Work-Efficient Parallel Derandomization II: Optimal Concentrations via Bootstrapping

Work-Efficient Parallel Derandomization II: Optimal Concentrations via Bootstrapping

Mohsen Ghaffari
MIT
Cambridge, MA, USA
ghaffari@mit.edu

Christoph Grunau
ETH Zurich
Zurich, Switzerland
cgrunau@inf.ethz.ch

ABSTRACT

In this paper, we present an efficient parallel derandomization method for randomized algorithms that rely on concentrations such as the Chernoff bound. This settles a classic problem in parallel derandomization, which dates back to the 1980s.

Concretely, consider the *set balancing* problem where m sets of size at most s are given in a ground set of size n, and we should partition the ground set into two parts such that each set is split evenly up to a small additive (discrepancy) bound. A random partition achieves a discrepancy of $O(\sqrt{s \log m})$ in each set, by Chernoff bound. We give a deterministic parallel algorithm that matches this bound, using near-linear work $\tilde{O}(m + n + \sum_{i=1}^{m} |S_i|)$ and polylogarithmic depth $\text{poly}(\log(mn))$. The previous results were weaker in discrepancy and/or work bounds: Motwani, Naor, and Naor [FOCS'89] and Berger and Rompel [FOCS'89] achieve discrepancy $s^{\varepsilon} \cdot O(\sqrt{s \log m})$ with work $\tilde{O}(m+n+\sum_{i=1}^{m} |S_i|) \cdot m^{\Theta(1/\varepsilon)}$ and polylogarithmic depth; the discrepancy was optimized to $O(\sqrt{s \log m})$ in later work, e.g. by Harris [Algorithmica'19], but the work bound remained prohibitively high at $\tilde{O}(m^4 n^3)$. Notice that these would require a large polynomial number of processors to even match the near-linear runtime of the sequential algorithm. Ghaffari, Grunau, and Rozhon [FOCS'23] achieve discrepancy $s/\text{poly}(\log(nm)) + O(\sqrt{s \log m})$ with near-linear work and polylogarithmic-depth. Notice that this discrepancy is nearly quadratically larger than the desired bound and barely sublinear with respect to the trivial bound of s.

Our method is different from prior work. It can be viewed as a novel bootstrapping mechanism that uses crude partitioning algorithms as a subroutine and sharpens their discrepancy to the optimal bound. In particular, we solve the problem recursively, by using the crude partition in each iteration to split the variables into many smaller parts, and then we find a constraint for the variables in each part such that we reduce the overall number of variables in the problem. The scheme relies crucially on an interesting application of the multiplicative weights update method to control the variance losses in each iteration.

Our result applies to the much more general *lattice approximation* problem, thus providing an efficient parallel derandomization of the randomized rounding scheme for linear programs.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649668

CCS CONCEPTS

• **Theory of computation → Dynamic graph algorithms**; **Pseudorandomness and derandomization**; **Parallel algorithms**.

KEYWORDS

Parallel Algorithms, Derandomization

ACM Reference Format:
Mohsen Ghaffari and Christoph Grunau. 2024. Work-Efficient Parallel Derandomization II: Optimal Concentrations via Bootstrapping. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649668

1 INTRODUCTION

This paper presents an efficient parallel method for derandomizing randomized algorithms that rely on concentrations of measure such as Chernoff and Hoeffding bounds. This settles one of the classic and central questions in parallel derandomization [5, 23].

Let us start with a concrete and simple-to-state problem, known as the *set balancing* or *set discrepancy* problem, which has been used as the primary target in this line of work[1]. By known reductions [23], our result applies to much more general problems such as *lattice approximation* [25].

Set balancing: Consider m subsets $S_1, S_2, \ldots, S_m \subseteq [n]$ in a ground set of n elements, where $|S_i| \leq s$ for each $i \in [m]$. We want to split the ground set into two parts such that each of the subsets is split as evenly as possible. Concretely, we want a vector $\chi \in \{-1, 1\}^n$ that minimizes the *set system's discrepancy* defined as $\max_{i \in [m]} disc(S_i)$ where the discrepancy of set S_i is defined as $disc(S_i) := |\sum_{j \in S_i} \chi_j|$. For a random $\chi \in \{-1, 1\}^n$, the Chernoff bound implies a discrepancy of $O(\sqrt{s \log m})$, via a union bound over all m sets.[2]

Results of Spencer [28] and Raghavan [25] present deterministic algorithms that achieve the same bound, via the method of conditional expectation. But these algorithms are inherently sequential.

[1]This problem and similar others arise naturally and frequently in algorithm design. Here is a simple example: suppose we want to partition the edges of a graph $G = (V, E)$ into k parts, such that each node has almost equal number of edges in different parts. Such a partition is useful, e.g., when computing an edge-coloring as different parts can be colored independently.

[2]A beautiful result of Spencer [29] gives a discrepancy bound of $O(\sqrt{n(1 + \log(m/n))})$, and algorithmic variants were provided in recent breakthroughs [4, 20]. However, our focus here is on bounds that have $\sqrt{s}$ as the leading factor, instead of $\sqrt{n}$, because this is frequently needed in algorithmics, e.g., in generalization to lattice approximation. Moreover, the difference between the second factors $\sqrt{\log(2m/n)}$ and $\sqrt{\log m}$ is considerable only in the special case where $m \ll n^{1.001}$.

Our objective is to achieve a similar result via a deterministic parallel algorithm. We next review the state of the art, after a quick recap on the parallel terminology.

Parallel model and terminology: depth, work, and work-efficiency: We follow the standard *work-depth* model [6, 14], where the algorithm runs on p processors with access to a shared memory. In any algorithm $\mathcal{A}$, its depth $D(\mathcal{A})$ is the longest chain of computational steps in $\mathcal{A}$ each of which depends on the previous ones. In other words, this is the time that it would take the algorithm to run even if we were given an infinite number of processors. The work $W(\mathcal{A})$ is the total number of the computational steps in $\mathcal{A}$. Given p processors, the algorithm clearly needs $\max\{D(\mathcal{A}), W(\mathcal{A})/p\}$ time. By Brent's principle [8], we can run the algorithm in $D(\mathcal{A}) + W(\mathcal{A})/p$ time using p processors. The objective in parallel computations is to devise algorithms that run faster than their sequential counterpart, ideally with a speed-up proportional to the number of processors p. In particular, this requires the parallel algorithm to have a work bound (asymptotically) equal to the best known sequential algorithm, in which case we call the algorithm *work-efficient*. There have been a number of exciting recent work on achieving work-efficient parallel algorithms (or *nearly work-efficient* algorithms where the work bound is relaxed by a polylogarithmic factor) for various problems; see, e.g., [1, 2, 7, 9–12, 15, 19, 26, 27].

1.1 State of the Art

Let us note that the sequential deterministic algorithms of [25, 28] achieve the desired discrepancy bound of $O(\sqrt{s\log m})$ in $\tilde{O}(n+m+\sum_{i=1}^{m}|S_i|)$ time, which is near-linear in the input size. Thus, the ideal parallel algorithm is one with the same near-linear work bound and polylogarithmic depth, achieving the same discrepancy. The state-of-the-art deterministic parallel algorithms achieve weaker results:

(1) Motwani, Naor, and Naor [23], and Berger and Rompel [5], presented parallel deterministic algorithms with a depth of $\text{poly}(\log(mn))$ that achieve discrepancy $s^{\varepsilon}\cdot O(\sqrt{s\log m})$, for $\varepsilon\in(0,0.5]$, using $\tilde{O}(n+m+\sum_{i=1}^{m}|S_i|)\cdot m^{\Theta(1/\varepsilon)}$ work. Motwani, Naor, and Naor [23] also present an improvement, which limits the work to a bound of roughly $\tilde{O}(n+m+\sum_{i=1}^{m}|S_i|)\cdot m^4$, though at the expense of increasing the depth to $\log^{\Theta(1/\varepsilon+1)}(mn)$. Via a parallel randomized automata fooling method of Karger and Koller [16], Mahajan, Ramos, and Subrahmanyam [22] optimized the discrepancy bound to $O(\sqrt{s\log m})$, using a high work bound of $\tilde{O}(m^{10}n^7)$, which was later improved by Harris [13] to $\tilde{O}(m^4n^3)$. All these algorithms are quite far from work efficiency, and require a large polynomial number of processors to even match the speed of sequential algorithms—a prohibitively high requirement.

