

Proof. As a sequential algorithm, the correctness of Algorithm 7 is established by Theorem 5.5 in the case μ is constant and Theorem C.2 when $\mu = 0$. We now show how to parallelize Algorithm 7.

As in Theorem 2.4, there are $n^{c-\mu}$ machines and each edge is assigned to one of the machines with $O(n^{1+\mu})$ edges per machine. Each vertex (and its adjacency list) is randomly assigned to one of the machines, so the space per machine is $O(n^{1+\mu})$ whp. Each edge e stores its original weight and maintains a bit indicating if $e \in E_i$.

The local ratio steps (lines 12-14) are performed sequentially on the central machine. The input to the central machine is the sets E'_v, together with the original weights of those edges. The total size of the input is proportional to $\sum_v |E'_v|$, which is guaranteed to be $O(\eta)$ by line 11. The central machine is stateful: it maintains values $\phi(v)$ for each vertex v, initially zero. The value $\phi(v)$ will always equal the total value of the weight reductions for all edges incident to v. Thus, for any edge $e = \{u, v\}$ that was never added to the stack, if its original weight is w_e, then its modified weight is $w_e - \phi(u) - \phi(v)$. The weight reduction operation for edge $e =$

$\{u, v\}$ is then straightforward: simply decrease both $\phi(u)$ and $\phi(v)$ by $w_e - \phi(u) - \phi(v)$.

The main step requiring parallelization is line 15. After the for-loop terminates, the central machine sends $\phi(v)$ to each vertex v, and informs each edge whether it was added to the stack. If so, the edge's modified weight is definitely non-positive so it cannot belong to E_{i+1}. Afterwards, each vertex v then sends $\phi(v)$ to each edge $e \in \delta(v)$. Each edge $e = \{u, v\}$ receives $\phi(u)$ and $\phi(v)$ and, if it was not added to the stack, computes its modified weight; if this is positive then e belongs to E_{i+1}. Given the knowledge of whether $e \in E_i$ and given $|E_i|$ (which is easy to compute in parallel), an edge can independently perform the sampling operation on line 9. □

b-matching. Using similar techniques, one can obtain a $(3-2/b+\varepsilon)$-approximation to b-matching; see Appendix D.

6 Vertex and edge colouring

In this section, we show how to colour a graph with maximum degree Δ using $(1+o(1))\Delta$ colours in a constant number of rounds. As is the case with MIS, $(\Delta+1)$-vertex colouring is one of the most fundamental problems of distributed computing. In the CREW PRAM model, both MIS and $(\Delta+1)$-vertex colouring have algorithms that can be easily translated to $O(\log n)$-round algorithms in the MapReduce model. Luby's randomized algorithms for both MIS [31] and $(\Delta + 1)$-vertex colouring [32] have clean MapReduce implementations by using one machine per processor in the CREW PRAM algorithm. Within the CREW PRAM and LOCAL model, both MIS and $(\Delta + 1)$-vertex colouring have well established lower bounds. However, we are unaware of non-trivial (i.e. non-constant) lower bounds on round complexity in the MapReduce model.

Although our algorithm does not use the randomized local ratio or the hungry-greedy paradigm developed in the previous sections, we feel that it is of independent interest as it is the first constant round algorithm for $\Delta + o(\Delta)$-vertex colouring within the MapReduce model that we are aware of.

Recall that the number of edges is n^{1+c}, so $\Delta \geq n^c$ since the maximum degree is at least the average degree. Our algorithm is very simple; first we randomly partition the vertex set into $\kappa := n^{(c-\mu)/2}$ groups. Within each group, the maximum degree is $(1 + o(1))\Delta/\kappa$ with high probability, so $(1+o(1))\Delta/\kappa+1$ colours suffices to colour each group. The colour of a vertex is determined by the group that it is in and its colour within each group. Hence, this gives a colouring with $(1 + o(1))\Delta$ colours. Furthermore, we show that the subgraph induced by each group has a small number of edges. As a corollary we get a MapReduce algorithm for $(1+o(1))\Delta$-colouring.

Lemma 6.1. For all $\mu > 0$ and $\kappa = n^{(c-\mu)/2}$ then we have $\Delta_i \leq (1 + n^{-\mu/2}\sqrt{6 \ln n})\Delta/\kappa$ for all i with probability at least $1 - 1/n$.

