Parallel Set Cover and Hypergraph Matching via Uniform Random Sampling

Laxman Dhulipala
Google Research, New York, NY, USA

Michael Dinitz
Johns Hopkins University, Baltimore, MD, USA

Jakub Łącki
Google Research, New York, NY, USA

Slobodan Mitrović
UC Davis, CA, USA

Abstract

The SetCover problem has been extensively studied in many different models of computation, including parallel and distributed settings. From an approximation point of view, there are two standard guarantees: an $O(\log \Delta)$-approximation (where Δ is the maximum set size) and an $O(f)$-approximation (where f is the maximum number of sets containing any given element).

In this paper, we introduce a new, surprisingly simple, model-independent approach to solving SetCover in unweighted graphs. We obtain multiple improved algorithms in the MPC and CRCW PRAM models. First, in the MPC model with sublinear space per machine, our algorithms can compute an $O(f)$ approximation to SetCover in $\hat{O}(\sqrt{\log \Delta} + \log f)$ rounds[1] and a $O(\log \Delta)$ approximation in $O(\log^{3/2} n)$ rounds. Moreover, in the PRAM model, we give a $O(f)$ approximate algorithm using linear work and $O(\log n)$ depth. All these bounds improve the existing round complexity/depth bounds by a $\log^{\Omega(1)} n$ factor.

Moreover, our approach leads to many other new algorithms, including improved algorithms for the HypergraphMatching problem in the MPC model, as well as simpler SetCover algorithms that match the existing bounds.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms; Theory of computation → Shared memory algorithms; Theory of computation → MapReduce algorithms

Keywords and phrases approximate maximum matching, set cover, hypergraph matching, PRAM, massively parallel computation

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.19

Related Version *Full Version*: `https://arxiv.org/abs/2408.13362`

Funding *Michael Dinitz*: Supported in part by NSF awards CCF-1909111 and CCF-2228995.
Slobodan Mitrović: Supported by the Google Research Scholar and NSF Faculty Early Career Development Program No. 2340048.

1 Introduction

There is perhaps no more central and important problem in the area of approximation algorithms than SetCover. It has been a testbed for various algorithmic techniques that have become central in the field: greedy algorithms, deterministic and randomized rounding, primal-dual, dual fitting, etc. Due to its importance, ubiquity, and the fact that many different algorithmic techniques can be used, it is widely considered a "textbook problem" and, for example, has been used to illustrate the very basics of approximation

[1] We use the $\hat{O}(x)$ notation to suppress poly log x and poly log log n terms.

algorithms [38, Chapter 1]. There are essentially two standard approximation bounds, both of which can be achieved through a number of different algorithms: an f-approximation, where f is the *frequency* (the maximum number of sets containing any given element), and an $H_\Delta = O(\log \Delta)$-approximation, where Δ is the maximum set size and H_k is the k'th harmonic number.[2]

Unsurprisingly, SetCover has also received significant attention in parallel and distributed models of computation. However, the simple sequential algorithms for SetCover are not "obviously" parallelizable, so new algorithms have been developed for these models. These lines of work range from classical complexity-theoretic models (e.g., showing that it can be approximated well in NC [7]), classical parallel models such as PRAMs [7, 34, 9], classical distributed models such as LOCAL [29, 28], and modern models such as MapReduce and Massively Parallel Computation (MPC) [36, 3]. Much of this work has been model-focused rather than model-independent, and ideas and techniques from one model can only sometimes be transferred to a different model.

In this paper, we introduce a new, simple, and model-independent technique for solving unweighted SetCover in parallel settings. Our technique, which involves careful independent random sampling of either the sets or elements, yields both a $(1+\epsilon)f$-approximation and a $(1+\epsilon)H_\Delta$-approximation and can be efficiently instantiated in multiple models of computation, including the MPC and PRAM models. Moreover, it can also be extended to solve the approximate HypergraphMatching problem in unweighted graphs. By applying our technique, we obtain efficient algorithms for SetCover and HypergraphMatching in MPC and PRAM models, which either improve upon or (essentially) match state-of-the-art algorithms for the problems. Importantly, our technique provides a unified and model-independent approach across HypergraphMatching and two variants of SetCover, and can be efficiently implemented in two fundamental models of parallel computation.

Our algorithms are obtained by parallelizing two classic f- and $O(\log \Delta)$-approximate SetCover algorithms. The f-approximate algorithm repeatedly picks an uncovered element and adds all sets containing it to the solution. The $O(\log \Delta)$-approximate in each step simply adds to the solution the set that covers the largest number of uncovered points.

Even though our parallelization of these algorithms is surprisingly direct, to the best of our knowledge, it has not been analyzed prior to our work. At a high level, our algorithms perform independent random sampling to find a collection of sets to be added to the solution, remove all covered elements from the instance, and then repeat. By combining the random sampling-based approach with modern techniques in parallel algorithms, we are able to give state-of-the-art bounds.

1.1 Our Contribution

We now present the main contributions of the paper. We study the unweighted version of SetCover. To formulate the bounds we obtain, we assume the SetCover problem is represented by a bipartite graph, in which vertices on one side represent the sets, and vertices on the other side represent elements to be covered. Edges connect elements with all sets that they belong to. We use Δ to denote the maximum degree of a vertex representing a set, and f to denote the maximum degree of a vertex representing an element. We use n to denote the number of vertices in the graph (equal to the number of sets plus the number of elements) and m to denote the number of edges (the total size of all sets).

[2] This result is often given as an $O(\log n)$-approximation since $\Delta \leq n$, but H_Δ is a technically stronger bound.

Table 1 Round complexity of SetCover algorithms in the Massively Parallel Computation model. We use n to denote the number of vertices in the graph (which is equal to the number of sets plus the number of elements) and m to denote the number of edges (the total size of all sets). $\delta \in (0,1)$ is a constant. In [22] $\phi \in (0,1]$ is any value satisfying $m \le n^{1+\phi}$ and $c < \phi$ controls the amount of space per machine. We use ‡ to denote that a bound holds with high probability.

Ref.	Space/Machine	Total Space	Approx. Factor	Det.	Round Complexity
[9]	$O(n^\delta)$	$O(m)$	$(1+\epsilon)H_\Delta$	No	$O(\log^2 n)^\ddagger$
Here	$O(n^\delta)$	$\tilde{O}(m)$	$(1+\epsilon)H_\Delta{}^\ddagger$	No	$\hat{O}(\log\Delta \cdot \sqrt{\log f})^\ddagger$
Here	$O(n^\delta)$	$\tilde{O}(m)$	$(1+\epsilon)f^\ddagger$	No	$\hat{O}(\sqrt{\log\Delta})^\ddagger$
[6]	$O(n^\delta)$	$O(m)$	$f+\epsilon$	Yes	$O(\log\Delta/\log\log\Delta)$
[16]	$O(n^\delta)$	$O(m)$	$(1+\epsilon)f$	Yes	$O(\log(f\Delta)/\log\log(f\Delta))$
[3]	$\tilde{O}(n)$	$\tilde{O}(m)$	$O(\log n)^\ddagger$	No	$O(\log n)^\ddagger$
Here	$\tilde{O}(n)$	$\tilde{O}(m)$	$(1+\epsilon)H_\Delta{}^\ddagger$	No	$O(\log\Delta)^\ddagger$
[22]	$O(fn^{1+c})$	$O(m)$	f	No	$O((\phi/c)^2)$

We start by presenting our results in the Massively Parallel Computation (MPC) model [25, 20, 4, 1]. MPC computation proceeds in *synchronous* rounds over M machines. We assume that the input to the computation is partitioned arbitrarily across all machines in the first round. Each machine has a local space of η bits. In one round of computation, a machine first performs computation on its local data. Then, the machines can communicate by sending messages: each machine can send messages to any other machine. The messages sent in one round are delivered at the beginning of the next round. Hence, within a round, the machines, given the messages received in this round, work entirely independently. Importantly, the total size of the messages sent or received by a machine in a given round is at most η bits.

In the context of graph algorithms, there are three main regimes of MPC defined with respect to the relation of the available space on each machine η to the number of vertices of the graph n. In the *super-linear* regime, $\eta = n^{1+c}$ for a constant $0 < c < 1$. The *nearly-linear* regime requires $\eta = n \operatorname{poly}\log n$. Finally, the most restrictive and challenging *sub-linear* regime requires $\eta = n^c$. In all the regimes, we require that the total space of all machines is only a $\operatorname{poly}\log n$ factor larger than what is required to store the input.

In our definition of the SetCover problem, the number of vertices is the number of sets plus the number of elements. We note that some SetCover algorithms in the linear space regime (both prior and ours) only require space near-linear in the number of sets plus sublinear in the number of elements, but we use a single parameter for simplicity.

SetCover in MPC

Our first result is a set of improved MPC algorithms for SetCover.

▶ **Result 1.** *Let $\epsilon \in (0, 1/2)$ be a constant. Denote by f the maximum number of sets an element appears in, and by Δ the largest set size. Then, SetCover can be solved in MPC with the following guarantees:*

- *$(1+\epsilon)H_\Delta$-approximation in $\hat{O}\left(\operatorname{poly}(1/\epsilon)\cdot\log\Delta\cdot\sqrt{\log f}\right)$ rounds in the sub-linear regime,*
- *$(1+\epsilon)H_\Delta$-approximation in $O(\log\Delta)$ rounds in the nearly-linear regime,*
- *$(1+\epsilon)f$-approximation in $\hat{O}\left(\operatorname{poly}(1/\epsilon)\cdot\left(\sqrt{\log\Delta}+\log f\right)\right)$ rounds in the sub-linear regime.*

The algorithms use $\tilde{O}(m)$ total space, and the round complexities hold with high probability.