(2) Recently, Ghaffari, Grunau, and Rozhon [12] gave a parallel deterministic algorithm using $\tilde{O}(n+m+\sum_{i=1}^{m}|S_i|)$ work and $\text{poly}(\log(mn))$ depth that achieves a discrepancy of $s/\text{poly}(\log(mn))+O(\sqrt{s\log m})$. That is they split the ground set into two parts such that for each of the m subsets of size at most s, the intersection with each part has size at most $s(1/2+1/\text{poly}(\log(mn)))$, for $s=\Omega(\text{poly}(\log(mn)))$. However, notice that the achieved discrepancy is almost quadratically higher than the desired bound.

1.2 Our Results

In this paper, we present the first work-efficient deterministic parallel algorithm that achieves the optimal discrepancy in polylogarithmic depth, thus essentially settling the above question.

Theorem 1.1. *Consider $m\geq 2$ subsets $S_1,S_2,\ldots,S_m\subseteq[n]$ and suppose $|S_i|\leq s$ for each $i\in[m]$. There is a deterministic parallel algorithm, with $\tilde{O}(n+m+\sum_{i=1}^{m}|S_i|)$ work and $\text{poly}(\log(mn))$ depth, that computes a vector $\chi\in\{-1,1\}^n$ such that, for each $i\in[m]$, we have $disc(S_i)=|\sum_{j\in S_i}\chi_j|=O(\sqrt{s\log m})$.*

Weighted set balancing: Our method, with some extra work, applies to the more general weighted discrepancy problem defined as follows: Consider an $m\times n$ matrix A, and m subsets $S_1,S_2,\ldots,S_m\subseteq[n]$. We want a vector $\chi\in\{-1,1\}^n$ that gives a small weighted discrepancy $disc(i):=|\sum_{j=1}^{n}a_{ij}\chi_j|$ for each set $i\in[m]$. For a random χ, Hoeffding's bound implies $disc(i)=O(\sqrt{\sum_{j=1}^{n}a_{ij}^2\cdot\log m})$, for each set S_i, with high probability. In the full version of this paper, we give a work-efficient deterministic parallel algorithm that gets the same guarantee [3].

Theorem 1.2. *Let $n,m\in\mathbb{N}$ with $m\geq 2$, and $A\in\mathbb{R}^{m\times n}$. There is a deterministic parallel algorithm algorithm with work $\tilde{O}(nnz(A)+n+m)$ and depth $\text{poly}(\log(mn))$ that computes a vector $\chi\in\{-1,1\}^n$ such that, for every $i\in[m]$, it holds that $disc(i)=|\sum_{j=1}^{n}a_{ij}\chi_j|=O(\sqrt{\sum_{j=1}^{n}a_{ij}^2\cdot\log m})$.*

As a very special case, this implies in the unweighted case that each set S_i has discrepancy $O(\sqrt{|S_i|\log m})$. With this weighted set balancing algorithm, via a reduction of [23], our result generalizes much further, to a problem known as *lattice approximation* [23, 25]. Informally, this is the problem of finding an integral point that approximates a fractional solution, for a number of given linear constraints.

Lattice approximation: Suppose we are given an $m\times n$ matrix A, with each entry $a_{ij}\in[0,1]$, as well as a vector $\mathbf{p}\in[0,1]^n$. The *lattice approximation* problem asks for a vector $\mathbf{q}\in\{0,1\}^n$ with a small bound on $|\sum_{j=1}^{n}a_{ij}q_j-\sum_{j=1}^{n}a_{ij}p_j|$ for each $i\in[m]$.

Notice that if we set $\mathbf{q}$ randomly where $Pr[q_j=1]=p_j$ for each $j\in[n]$, by Chernoff bound we get a solution such that, with high probability, for each $i\in[m]$, we have $|\sum_{j=1}^{n}a_{ij}q_j-\sum_{j=1}^{n}a_{ij}p_j|\leq O(\sqrt{\mu_i\log m}+\log m)$, where $\mu_i=\sum_{j=1}^{n}a_{ij}p_j$. This is what Raghavan called *randomized rounding for linear programs* [25]. Motwani, Naor, and Naor [23] provided a parallel algorithm that achieves discrepancy $O(\mu^{\varepsilon}\cdot\sqrt{\mu_i\log m}+\log^{1/(1-2\varepsilon)}m)$ for each $i\in[m]$, using a large polynomial work, with $\Theta(1/\varepsilon)$ in the exponent of the polynomial. The discrepancy was improved later to the ideal bound [13, 22], but using work bounds that are large polynomials and are thus far from work efficiency. The method of Ghaffari, Grunau, and Rozhon [12] gives a work-efficient algorithm, but achieving only

[3]The full version is accessible at https://arxiv.org/abs/2311.13771.

a slightly sublinear discrepancy of $\mu_i/\text{poly}(\log(mn))$ when the expectation μ_i is lower bounded by a sufficiently high $\text{poly}(\log(mn))$. Our result stated below provides an efficient deterministic parallel algorithm that achieves the same result as the Chernoff bound. It thus can be viewed as an efficient parallel derandomization of randomized rounding for linear programs.

Theorem 1.3. *Suppose we are given an $m \times n$ matrix A, with each entry $a_{ij} \in [0,1]$, as well as a vector $\mathbf{p} \in [0,1]^n$. There is a deterministic parallel algorithm that computes a vector $\mathbf{q} \in \{0,1\}^n$ such that, for each $i \in [m]$, we have $|\sum_{j=1}^{n} a_{ij}q_j - \sum_{j=1}^{n} a_{ij}p_j| \leq O(\sqrt{\mu_i \log m} + \log m)$, where $\mu_i = \sum_{j=1}^{n} a_{ij}p_j$. The algorithm has $\text{poly}(\log(mn))$ depth and $\tilde{O}(n + m + nnz(A))$ work, where $nnz(A)$ denotes the number of nonzero entries in the matrix A.*

This result follows from Theorem 1.2 along with a reduction of Motwani, Naor, Naor detailed in their journal version [24, Sections 8]. We provide a proof sketch in the full version of this paper.

Example application—edge coloring: By Vizing's theorem, any graph with maximum degree Δ admits an edge coloring with at most $\Delta + 1$ colors. It is not known how to achieve this result using a $\text{poly}(\log n)$ depth parallel algorithm (in general graphs; the bipartite case is easy and known even with Δ colors [18].) However there are algorithms that come close to this bound, and our focus is on deterministic parallel algorithms. Motwani, Naor, and Naor [23] presented a $\text{poly}(\log n)$-depth deterministic parallel algorithm with $\Delta + \Delta^{\varepsilon} \cdot O(\sqrt{\Delta \log n})$ colors, and using work $m^{\Theta(1/\varepsilon)}$. This was essentially a direct application of the set balancing problem, by following the edge coloring framework of Karloff and Shmoys [17]. The improvements of Mahajan et al [22] and Harris [13] improved the number of colors to $\Delta + O(\sqrt{\Delta \log n})$, but using prohibitively large polynomial work bounds. Plugging our set balancing algorithm instead, we get a deterministic parallel edge-coloring algorithm with the number of colors improved to $\Delta + O(\sqrt{\Delta \log n})$, with near-linear work. A proof sketch is presented in the full version of this paper.