Proof. For a single vertex v of degree d, the probability that v has degree greater than $(1 + \varepsilon)\Delta/\kappa$ (in the subgraph induced by its group) is at most $\exp\left(-\varepsilon^2 \frac{\Delta}{3\kappa}\right)$ by a standard Chernoff bound (Theorem E.1). Since $\Delta/\kappa \geq n^{c/2+\mu/2} \geq n^{\mu}$, we may take $\varepsilon = n^{-\mu/2}\sqrt{6 \ln n}$ in which case the probability that a vertex v has degree greater than $(1+\varepsilon)\Delta/\kappa$ is at most $1/n^2$. Taking a union bound then yields the lemma. □

Algorithm 5 $(1 + o(1))\Delta$-vertex colouring

```
1: procedure VertexColouring(Graph $G = (V, E)$)
2:     Randomly partition $V$ into $\kappa$ groups, $V_1, \ldots, V_\kappa$
3:     Define $E_i = E[V_i]$ and $G_i = (V_i, E_i)$
4:     if $\exists i$ such that $|E_i| > 13n^{1+\mu}$ then
5:         Fail
6:     for every vertex $v$ in parallel do
7:         if $v \in V_i$ then send $N(v) \cap V_i$ to central machine $i$
8:     for every central machine $i$ in parallel do
9:         Let $\Delta_i$ be max degree of $G_i$
10:        Colour $G_i$ using the standard $(\Delta_i + 1)$-vertex colouring algorithm
11:        for every $v \in V_i$ do
12:            Let $c_i(v)$ be colour of $v \in V_i$ in this colouring
13:            Output $(i, c_i(v))$ as colour for $v$.
```

Lemma 6.2. We have $|E_i| \leq 13n^{1+\mu}$ with probability at least $1 - n^2 \cdot \exp(-n^{\mu})$.

Proof. By the Hajnal-Szemerédi Theorem (Theorem E.2) applied to the line graph of G, we may partition the edges into 2Δ sets $F_1, \ldots, F_{2\Delta}$ such that the following holds for all $j \in [2\Delta]$:

- $|F_j| \geq n^{1+c}/(4\Delta)$; and
- if $e, e' \in F_j$ and $e \neq e'$ then $e \cap e' = \emptyset$.

For analysis, fix E_i and an edge class F_j. For each $e \in F_j$, let $X_{j,e}$ be the indicator random variable which is 1 if edge e ends up in E_i. Then $\mathbb{E}\sum_{e \in F_j} X_{j,e} = |F_j|/\kappa^2$. Since any distinct $e, e' \in F_j$ do not share a common vertex, it follows that $\{X_{j,e}\}_{e \in F_j}$ are mutually independent random variables. Therefore, we may apply a Chernoff bound (Theorem E.1) to get

$$\Pr\Big[\sum_{e \in F_j} X_{j,e} > 13|F_j|/\kappa^2\Big] \leq \exp\left(-12\frac{|F_j|}{3\kappa^2}\right)$$
$$\leq \exp\left(-\frac{n^{1+c}}{\Delta\kappa^2}\right) \leq \exp\left(-\frac{n^c}{\kappa^2}\right) = \exp\left(-n^{\mu}\right).$$

Taking a union bound over all i and j gives the claim. □

Corollary 6.3. Algorithm 5 returns a $\left(1 + n^{-\mu/2}\sqrt{6 \ln n} + n^{-\mu}\right)\Delta$-colouring of G with high probability.

Proof. We can colour each G_i with $\Delta_i + 1$ colours using the standard greedy colouring algorithm. By Lemma 6.1 $\Delta_i \leq (1 + n^{-\mu/2}\sqrt{6 \ln n})\Delta/\kappa$ with probability at least $1 - 1/n$. Hence, we use at most $\kappa(\Delta_i + 1) \leq \left(1 + n^{-\mu/2}\sqrt{6 \ln n} + n^{-\mu}\right)\Delta$ colours. □

Theorem 6.4. If $\mu = \omega(\log\log n/\log n)$ and the memory on each machine is $O(n^{1+\mu})$ then there is a MapReduce algorithm for $(1 + o(1))\Delta$-colouring which succeeds with high probability.

Remark 6.5. $(1 + o(1))\Delta$-edge colouring can be achieved with almost the same algorithm, partitioning the edges into groups instead of vertices and colouring the groups with the algorithm of Misra and Gries [36].

Theorem 6.6. There are constant-round MapReduce algorithms for $(1+o(1))\Delta$-vertex colouring and $(1+o(1))\Delta$-edge colouring that succeed w.h.p.

A Improved algorithm for maximal independent set

In this section, we discuss how to improve the algorithm in Section 3 to obtain an algorithm that rounds in $O(c/\mu)$ rounds instead of $O(1/\mu^2)$. The basic idea is that if we sample $n^{(i+1)\alpha}$ vertices from the set of vertices that have degree $[n^{1-i\alpha}, n^{1-(i-1)\alpha})$ and proceed as in Algorithm 2, then the degree of almost all vertices decreases by a factor of n^{α}. In particular, this implies that the number of edges decrease by a factor of n^{α} which gives a algorithm using $O(c/\alpha)$ rounds. We will set $\alpha = \mu/8$ in the analysis below.