Table 2 Parallel cost bounds (work and depth) of f-approximate SetCover algorithms in the CRCW PRAM. m denotes the sum of the sizes of all sets (or the number of edges in the bipartite representation of SetCover), n denotes the number of elements, f denotes the maximum number of sets any element is contained in, and $\epsilon \in (0, 1/2)$ is an arbitrary constant. We use * to denote that a bound holds in expectation, and ‡ to denote that a bound holds with high probability.

Ref.	Approx. Factor	Det.	Work	Depth	Notes
[27]	$(1+\epsilon)f$	Yes	$O(fm)$	$O(f \log^2 n)$	
[28]	2	No	$O(m)^*$	$O(\log n)^\ddagger$	For weighted instances with $f = 2$.
Here	$(1+\epsilon)f^*$	No	$O(m)$	$O(\log n)$	

Before our work, the best-known round complexity for the $(1+\epsilon)H_\Delta$ SetCover in the sub-linear regime was $O(\log \Delta \cdot \log f)$; this complexity is implicit in [7]. Our algorithm improves this bound by a $\sqrt{\log f}$ factor. In the nearly-linear space regime, it is possible to achieve $O(\log n)$-approximation in $O(\log n)$ rounds by building on [3]. It is unclear how to transfer this approach to the sub-linear regime. We improve the approximation ratio to H_Δ, which is better, especially when $\Delta \ll n$.

In terms of $(1+\epsilon)f$-approximation, the most efficient SetCover algorithm in MPC follows by essentially a direct adaption of the Congest/Local $O(\log \Delta / \log \log \Delta)$ round algorithms in [6, 16] to MPC. Hence, for $f \le 2^{O(\sqrt{\log n})}$, our work improves the MPC round complexity nearly quadratically.

SetCover in PRAM

Since our main algorithmic ideas are model-independent, they also readily translate to the PRAM setting, giving a new result for $(1+\epsilon)f$-approximate SetCover that improves over the state-of-the-art, and a streamlined $(1+\epsilon)H_\Delta$-approximation algorithm for SetCover [9].

▶ **Result 2.** *Let $\epsilon \in (0, 1/2)$ be an absolute constant. Let f be the maximum number of sets an element appears in, and let Δ be the largest set size. Then, SetCover can be solved in CRCW PRAM with the following guarantees:*

- *$(1+\epsilon)f$-approx. in expectation with* deterministic *$O(n+m)$ work and $O(\log n)$ depth.*
- *$(1+\epsilon)H_\Delta$-approx. in expectation with* deterministic *$O(n+m)$ work and $O(\log^2 n \log \log n)$ depth.*

In the context of $(1+\epsilon)f$-approximation, our result improves the state-of-the-art [27] total work by f while depth is improved by an $f \log n$ factor. For $(1+\epsilon)H_\Delta$-approximation, our result obtaining deterministic $O(\log^2 n \log \log n)$ depth and providing the approximation guarantee in expectation should be compared to the state-of-the-art PRAM algorithm of Blelloch, Peng, and Tangwongsan [9], which provides a depth guarantee in expectation and a worst-case guarantee for the approximation ratio. While the expected depth bound reported in [9] is $O(\log^3 n)$, we believe it can be improved to $O(\log^2 \log \log n)$ using some of the implementation ideas in our PRAM algorithm (see the full version for more discussion). A more detailed comparison between the prior and our results in PRAM is given in Table 2.[3]

[3] Our algorithms provide approximation in expectation. Nevertheless, this can be lifted to "with high probability" guarantees by executing $O(\log n/\epsilon)$ independent instances of our algorithm and using the smallest set cover. It incurs an extra $O(\log n/\epsilon)$ factor in the total work while not affecting the depth asymptotically.

HypergraphMatching in MPC

Finally, we also obtain an improved MPC algorithm for finding hypergraph matchings, i.e., for finding matchings in graphs where an edge is incident to (at most) h vertices.

▶ **Result 3.** *Let $\epsilon \in (0, 1/2)$ be an absolute constant. There is an MPC algorithm that, in expectation, computes a $(1-\epsilon)/h$ approximate maximum matching in a rank h hypergraph in the sub-linear space regime. This algorithm succeeds with high probability, runs in $\hat{O}\left(\mathrm{poly}(1/\epsilon) \cdot \left(h^4 + h \cdot \sqrt{\log \Delta}\right)\right)$ MPC rounds and uses a total space of $\tilde{O}(m)$.*

Prior work [21] shows how to solve HypergraphMatching in rank h hypergraphs in $O(\log n)$ rounds in the *nearly-linear* space regime. So, for $h \in O(1)$, Result 3 improves quadratically over the known upper bound and, in addition, extends to the sub-linear space regime at the cost of slightly worsening the approximation ratio. For simple graphs, i.e., when $h = 2$, the work [19] already provides $\tilde{O}(\sqrt{\log \Delta})$ round complexity algorithm for computing $\Theta(1)$-approximate, and also maximal, matching. Nevertheless, our approach is arguably simpler than the one in [19] and, as such, lands gracefully into the MPC world.

1.2 Further Related Work

SetCover in the MPC Model

Both SetCover and VertexCover, i.e., SetCover with $f = 2$, have been extensively studied in the MPC model. Stergiou and Tsioutsiouliklis [36] studied the SetCover problem in MapReduce and provided an empirical evaluation. Their main algorithm is based on bucketing sets to within a $(1+\epsilon)$ factor with respect to the set sizes and then processing all the sets within the same bucket on one machine. Their algorithm, when translated to the MPC model, runs in $O(\log \Delta)$ iterations, but does not come with a bound on the required space per machine, which in the worst case can be linear in the input size.

Harvey, Liaw, and Liu [22] studied weighted VertexCover and SetCover in the MPC model and obtained results for both f and $(1+\epsilon)H_\Delta$-approximation. Their results exhibit a tradeoff between the round complexity and the space per machine. For f-approximation, they gave a $O((\phi/c)^2)$ round algorithm with space per machine $O(fn^{1+c})$ by applying filtering [30] to a primal dual algorithm. When the space per machine is nearly-linear, i.e., $c = O(1/\log n)$, this approach results in $O(\phi^2 \log^2 n)$ rounds, which is quadratically slower than our algorithm.

Bateni, Esfandiari, and Mirrokni [3] developed a MapReduce algorithm for the k-cover problem that uses $\tilde{O}(n)$ space per machine. In this problem, one is given an integer k and is asked to choose a family of at most k sets that cover as many elements as possible. The problem, since it is a submodular maximization under cardinality constraint, admits a $\Theta(1)$-approximation. Their algorithm can be turned into an $O(\log n)$-approximate one for SetCover that uses $O(\log n)$ MPC rounds as follows. Assume that k is the minimum number of sets that covers all the elements; this assumption can be removed by making $O(\log n/\epsilon)$ guesses of the form $k = (1+\epsilon)^i$. Then, each time [3] is invoked, it covers a constant fraction of the elements. So, repeating that process $O(\log n)$ times covers all the elements using $O(k \cdot \log n)$ many sets. Our result provides tighter approximation and, when $\Delta \ll n$, also lower round complexity.

Since k-cover is a submodular maximization problem, the work [32] yields $O(\log n)$ MPC round complexity and $O(\log n)$ approximation for SetCover. In the context of k-cover or SetCover, it is worth noting that the algorithm of [32] sends $\Theta(\sqrt{nk})$ sets to a machine. It is unclear whether all those sets can be compressed to fit in $O(n)$ or smaller memory.

Ghaffari and Uitto [19] developed a $\tilde{O}(\sqrt{\log \Delta})$ round complexity algorithm for VertexCover in the sub-linear space regime. They first compute a maximal independent set, which is then used to obtain a maximal matching in the corresponding line graph. Finally, by outputting the endpoints of the edges in that maximal matchings, the authors provide a 2-approximate VertexCover. Our algorithm has a matching round complexity while, at the same time, it is arguably simpler. For both f-approximation and H_Δ-approximation, we are unaware of any MPC algorithms that run in the sub-linear space regime. However, we note that the PRAM algorithm of Blelloch, Peng, and Tangwongsan [9] can be simulated in this setting to obtain a round complexity of $O(\log^2 n)$ with $O(m)$ total space.

f-Approximate SetCover in PRAM

The first f-approximation algorithms for SetCover in the sequential setting are due to Hochbaum [23]. In the unweighted case, we can sequentially obtain an f-approximation in $O(m)$ work by picking any element, adding all of $\leq f$ sets containing it to the cover, and removing all newly covered elements. For parallel algorithms aiming for f-approximation, Khuller, Vishkin, and Young [27] gave the first parallel $(1+\epsilon)f$-approximation for weighted SetCover that runs in $O(fm\log(1/\epsilon))$ work and $O(f\log^2 n\log(1/\epsilon))$ depth. Their method uses a deterministic primal-dual approach that in each iteration raises the dual values $p(e)$ on every uncovered element e until the primal solution, which is obtained by rounding every set s where $\sum_{e\in s} p(e) \geq (1-\epsilon)w(s)$, is a valid set cover. Their work analysis bounds the total number of times an element is processed across all $O(f\log n)$ iterations by m, giving a total work of $O(f\cdot m)$, which is not work-efficient. Their algorithm also has depth linear in f, which means that the number of iterations of their algorithm can be as large as $O(\log^2 n)$ for $f \leq \log n$, and the depth therefore as large as $O(\log^3 n)$.

For weighted VertexCover, Koufogiannakis and Young [28] gave an elegant 2-approximation that runs in $O(m)$ work in expectation and $O(\log n)$ depth. They generalize their algorithm to work for f-approximate weighted SetCover in the *distributed* setting using Linial-Saks decomposition [33]; however, this does not imply an NC or RNC algorithm when $f > 2$.[4]

Unlike the deterministic $(1+\epsilon)f$-approximation of Khuller et al. [27], our algorithm is randomized and produces a set cover with the same approximation guarantees in expectation. By contrast, our algorithm is easy to understand, analyze (with Lemma 3 as a given) and argue correctness. Our algorithm is well suited for implementation and has small constant factors, since every element set or element and their incident edges are processed *exactly once* when the element is sampled or when the set is chosen. We note that our algorithm also implies that $(1+\epsilon)f$-approximate SetCover is in RNC^1 for any f; the work of [27] only implies this result for $f = O(1)$.