Corollary 1.4. *There is a deterministic parallel algorithm that, given an undirected graph $G = (V, E)$ with maximum degree Δ, computes an edge-coloring of it with $\Delta + O(\sqrt{\Delta \log n})$ colors, using $\text{poly}(\log n)$ depth and $\tilde{O}(m)$ work, where $n = |V|$ and $m = |E|$.*

1.3 Method Overview

Our method is different than those of the prior work [5, 12, 13, 22, 23]. In short, the method of [5, 23] works via an efficient binary search in a k-wise independent space for $k = \log m/(\varepsilon \log s)$, trying to find a point in this space that satisfies all the discrepancy constraints. Methods of [13, 22], which in part rely on those of [16], work via a scheme of building pseudorandom spaces that fool certain automata, and they seem to inherently require a large polynomial work. Finally, the method of [12] determines the n random variables as a result of a $\text{poly}(\log(mn))$-iteration random walk, with merely pairwise independence in each iteration, and derandomizes each iteration work-efficiently by maintaining certain pairwise objectives.

The method we present in this paper can be viewed as an orthogonal approach, which makes use of algorithms with weaker discrepancy bounds as a basic partitioning tool, and sharpens their discrepancy via bootstrapping. Indeed, our method can be applied on top of the previous methods of [5, 12, 23] to improve their discrepancy to the optimal bound (though with different work bounds). We use in particular the latter result since this yields a work-efficient algorithm overall. Below, we provide a high-level and intuitive (though, admittedly imprecise) overview of our approach. The actual technical proofs will deviate from this outline in some parts, due to details not discussed here.

High-level approach: On a high level, our general approach is to partition the n random variables $\chi_j \in \{-1, 1\}$ for $j \in [n]$ into $L \leq n/2$ parts $P_1 \sqcup \cdots \sqcup P_L$ and constrain the variables in each part to a one-dimensional space. That is, for each part P_t, we compute a *tentative* χ_j for $j \in P_t$, but we allow that potentially all of these variables in part P_t can be negated. More concretely, for each $j, j' \in P_t$, their product $\chi_j \chi_{j'}$ in the final χ is fixed, so once we know the final χ_j, we know $\chi_{j'}$ for all $j' \in P_t$. Then the remaining problem will be to determine whether each part's tentative assignment should be negated or not. This is like computing a vector in $\{-1, 1\}^L$, indicating the negations, such that we satisfy some linear constraints. We solve that recursively as another (*weighted*) instance of the set balancing problem, with $L \leq n/2$ variables. In the base case, once the number of variables is as small as $\text{poly}(\log(nm))$, we can fix them one by one via conditional expectations [25].

The crucial point is how to perform the recursive scheme. Note that the desired output is that each set's discrepancy is within an $O(\sqrt{\log m})$ of its standard deviation upper bound $\sqrt{s}$. Also, this is roughly the best one can hope for, given that we want to achieve this for m sets simultaneously. Thus, we need to perform the recursion such that, in each recursion level, we do not increase the *standard deviation* of each set (by more than a $1 + 1/\log n$ factor, as we will have roughly $\log n$ recursion levels). Furthermore, some care and much extra work will be needed due to the *weightedness* introduced in the recursion, even if we start with an unweighted instance; we will ignore that for now for this brief overview.

Each recursion level, and how to fix the variables in each part: Let us discuss how we fix the variables inside each part. We later discuss how the partition is computed, once we understand the properties needed from the partition. Consider a fixed part P_t, and a set S_i for a fixed $i \in [m]$. Consider a random $\chi \in \{-1, 1\}^n$, let us define its signed discrepancy $sdisc(S_i) = \sum_{j \in S_i} \chi_j$, in contrast to the absolute discrepancy $disc(S_i) = |\sum_{j \in S_i} \chi_j|$. We have $\mathbb{E}[sdisc(S_i)] = 0$. Instead of working with the *standard deviation*, we zoom in on the variance of $sdisc(S_i)$, due to its nice additiveness properties. We have $Var(sdisc(S_i)) = \mathbb{E}[sdisc^2(S_i)] - (\mathbb{E}[sdisc(S_i)])^2 = \mathbb{E}[disc^2(S_i)]$. In particular, notice that

$$\mathbb{E}[disc^2(S_i)] = \mathbb{E}[(\sum_{j \in S_i} \chi_j)^2] = \mathbb{E}[\sum_{j \in S_i} (\chi_j)^2] = |S_i| \leq s.$$

Initially, the variables in $S_i \cap P_t$ contribute exactly $|S_i \cap P_t|$ to $Var(sdisc(S_i)) = \mathbb{E}[disc^2(S_i)]$. After constraining these variables, their contribution to $\mathbb{E}[disc^2(S_i)]$ is $(\sum_{j \in S_i \cap P_t} \chi_j)^2$, as it will be independent of the contributions of other parts. Thus, we want to determine χ_j for $j \in P_t$ such that $(\sum_{j \in S_i \cap P_t} \chi_j)^2$ remains almost the same bound as $|S_i \cap P_t|$.

Unfortunately, that is impossible! This is exactly a set balancing problem for sets $S_1 \cap P_t, S_2 \cap P_t, \ldots, S_m \cap P_t \subseteq P_t$, and we know that, regardless of how we choose χ_j for $j \in P_t$, some set S_i will have $\frac{(\sum_{j \in S_i \cap P_t} \chi_j)^2}{|S_i \cap P_t|} = \Omega(\log m)$; this is a $\Theta(\log m)$ factor loss in the variance. Thus, it seems we would lose a $\Theta(\log m)$ factor in variance in each recursion level. See the warm-up in Theorem 3.1 where we follow this scheme for a two-level recursion, getting an $\tilde{O}(\sqrt{n})$ depth parallel algorithm with a $\sqrt{\log m}$ loss in discrepancy. If we follow this for $\log n$ recursion levels, the variance losses would multiply to $(\log m)^{\log n}$—a useless bound.

Enforcing an average-guarantee, and leveraging it via multiplicative weights update: There is something to be optimistic about: even though the worst variance loss factor among the m sets $S_1 \cap P_t, S_2 \cap P_t, \ldots, S_m \cap P_t$ will be $\Theta(\log m)$, the average loss across these sets should be smaller. Indeed, by Chernoff bound, for a random χ, the probability of a z factor loss is $exp(-\Theta(z^2))$. More useful for us, we have $\mathbb{E}[\sum_{i=1}^m (\sum_{j \in S_i \cap P_t} \chi_j)^2] = \sum_{i=1}^m |S_i \cap P_t|$. We will be able to enforce this (and even a weighted variant of it) as an actual constraint in the derandomization process, in a work-efficient manner by utilizing that it uses only pairwise independence. However, this average across different sets $S_1 \cap P_t, \ldots, S_m \cap P_t$ in one part P_t is not useful on its own. We want *each* set S_i to have a small discrepancy. So we need a different averaging, for each set S_i across different parts P_t. That is, we need each set S_i not to experience too much loss, once we add up the new variances in $S_i \cap P_1, S_i \cap P_2, \ldots, S_i \cap P_L$. That is, we want $\sum_{t=1}^L (\sum_{j \in S_i \cap P_t} \chi_j)^2 \approx s$ where the approximation can tolerate a $1 + 1/\log n$ factor.

For that, we appeal to the Multiplicative Weights Update (MWU) method [3]. Suppose we work through the parts $P_1, P_2, \ldots, P_L$ sequentially in L rounds (the algorithm will be different, as we will discuss, since depth L would be expensive). In each round, each set S_i will have an importance value $imp(i)$, such that sets with larger hitherto variance losses have higher importance. We then enforce an importance-weighted variant of the previous average guarantee across the sets $S_1 \cap P_t, \ldots, S_m \cap P_t$. This in effect tries to have smaller variance losses in this round for sets S_i that have higher importance. Roughly speaking, this will force an averaging for each set S_i across different parts P_t. As a result, in the end, the overall variance loss for each set S_i summed up over all its parts $S_i \cap P_1, S_i \cap P_2, \ldots, S_i \cap P_L$ will be small and we get $\sum_{t=1}^L (\sum_{j \in S_i \cap P_t} \chi_j)^2 \leq s(1 + 1/\log n)$. See the warm-up in Theorem 3.4 where we use this idea to sharpen the discrepancy of the previously discussed $\tilde{O}(\sqrt{n})$-depth algorithm.