Algorithm 6 Improved algorithm for maximal independent set. Blue lines are centralized.

```
1:  procedure MIS2($G = (V, E)$)
2:      $I_1 \leftarrow \{v\colon d(v) = 0\}, E_1 \leftarrow E, V_1 \leftarrow V, k \leftarrow 1$
3:      while $|E_k| \geq n^{1+\mu}$ do
4:          $I \leftarrow I_k$
5:          Let $V_{k,i} = \{v \in V_k\colon n^{1-i\alpha} \leq d_I(v) < n^{1-(i-1)\alpha}\}$, for $i \in \{1, \ldots, 1/\alpha\}$.
6:          for $i = 1, \ldots, 1/\alpha$ do
7:              Draw $n^{(i+1)\alpha}$ groups of $n^{\mu/2}$ vertices from $V_{k,i}$, say $\mathcal{X}_{i,1}, \ldots, \mathcal{X}_{i,n^{(i+1)\alpha}}$
8:          for $i = 1, \ldots, 1/\alpha$ do
9:              for $j = 1, \ldots, n^{(i+1)\alpha}$ do
10:                 if $\exists v_{i,j} \in \mathcal{X}_{i,j}$ such that $d_I(v_{i,j}) \geq n^{1-(i+1)\alpha}$ then
11:                     $I \leftarrow I \cup \{v_{i,j}\}$
12:         $I_{k+1} \leftarrow I, V_{k+1} \leftarrow \{v\colon d_{I_{k+1}}(v) > 0\}, E_{k+1} \leftarrow E[V_{k+1}]$
13:         $k \leftarrow k + 1$
14:     Find maximal independent set in $(V_k, E_k)$ and add it to $I$ ▷ Total edges is $< n^{1+\mu}$
15:     return $I$
```

In the algorithm, the set $V_{k,i}$ corresponds to the set of vertices in iteration k with degree between $n^{1-i\alpha}$ and $n^{1-(i-1)\alpha}$. We first show that most vertices in $V_{k,i}$ lose a significant number of their neighbours. This will be important to show good progress in removing the edges.

Lemma A.1. Let $V_{k,i}$ be as in the algorithm and $V'_{k,i} = \{v \in V_{k,i} : d_{I_{k+1}}(v) \geq n^{1-(i+1)\alpha}\}$. In other words, $V'_{k,i}$ is the set of vertices within $V_{k,i}$ that did not decrease their degree by at least a factor of $n^{i\alpha}$. Then $|V'_{k,i}| \leq |V_{k,i}|/n^{\mu/4}$ w.h.p.

Proof. Suppose we are currently processing group $\mathcal{X}_{i,j}$. If the number of vertices v in $V_{k,i}$ with $d_I(v) \geq n^{1-(i+1)\alpha}$ is at most $|V_{k,i}|/n^{\mu/4}$ then we are done. Otherwise, with probability at least $1-(1-n^{-\mu/4})^{n^{\mu/2}} \geq 1-\exp(-n^{\mu/4})$, the group $\mathcal{X}_{i,j}$ contains a vertex $v_{i,j}$ with $d_I(v_{i,j}) \geq n^{1-(i+1)\alpha}$ and we add it to the independent set. This can only happen at most $n^{(i+1)\alpha}$ times because if it happens $n^{(i+1)\alpha}$ times then there are no more vertices in the graph. Since we sampled precisely $n^{(i+1)\alpha}$ groups of vertices, we expect $|V'_{k,i}| \leq |V_{k,i}|/n^{\mu/4}$ w.h.p. □

By choosing $\alpha = \mu/8$, we can guarantee that every vertex class $V_{k,i}$ decrease their degree by a factor of n^{α} with high probability. This gives the following lemma.

Lemma A.2. We have $|E_{k+1}| \leq 2|E_k|/n^{\mu/8}$ w.h.p.

This lemma implies that after $O(c/\mu)$ rounds, we can fit all the data onto a single machine.

Proof. Let $S \subseteq V$ and define $d_I(S) = \sum_{v\in S} d_I(v)$. We will lower bound $d_{I_k}(V_{k,i})$ and upper bound $d_{I_{k+1}}(V_{k,i})$. We have $d_{I_k}(V_{k,i}) \geq |V_{k,i}|n^{1-i\alpha}$. On the other hand,

$$\begin{aligned} d_{I_{k+1}}(V_{k,i}) &\leq |V'_{k,i}|n^{1-(i-1)\alpha} + |V_{k,i} \setminus V'_{k,i}|n^{1-(i+1)\alpha} \\ &\leq |V_{k,i}|n^{-\mu/4}n^{1-(i-1)\alpha} + |V_{k,i}|n^{1-(i+1)\alpha} \\ &= 2|V_{k,i}|n^{1-(i+1)\mu/8}, \end{aligned}$$

by Lemma A.1 and using $\alpha = \mu/8$. Since the sets $V_{k,i} \cap V_{k+1}$ form a partition of V_{k+1}, we have

$$\begin{aligned} 2|E_{k+1}| &= \sum_i d_{I_{k+1}}(V_{k,i} \cap V_{k+1}) \\ &= \sum_i d_{I_{k+1}}(V_{k,i}) \leq 2\sum_i |V_{k,i}|n^{1-(i+1)\mu/8} \\ &\leq 2\sum_i d_{I_k}(V_{k,i})n^{-\mu/8} \\ &= 4|E_k|n^{-\mu/8}, \end{aligned}$$

proving the claim. □

Theorem A.3. There is a MapReduce algorithm to find a maximal independent set in $O(c/\mu)$ rounds and $O(n^{1+\mu})$ space per machine, w.h.p.