Matching and HypergraphMatching in the Massively Parallel Computation Model

The study of approximate matchings in MPC was initiated by Lattanzi et al. [30], who developed an $O(1)$ round algorithm for finding a maximal matching when the space per machine is $n^{1+\mu}$, for any constant $\mu > 0$. In the linear space regime, a line of work [14, 17, 2, 5]

[4] NC contains all problems that admit log-space uniform circuits of polynomial size and poly-logarithmic depth and is the primary complexity class of interest when designing parallel algorithms. RNC extends NC by allowing the circuit access to randomness. By known simulation results [26], polynomial work and poly-logarithmic depth (randomized) PRAM algorithms also imply membership in NC (RNC).

culminated in $O(\log\log n)$ MPC round complexity. In the sublinear space regime, Ghaffari and Uitto [19] developed a method that finds a maximal matching in $\tilde{O}(\sqrt{\log n})$ rounds. When each machine has at least $O(nr)$ space, Hanguir and Stein [21] show how to find a maximal matching in r-hypergraph in $O(\log n)$ MPC rounds. Their approach follows the filtering idea developed in [30]. Our work does not only provide nearly quadratically lower round complexity compared to [21], but it also extends to the sub-linear space regime.

H_Δ-approximate SetCover in PRAM

Sequentially, H_Δ-approximate SetCover can be solved in $O(n+m)$ work by repeatedly selecting the set incident to the largest number of uncovered elements. The first parallel approximation algorithm for SetCover was due to Berger, Rompel and Shor [7], who gave a $(1+\epsilon)H_\Delta$-approximation that runs in $O(m\log^5 n)$ work and $O(\log^5 n)$ depth whp. Their algorithm buckets the sets based on their sizes into $O(\log\Delta)$ buckets. It then runs $O(\log f)$ subphases, where the j-th subphase ensures that all elements have degrees at most $(1+\epsilon)^j$ (the subphases are run in decreasing order). Each subphase performs $O(\log n)$ steps that work by either selecting sets that cover a constant fraction of certain large edges or otherwise independently sampling the remaining sets with probability $(1+\epsilon)^{-j}$. Our approach also uses independent sampling but does not require handling two cases separately. As a result, our approach can be implemented efficiently by fixing the random choices upfront (see Section 3.1). Subsequent work by Rajagopalan and Vazirani [34] improved the work and depth, obtaining a parallel primal-dual algorithm with $O(m\log^3 n)$ work and $O(\log^3 n)$ depth with high probability, but a weaker approximation guarantee of $2(1+\epsilon)H_\Delta$.

More recently, Blelloch, Peng and Tangwongsan [9] revisited parallel approximate SetCover with the goal of designing work-efficient algorithms. Their algorithm achieves a $(1+\epsilon)H_\Delta$-approximation in $O(m)$ expected work and $O(\log^3 n)$ depth with high probability on the CRCW PRAM. They propose a general primitive inspired by the approach of [34] called a Maximal Nearly-Independent Set (MaNIS), which, given a collection of sets chooses a subset of them while ensuring that the chosen sets are (1) nearly independent and thus do not have significant overlap, and (2) maximal, so that any unchosen sets have significant overlap with chosen ones. Blelloch, Simhadri, and Tangwongsan [10] later studied the algorithm in the Parallel Cache Oblivious model, and provided an efficient parallel implementation.

Compared to this prior work, we obtain a streamlined $(1+\epsilon)H_\Delta$-approximate algorithm that shares some ideas with the previously discussed algorithms. We also bucket the sets by size, and like [34, 9] each round finds a subset of sets with low overlap; the main difference is that our method is arguably simpler. Our algorithm is also potentially very efficient in practice, since after we fix the randomness up-front (see Section 3.1), we process every set in a bucket exactly once, unlike other implementations of MaNIS which can process a set within a bucket potentially many times [15]. Overall, our algorithm is work-efficient and runs in $O(\log^2 n\log\log n)$ depth on the CRCW PRAM. Although this is an improvement over known depth bounds for PRAM algorithms, one can obtain similar bounds (in expectation) for the algorithm of [9] by applying similar PRAM techniques.We also note that both algorithms achieve $O(\log^3 n)$ depth in the binary-forking model [8], and no parallel H_Δ-approximate algorithms exist with $o(\log^3 n)$ depth in this model. Experimentally comparing our algorithm with existing implementations of [9] is an interesting direction for future work.

1.3 Outline

The rest of the paper is organized as follows. In Section 2 we introduce notation that we use in the paper. Section 3 contains a technical overview of our results. In particular, it describes our algorithms and outlines how they can be analyzed and efficiently implemented in the MPC and PRAM models. Then, in Section 4 we provide the approximation analysis of our basic algorithms. Finally, in Appendix A we provide the formal analysis of the random process which we use to model our algorithms. Due to space constraints, the remaining details, including the detailed descriptions of the MPC and PRAM algorithms are deferred to the full version of this paper.

2 Preliminaries

In the SetCover problem, we have a collection of *elements* T and a family of *sets* S, which we can use to cover elements of T. We represent an instance of the problem with a bipartite graph $G = ((S \cup T), E)$, where $st \in E$ if and only if element t belongs to the set s. For a vertex $x \in S \cup T$ we use $N(x)$ to denote the set of its neighbors. Since G is bipartite, $x \in S$ implies $N(x) \subseteq T$ and $x \in T$ implies $N(x) \subseteq S$. In particular, for $x \in S$, $|N(x)|$ is the size of the set x.[5] We use Δ to denote the maximum set size (i.e., the maximum degree of any vertex in S) and f to denote the largest number of sets that contain any element (the maximum degree of any vertex in T). Note that some of our algorithms modify the input graphs along the way, but we assume Δ and f to be constant and refer to the corresponding quantities in the input graph.

The VertexCover problem is defined as follows. The input is an undirected graph $G = (V, E)$ and the goal is to find the smallest set $C \subseteq E$ such that each edge has at least one endpoint in C. We note that this problem is equivalent to the SetCover problem in which each element belongs to exactly 2 sets, except that the graph representing an instance is constructed a bit differently.

In the HypergraphMatching problem, the input is a hypergraph G consisting of a set of vertices V and a set of edges E. Each edge is a nonempty subset of V. The *rank* of a hypergraph G is the maximum size of any edge. In the HypergraphMatching problem the goal is to find a subset $M \subseteq E$ which contains pairwise disjoint edges and has maximum possible size. As opposed to SetCover, this is a maximization problem, and thus we say that the solution M to the HypergraphMatching problem is α-approximate, for $\alpha \in (0, 1]$, when $|M| \geq \alpha \cdot |\mathrm{OPT}|$, where OPT is an optimal solution to the HypergraphMatching problem. The Matching problem is the HypergraphMatching problem in simple graphs, i.e., in graphs with all edges of size 2.

Notation. We use $\tilde{O}(x)$ to hide logarithmic factor in x, i.e., $\tilde{O}(x)$ denotes $O(x \cdot \operatorname{poly}\log x)$. Throughout this paper, we use n to refer to the number of vertices and m to refer to the number of edges of an input graph. When it is stated that a guarantee holds "with high probability", or whp for short, it means that it holds with probability $1 - 1/n^c$, where c is a constant. In our proofs with whp guarantees, c can be made arbitrarily large by paying a constant factor in the round, space, total work, or depth complexity. Hence, we often omit the exact value of c.

[5] Technically, $|x|$ is also the size of the set x. However, in our algorithms, we repeatedly remove some elements from T (together with their incident edges), and so we use $|N(x)|$ to make it clear that we refer to the *current* size of the set.

Probability tools. In our analysis, we extensively apply the following well-known tool from probability.

▶ **Theorem 1** (Chernoff bound)**.** *Let $X_1, \ldots, X_k$ be independent random variables taking values in $[0,1]$. Let $X \overset{\text{def}}{=} \sum_{i=1}^k X_i$ and $\mu \overset{\text{def}}{=} \mathbb{E}[X]$. Then,*

(A) *For any $\delta \in [0,1]$ it holds $\Pr[X \le (1-\delta)\mu] \le \exp\left(-\delta^2\mu/2\right)$.*

(B) *For any $\delta \in [0,1]$ it holds $\Pr[X \ge (1+\delta)\mu] \le \exp\left(-\delta^2\mu/3\right)$.*

Work-Depth Model. We study our algorithms in the shared-memory setting using the concurrent-read concurrent-write (CRCW) parallel random access machine model (PRAM). We state our results in terms of their work and depth. The *work* of a PRAM algorithm is equal to the total number of operations required, and the *depth* is equal to the number of time steps required [24]. Algorithms with work W and depth D can be scheduled to run in $W/P + O(D)$ time [24, 11] on a P processor machine. The main goal in parallel algorithm design is to obtain work-efficient algorithms with low (ideally poly-logarithmic) depth. A *work-efficient* algorithm asymptotically requires the same work as the fastest sequential algorithm. Since the number of processors, P, is still relatively small on modern multicore machines, minimizing W by designing work-efficient algorithms is critical in practice. Our algorithms make use of several PRAM primitives, including parallel prefix sum [24], parallel integer sort [35], and approximate prefix sums [23].