Processing L parts sequentially would not yield a polylogarithmic depth (unless L itself is just polylogarithmic, which brings other issues, as each part would have many variables, and they are solved one after the other). Instead of processing one part in each round, we process L/T many of them in one round in parallel and independently. The number of rounds will be set to $T = \text{poly}(\log(mn))$, and this will be sufficient for MWU to bring down the average loss in each set to $1 + 1/\log n$. Roughly speaking, this is because the worst-case loss in each round is at most an $O(\log m)$ higher than the average loss factor of 1.

We will set the parts such that each part induces an easy problem: in one application of the scheme each part will consist of only $\text{poly}(\log(mn))$ variables, and thus we will be able to fix these variables via sequential derandomization [25]. In another application, the number of variables in the part will be large but each set will have an intersection of size $\text{poly}(\log(mn))$ with this part, and therefore we will be able to use pairwise parallel derandomization, a la Luby [21].

Partitioning: For the scheme described above to work, we need that, in each set S_i, the different $T = \text{poly}(\log(mn))$ rounds of MWU process portions of S_i that expect roughly the same variance, up to $1 + 1/\text{poly}(\log n)$ factors. To produce this $\text{poly}(\log(mn))$-way partitioning, we use the work-efficient result of Ghaffari, Grunau, Rozhon [12], as the base partitioning tool, in an essentially black-box way. It is crucial that we do not need partitions with optimal or near-optimal additive loss here, and the multiplicative error $1+1/\text{poly}(\log(mn))$ of [12] will be tolerable in our overall scheme[4].

Finally, the above outline discusses only the unweighted instance and an intuitive explanation of how we perform one level of recursion. However, even if we start with the unweighted set balancing problem, the recursion creates a weighted problem for the next iteration. Fortunately, in the case of the unweighted set balancing problem, the "weights" will remain within a relatively close range, and thus an algorithm along the lines discussed above will still work. We present this in Section 4 as the proof of Theorem 1.1.

For our weighted set balancing result, Theorem 1.2, the partitioning is more complex. We do not provide an overview here but just mention this: in each constraint, a few variables of very large weight cannot be handled as before, and we will need the partitioning to put essentially all of these into separate parts (modulo a loss, that will have to be controlled very tightly). We present the proof of Theorem 1.2 in the full version of this paper.

2 PRELIMINARIES: NOTATIONS AND TOOLS FROM PRIOR WORK

Notations: Throughout, we work with the ground set $[n]$ and usually m sets in it $S_1, S_2, \ldots, S_m \subseteq [n]$. For a given *value assignment* vector $\chi \in \{-1,1\}^n$, the discrepancy of each set S_i is defined as $disc(S_i) = |\sum_{j \in S_i} \chi_j|$. We generally assume that $m \geq 2$, as for $m = 1$ the problem is trivial. In the weighted generalization, we are given a matrix $A \in \mathbb{R}^{m \times n}$, where the entry i, j is denoted by a_{ij}, and the discrepancy of the i^{th} constraint is defined as $disc(i) = |\sum_{j=1}^n a_{ij}\chi_j|$. Sometimes, instead of focusing on the absolute value of the discrepancy, we talk about the signed discrepancy $sdisc(i) = \sum_{j=1}^n a_{ij}\chi_j$.

Often, we work with a partition of the ground set into parts $P_1 \sqcup P_2 \sqcup \cdots \sqcup P_L = [n]$, and we need to determine a value assignment for the variables in each part. We use the notation $\chi^t \in \{-1,1\}^{[P_t]}$ to indicate the part of the value assignment in part P_t, which is a binary vector of length $|P_t|$, but with the convenient indexing

[4]Let us clarify the distinction between the additive loss and the multiplicative error: Consider for instance their scheme for partitioning into two parts, and m sets of size at most s. The ideal additive discrepancy would be $O(\sqrt{s \log m})$. The additive discrepancy of [12] will be up to $s/\text{poly}(\log(mn))$. This means, in each set, each of the two halves will have size in $s/2(1 \pm 1/\text{poly}(\log(mn)))$. Thus the multiplicative error is $1+1/\text{poly}(\log(mn))$. In our scheme, this $1/\text{poly}(\log(mn))$ partitioning loss will add up with the $1/\text{poly}(\log(mn))$ loss of the MWU method, and still keep the overall loss in variance per level below a $1 + 1/\log n$ factor.

inherited from $[n]$ such that their concatenation forms the output vector $\chi \in \{-1,1\}^n$, that is, $\chi_j = \chi_j^t$ for $j \in P_t$.

2.1 Crude Partitioning

We use the algorithm of Ghaffari, Grunau, and Rozhon [12] as a crude partitioning tool. We next state their key result here, as well as two simple generalizations that we derive here with some extra work. The proofs of these extensions are deferred to the full version of this paper.

Theorem 2.1 (Ghaffari, Grunau, Rozhon [12]). *Let $n, m, k \in \mathbb{N}$ with $m \geq 2$ and let $\{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $[n]$. Then, there exists a deterministic parallel algorithm with work $\tilde{O}(n + m + \sum_{i=1}^m |S_i|)\operatorname{poly}(k)$ and depth $\operatorname{poly}(\log(nm)k)$ that computes a partition $P_1 \sqcup P_2 = [n]$ satisfying that $\max(|S_i \cap P_1|, |S_i \cap P_2|) = |S_i|/2 + O(\sqrt{|S_i|\log m}) + \frac{|S_i|}{k}$ for every $i \in [m]$.*

In particular, the following states the variant when we partition into L parts, instead of just two parts:

Lemma 2.2 (Unweighted multi-way partition). *Let $n, m, L \in \mathbb{N}$ with $m \geq 2$, $\varepsilon \in (0, 0.5]$, and let $\{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $[n]$. Also, let L be a power of two. Then, there exists a deterministic parallel algorithm with work $\tilde{O}(n + m + \sum_{i=1}^m |S_i|) \cdot \operatorname{poly}(1/\varepsilon)$ and depth $\operatorname{poly}(\log(nm)/\varepsilon)$ that computes a partition $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L = [n]$ satisfying that $|S_i \cap P_\ell| \leq (1+\varepsilon)|S_i|/L + O(\log(m)/\varepsilon^2)$ for every $i \in [m]$ and $\ell \in [L]$.*

We also use the following weighted variant, which follows roughly speaking by managing together in each constraint the variables of almost the same weight.

Lemma 2.3 (Weighted multi-way partition). *Let $n, m, L \in \mathbb{N}$ with L being a power of two, $m \geq 2$, $A \in \mathbb{R}^{n \times m}$ and let $\varepsilon \in (0, 1]$. There exists a deterministic parallel algorithm with work $\tilde{O}(n + m + nnz(A)) \cdot \operatorname{poly}(1/\varepsilon)$ and depth $\operatorname{poly}(\log(nm)/\varepsilon)$ that computes a partition $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L = [n]$ satisfying the following for every $\ell \in [L]$:*

- *$|P_\ell| \leq (1+\varepsilon)n/L + O(\log(nm)/\varepsilon^2)$,*
- *$|P_\ell \cap \{j \in [n] \colon a_{ij} \neq 0\}| \leq \frac{(1+\varepsilon)}{L}|\{j \in [n] \colon a_{ij} \neq 0\}| + O(\log(nm)/\varepsilon^2)$ for every $i \in [m]$ and*
- *$\sum_{j \in P_\ell} a_{ij}^2 \leq \frac{1+\varepsilon}{L}\sum_{j=1}^n a_{ij}^2 + O(\log^2(nm)/\varepsilon^3)(a_{max})^2$ for every $i \in [m]$,*

where we define $a_{max} = \max_{i \in [m], j \in [n]} |a_{ij}|$.