Proof sketch. The pseudocode in Algorithm 6 is written in the manner of a sequential algorithm, but it is easy to parallelize. The main step that is performed in parallel is the sampling (lines 6-7). The sets $\mathcal{X}_{i,j}$ are then sent to a central machine. Lines 8-11 are performed sequentially on the central machine. All other steps are straightforward to perform in the MapReduce model.

The space usage on the central machine is dominated by the total size of the neighbourhoods of all vertices in the sets $\mathcal{X}_{i,j}$. Thus, the required space per machine is proportional to

$$\begin{aligned} \sum_{i=1}^{1/\alpha} \sum_{j=1}^{n^{(i+1)\alpha}} \sum_{v\in\mathcal{X}_{i,j}} d_I(v) &\leq \sum_{i=1}^{1/\alpha} n^{(i+1)\alpha} \cdot n^{\mu/2} \cdot n^{1-(i-1)\alpha} \\ &= (1/\alpha) \cdot n^{1+\mu/2+2\alpha}. \end{aligned}$$

Since $\alpha = \mu/8$, this gives the claimed space bound. □

B Maximal clique

We can compute a maximal clique via a reduction to MIS. This is, however, nontrivial because naively complementing the graph could result in a graph that is too large to fit into memory. To resolve this, we will use a relabeling scheme which satisfies the following properties:

- if there are k active vertices then every active vertex has a label in $[k]$.

- every vertex knows k and the label of all its neighbours.

Suppose we had such a labelling and consider a vertex v. Suppose N is the set of active neighbours of v. Then its degree in the complement graph is $k - |N|$ and its active neighbourhood in the complement graph is $[k] \setminus N$. Observe that, while the complement graph could have $\Omega(n^2)$ edges, there is no space issue because each round only requires us to compute $O(n^{1+\mu})$ edges of the complement graph *in total.*

We now describe the relabelling scheme. Initially, the two conditions are trivially satisfied if we assume each vertex has a unique label in $[n]$. To maintain the invariant, we add the following relabeling procedure:

(1) Suppose there are k active vertices (the central machine always knows which vertices are active). Pick a permutation $\sigma\colon V \longrightarrow [n]$ such that if v is active then $\sigma(v) \in [k]$ and if v is inactive then $\sigma(v) > k$. Send $\sigma(v)$ and k to vertex v.
(2) Each vertex v knows $\sigma(v)$. If u is a neighbour of v then v queries u for $\sigma(u)$. It then knows which of its neighbours are still active (since it knows k).

Corollary B.1. There is a MapReduce algorithm to find a maximal clique in $O(1/\mu)$ rounds w.h.p.

C Matching with $O(n)$ space per machine

In this section we discuss the matching algorithm (Algorithm 7) in the case $\mu = 0$, which corresponds to having $O(n)$ space per machine.

Lemma C.1. Suppose $\eta = n$. Then none of the iterations fail w.h.p. Moreover, conditioned on none of the iterations failing, we have $\mathbb{E}\left[|E_{i+1}| \mid E_i\right] \le 0.975|E_i|$.

Proof. The first part of the lemma is by Claim 5.2.

Let $H_i = \{v : d_i(v) \ge |E|/n\}$ and $L_i = V \setminus H_i$. We call H_i the heavy vertices at iteration i and L_i the light vertices at iteration i. Let $c = 0.9$. We claim that if v is heavy then with probability at least $1/2$, we have $d_{i+1}(v) \le c \cdot d_i(v)$. Let k_v be the number of edges incident to v with positive weight when we reach v in the for loop in Line 13. If $k_v \le c \cdot d_i(v)$ then we are done so suppose $k_v > c \cdot d_i(v)$. The probability that we do not sample any of the heaviest ck_v edges that are currently incident to v is at most $\left(1 - \frac{n}{|E|}\right)^{ck_v} \le \exp\left(-c^2\right) < 1/2$.

Observe that $\sum_{v \in L_i} d_i(v) \le |E_i|$ so $\sum_{v \in H_i} d_i(v) \ge |E_i|$. Hence, conditioned on E_i, we have

$$\begin{aligned} \mathbb{E}|E_{i+1}| &\le \frac{1}{2}\left(\sum_{v \in L_i} d_i(v) + \sum_{v \in H_i}\left[\frac{1}{2}d_i(v) + \frac{1}{2}cd_i(v)\right]\right) \\ &\le \frac{3+c}{4}|E_i| = 0.975|E_i|, \end{aligned}$$

proving the second part of the lemma. ☐

Theorem C.2. With probability at least $1-1/n$, Algorithm 4 terminates in $O(\log n)$ iterations and returns a 2-approximate maximum matching.