3 Technical Overview

In this section we demonstrate the main ideas behind our results. We start by presenting our sequential algorithms for the SetCover problem. For any set X and probability $p \in [0,1]$ we write Sample(X, p) to denote a procedure that returns a random subsample of X in which each element of X is included independently with probability p. Our algorithms work by repeatedly sampling sets or elements independently using a sequence of probabilities p_i, which is defined as follows for any $\epsilon > 0$.

$$b \overset{\text{def}}{=} \lceil \log(2+2\epsilon)/\epsilon \rceil \tag{1}$$

$$p_i \overset{\text{def}}{=} (1+\epsilon)^{-\lceil i/b \rceil} \qquad \text{for any } i \in \mathbb{N}. \tag{2}$$

Throughout the paper, we use log to denote the natural logarithm function. We can now present our algorithms for SetCover, which are given as Algorithm 1 and Algorithm 2.

Algorithm 1 is a natural parallelization of the sequential f-approximate algorithm. Instead of picking one element at a time, we sample multiple elements at random and add to the solution the sets containing them. The sampling probability is slowly increased in each step (or, more precisely, every $O(1/\epsilon)$ steps). Algorithm 2 in turn parallelizes the $O(\log \Delta)$ approximate algorithm. The outer loop iterates over different set sizes (rounded to the power of $1+\epsilon$) starting from the largest ones. For a fixed set size, the inner loop adds to the solution a uniformly random sample of sets, again slowly increasing the sampling probability.

We start by analyzing Algorithm 1. Clearly, the algorithm runs in $O(\log n)$ iterations. Iteration i samples each element independently with probability p_i and adds all sets covering the sampled elements to the solution. Then, all chosen sets and elements that became covered are removed. Crucially, the sampling probability in the first step is at most $(1+\epsilon)^{-\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil} \le \epsilon/\Delta$, which implies that the expected number of elements sampled within each set is at most ϵ. Moreover, the sampling probability increases very slowly, as it increases by a $1+\epsilon$ factor every b steps.

Algorithm 1 $(1+\epsilon)f$-approximate algorithm for SetCover.

```
1: function SetCover($G, \epsilon$)                                    ▷ $G = (S \cup T, E)$
2:     $C \leftarrow \emptyset$
3:     for $i = b\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil$ down to $0$ do
4:         $D \leftarrow$ Sample($T, p_i$)
5:         $C \leftarrow C \cup N(D)$
6:         Remove from $G$ all sets in $N(D)$ and all elements they cover
7:     return $C$
```

Algorithm 2 $(1+\epsilon)H_\Delta$-approximate algorithm for SetCover.

```
1: function SetCover($G, \epsilon$)                                    ▷ $G = (S \cup T, E)$
2:     $C \leftarrow \emptyset$
3:     for $j = \lfloor \log_{1+\epsilon} \Delta \rfloor$ down to $0$ do
4:         for $i = b\lceil \log_{1+\epsilon}(f/\epsilon) \rceil$ down to $0$ do
5:             $D \leftarrow$ Sample($\{s \in S \mid |N(s)| \geq (1+\epsilon)^j\}, p_i$)
6:             $C \leftarrow C \cup D$
7:             Remove from $G$ all sets in $D$ and all elements they cover
8:     return C
```

Let us now present the main ideas behind the analysis of the approximation ratio of the algorithm. For simplicity of presentation, let us consider the case when each element is contained in exactly two sets (which implies $f = 2$). In other words, we consider the VertexCover problem. Specifically, since the degree of each vertex of T is 2, we can dissolve vertices of T (equivalently, contract each such vertex into its arbitrary neighbor) and obtain a graph $H = (V, E)$ (where $V = S$) on which we would like to solve the VertexCover problem. We note that the solution and analysis of Algorithm 1 for VertexCover generalizes easily to the case of arbitrary f.

If we translate Algorithm 1 to an algorithm running on H, we see that it repeatedly samples a set of *edges* of H, and for each sampled edge e adds both endpoints of e to the solution, and removes both endpoints of e from H together with their incident edges. In order to prove the approximation guarantee, we show the following.

▶ **Lemma 2.** *Let D be the subset of T picked across all iterations of Algorithm 1. For each vertex v, $\mathbb{E}[\deg_D(v))] \leq 1 + O(\epsilon)$.*

Here $\deg_D(v)$ denotes the number of elements of D contained in v. We prove this lemma formally in Section 4 (see Lemma 6).

Notice that when an element $x \in T$ is sampled to D in Algorithm 1, *all* the sets containing x are added to C. So if w sampled elements belong to the same set, then the algorithm could add $\Theta(wf)$ many sets, although only one of the sets suffices to cover all the w sampled elements. Intuitively, Lemma 2 states that the value of w is at most $1 + O(\epsilon)$ in expectation, which we turn into an approximation guarantee in Lemma 7.

To prove Lemma 2, we model the sampling process in the algorithm as follows. Fix a vertex $v \in V$. Let A be the set of edges incident to v in G. Algorithm 1 runs a sequence of $b\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil + 1$ steps, indexed by $b\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil, \ldots, 0$. Note that the step indices are *decreasing*. Moreover, $\Delta \geq |A|$, since Δ is the maximum vertex degree in G. In step i, each element of A is sampled independently with probability p_i. As soon as at least one element of A is sampled, v is added to the cover. When this happens all elements of A are deleted and the random process stops. Moreover, even if no element of A is sampled, due to other random choices of the algorithm some elements of A may get deleted. In particular, when a neighbor w of v is added to the set cover, the edge wv is deleted from A. Hence, it is possible that all elements of A are deleted before any of them is sampled.

To analyze this we introduce the following *set sampling process*, henceforth denoted as SSP, which gives a more abstract version of the above sampling. The analysis of this process forms our core sampling lemma, which will be useful not just for Lemma 2 but to analyze all of our algorithms. Let A be a fixed set and k be an integer. The process proceeds in $k+1$ steps indexed $k, k-1, \ldots, 0$ and constructs a family of sets $A = A_k \supseteq A_{k-1} \supseteq \cdots \supseteq A_1 \supseteq A_0$ as well as a family $R_k, \ldots, R_0$, such that $R_i \subseteq A_i$. In each step i ($k \geq i \geq 0$) we first construct a set A_i. We have that $A_k = A$, and for $i \in [0, k)$ the set $A_i \subseteq A_{i+1}$ is constructed by a (possibly randomized) *adversary*, who is aware of the sets A_j and R_j for $j > i$. In our analysis of SSP, the goal is to argue that certain guarantees hold regardless of what the adversary does. After the adversary constructs A_i, we sample $R_i = \textsc{Sample}(A_i, p_i)$.

We note that we assume that the updates are adversarial to simplify the overall proof. This makes our claims about SSP more robust, and analyzing SSP with an adversary does not introduce significant complications.

For $i \in [0, k]$, we define $n_i \overset{\text{def}}{=} |A_i|$. Whenever we apply SSP we have that $k \geq b\lceil \log_{1+\epsilon}(n_k/\epsilon) \rceil$, and for simplicity we make this assumption part of the construction. Note that this condition simply ensures that the initial sampling probability is at most ϵ/n_k. Finally, we let z be the maximum index such that $R_z \neq \emptyset$. We stress that the SSP steps are indexed in *decreasing order*, and hence z is the index of the *first* step such that R_z is nonempty. If all R_i are empty, we set $z = -1$ and assume $R_{-1} = \emptyset$. We say that z is the step when the SSP stops.

Observe that in order to analyze the properties of the set of sampled edges in Algorithm 1, it suffices to analyze the properties of the set R_z. Our main lemma analyzing SSP is given below. It captures the single property of SSP which suffices to prove the approximation ratio of both Algorithm 1 and Algorithm 2. In particular, it directly implies Lemma 2.

▶ **Lemma 3.** *Consider the SSP using any adversary and $\epsilon > 0$. Then, $\mathbb{E}[|R_z|] \leq 1 + 4\epsilon$.*

Let us now describe the intuition behind the proof of Lemma 3. To simplify presentation, let us assume that the sets $A_0, \ldots, A_k$ are fixed upfront (i.e., before any set R_i is sampled). We show in Observation 11 that if we are interested in analyzing the properties of R_z, this can be assumed without loss of generality. Observe that as long as the process executes steps where $p_i \cdot n_i \leq \epsilon$, the desired property holds. Indeed, with this assumption we have that $\mathbb{E}[|R_i| \mid R_i \neq \emptyset] \leq 1 + \epsilon$. This is because even if one element is sampled, the expected size of the sample among all remaining elements is at most ϵ (for a formal proof, see Claim 16).

In order to complete the proof, we show that reaching a step where $p_i \cdot n_i \gg \epsilon$ is unlikely. Specifically, the value of $p_i \cdot n_i$ can increase very slowly in consecutive steps, as p_i increases only by $(1+\epsilon)$ factor every b steps, and n_i can only decrease. By picking a large enough value of b, we can ensure that the process most likely stops before $p_j \cdot n_j$ becomes large, i.e., the expected value of $p_z \cdot n_z$ is $O(\epsilon)$. Indeed, in each step where $p_i \cdot n_i \geq \epsilon$, the process stops with probability $\Omega(\epsilon)$. Hence, if we repeat such a step roughly $1/\epsilon$ times (which can be achieved by tweaking b), the process will stop with constant probability (independent of ϵ). In the end we fix b, such that the probability of $p_i \cdot n_i$ increasing by a factor of $1+\epsilon$ is at most $1/(2+2\epsilon)$. As a result, thanks to a geometric sum argument, the expected value of $p_z \cdot n_z$ is $O(\epsilon)$, which implies Lemma 3.