2.2 Sequential Derandomization

We also make use of the sequential derandomization method of Raghavan [25], which solves the weighted set balancing problem by fixing the random variables one by one, using the method of conditional expectations. The following theorem abstracts this result. We do not provide a proof for this version, but we will later state and use a more general result as Theorem 3.2, and we present a proof for that in the full version of

Theorem 2.4 (Sequential derandomization). *There exists an absolute constant $C > 0$ for which the following holds. Let $n, m \in \mathbb{N}$, and $A \in \mathbb{R}^{m \times n}$. There exists a deterministic parallel algorithm algorithm with work $\tilde{O}(nnz(A) + n + m)$ and depth $n\operatorname{poly}(\log m)$ that computes a vector $\chi \in \{-1,1\}^n$ such that, for every $i \in [m]$, it holds that $disc_i^2 = C\log m \cdot \sum_{j=1}^n a_{ij}^2$.*

3 WARM-UP

In this section, we present two warm-up results, and in their context, we discuss two of the ideas that we use in our main results. Suppose we are given m sets $S_1, S_2, \ldots, S_m \subseteq [n]$, for $m \geq 2$, such that $|S_i| \leq s$ for each $i \in [m]$. By Chernoff bound, we know that a random value assignment $\chi \in \{-1,1\}^n$ creates a discrepancy of at most $disc(S_i) = |\sum_{j \in S_i} \chi_j| = O(\sqrt{s\log m})$ in each of the m set. In the first result, Theorem 3.1, we show a deterministic parallel algorithm that achieves a slightly suboptimal discrepancy of $O(\sqrt{s}\log m)$ in near-linear work and $\tilde{O}(\sqrt{n})$ depth. In the second result, Theorem 3.4, we show how to improve the discrepancy to the optimal bound of $O(\sqrt{s\log m})$ while keeping near-linear work and $\tilde{O}(\sqrt{n})$ depth. As stated before, these are warm-up results, presented chiefly as contexts for introducing two of the ideas that we use frequently in our main results. Next, we discuss these two ideas from a high-level and informal viewpoint.

The key idea that we will present in the first result is how to create some parallelism in the task of computing a low-discrepancy value assignment, by partitioning the variables. Roughly speaking, we partition the ground set $[n]$ into about $\sqrt{n}$ parts and we find a value assignment in each part independently and all in parallel. This is such that the discrepancy of each set in each part is "small". Then, in the end, we need to find a good mixture of the assignments of the different parts. Intuitively, this mixture selection will involve negating the solution coming from some of the parts; we will choose the negated parts carefully to achieve a good discrepancy in the output for all the m sets.

As we will see, compared to the standard deviation upper bound of $\sqrt{s}$, the above approach loses a $\sqrt{\log m}$ in the discrepancy in the parts and another $\sqrt{\log m}$ in finding a good mixture, thus creating the suboptimal discrepancy of $O(\sqrt{s}\log m)$. To remedy this, in the second result, we create a certain multi-round game for the process of determining the solutions in different parts, and we use an instantiation of the Multiplicative Weights Update method to "average out" the losses of each set throughout the rounds of this game. As a result, we will be able to remove one of the $\sqrt{\log m}$ factor losses essentially completely.

3.1 Near-Optimal Discrepancy with $\tilde{O}(\sqrt{n})$ Depth

Theorem 3.1. *Let $n, m \in \mathbb{N}$ with $m \geq 2$, and let $\{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $[n]$. Then, there exists a deterministic parallel algorithm that can compute a vector $\chi \in \{-1,1\}^n$ with $\tilde{O}(n + m + \sum_{i=1}^m |S_i|)$ work and $\tilde{O}(\sqrt{n})$ depth such that, for every $i \in [m]$, it holds that $disc^2(S_i) = (\sum_{j \in S_i} \chi_j)^2 = O(|S_i|\log^2 m)$.*

Proof. Recall that the sequential derandomization method (Theorem 2.4) provides a method to fix the n variables χ_j for $j \in [n]$ one by one, in depth $\tilde{O}(n)$, such that we have $(\sum_{j \in S_i} \chi_j)^2 = O(|S_i|\log m)$ for every $i \in [m]$. To reduce this $\tilde{O}(n)$ depth to $\tilde{O}(\sqrt{n})$, we use the following approach: (1) first we partition the variables into roughly $\sqrt{n}$ parts each with $O(\sqrt{n})$ variables, (2) we perform a sequential derandomization inside each part and all in parallel, and then (3) we determine how to merge the $\sqrt{n}$ parts together via another sequential derandomization. We next make this outline concrete.

(1) We compute a partitioning of the $[n]$ into $L = 2^{\lceil \log_2(\sqrt{n}) \rceil}$ parts $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L = [n]$ such that for each $t \in [L]$, we have $|P_t| \leq 2\sqrt{n}$. This can be achieved easily via the unweighted partitioning recalled in Lemma 2.2, in $\tilde{O}(n + m + \sum_{i=1}^{m} |S_i|)$ work and poly$(\log(nm))$ depth.

(2) As a result of the above partitioning, we have L independent discrepancy problems on disjoint variables: problem t consists of sets $S_1 \cap P_t, S_2, \cap P_t, \ldots, S_m \cap P_t$. We solve these problems in parallel and independently of each other, each using sequential derandomization. Since each part has at most $2\sqrt{n}$ variables, we can invoke the sequential derandomization method (cf. Theorem 2.4) to solve each part t in $\tilde{O}(\sqrt{n})$ depth, getting a vector $\bar{\chi}^t \in \{-1,1\}^{[P_t]}$ such that for each $i \in [m]$, we have $(\sum_{j \in S_i \cap P_t} \bar{\chi}^t_j)^2 = |S_i \cap P_t| \cdot O(\log m)$. This works for all the L parts in parallel, in $\tilde{O}(\sqrt{n})$ depth, and using a total of $\tilde{O}(n + m + \sum_{i=1}^{m} |S_i|)$ work. To ensure this work bound, we need a simple clean-up before invoking the sequential derandomization in each part t: we discard from each part P_t sets i for which $S_i \cap P_t = \emptyset$.

(3) Finally, we need to determine how to merge these $L \approx \sqrt{n}$ solutions $\bar{\chi}^t$ for $t \in [L]$. Let us first discuss the situation from an intuitive viewpoint. Notice that naively taking the output vector $\chi \in \{-1,1\}^n$ as the "concatenation" of vectors $\bar{\chi}^t \in \{-1,1\}^{[P_t]}$ computed in different parts can result in a large discrepancy. For instance, in the absence of any further guarantee on how the parts are mixed, for set S_i, we can have a discrepancy as large as

$$\begin{aligned}|\sum_{j \in S_i} \chi_j| &= |\sum_{t=1}^{L} \sum_{j \in S_i \cap P_t} \chi_j| \\ &= \sqrt{|S_i \cap P_1| \cdot O(\log m)} + \sqrt{|S_i \cap P_2| \cdot O(\log m)} + \ldots \\ &+ \sqrt{|S_i \cap P_L| \cdot O(\log m)}.\end{aligned}$$

This can reach $\sqrt{|S_i| L \cdot O(\log m)}$, which is an $\sqrt{L}$ factor loss compared to the ideal bound of $\sqrt{|S_i| \cdot O(\log m)}$. This loss stems from mixing L variables, each expected to be roughly $\sqrt{|S_i|/L \cdot O(\log m)}$, in an arbitrary way. This arbitrary way allows all of the L terms to contribute positively and thus the absolute values add up. The challenge is that this can happen in any one of the m sets.

To remedy the above issue, we need to find a *good mixture* of the solutions. More concretely, to obtain the output vector $\chi \in \{-1,1\}^n$, for each part t, we can determine whether to *take* $\bar{\chi}^t$ *as is or to negate it.* The negation of course does not change the absolute value of discrepancy in each part. However, if we choose it wisely, it can help us avoid the unfortunate case of discrepancies of the L different parts adding up in the same direction.