Proof. Let us consider a modified version of Algorithm 4 where the condition to terminate when $|E_i| < 8\eta$ is removed. Let E'_i denote the sequence of edges in the modified algorithm and E_i to denote the sequence of edges in the unmodified version of Algorithm 4. There is a simple coupling which ensures that:

- $E'_1 = E_1$;
- $E'_{i+1} = E_{i+1}$ if $|E_i| \ge 8n$; and
- if $|E_i| < 8n$ then Algorithm 4 terminates in iteration $i+1$.

Observe that the final condition implies that

$$\Pr[\text{Algorithm 7 terminates by iteration } \tau + 1] \ge \Pr[|E'_\tau| < 8n].$$

Choose $\tau = 200\log(n) > \log(n^2)/\log(1/0.975)$. By the previous lemma, we have $\mathbb{E}|E'_\tau| \le 1$ so by Markov's Inequality we have $\Pr[|E'_\tau| \ge 8n] \le 1/(8n)$. In particular, Algorithm 7 terminates in $O(\log(n))$ iterations with probability at least $1 - 1/n$.

The correctness follows from Theorem 5.1. ☐

D Maximum weight b-matching

D.1 Local ratio method

Sequential local ratio algorithm for maximum weight b-matching

Arbitrarily select an edge $e = (u, v)$ with positive weight, say $w(e)$. Reduce the weight of e by $w(e)$. For all other $e' \ni u$, reduce its weight by $w(e)/b(u)$. For all other $e' \ni v$, reduce its weight by $w(e)/b(v)$. Push e onto a stack and repeat this procedure until there are no positive weight edges remaining. At the end, unwind the stack adding edges greedily to the matching.

Theorem D.1. Let $b = \max_v b(v)$. The above algorithm returns a $(3 - 2/\max\{2, b\})$-approximate matching.

Proof. We assume $b \ge 2$ since the case $b = 1$ is exactly Theorem 5.1.

Let $M_0 = \emptyset$ and, for $i \ge 1$, let M_i be the matching maintained after we have unwinded i edges from the stack. Next, let G_0 be the (weighted) graph just before we begin unwinding the edges and, for $i \ge 1$, let G_i be the graph with the last i weight reductions reversed. (So, $G_0, G_1, \ldots$ all have the same vertex and edge sets; only the edge weights are different.)

We claim that M_i is a $(3 - 2/b)$-approximate matching to G_i for all $i = 0, 1, 2, \ldots$ which proves the claim. The case $i = 0$ is trivial since G_0 has no positive edge weights, so an empty matching is an optimal matching.

Now suppose that M_{i-1} is a $(3 - 2/b)$-approximate matching to G_{i-1}. Let $\mathrm{OPT}_{i-1}, \mathrm{OPT}_i$ denote an optimal matching of G_{i-1}, G_i, respectively and w_{i-1}, w_i be the respective weight functions. Let $e_i = (u_i, v_i)$ be the ith edge that was popped from the stack. Note that

$$
\begin{aligned}
w_i(\mathrm{OPT}_i) &\le w_{i-1}(\mathrm{OPT}_{i-1}) + w_i(e_i) + \left(\frac{b(u_i)-1}{b(u_i)} + \frac{b(v_i)-1}{b(v_i)}\right) w_i(e_i) \\
&= w_{i-1}(\mathrm{OPT}_{i-1}) + w_i(e_i)\left(3 - \frac{1}{b(u_i)} - \frac{1}{b(v_i)}\right) \\
&\le w_{i-1}(\mathrm{OPT}_{i-1}) + w_i(e_i)(3 - 2/b).
\end{aligned}
$$

Next, we give a lower bound for $w_i(M_i)$. Suppose first that we add e_i to the matching, i.e. $M_i = M_{i-1} \cup \{e_i\}$. Then

$$w_i(M_i) \ge w_{i-1}(M_{i-1}) + w_i(e_i).$$

On the other hand, suppose we do not add e_i to the matching, i.e. $M_i = M_{i-1}$. In this case, M_i must contain at least $b(u_i)$ edges incident to e_i or $b(v_i)$ edges incident to e_i. This implies that $w_i(M_i) \ge w_{i-1}(M_{i-1}) + w_i(e_i)$ because undoing the weight reduction increases the weight of the edges incident to u_i (other than e_i) by $w(e_i)/b(u_i)$ and similarly for v_i. In either case, we have

$$
\begin{aligned}
(3-2/b)w_i(M_i) &\ge (3-2/b)(w_{i-1}(M_{i-1}) + w_i(e_i)) \\
&\ge w_{i-1}(\mathrm{OPT}_{i-1}) + (3-2/b)w_i(e_i) \\
&\ge w_i(\mathrm{OPT}_i),
\end{aligned}
$$

where the second inequality used the assumption that M_{i-1} is a $(3-2/b)$-approximate matching to G_{i-1}. □

D.2 Randomized local ratio

Unfortunately, we cannot translate the above local ratio method for b-matching to the MapReduce model in the same way that we did for matching. The issue is as follows. Consider a vertex and suppose that it has b edges all of unit weight. In the matching case (i.e. $b = 1$) if any of these edges were chosen by the local ratio algorithm then every edge incident to the vertex would be killed off by the weight reduction step. On the other hand if $b > 1$ then the weight of the remaining edges after looking at $t < b$ edges is $(1-1/b)^t > 0$. In particular, we have only managed to kill all the edges after we have looked at all the edges!