Algorithm 3 $(1+\epsilon)f$-approximate algorithm for SetCover.

```
1:  function SetCover($G, \epsilon$)                          ▷ $G = (S \cup T, E)$
2:      $C \leftarrow \emptyset$
3:      $k \leftarrow b\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil$
4:      $B_i \leftarrow \emptyset$ for all $i \in [0, k]$
5:      for each element $t \in T$ do
6:          Sample $X_t \in [0, k]$, where $P(X_t = i) = \tilde{p_i}$ and add $t$ to $B_{X_t}$
7:      for $i = k$ down to $0$ do
8:          $D \leftarrow$ all elements of $B_i$ which are not marked
9:          $C \leftarrow C \cup N(D)$
10:         Remove from $G$ all sets in $N(D)$ and mark all elements they cover
11:     return $C$
```

3.1 Fixing the Random Choices Upfront

In order to obtain efficient implementations of our algorithms, we reformulate them into equivalent versions where the sampling happens upfront. Specifically, consider the main loop of Algorithm 1. Observe that each element is sampled at most once across all iterations, since as soon as an element is sampled it is removed from further consideration. A similar property holds for each set across all iterations of the inner for loop of Algorithm 2. Moreover, in both cases, the probability of being sampled in a given iteration is fixed upfront and independent of the algorithm's actions in prior iterations. It follows easily that we can make these per-element or per-set random choices upfront. Specifically, let $k = b\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil$. Then, Algorithm 1 executes $k+1$ iterations indexed $k, k-1, \ldots, 0$. We can randomly partition the input elements into $k+1$ *buckets* $B_k, \ldots, B_0$ using a properly chosen distribution and then in iteration i consider the elements of B_i which have not been previously removed as the sample to be used in this iteration.

Observe that since $p_0 = 1$ (see Equation (2)), each element that is not removed before the last step is sampled. Specifically, let $\tilde{p_0}, \ldots, \tilde{p_k}$ be a probability distribution such that $\tilde{p_i} = p_i \cdot \prod_{j=i+1}^{k}(1 - p_j)$. Observe that $\tilde{p_i}$ is the probability that an element should be put into bucket B_i.

Algorithm 3 shows a version of Algorithm 1 in which the random choices are made upfront. It should be clear that Algorithms 1 and 3 produce the same output. Moreover, an analogous transformation can be applied to the inner loop of Algorithm 2. The benefit of making the random choices upfront is twofold. In the MPC model, we use the sampling to simulate r iterations of the algorithms in $O(\log r)$ MPC rounds. The efficiency of this simulation crucially relies on the fact that we only need to consider the edges sampled within the phase and we can determine (a superset of) these edges upfront.

In the PRAM model, the upfront sampling allow us to obtain an improved work bound: instead of tossing a coin for each element separately in each iteration, we can bucket the elements initially and then consider each element in exactly one iteration. In order to bucket the elements efficiently we can use the following lemma.

▶ **Lemma 4** ([37]). *Let $r_0, r_1, \ldots, r_k$ be a sequence of nonnegative real numbers which sum up to 1. Let $X \to [0, k]$ be a discrete random variable, such that for each $i \in [k]$, $P(X = i) = r_i$. Then, there exists an algorithm which, after preprocessing in $O(k)$ time, can generate independent samples of X in $O(1)$ time.*

3.2 MPC Algorithms

Simulating Algorithm 3 in the sub-linear MPC model is a relatively straightforward application of the graph exponentiation technique [31, 19, 18, 13, 12]. For simplicity, let us again consider the VertexCover problem. We will show how to simulate $r = O(\sqrt{\log n})$ iterations of the for loop in only $O(\log r) = O(\log\log n)$ MPC rounds. Let us call these r iterations of the algorithm a *phase*. We first observe that to execute a phase we only need to know edges in the buckets corresponding to the iterations within the phase. Let us denote by G_r the graph consisting of all such edges. Moreover, let p be the sampling probability used in the first iteration of the phase. The crucial observation is that the maximum degree in G_r is $2^{\tilde{O}(\sqrt{\log n})}$ with high probability. This can be proven in three steps. First, we show that by the start of the phase the maximum degree in the original graph G drops to $O(1/p \cdot \log n)$ with high probability. Indeed, for any vertex v with a higher degree the algorithm samples an edge incident to v with high probability, which causes v to be removed. Second, we observe that the sampling probability increases to at most $p \cdot 2^{O(\sqrt{\log n})}$ within the phase, and so the expected number of edges incident to any vertex of G_r is at most $O(1/p \cdot \log n) \cdot p \cdot 2^{O(\sqrt{\log n})} = 2^{\tilde{O}(\sqrt{\log n})}$. Third, we apply a Chernoff bound.

At this point, it suffices to observe that running r iterations of the algorithm can be achieved by computing for each vertex v of G_r a subgraph S_v consisting of all vertices at distance $O(r)$ from v and then running the algorithm separately on each S_v. In other words, running r iterations of the algorithm is a $O(r)$ round Local algorithm. Computing S_r can be done using graph exponentiation in $\log r = O(\log\log n)$ MPC rounds using $2^{O(\sqrt{\log n})\cdot r} = n^\alpha$ space per machine and $n^{1+\alpha}$ total space, where $\alpha > 0$ is an arbitrary constant.

The space requirement can also be reduced to $\tilde{O}(m)$. We now sketch the high-level ideas behind this improvement. We leverage the fact that if we sample each edge of an m-edge graph independently with probability p, then only $O(p \cdot m)$ vertices have an incident sampled edge, and we can ignore all the remaining vertices when running our algorithm. Hence, we only need to run the algorithm for $O(p \cdot m)$ vertices and thus have at least $S = m/O(p \cdot m) = \Omega(1/p)$ available space per vertex, even if we assume that the total space is $O(m)$. As argued above, with space per vertex S, we can simulate roughly $\sqrt{\log S}$ steps of the algorithm. In each of these steps, the sampling probability increases by a constant factor, so overall, it increases by a factor of $2^{\Omega(\sqrt{\log S})}$ across the $\sqrt{\log S}$ steps that we simulate. After repeating this simulation $t = \sqrt{\log S} = O(\sqrt{\log \Delta})$ times, the sampling probability increases by a factor of at least $2^{\Omega(t\cdot\sqrt{\log S})} = 2^{\Omega(\log S)} = S^{\Omega(1)}$. Overall, after roughly $O(\sqrt{\log \Delta})$ repetitions the space per vertex reduces from S to $S^{1-\Omega(1)}$. Similarly, the sampling probability increases from p to $p^{1+\Omega(1)}$. Hence, it suffices to repeat this overall process $\log\log \Delta$ times to simulate all $O(\log \Delta)$ steps.

3.3 PRAM algorithms

Algorithm 3 also almost immediately yields a work-efficient algorithm with $O(\log n)$ depth in the CRCW PRAM. Obtaining a work-efficient and low-depth implementation of Algorithm 2 is only a little more involved. One challenge is that the set sizes change as elements get covered. Since we run $O(\log n)$ steps per round, we can afford to exactly compute the sizes at the start of a round, but cannot afford to do so on every step without incurring an additional $O(\log n)$ factor in the depth. We first use the randomness fixing idea described in Section 3.1 to identify the step in the algorithm when a set will be sampled. Then, in every step, for the sets sampled in this step, we approximate the set sizes up to a $(1 + \delta)$

Algorithm 4 Algorithm for HypergraphMatching.

```
1: function HypergraphMatching($G, \epsilon$)                    ▷ $G = (V, E)$
2:     $\Delta \leftarrow$ the maximum degree in $G$
3:     $C \leftarrow \emptyset$
4:     for $i = b\lceil \log_{1+\epsilon}(\Delta/\epsilon) \rceil$ down to 0 do
5:         $D \leftarrow$ Sample($E(G), p_i$)
6:         $C \leftarrow C \cup D$
7:         Remove from $G$ all endpoints of edges in $D$
8:     return edges independent in $C$
```

factor, which can be done deterministically and work-efficiently in $O(\log\log n)$ depth and use these estimates in our implementation of Algorithm 2. The resulting algorithm still obtains a $(1+\epsilon)H_\Delta$-approximation in expectation while deterministically ensuring work efficiency and $O(\log^2 n \log\log n)$ depth.

3.4 HypergraphMatching in MPC

We show that our techniques for solving SetCover can be further applied to solve approximate HypergraphMatching. For the purpose of this high-level overview we consider the special case of approximate Matching in simple graphs, i.e., hypergraphs in which each edge has exactly 2 endpoints. Generalizing our approach to arbitrary hypergraphs does not require any additional ideas. Our algorithm for HypergraphMatching is shown as Algorithm 4, and works similarly to Algorithm 1. Specifically, if we consider the simple graph setting and the VertexCover problem, Algorithm 1 samples a set of edges of the graph and then returns the set of endpoints of these edges as the solution. Algorithm 4 also samples a set of edges, but the difference is in how it computes the final solution. Namely, it returns all sampled edges which are independent, i.e., not adjacent to any other sampled edge. Clearly, the set of edges returned this way forms a valid matching. To argue about its cardinality, we show that the number of edges that are returned is a constant factor of all edges that have been sampled. To this end, we show a second fact about the SSP, which says that any sampled element is *not* sampled by itself with only small constant probability.

▶ **Lemma 5.** *Let $a \in A_k$ and let $\epsilon \le 1/2$. Then $P(|R_z| > 1 \mid a \in R_z) \le 6\epsilon$.*

The high-level idea behind the proof of Lemma 5 is similar to the proof of Lemma 2: in the steps where the expected number of sampled elements is $\le \epsilon$, the property follows in a relatively straightforward way. Moreover, we are unlikely to reach any step where the expected number of sampled elements is considerably larger, and so to complete the proof we also apply a geometric sum-based argument. With the above Lemma, the analysis of Algorithm 4 becomes straightforward and shows that the approximation ratio of the algorithm is $\frac{1-h\cdot 6\epsilon}{h}$ (see Lemma 10), where h is the rank of the hypergraph.