Formally, we set up a new discrepancy problem with a variable $\chi' \in \{-1,1\}^L$—which determines whether each part is negated or not—with the interpretation that we will set the overall output vector as $\chi_j = \bar{\chi}^t_j \cdot \chi'_t$ where $j \in P_t$. For each set $i \in [m]$, we have

$$\begin{aligned}(\sum_{j \in S_i} \chi_j)^2 = (\sum_{t=1}^{L} \sum_{j \in S_i \cap P_t} \chi_j)^2 &= (\sum_{t=1}^{L} \sum_{j \in S_i \cap P_t} \bar{\chi}^t_j \cdot \chi'_t)^2 \\ &= (\sum_{t=1}^{L} \chi'_t \cdot (\sum_{j \in S_i \cap P_t} \bar{\chi}^t_j))^2.\end{aligned}$$

To summarize, we have a new "weighted" discrepancy problem with an output vector $\chi' \in \{-1,1\}^L$, which consists of m sets where each set S_i has L elements and element t has weight $a_{i,t} = (\sum_{j \in S_i \cap P_t} \bar{\chi}^t_j)$ in the discrepancy of set S_i. That is, we have $disc(S_i) = (\sum_{t=1}^{L} a_{i,t} \cdot \chi'_t)$. The guarantee provided is that for each $t \in [L]$, we have $a^2_{i,t} = (\sum_{j \in S_i \cap P_t} \bar{\chi}^t_j)^2 = |S_i \cap P_t| \cdot O(\log m)$.

We can solve this discrepancy problem via sequential derandomization of Theorem 2.4 in depth $\tilde{O}(L) = \tilde{O}(\sqrt{n})$. Also, the work is no more than $\tilde{O}(n + m + \sum_{i=1}^{m} |S_i|)$. We get a vector $\chi' \in \{-1,1\}^L$ such that $disc^2(S_i) = (\sum_{t=1}^{L} a_{i,t} \cdot \chi'_t)^2 = (\sum_{t=1}^{L} a^2_{i,t} \cdot O(\log m))$. Hence, overall, we have an output vector $\chi \in \{-1,1\}^n$—with the definition $\chi_j = \bar{\chi}^t_j \cdot \chi'_t$ where $j \in P_t$—such that for each $i \in [m]$, we have

$$\begin{aligned}disc^2(S_i) &= (\sum_{j \in S_i} \chi_j)^2 \\ &= \sum_{t=1}^{L} a^2_{i,t} \cdot O(\log m) \\ &= \sum_{t=1}^{L} |S_i \cap P_t| \cdot O(\log m) \cdot O(\log m) \\ &= |S_i| \cdot O(\log^2 m).\end{aligned}$$

□

3.2 Optimal Discrepancy with $\tilde{O}(\sqrt{n})$ Depth

An intuitive/informal discussion of the suboptimality in Theorem 3.1 and how we remedy it: Theorem 3.1 has a suboptimal discrepancy of $disc(S_i) = \sqrt{|S_i|} \cdot O(\log m)$, instead of the ideal bound of $disc(S_i) = \sqrt{|S_i|} \cdot O(\sqrt{\log m})$. Intuitively, there are two $O(\sqrt{\log m})$ factors in the achieved discrepancy: one $O(\sqrt{\log m})$ factor from the sequential derandomization in solving each of the $\sqrt{n}$ parts as described in step (2), and another $O(\sqrt{\log m})$ factor from the process of finding a good mixture of the parts as described in part (3). In this subsection, we discuss how to remedy this loss.

Let us zoom in on the first loss: the $O(\sqrt{\log m})$ loss in solving each of the $\sqrt{n}$ parts in step (2). This loss is optimal in the worst-case sense, meaning that in each part, there will be at least one set that experiences an $O(\sqrt{\log m})$ factor loss. However, there is something to be optimistic about: in each part, most of the sets should not experience such a loss. As a matter of fact, the $\sqrt{\log m}$ factor in the sequential derandomization is to allow a union bound over all the m sets, by reducing the probability of each set breaking the bound to $1/m$. But most sets should have a much smaller loss.

Indeed, we observe that, with some extra work, we can create a variant of the sequential derandomization result stated in Theorem 2.4 that ensures the average of the losses to be only a $1 + o(1)$ factor. We next state this variant (for the first reading, the reader can think of all $imp(i)$ as equal to 1; later we will need the importance-weighted generality).

Theorem 3.2 (Sequential derandomization, augmented with importance-weighted averaging). *There exists an absolute constant $C > 0$ for which the following holds. Let $n, m \in \mathbb{N}$, $M \in \mathbb{R}$ with $M \geq \max(m, 2)$, $A \in \mathbb{R}^{m\times n}$, and $imp(i) \in \mathbb{R}_{\geq 0}$ for each $i \in [m]$. There exists a deterministic parallel algorithm with work $\tilde{O}(nnz(A) + n + m))$ and depth $n\,\mathrm{poly}(\log M)$ that computes a vector $\chi \in \{-1, 1\}^n$ such that $\sum_{i=1}^m imp(i) \cdot disc_i^2 \leq \left(1 + \frac{1}{M}\right) \sum_{i=1}^m imp(i) \cdot \left(\sum_{j=1}^n a_{ij}^2\right)$. Moreover, for every $i \in [m]$, it also holds that $disc_i^2 = C \log M \cdot \sum_{j=1}^n a_{ij}^2$.*

The proof of this variant is deferred to the full version of this paper. Intuitively, this variant says that the average loss among the sets is "negligible". However, the average loss among the sets is not directly helpful. We somehow need the overall loss of each of the m sets to be small. That is, we need a mechanism that ensures that the average loss of each set, averaged over all the parts, is small (ideally $1 + o(1)$).

For this mechanism, we appeal to the Multiplicative Weights Update (MWU) method. This method uses varying degrees of importance for different sets based on the losses they have experienced so far, ensuring that sets with large hitherto losses have large importance. Theorem 3.2 allows us to make use of this, by enforcing that sets with larger importance experience smaller losses in the next part. To make this MWU averaging work, we need to set up some sequential dependency between the parts (previously, the parts were solved in parallel, independently). Next, we first provide a reminder on MWU, phrased concretely for our usage, and then present the algorithm that formalizes this intuition. A proof of this MWU lemma is presented in the full version of this paper.

Lemma 3.3 (Multiplicative Weights Update). *Consider a multi-round game with m constraints and an oracle. Each constraint $i \in [m]$ has an importance value $imp(i) \in \mathbb{R}^+$, which is initially set $imp^1(i) = 1$ and changes over the rounds. In each round t, we give the oracle the importance $imp^t(.)$ of the constraints, and the oracle gives back for each constraint a value $gap^t(i) \in [0, W]$, with the guarantee that $\sum_{i=1}^m imp^t(i) \cdot gap^t(i) \leq \sum_{i=1}^m imp^t(i)$. For any given value $\varepsilon \in [0, 1/2]$, there is a way to set the importance values during the game such that, at the end of a game with $T = \Omega(W \log m/\varepsilon^2)$ rounds, for each constraint, we have $(\sum_{t=1}^T gap^t(i))/T \leq 1 + \varepsilon$. The rule for updating the importance values is simple: in each round t, set $imp^{t+1}(i) = imp^t(i) \cdot (1 + \eta \cdot gap^t(i))$ where $\eta = \varepsilon/(3W)$.*

We are now ready for our second warm-up, which achieves the optimal discrepancy in $\tilde{O}(\sqrt{n})$ depth.

Theorem 3.4. *Let $n, m \in \mathbb{N}$ with $m \geq 2$, and let $\{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $[n]$ such that $|S_i| \leq s$ for all $i \in [m]$. Then, there exists a deterministic parallel algorithm that can compute a vector $\chi \in \{-1, 1\}^n$ with $\tilde{O}(n+m+\sum_{i=1}^m |S_i|)$ work and $\tilde{O}(\sqrt{n})$ depth such that, for every $i \in [m]$, it holds that $disc^2(S_i) = (\sum_{j\in S_i} \chi_j)^2 = O(s \log m)$.*

Proof. Set $\varepsilon = 1/(10 \log m)$. Also, we assume $n \geq \log^5 m$ (otherwise, we can solve the entire problem in poly$(\log m)$ depth and near-linear work using sequential derandomization), and that $s \geq \Omega(\log^{10} m)$ (otherwise, the statement of the theorem follows from Theorem 2.1, by setting $k = \sqrt{s}$).