To fix this, we will consider an ε-adjusted local ratio method which is inspired by the algorithm of Paz and Schwartzman [37] for weighted matchings in the semi-streaming model. As in Theorem 5.6, we maintain a variable $\phi(v)$ for each vertex v which corresponds to the sum of the weight reductions incident to vertex v. However, rather than killing an edge (u,v) if $w_{(u,v)} \le \phi(u) + \phi(v)$, we kill the edge if $w_{(u,v)} \le (1+\varepsilon)(\phi(u)+\phi(v))$. This corresponds to applying a reduction with a multiplier of either 1 or $1+\varepsilon$. It is not hard to show that this gives a $(3-2/\max\{2,b\}+2\varepsilon)$-approximate b-matching. Indeed, the only change to the above proof is to replace the inequality $w_i(\mathrm{OPT}_i) \le w_{i-1}(\mathrm{OPT}_{i-1}) + (3-2/b)w_i(e_i)$ with $w_i(\mathrm{OPT}_i) \le w_{i-1}(\mathrm{OPT}_{i-1}) + (3-2/b+2\varepsilon)w_i(e_i)$.

Let us now give some intuition for the MapReduce implementation of the ε-adjusted local ratio method. Consider a fixed a vertex v and suppose we sample approximately $2b(v)$ of its edges uniformly at random. In expectation, $b(v)$ of the sampled edges will have weight at least the median weight edge adjacent to v. By reducing all of these edges in the local ratio algorithm, we can effectively remove half of the edges adjacent to v. Similarly, by drawing $\tilde{O}(b(v)n^{\mu})$ edges, we decrease the degree of v by a factor of $n^{-\mu}$.

To implement the sequential local ratio algorithm in MapReduce, we essentially use the same algorithm as the one for maximum matching, adding up to $b \log(\delta^{-1})$ edges to the stack for each vertex, where $\delta = \varepsilon/(1+\varepsilon)$. To simplify the analysis of the algorithm, we randomly sample a fixed number of edges from each vertex instead of sampling with uniform probability from the graph.

Algorithm 7 $(3-2/b+2\varepsilon)$-approximation for maximum weight b-matching. Blue lines are centralized.

```
1:  procedure ApproxBMaxMatching($G = (V,E)$)
2:      $\delta \leftarrow \varepsilon/(1+\varepsilon)$
3:      $E_1 \leftarrow E$, $d_1(v) \leftarrow d(v)$, $i \leftarrow 1$
4:      $S \leftarrow \emptyset$                              ▷ Initialize an empty stack.
5:      while $E_i \ne \emptyset$ do
6:          for each vertex $v \in V$ do
7:              if $|E_i| < 2b \ln(\delta^{-1}) n^{1+\mu}$ then
8:                  Let $E'_v$ be all edges in $E_i$ incident to $v$
9:              else
10:                 Randomly sample $b(v) \ln(\delta^{-1}) n^{\mu}$ edges from $E_i$ incident to $v$ and add to $E'_v$
11:         for each vertex $v \in V$ do
12:             $j \leftarrow 1$
13:             while $j \le b(v) \ln(\delta^{-1})$ do
14:                 Let $e \in E'_v$ be the heaviest edge and apply an $\varepsilon$-adjusted weight reduction
15:                 Push $e$ onto the stack $S$
16:                 Remove $e$ from $E'_v$
17:                 $j \leftarrow j+1$
18:         Let $E_{i+1}$ be the subset of $E_i$ with positive weights
19:         $d_{i+1}(v) \leftarrow |\{e \in E_{i+1} : v \in e\}|$
20:         $i \leftarrow i+1$
21:     Unwind $S$, adding edges greedily to the $b$-matching $M$
22:     return $M$
```

The analysis of the algorithm is similar to the proof for maximum matching. Instead of repeating the entire proof, we show a variant of Lemma 5.4 below and note that all the other lemmas extend similarly.

Lemma D.2. Suppose $\eta = n^{1+\mu}$ for some constant $\mu > 0$. Let $\Delta_i = \max_v d_i(v)$. For $i > 2$ with probability at least $1 - \frac{n^2}{\delta} \cdot \exp(-n^{\mu/2}/2)$, it holds that $\Delta_{i+1} \le \Delta_i / n^{\mu/4}$.