Algorithm 4 can be seen as a simplification of the "warm-up" algorithm of [19], which alternates between sampling edges incident to high-degree vertices and peeling high-degree vertices. Our algorithm simply samples from *all edges* and does not peel vertices. This makes the proof of the approximation ratio trickier since there is less structure to leverage. However, the simplification results in a straightforward application of round compression and enables extending the algorithm to hypergraphs.

$$\begin{array}{lll} \text{minimize} & \sum_{s\in S} x_s & \\ \text{subject to} & \sum_{s\ni t} x_s \geq 1 & \text{for } t \in T \\ & x_s \geq 0 & \text{for } s \in S \end{array} \qquad \begin{array}{lll} \text{maximize} & \sum_{t\in T} y_t & \\ \text{subject to} & \sum_{t\in s} y_t \leq 1 & \text{for } s \in S \\ & y_t \geq 0 & \text{for } t \in T \end{array}$$

Figure 1 LP relaxation of the SetCover LP (left) and its dual (right).

4 Approximation Analysis of the Algorithms

In this section, we analyze the approximation ratio and analysis of our algorithms. We note that the correctness of all algorithms is essentially immediate. Specifically, in Algorithm 1 and Algorithm 2 we only remove an element when it is covered, and in the last iteration of the inner **for** loop in both algorithms (which in Algorithm 1 is the only loop) the sampling probability is 1, so we add all remaining sets (in Algorithm 2, limited to the large enough size) to the solution. Similarly, Algorithm 4 clearly outputs a valid matching, thanks to the final filtering step in the **return** statement.

▶ **Lemma 6.** *Let $\bar{D}$ be the union of all elements picked in all iterations of Algorithm 1. For each set $s \in S$ we have $\mathbb{E}[|s \cap \bar{D}|] \leq 1 + 4\epsilon$.*

Proof. This follows from Lemma 3 applied to the set s. ◀

▶ **Lemma 7.** *Algorithm 1, called with $e' = \epsilon/4$, computes an $(1+\epsilon)f$-approximate solution to SetCover.*

Proof. The key property that we utilize in the analysis is stated in Lemma 6. The proof is a relatively simple generalization of the dual fitting analysis of the standard f-approximate SetCover algorithm. The generalization needs to capture two aspects: the fact that the property stated in Lemma 6 holds only in expectation and allows for a slack of 4ϵ.

We use the relaxation of the SetCover IP and its dual given in Figure 1. Let $\bar{D}$ be the union of all elements picked in all iterations of Algorithm 1. We construct a dual solution that corresponds to $\bar{D}$ as follows. First, for each $t \in \bar{D}$, we set $\bar{y}_t = 1/(1+4\epsilon)$, and for $t \notin \bar{D}$ we set $\bar{y}_t = 0$. Recall that Algorithm 1 returns a solution of size $|C|$. For any run of the algorithm we have $|C| \leq f \sum_{t\in T} \bar{y}_t(1+4\epsilon)$.

We now define a set dual of variables by setting $y_t = \mathbb{E}[\bar{y}_t]$ for each $t \in T$. This set forms a feasible dual solution, since for every $s \in S$ we have

$$\sum_{t\in s} y_t = \sum_{t\in s} \mathbb{E}[\bar{y}_t] = \mathbb{E}[|s \cap \bar{D}|/(1+4\epsilon)] \leq 1,$$

where in the last inequality we used Lemma 6. Moreover, we have

$$\mathbb{E}[|C|] \leq f \cdot \sum_{t\in T} y_t(1+4\epsilon),$$

which implies that the solution's expected size is at most $f(1+4\epsilon)$ times larger than a feasible dual solution. Hence, the lemma follows from weak LP duality. ◀

Now let us consider Algorithm 2.

▶ **Lemma 8.** *Algorithm 2 adds sets to the solution in batches. When a batch of sets D is added to the solution we have that (a) the residual size of each set in D is at most $(1+\epsilon)$ smaller than the maximum residual size of any set at that moment, and (b) for each newly covered element t, the expected number of sets in a batch that cover it is at most $(1+4\epsilon)$.*

Proof. Observe that for $i = 0$ and the current value of j, each of the remaining sets of size $(1+\epsilon)^j$ or more is included in D. By applying this observation inductively, we see that each iteration of the outer loops starts with the maximum set size being less than $(1+\epsilon)^{j+1}$ and results in all sets of size at least $(1+\epsilon)^j$ being either added to the solution or removed from the graph. This implies claim (a). Claim (b) follows directly from Lemma 3. ◀

To bound the approximation guarantee of Algorithm 2, we show the following, which, similarly to the proof of Lemma 7, uses a dual-fitting analysis.

▶ **Lemma 9.** *Any algorithm that computes a valid* SetCover *solution and satisfies the property of Lemma 8, computes an $(1+\epsilon)(1+4\epsilon)H_\Delta$-approximate (in expectation) solution to* SetCover.

▶ **Lemma 10.** *Algorithm 4 ran on a rank h hypergraph in expectation computes a $\frac{1-h\cdot 6\epsilon}{h}$ approximate matching.*

Proof. Let $G = (V, E)$ be input to Algorithm 4, and let C' be the set C after the execution of the for-loop. Observe that $V(C')$ is a vertex cover of G: all the edges not covered by the time we reach $i = 0$ are included in D and, so, in C.

We want also to lower-bound the size of independent edges in C'. Fix an edge e and consider a vertex $v \in e$. Once e is included in C, all the endpoints of e are removed from G. Hence, if e is not independent in C', then it is the case because, in the same iteration, an edge e' adjacent to e is also included in D. To upper-bound the probability of e' and e being included in D, we use Lemma 5. How do we use Lemma 5 in the context of Algorithm 4? For a fixed vertex v, A_i is the set of edges incident to v at the i-th iteration of the for-loop of Algorithm 4. In particular, the set $A_k = A$ defined in Appendix A equals all the edges of the input graph G containing v.

By Lemma 5, the probability of v being incident to more than one sampled edge is at most 6ϵ. Thus, by union bound, e and an edge adjacent to e are included in D with probability at most $h \cdot 6\epsilon$. Therefore, with probability $1 - h \cdot 6\epsilon$ at least, a fixed edge in C' is independent. This implies that in expectation $|C'|(1 - h \cdot 6\epsilon)$ edges in C' are independent and, so, Algorithm 4 outputs a matching that in expectation has size at least $|C'|(1 - h \cdot 6\epsilon)$. Since there is a vertex cover of size $|V(C')| \le h|C'|$ at most, it implies that Algorithm 4 in expectation produces a $\frac{1-h\cdot 6\epsilon}{h}$-approximate maximum matching. ◀

References

1 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *ACM Symposium on Theory of Computing (STOC)*, pages 574–583, 2014. doi:10.1145/2591796.2591805.

2 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1616–1635. SIAM, 2019. doi:10.1137/1.9781611975482.98.

3 MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. Optimal distributed submodular optimization via sketching. In *ACM International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1138–1147, 2018. doi:10.1145/3219819.3220081.

4 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):1–58, 2017. doi:10.1145/3125644.

5 Soheil Behnezhad, Mohammad Taghi Hajiaghayi, and David G Harris. Exponentially faster massively parallel maximal matching. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1637–1649. IEEE, 2019.

6 Ran Ben Basat, Guy Even, Ken-ichi Kawarabayashi, and Gregory Schwartzman. Optimal distributed covering algorithms. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 104–106, 2019. doi:10.1145/3293611.3331577.

7 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. doi:10.1016/S0022-0000(05)80068-6.

8 Guy E Blelloch, Jeremy T Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 89–102, 2020. doi:10.1145/3350755.3400227.

9 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*, pages 23–32, 2011. doi:10.1145/1989493.1989497.

10 Guy E Blelloch, Harsha Vardhan Simhadri, and Kanat Tangwongsan. Parallel and i/o efficient set covering algorithms. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 82–90, 2012. doi:10.1145/2312005.2312024.

11 Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748, 1999. doi:10.1145/324133.324234.

12 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the linear-memory barrier in MPC: Fast MIS on trees with strongly sublinear memory. *Theoretical Computer Science*, 849:22–34, 2021. doi:10.1016/J.TCS.2020.10.007.

13 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\delta + 1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 471–480, 2019. doi:10.1145/3293611.3331607.

14 Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. In *ACM Symposium on Theory of Computing (STOC)*, pages 471–484, 2018.

15 Laxman Dhulipala, Guy Blelloch, and Julian Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 293–304, 2017. doi:10.1145/3087556.3087580.

16 Guy Even, Mohsen Ghaffari, and Moti Medina. Distributed set cover approximation: primal-dual with optimal locality. In *International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

17 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover. In *ACM Symposium on Principles of Distributed Computing*, pages 129–138, 2018. doi:10.1145/3212734.3212743.

18 Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. Improved parallel algorithms for density-based network clustering. In *International Conference on Machine Learning (ICML)*, pages 2201–2210. PMLR, 2019. URL: http://proceedings.mlr.press/v97/ghaffari19a.html.

19 Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1636–1653. SIAM, 2019. doi:10.1137/1.9781611975482.99.

20 Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011. doi:10.1007/978-3-642-25591-5_39.

21 Oussama Hanguir and Clifford Stein. Distributed algorithms for matching in hypergraphs. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 30–46. Springer, 2021.

22 Nicholas JA Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 43–52, 2018.

23 Dorit S Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982. doi:10.1137/0211045.

24 Joseph JáJá. *An Introduction to Parallel Algorithms.* Addison Wesley Longman Publishing Co., Inc., USA, 1992.

25 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948. SIAM, 2010. doi:10.1137/1.9781611973075.76.

26 Richard M Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines, 1988.

27 Samir Khuller, Uzi Vishkin, and Neal Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17(2):280–289, 1994. doi:10.1006/JAGM.1994.1036.

28 Christos Koufogiannakis and Neal E Young. Distributed algorithms for covering, packing and maximum weighted matching. *Distributed Computing*, 24:45–63, 2011. doi:10.1007/S00446-011-0127-7.

29 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

30 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 85–94, 2011. doi:10.1145/1989493.1989505.