First, we partition the variables into $T = \Theta(\log^5 m)$ parts $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_T = [n]$ with the following guarantee: For each set $i \in [m]$ and each part $t \in [T]$, we should have $|S_i \cap P_t| \leq (1+\varepsilon)(|S_i|/T + \delta)$ for $\delta = O(\log m/\varepsilon^2)$, and moreover, for each $t \in T$, we should have $|P_t| \leq (1+\varepsilon)n/T$. This partition can be done directly via Lemma 2.2, in $\tilde{O}(n+m+\sum_{i=1}^m)$ work and poly$(\log(nm))$ depth. We will process these $T = \Theta(\log^5 m)$ parts sequentially, using MWU, in the sense that processing part $t \in T$ will be regarded as round t of the MWU game, as described in Lemma 3.3.

Let us first discuss what we do in each part. Consider part t, which has $|P_t| = \Theta(n/\log^5 m)$ variables, and the sets $S_1 \cap P_t, S_2 \cap P_t, \ldots, S_m \cap P_t$. To solve this part in $\tilde{O}(\sqrt{n})$ depth, we perform something similar to steps (1) and (2) of the proof of Theorem 3.1. Concretely, first we partition P_t further into $L = 2^{\lceil \log \sqrt{n/\log^5 m}\rceil} \approx \sqrt{n/\log^5 m}$ *pieces* $P_{t,1} \sqcup P_{t,2} \sqcup \ldots \sqcup P_{t,L}$ such that for each $t' \in [L]$, we have $|P_{t,t'}| \leq 2|P_t|/L = O(\sqrt{n/\log^5 m})$. This can be done via Lemma 2.2, in poly$(\log(nm))$ depth and using $\tilde{O}(n + m + \sum_{i=1}^m)$ total work (again, we need to perform the simple clean-up of removing from each part sets that have an empty intersection with the part). Then, we use sequential derandomization inside each of the pieces, all independently and in parallel. In particular, by invoking Theorem 3.2 in each piece $t' \in [L]$, we get an output $\bar{\chi}^{t,t'} \in \{-1, 1\}^{[P_{t,t'}]}$ with the following two guarantees:

- For each set i, we have $(\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2 \leq |S_i \cap P_{t,t'}| \cdot O(\log m)$
- $\sum_{i=1}^m imp(i) \cdot (\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2 \leq (1+\varepsilon) \sum_{i=1}^m imp(i) \cdot |S_i \cap P_{t,t'}|$

Over all the pieces t', which are solved in parallel, the algorithm works in $\tilde{O}(n + m + \sum_{i=1}^m |S_i|)$ work and $\tilde{O}(\sqrt{n})$ depth. Also, from the second inequality, we can deduce that

$$\begin{aligned}
&\sum_{i=1}^m imp(i) \cdot \left(\sum_{t'=1}^L (\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2\right) \\
&= \sum_{t'=1}^L \sum_{i=1}^m imp(i) \cdot (\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2 \\
&\leq \sum_{t'=1}^L (1+\varepsilon) \sum_{i=1}^m imp(i) \cdot |S_i \cap P_{t,t'}| \\
&= (1+\varepsilon) \sum_{i=1}^m imp(i) \cdot \left(\sum_{t'=1}^L |S_i \cap P_{t,t'}|\right) \\
&= (1+\varepsilon) \sum_{i=1}^m imp(i) \cdot |S_i \cap P_t| \\
&\leq (1+\varepsilon) \sum_{i=1}^m imp(i) \cdot \left((1+\varepsilon)(\frac{s}{T} + \delta)\right) \\
&= (1+\varepsilon)^2(\frac{s}{T} + \delta) \cdot \sum_{i=1}^m imp(i).
\end{aligned}$$

Now, let us define

$$gap^t(i) = \frac{\sum_{t'=1}^{L}(\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2}{(1+\varepsilon)^2(\frac{s}{T}+\delta)}.$$

Notice that

$$\begin{aligned} gap^t(i) &= \frac{\sum_{t'=1}^{L}(\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2}{(1+\varepsilon)^2(\frac{s}{T}+\delta)} \\ &\le \frac{\sum_{t'=1}^{L}|S_i\cap P_{t,t'}|\cdot O(\log m)}{(1+\varepsilon)^2(\frac{s}{T}+\delta)} \\ &= \frac{|S_i\cap P_t|\cdot O(\log m)}{(1+\varepsilon)^2(\frac{s}{T}+\delta)} \le \frac{O(\log m)}{(1+\varepsilon)^2}, \end{aligned}$$

and moreover,

$$\begin{aligned} \sum_{i=1}^{m} imp(i)\cdot gap^t(i) &= \sum_{i=1}^{m} imp(i)\cdot \frac{\sum_{t'=1}^{L}(\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2}{(1+\varepsilon)^2(\frac{s}{T}+\delta)} \\ &\le \frac{(1+\varepsilon)^2(\frac{s}{T}+\delta)\cdot\sum_{i=1}^{m} imp(i)}{(1+\varepsilon)^2(\frac{s}{T}+\delta)} = \sum_{i=1}^{m} imp(i). \end{aligned}$$

Hence, the values $gap^t(i)$ satisfy the two properties of the MWU statement in Lemma 3.3 with $W = O(\log m)$. Thus, after running the game for T rounds by going through the parts $P_1, P_2, \ldots, P_T$ and doing the above for each of them sequentially, since $T = \Omega(W\log m/\varepsilon^2)$, we get that for each set $i \in [m]$, we have $\sum_{t=1}^{T} gap^t(i) \le (1+\varepsilon)T$. This means

$$\sum_{t=1}^{T}\sum_{t'=1}^{L}(\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2 \le (1+\varepsilon)^3(s+T\delta) = (1+\varepsilon)^3 s + O(\log^8 m).$$

Thus, we have $TL = \Theta(\sqrt{n\log^5 m})$ solutions $\bar{\chi}^{t,t'} \in \{-1,1\}^{[P_{t,t'}]}$ for $t \in [T]$ and $t' \in [L]$ and we need to find a good mixture of them. That is, we want to find a mixture vector $\chi' \in \{-1,1\}^{TL}$ which determines for each of these solutions whether to take itself or its negation, by setting the output vector $\chi \in \{-1,1\}^n$ as $\chi_j = \bar{\chi}_j^{t,t'}\cdot\chi'_{t,t'}$ where $j \in P_{t,t'}$. This part is quite similar to step (3) in the proof of Theorem 3.1. In particular, define $a_{i,(t-1)T+t'} = (\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})$. By invoking the sequential derandomization of Theorem 3.2, which runs in $\tilde{O}(\sqrt{n})$ depth and with $\tilde{O}(n+m+\sum_{i=1}^{m}|S_i|)$ work, we get a mixture vector $\chi' \in \{-1,1\}^{TL}$ such that $disc^2(i) = (\sum_{k=1}^{TL} a_{i,k}\cdot\chi'_k)^2 = (\sum_{k=1}^{TL} a_{i,k}^2\cdot O(\log m))$. Hence, overall, for each $i \in [m]$, we have

$$\begin{aligned} disc^2(S_i) &= (\sum_{j\in S_i}\chi_j)^2 \\ &= (\sum_{k=1}^{TL} a_{i,k}\cdot\chi'_k)^2 \\ &\le \sum_{k=1}^{TL} a_{i,k}^2\cdot O(\log m) \\ &= \sum_{t=1}^{T}\sum_{t'=1}^{L}(\sum_{j\in S_i\cap P_{t,t'}} \bar{\chi}_j^{t,t'})^2\cdot O(\log m) \\ &\le ((1+\varepsilon)^3 s + O(\log^8 m))\cdot O(\log m) \\ &= O(s\log m). \end{aligned}$$

Here, the last inequality uses that $\varepsilon = \frac{1}{10\log m}$ and $s \ge \Omega(\log^{10} m)$. □

4 OPTIMAL DISCREPANCY IN POLYLOGARITHMIC DEPTH — UNWEIGHTED

In this section, we present a deterministic parallel algorithm that achieves an asymptotically optimal discrepancy for the (unweighted) set balancing problem—i.e., matching what follows from the Chernoff bound—using near-linear work and polylogarithmic depth, therefore proving Theorem 1.1.