Proof. If $|E_i| < 2b\ln(\delta^{-1})n^{1+\mu}$ then the local ratio algorithm is performed on the entire graph, and so $E_{i+1} = \emptyset$ and the lemma is trivial. So, assume that $|E_i| \ge 2b\ln(\delta^{-1})n^{1+\mu}$.

Let k_v be the number of edges incident to v with positive weight when we reach v in the for loop in line 13. If $k_v \le \Delta_i/n^{\mu/4}$ then we are done, so suppose $k_v > \Delta_i/n^{\mu/4}$.

Define an edge to be heavy if it is one of the top $k_v/n^{\mu/4}$ heaviest edges. We claim that we sampled at least $b(v)\ln(1/\delta)$ distinct heavy edges w.h.p. To see this, first split the sampled edges into $b(v)\ln(1/\delta)$ groups each with n^{μ} edges. Then, using a union bound, the probability that we *cannot* pick a distinct heavy edge from each group is bounded above by

$$\begin{aligned}&\sum_{t=1}^{b(v)\ln(1/\delta)}\left(1-\frac{k_v/n^{\mu/4}-t}{\Delta_i}\right)^{n^\mu}\\ &\leq b(v)\ln(1/\delta)\left(1-\frac{\Delta_i/n^{\mu/2}-b(v)\ln(1/\delta)}{\Delta_i}\right)^{n^\mu}.\end{aligned}\tag{D.1}$$

By assumption, we have $|E_i| \geq 2b(v)\ln\left(\delta^{-1}\right)n^{1+\mu}$, so since maximum degree is at least average degree, $\Delta_i/n^{\mu/2} \geq 2b(v)\log(1/\delta)$. Hence, (D.1) is at most

$$b(v)\ln(1/\delta)\left(1-\frac{1}{2n^{\mu/2}}\right)^{n^\mu} \leq b(v)\ln(1/\delta)\exp\left(-n^{\mu/2}/2\right).$$

So we have sampled at least $b(v)\ln(1/\delta)$ heavy edges with probability at least $1-b(v)\ln(1/\delta)\exp\left(-n^{\mu/2}/2\right)$. We will condition on this event for the rest of the proof.

Recall that in the algorithm we add the top $b(v)\ln(1/\delta)$ edges to the stack (note that the ordering of the remaining weights are unchanged in the while loop starting at Line 13 because we subtract the same quantity from each edge). Let w_L be the weight of the lightest of these edges *before* we perform the weight reductions. Imagine for now that the algorithm performs ordinary weight reductions rather than ε-adjusted reductions. Then, after performing the weight reductions, the weight of all non-heavy edges is at most $w_L(1-1/b(v))^{b(v)\ln(1/\delta)} \leq w_L\delta$. Since we choose $\delta = \varepsilon/(1+\varepsilon)$, this implies that we have reduced the weight of all non-heavy edges by a $1/(1+\varepsilon)$ fraction of their original weight. So, since the algorithm actually performs ε-adjusted weight reductions, it actually reduces the weight of the non-heavy edges to a non-positive value. These edges can now be safely discarded from the graph, so we have $d_{i+1}(v) \leq k_v/n^{\mu/4}$ after the end of the weight reduction phase (line 19). Taking a union bound completes the proof. □

Theorem D.3. There is a MapReduce algorithm that computes a $(3-2/b+2\varepsilon)$-approximation to the maximum weight b-matching in $O(c/\mu)$ rounds when $\mu > 0$ and $O(\log n)$ rounds when $\mu = 0$. The memory requirement is $O(b\log(1/\varepsilon)n^{1+\mu})$.

E Auxiliary Results

Theorem E.1 (Chernoff bound). Let $X_1,\ldots,X_n$ be independent random variables such that $X_i \in [0,1]$ with probability 1. Define $X = \sum_{i=1}^n X_i$ and let $\mu = \mathbb{E}X$. Then, for any $\varepsilon > 0$, we have

$$\Pr[X \geq (1+\varepsilon)\mu] \leq \exp\left(-\frac{\min\{\varepsilon,\varepsilon^2\}\mu}{3}\right).$$

Theorem E.2 (Hajnal-Szemerédi). Every graph with n vertices and maximum degree bounded by $k-1$ can be coloured using k colours so that each colour class has at least $\lfloor n/k\rfloor \geq n/(2k)$ vertices.

References

[1] Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 202–211, June 2015.

[2] Sepehr Assadi. Simple round compression for parallel vertex cover. *arXiv preprint arXiv:1709.04599*, 2017.

[3] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet edcs: Algorithms for matching and vertex cover on massive graphs. *arXiv preprint arXiv:1711.03076*, 2017.

[4] Sepehr Assadi and Sanjeev Khanna. Randomized composable coresets for matching and vertex cover. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017.

[5] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1995–2003, 2013.