31 Christoph Lenzen and Roger Wattenhofer. Brief announcement: Exponential speed-up of local algorithms using non-local communication. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 295–296, 2010. doi:10.1145/1835698.1835772.

32 Paul Liu and Jan Vondrak. Submodular optimization in the mapreduce model. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019.

33 Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996. doi:10.1006/JAGM.1996.0017.

34 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. doi:10.1137/S0097539793260763.

35 Sanguthevar Rajasekaran and John H Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM Journal on Computing*, 18(3):594–607, 1989. doi:10.1137/0218041.

36 Stergios Stergiou and Kostas Tsioutsiouliklis. Set cover at web scale. In *ACM International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1125–1133, 2015. doi:10.1145/2783258.2783315.

37 Alastair J Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 8(10):127–128, 1974.

38 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, USA, 1st edition, 2011.

A Analysis of the Set Sampling Process

In this section we prove two properties of the SSP, which are key in analyzing our algorithms. Recall the set sampling process SSP: initially $A_k = A$, and R_k is obtained by including each element of A_k independently with probability p_k. Then, for every i from $k-1$ down to 0, an adversary chooses $A_i \subseteq A_{i+1}$ (possibly with randomization) and then R_i is obtained by including every element of A_i independently with probability p_i.

Note two things about this process. First, the adversary can be randomized. Second, the adversary can be *adaptive*: its choice of A_i can depend on A_j and R_j for $j > i$. Recall that after running this process, z is the largest index such that R_z is nonempty. Our goal in this section is to prove the following lemma:

▶ **Lemma 3.** *Consider the SSP using any adversary and $\epsilon > 0$. Then, $\mathbb{E}[|R_z|] \leq 1 + 4\epsilon$.*

To get some intuition for why Lemma 3 might be true, observe that the sampling probability p_i increases very slowly; specifically, it increases by a $(1+\epsilon)$ factor every b steps. So the algorithm gets many chances at each (low) probability to obtain a non-empty R_i, and so it is not very likely to get more than 1 element in R_z.

To prove this lemma, we start with a simple but extremely useful observation: we may assume without loss of generality that the adversary is *nonadaptive*: its choice of A_i does not depend on R_j for $j > i$ (it can still depend on A_j for $j > i$). In other words, a nonadaptive adversary must pick the entire sequence of A_i's before seeing the results of any of the R_i's. Moreover, we may assume that the adversary is *deterministic*.

▶ **Observation 11.** *Without loss of generality, the adversary is nonadaptive and deterministic, i.e., it is a single fixed sequence $A_k, A_{k-1}, \dots, A_0$.*

Proof. We begin by showing that the adversary is nonadaptive without loss of generality. To see this, suppose there is some adaptive adversary P. Then let P' be the nonadaptive adversary obtained by simply running P under the assumption that every $R_i = \emptyset$. Clearly, this gives a (possibly randomized) sequence $A_k, A_{k-1}, \dots, A_0$ without needing to see the R_i's, and so is nonadaptive. Clearly, P and P' behave identically until $z-1$, i.e., until just *after* the first time that some R_i is nonempty (since P' is just P under the assumption that all R_i's are empty). But indices $z-1$ down to 0 make no difference in Lemma 3! Hence if Lemma 3 holds for nonadaptive adversaries, it also holds for adaptive adversaries.

So we assume that the adversary is nonadaptive, i.e., the adversarial choice is simply a distribution over sequences $A_k, A_{k-1}, \dots, A_0$. This means that the expectation in Lemma 3 is taken over both the adversary's random choices and the randomness from sampling the R_i's once the A_i's are fixed. These are intermixed for an adaptive adversary but for a nonadaptive adversary, which we may assume WLOG, we can separate these out by first choosing the random A_i's and then subsampling to get the R_i's. So we want to prove that

$$\mathbb{E}_{A_k,\dots,A_0}\left[\mathbb{E}_{R_k,\dots R_0}[|R_z|]\right] \leq 1 + 4\epsilon.$$

Suppose we could prove Lemma 3 for a deterministic nonadaptive adversary, i.e., for a fixed $A_k, A_{k-1}, \dots, A_0$. In other words, suppose that $\mathbb{E}_{R_k,\dots,R_0}|R_z| \leq 1 + 4\epsilon$ for all sequences $A_k, A_{k-1}, \dots, A_0$. Then clearly

$$\mathbb{E}_{A_k,\dots,A_0}\left[\mathbb{E}_{R_k,\dots R_0}[|R_z|]\right] \leq \mathbb{E}_{A_k,\dots,A_0}[1 + 4\epsilon] = 1 + 4\epsilon.$$

Thus if we can prove Lemma 3 against a nonadaptive deterministic adversary, we have proved Lemma 3 against an adaptive and randomized adversary, as desired. ◀

So from now on, we may assume that the family $A_k, A_{k-1}, \ldots, A_0$ is **fixed**. Note that in this setting, the sets R_i are independent of each other; this holds as the sets A_i are fixed, and the randomness used to obtain R_i is independent of the randomness used to sample other R_j sets. Before proving Lemma 3, we first show several auxiliary observations (all of which are in the setting where $A_k, A_{k-1}, \ldots, A_0$ are fixed).

Our first observation is that for the first b rounds of the SSP, the expected number of sampled elements is small.

▶ **Observation 12.** *Assume that $k \geq b\lceil \log_{1+\epsilon}(n_k/\epsilon)\rceil$. Then, for each $j \in (k-b, k]$, $p_j \cdot n_j \leq \epsilon$.*

Proof. We have

$$\lceil j/b \rceil = \lceil k/b \rceil \geq \log_{1+\epsilon}(n_k/\epsilon),$$

which gives

$$p_j \cdot n_j = (1+\epsilon)^{-\lceil j/b \rceil} \cdot n_j \leq (1+\epsilon)^{-\log_{1+\epsilon}(n_k/\epsilon)} \cdot n_k = \epsilon. \qquad ◀$$

We can also show that probabilities and the expected number of sampled elements do not increase much in any consecutive b steps.

▶ **Observation 13.** *For each $i \in [0, k)$, and any $j \in [i+1, i+b]$, $p_i \leq (1+\epsilon)p_j$ and $p_i \cdot n_i \leq (1+\epsilon)p_j \cdot n_j$.*

Proof. This first claim follows from the definition of p_i. The second claim additionally uses the fact that $n_0, \ldots, n_k$ is a non-decreasing sequence. ◀

This observation now allows us to show that if we have a relatively large expected number of elements in R_i, then the probability that we have not yet sampled any elements in R_j for $j > i$ is notably smaller than the probability that we haven't sampled any elements in R_j for $j > i + b$.

▶ **Lemma 14.** *Assume that $p_i \cdot n_i \geq \epsilon(1+\epsilon)$ for some $i \in [0, k-b]$. Then, $P(z \leq i) \leq P(z \leq i+b)/(2+2\epsilon)$.*

Proof. Denote by E_j the event that $R_j = \emptyset$. Recall that z is the maximum index such that $R_z \neq \emptyset$. Observe that the event that $z \leq x$ is equivalent to $\bigcap_{j>x} E_j$ and the individual events E_j are independent. Hence $P(z \leq i) = P(z \leq i+b) \cdot \prod_{j=i+1}^{i+b} P(E_j)$. To complete the proof we will show that $\prod_{j=i+1}^{i+b} P(E_j) \leq 1/(2+2\epsilon)$.

By Observation 13, we have that for $j \in [i+1, \ldots, b]$, $(1+\epsilon)p_j \cdot n_j \geq p_i \cdot n_i \geq \epsilon(1+\epsilon)$, which implies $p_j \cdot n_j \geq \epsilon$. Hence,

$$P(E_j) = (1-p_j)^{n_j} \leq e^{-p_j \cdot n_j} \leq e^{-\epsilon}.$$

By using the above, we get

$$\prod_{j=i+1}^{i+b} P(E_j) \leq e^{-b\cdot\epsilon} = e^{-\lceil \log(2(1+\epsilon))/\epsilon \rceil \epsilon} \leq e^{-\log(2(1+\epsilon))} = \frac{1}{2+2\epsilon}$$

which finishes the proof. ◀

This now allows us to give an absolute bound on the probability that we have not sampled any elements before we sample R_j, assuming that we have a pretty high probability of sampling an element in R_j.

▶ **Lemma 15.** *Let $j \in [0,k]$ be such that $p_j \cdot n_j \geq \epsilon(1+\epsilon)^c$ for a nonnegative integer c. Then $P(z \leq j) \leq (2+2\epsilon)^{-c}$.*

Proof. We prove the claim using induction on c. For $c = 0$, $(2+2\epsilon)^{-c}$ is 1, and the claim is trivially true.

Now, fix $c \geq 1$. By Observation 12, we have that $j \leq k - b$, and so we apply Lemma 14 to obtain $P(z \leq j) \leq P(z \leq j+b)/(2+2\epsilon)$. Hence, to complete the proof, it suffices to show $P(z \leq j+b) \leq (2+2\epsilon)^{-c+1}$.

We achieve that by applying the induction hypothesis to $j' = j+b$. Indeed, by Observation 13, $p_{j'} \cdot n_{j'} \geq \epsilon(1+\epsilon)^{c-1}$, and so the assumptions of the inductive hypothesis hold. As a result, we obtain $P(z \leq j+b) = P(z \leq j') \leq (2+2\epsilon)^{-c+1}$, as required. ◀

Before finally proving Lemma 3, we first show two more useful claims.

▷ Claim 16. For each $i \in [0,k]$, it holds that $\mathbb{E}[|R_i| \mid R_i \neq \emptyset] \leq 1 + p_i \cdot n_i$.