[6] Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms in memoriam: Shimon Even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, 2004. doi:10.1145/1041680.1041683.

[7] Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985. doi:10.1016/S0304-0208(08)73101-3.

[8] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring*. Morgan & Claypool, 2017. Draft manuscript.

[9] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, pages 273–284, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2463664.2465224, doi:10.1145/2463664.2465224.

[10] Bonnie Berger, John Rompel, and Peter W. Shor. Efficient nc algorithms for set cover with applications to learning and geometry. *Journal of Computer and System Sciences*, 49(3):454 – 477, 1994. 30th IEEE Conference on Foundations of Computer Science. URL: http://www.sciencedirect.com/science/article/pii/S0022000005800686, doi:https://doi.org/10.1016/S0022-0000(05)80068-6.

[11] Guy E. Blelloch, Richard Peng, and Kanat Tangwongsan. Linear-work greedy parallel approximate set cover and variants. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 23–32, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/1989493.1989497, doi:10.1145/1989493.1989497.

[12] Vasek Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, August 1979.

[13] Stephen A. Cook. An overview of computational complexity. *Communications of the ACM*, 26(6):400–408, June 1983.

[14] Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

[15] Artur Czumaj, Jakub ÅĄÄĚcki, Aleksander MÄĚdry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms, 2017. URL: http://arxiv.org/abs/1707.03478.

[16] Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science*, 2016.

[17] Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '17, pages 293–304, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3087556.3087580, doi:10.1145/3087556.3087580.

[18] Mohsen Ghaffari. Space-optimal semi-streaming for $(2+\epsilon)$-approximate matching, 2017. URL: http://arxiv.org/abs/1701.03730.

[19] Michael T. Goodrich. Simulating parallel algorithms in the MapReduce framework with applications to parallel computational geometry. 2010. URL: http://arxiv.org/abs/1004.4708.

[20] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the MapReduce framework. In *22nd International Symposium on Algorithms and Computation (ISAAC)*, 2011.

[21] Elena Grigorescu, Morteza Monemizadeh, and Samson Zhou. Streaming weighted matchings: Optimal meets greedy. 2016. URL: http://arxiv.org/abs/1608.01487.

[22] Sungjin Im, Benjamin Moseley, and Xiaorui Sun. Efficient massively parallel methods for dynamic programming. In *ACM Symposium on Theory of Computing (STOC)*, 2017.

[23] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Symposium on Principles of Database Systems (PODS)*, 2014.

[24] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. HADI: mining radii of large graphs. *TKDD*, 5(2):8:1–8:24, 2011. URL: http://doi.acm.org/10.1145/1921632.1921634, doi:10.1145/1921632.1921634.

[25] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.

doi:10.1137/1.9781611973075.76.

[26] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in MapReduce and streaming. *TOPC*, 2(3):14:1–14:22, 2015. doi:10.1145/2809814.

[27] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 85–94, 2011. doi:10.1145/1989493.1989505.

[28] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 177–187, 2005. URL: http://doi.acm.org/10.1145/1081870.1081893, doi:10.1145/1081870.1081893.

[29] Jure Leskovec, Anand Rajaraman, and Jeff Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[30] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *J. ACM*, 62(5):38:1–38:17, 2015. URL: http://doi.acm.org/10.1145/2786753, doi:10.1145/2786753.

[31] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. URL: https://doi.org/10.1137/0215074, doi:10.1137/0215074.

[32] Michael Luby. Removing randomness in parallel computation without a processor penalty. *J. Comput. Syst. Sci.*, 47(2):250–286, 1993. URL: https://doi.org/10.1016/0022-0000(93)90033-S, doi:10.1016/0022-0000(93)90033-S.

[33] Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized composable coresets for distributed submodular maximization. In *ACM Symposium on Theory of Computing (STOC)*, pages 153–162, 2015.

[34] Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. Distributed submodular cover: Succinctly summarizing massive data. In *Neural Information Processing Systems (NIPS)*, 2015.

[35] Baharan Mirzasoleiman, Morteza Zadimoghaddam, and Andreas Krause. Fast distributed submodular cover: Public-private data summarization. In *Neural Information Processing Systems (NIPS)*, 2016.

[36] J. Misra and David Gries. A constructive proof of vizing's theorem. *Inf. Process. Lett.*, 41(3):131–133, March 1992. URL: http://dx.doi.org/10.1016/0020-0190(92)90041-S, doi:10.1016/0020-0190(92)90041-S.

[37] Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2153–2161, 2017. doi:10.1137/1.9781611974782.140.

[38] Sridhar Rajagopalan and Vijay V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):525–540, 1998. URL: https://doi.org/10.1137/S0097539793260763, arXiv:https://doi.org/10.1137/S0097539793260763, doi:10.1137/S0097539793260763.

[39] Stergios Stergiou and Kostas Tsioutsiouliklis. Set cover at web scale. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

[40] Leslie G. Valiant. Parallel computation. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science*, 1982.

[41] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[42] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[43] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.