Proof. We have

$$\mathbb{E}[|R_i| \mid R_i \neq \emptyset] = \frac{\mathbb{E}[|R_i|]}{P(R_i \neq \emptyset)} = \frac{p_i \cdot n_i}{1-(1-p_i)^{n_i}} \leq \frac{p_i \cdot n_i}{1 - \frac{1}{e^{p_i \cdot n_i}}} \leq \frac{p_i \cdot n_i}{1 - \frac{1}{1+p_i \cdot n_i}} = \frac{p_i \cdot n_i}{\frac{p_i \cdot n_i}{1+p_i \cdot n_i}} = 1 + p_i \cdot n_i.$$

Note that in the first inequality we used the fact that $1 - p_i \leq e^{-p_i}$, while in the second we used $e^{p_i \cdot n_i} \geq 1 + p_i \cdot n_i$. ◁

▷ Claim 17. Recall that z is the maximum index such that $R_z \neq \emptyset$. It holds that $\mathbb{E}[|R_z|] \leq 1 + \mathbb{E}[p_z \cdot n_z]$.

Proof. Denote by X_i the event that $R_j = \emptyset$ *for all* $j \in [i+1,k]$. Note that $z = i$ is the intersection of events $R_i \neq \emptyset$ and X_i.

$$\begin{aligned}
\mathbb{E}[|R_z|] &= \sum_{i=0}^{k} \mathbb{E}[|R_i| \mid z = i] \cdot P(z=i) \\
&= \sum_{i=0}^{k} \mathbb{E}[|R_i| \mid R_i \neq \emptyset \cap X_i] \cdot P(z=i) = \sum_{i=0}^{k} \mathbb{E}[|R_i| \mid R_i \neq \emptyset] \cdot P(z=i) \\
&\leq \sum_{i=0}^{k} (1 + p_i \cdot n_i) \cdot P(z=i) = \sum_{i=0}^{k} P(z=i) + \sum_{i=0}^{k} p_i \cdot n_i \cdot P(z=i) = 1 + \mathbb{E}[p_z \cdot n_z],
\end{aligned}$$

where the final inequality is from Claim 16. We used the fact that $\mathbb{E}[|R_i| \mid R_i \neq \emptyset \cap X_i] = \mathbb{E}[|R_i| \mid R_i \neq \emptyset]$, which follows from the fact that R_i is independent from X_i. ◁

We are now ready to prove Lemma 3.

Proof of Lemma 3. We know from Observation 11 that without loss of generality, the sequence $A_k, A_{k-1}, \ldots, A_0$ is fixed. By Claim 17 it suffices to show that $\mathbb{E}[p_z \cdot n_z] \leq 4\epsilon$. Let us define $X := p_z \cdot n_z$ to shorten notation.

$$\begin{aligned}
\mathbb{E}[X] &\leq P(X < \epsilon) \cdot \epsilon + \sum_{c=0}^{\infty} P\left(X \in \left[\epsilon(1+\epsilon)^c, \epsilon(1+\epsilon)^{c+1}\right)\right) \cdot \epsilon(1+\epsilon)^{c+1} \\
&\leq \epsilon + \sum_{c=0}^{\infty} P\left(X \geq \epsilon(1+\epsilon)^c\right) \cdot \epsilon(1+\epsilon)^{c+1} \\
&\leq \epsilon + \sum_{c=0}^{\infty} 2^{-c}(1+\epsilon)^{-c} \cdot \epsilon(1+\epsilon)^{c+1} \\
&\leq \epsilon(1 + 2(1+\epsilon)) \leq 4\epsilon.
\end{aligned}$$

Note that we used the bound on $P(X \geq \epsilon(1+\epsilon)^c) \leq (2+2\epsilon)^{-c}$ which follows directly from Lemma 15. ◀

A.1 Probability of the Sampled Element Being Not Unique

We use the following lemma to analyze our HypergraphMatching algorithm. Specifically, it upper bounds the probability that an edge is sampled in Algorithm 4, but *not* included in the final matching. In the proof of Lemma 10, we specify how to map our HypergraphMatching algorithm to the setup in this section.

▶ **Lemma 5.** *Let $a \in A_k$ and let $\epsilon \leq 1/2$. Then $P(|R_z| > 1 \mid a \in R_z) \leq 6\epsilon$.*

Proof. As for the previous proof in this section, first assume that the sets $A_0, \dots, A_k$ are fixed.

Our goal is to upper bound

$$P(|R_z| > 1 \mid a \in R_z) = \frac{P(|R_z| > 1 \cap a \in R_z)}{P(a \in R_z)} \tag{3}$$

Let $m_a = \min\{i \in [0,k] \mid a \in A_i\}$ be the index of the last step before a is removed from the sets $A_0, \dots, A_k$. We obtain:

$$\begin{aligned} P(|R_z| > 1 \cap a \in R_z) &= \sum_{i=m_a}^{k} P(z = i \cap |R_i| > 1 \cap a \in R_i) \\ &= \sum_{i=m_a}^{k} P(z \leq i \cap |R_i| > 1 \cap a \in R_i) && \text{since } a \in R_i \text{ implies } z \geq i \\ &= \sum_{i=m_a}^{k} P(z \leq i) P(|R_i| > 1 \cap a \in R_i) && \substack{z \leq i \text{ is equivalent to } R_j = \emptyset \text{ for all } j > i \\ \text{these events are independent of } R_i} \end{aligned}$$

Observe that the event $|R_i| > 1 \cap a \in R_i$ happens when a is sampled and at least one out of the remaining $n_i - 1$ elements of A_i are sampled. Hence,

$$P(|R_i| > 1 \cap a \in R_i) = p_i \cdot (1 - (1-p_i)^{n_i - 1}) \leq p_i \cdot (1 - (1 - p_i \cdot (n_i - 1))) \leq p_i^2 \cdot n_i,$$

and so we finally obtain $P(|R_z| > 1 \cap a \in R_z) \leq \sum_{i=m_a}^{k} P(z \leq i) p_i^2 \cdot n_i$. For the first inequality above, we used Bernoulli's inequality which states that $(1+x)^r \geq 1 + rx$ for every integer $r \geq 1$ and a real number $x \geq -1$. Analogous reasoning allows us to show a similar identity for the denominator:

$$P(a \in R_z) = \sum_{i=m_a}^{k} P(z \leq i) P(a \in R_i) = \sum_{i=m_a}^{k} P(z \leq i) p_i$$

Hence we can upper bound Equation (3) as follows

$$P(|R_z| > 1 \mid a \in R_z) \leq \frac{\sum_{i=m_a}^{k} P(z \leq i) P(a \in A_i) p_i^2 \cdot n_i}{\sum_{i=m_a}^{k} P(z \leq i) P(a \in A_i) p_i}. \tag{4}$$

▷ Claim 18. Let $I = \{i \in [m_a, k] \mid p_i \cdot n_i < \epsilon(1+\epsilon)\}$ be a set of indices. Then $\sum_{i=m_a}^{k} P(z \leq i) p_i^2 \cdot n_i \leq 2/(1-\epsilon) \sum_{i \in I} P(z \leq i) p_i^2 \cdot n_i$

Proof. Consider the sum

$$\sum_{i=m_a}^{k} P(z \leq i)p_i^2 \cdot n_i.$$

We are going to charge the summands indexed by $[m_a, k] \setminus I$ to the summands indexed by I. Formally, the charging is defined by a function $f : [m_a, k] \to [m_a, k]$. We define $f(i)$ to be the smallest index $j \in \{i, i+b, i+2b, \dots\}$ such that $j \in I$, that is $p_j \cdot n_j < \epsilon(1+\epsilon)$. We note that $f(i)$ is well-defined since by Observation 12 we have that $(k-b, k] \subseteq I$.

Now we charge each summand i to $f(i) \in I$ and show that the total charge of each summand in I increases by at most a constant factor. Let us now fix any $j \in I$ and consider the sum $\sum_{i \in f^{-1}(j)} P(z \leq i)p_i^2 \cdot n_i$. Let $h := |f^{-1}(j)|$. Then $f^{-1}(j) = \{j, j-b, \dots, j-(h-1)b\}$. We now show that the summands in the considered sum are geometrically decreasing (if we consider the indices in decreasing order). Indeed, consider $x \in f^{-1}(j) \setminus \{j\}$. We are now going to use the following facts.

- By Lemma 14 we have $P(z \leq x) \leq P(z \leq x+b)/(2+2\epsilon)$.
- By Observation 13, $p_x \cdot n_x \leq (1+\epsilon)p_{x+b} \cdot n_{x+b}$.
- By Observation 13, $p_x \leq (1+\epsilon)p_{x+b}$.

These three facts together imply that for any $x \in f^{-1}(j) \setminus \{j\}$

$$P(z \leq x)p_x^2 \cdot n_x \leq (1+\epsilon)/2 \cdot P(z \leq x+b)p_{x+b}^2 \cdot n_{x+b}.$$

Hence, the summands in $f^{-1}(j)$ can be arranged into a sequence in which the largest element is the summand corresponding to j, and each subsequent summand is at least a factor of $(1+\epsilon)/2$ smaller. As a result, the total charge of the summand j is $1/(1-(1+\epsilon)/2) = 2/(1-\epsilon)$. ◁

Using the above claim, we upper bound Equation (4).

$$\frac{\sum_{i=m_a}^{k} P(z \leq i)p_i^2 \cdot n_i}{\sum_{i=m_a}^{k} P(z \leq i)p_i} \leq \frac{2 \cdot \sum_{i \in I} P(z \leq i)p_i^2 \cdot n_i}{(1-\epsilon) \cdot \sum_{i \in I} P(z \leq i)p_i} < \frac{2 \cdot \epsilon(1+\epsilon) \cdot \sum_{i \in I} P(z \leq i)p_i}{(1-\epsilon) \cdot \sum_{i \in I} P(z \leq i)p_i} \leq 6\epsilon.$$

The proofs are stated while assuming that the sets $A_0, \dots, A_k$ are fixed. As given by Observation 11, this assumption can be made without loss of generality. ◀