In Section 4.1, we present the core ingredients in this result, in the format of a lemma that achieves $O(\sqrt{n\log m})$ discrepancy. The lemma will actually be somewhat more general for two reasons: (1) to make way for its own proof via recursion, and (2) to facilitate its later usage. Later, in Section 4.2, we use this result and some extra helper lemma to get discrepancy $O(\sqrt{s\log m})$.

4.1 A Polylogarithmic-Depth Recursive Algorithm for $O(\sqrt{n\log m})$ Discrepancy

In this subsection, we prove the following result:

Lemma 4.1. *There is an absolute constant $C' > 0$ for which the following holds. Let $n, m \in \mathbb{N}$ with $m, n \ge 2$ and $\Delta \in \mathbb{R}_{\ge 0}$. Let $A \in \mathbb{R}^{m\times n}$ satisfying that $\sum_{j=1}^{n} a_{ij}^2 \le \Delta$ for every $i \in [m]$ and $\max_{i\in[n],j\in[m]} a_{ij}^2 \le \frac{(\log^5(nm))}{n}\Delta$. Then, there exists a deterministic parallel algorithm that can compute a vector $\chi \in \{-1,1\}^n$ with $\tilde{O}(n+m+nnz(A))$ work and $\mathrm{poly}(\log(nm))$ depth such that, for every $i \in [m]$, it holds that $(\sum_{j=1}^{n} a_{ij}\chi_j)^2 = (2-1/\log n)\cdot(C'\log m)\cdot\Delta$.*

This result itself can be viewed as a generalization of achieving discrepancy $O(\sqrt{n\log m})$ in the unweighted setting. In particular, for that purpose, the reader can interpret $a_{ij} = 1$ for $j \in S_i$ and $a_{ij} = 0$ otherwise, and $\Delta = n$. The lemma generalizes the statement mainly by allowing a polylogarithmic range of weights for nonzero coefficients a_{ij}. As mentioned before, this generality is necessary for our recursive algorithm that proves the lemma, and moreover, it helps in later applications of the result.

The key ingredient in proving Lemma 4.1 is a helper lemma, which we next state as Lemma 4.2. The former provides a certain partitioning scheme for the variables, along with a value assignment inside each part, which basically sets up the recursion. Then the task of finding a good mixture of these assignments will be solved by recursion. We first prove Lemma 4.2, and then go back to proving Lemma 4.1.

Lemma 4.2. *There exists an absolute constant $c > 0$ such that the following holds. Let $n, m \in \mathbb{N}$ with $m \geq 2$ and $n \geq c\log^{30}(m)$ and $\Delta \in \mathbb{R}_{\geq 0}$. Let $A \in \mathbb{R}^{m\times n}$ satisfying that $\sum_{j=1}^{n} a_{ij}^2 \leq \Delta$ and $\max_{i\in[m],j\in[n]} a_{ij}^2 \leq \frac{(\log^5(nm))}{n}\Delta$. Then, there exists a deterministic parallel algorithm that can compute a vector $\chi \in \{-1,1\}^n$ and a partition $[n] = P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L$ for some $L \leq n/2$ with $\tilde{O}(n+m+nnz(A))$ work and $\tilde{O}(1)$ depth satisfying that*

(A) $\sum_{\ell\in L}\left(\sum_{j\in P_\ell} a_{ij}\chi_j\right)^2 \leq \left(1+\frac{1}{10\log^2 n}\right)\Delta$, *and*

(B) *for every $i \in [m]$ and $\ell \in [L]$,* $\left(\sum_{j\in P_\ell} a_{ij}\chi_j\right)^2 \leq \frac{O(\log(nm))}{L}\Delta$.

Proof. Set $\varepsilon = 1/(100\log^2(nm))$. First, we partition the n variables into $L = 2^{\lceil \log(\frac{n}{\log^{20}(nm)})\rceil} \approx \frac{n}{\log^{20}(nm)}$ parts $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L = [n]$ with the following two properties:

(I) For each part $\ell \in [L]$, we have $|P_\ell| \leq 2n/L = O(\log^{20}(nm))$

(II) For each part $\ell \in [L]$ and each set $i \in [m]$, we have $\sum_{j\in P_\ell} a_{ij}^2 \leq (1+2\varepsilon)\Delta/L$.

This can be computed via Lemma 2.3 using $\tilde{O}(nnz(A)+n+m))$ work and $\text{poly}(\log(mn))$ depth. In particular, the third property in Lemma 2.3 implies that for each set $i \in [m]$ and each part $\ell \in [L]$, we have

$$\begin{aligned}\sum_{j\in P_\ell} a_{ij}^2 &\leq (1+\varepsilon)\frac{\Delta}{L} + O(\frac{\log^2(nm)}{\varepsilon^3})\cdot(\max_{i\in[m],j\in[n]} a_{ij}^2)\\ &\leq (1+\varepsilon)\frac{\Delta}{L} + O(\frac{\log^2(nm)}{\varepsilon^3})\cdot\frac{(\log^5(nm))}{n}\Delta\\ &= (1+\varepsilon)\frac{\Delta}{L} + O(\frac{\log^2(nm)}{\varepsilon^3})\cdot\frac{(\log^5(nm))}{L\log^{20}(nm)}\Delta\\ &\leq (1+\varepsilon)\frac{\Delta}{L} + \varepsilon\frac{\Delta}{L}\\ &= (1+2\varepsilon)\frac{\Delta}{L}.\end{aligned}$$

Notice that we can easily apply sequential derandomization (Theorem 3.2) in each part $\ell \in [L]$ to obtain a vector $\chi^\ell \in \{-1,1\}^{[P_\ell]}$ such that for every $i \in [m]$, we have $\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2 \leq \frac{O(\log(m))}{L}\Delta$. This would satisfy guarantee (B) in the lemma we are proving. Furthermore, since each part has only $2n/L = O(\log^{20}(nm))$ variables, and we solve different parts in parallel, this would take depth $\text{poly}(\log(mn))$ and work $\tilde{O}(nnz(A)+n+m))$. But that alone would not provide the more important guarantee (A). To achieve (A), we work somewhat differently by appealing to the Multiplicative Weight Updates (MWU) method, as recalled in Lemma 3.3, in a manner similar to what we did in the proof of Theorem 3.4.

Concretely, we break the parts into $T = \Theta(\log^2(nm)/\varepsilon^2)$ groups, by viewing parts $(t-1)(L/T)+1$ to tL/T as group t, for each $t \in [T]$. Here, we choose T to be a power of 2 so that $T|L$. We will process the groups sequentially, each as one round of MWU. Initially, we set the importance value of each set $i \in [m]$ as $imp(i) = 1$. Then, we process the groups one by one and adjust the importance values as we will describe. Let us zoom in on one round.

Consider round $t \in [T]$ and the corresponding group of L/T parts $P_{(t-1)(L/T)+1}, P_{(t-1)(L/T)+2}, \ldots, P_{t(L/T)}$. We invoke the sequential derandomization of Theorem 3.2 in each of the parts independently, all with the current importance value $imp(i)$ for each set $i \in [m]$. Since each part has at most $2n/L = O(\log^{20}(nm))$ variables, and we solve different parts in parallel, this takes depth $\text{poly}(\log(mn))$. From Theorem 3.2 (setting $M = mn$), we get two properties for each set $i \in [m]$ and each $\ell \in [(t-1)(L/T)+1, t(L/T)]$:

- $\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2 \leq \frac{O(\log(nm))}{L}\Delta$
- $\sum_{i=1}^{m} imp(i)\cdot\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2 \leq (1+\varepsilon)\sum_{i=1}^{m} imp(i)\cdot(1+2\varepsilon)\frac{\Delta}{L}$.

As mentioned before, the first already gives property (B) of the lemma. We next examine property (A). Let us define

$$gap^t(i) = \frac{\sum_{\ell=(t-1)(L/T)+1}^{t(L/T)}\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2}{(1+\varepsilon)(1+2\varepsilon)\frac{\Delta}{T}}$$

From the above two properties, we conclude the following two guarantees about $gap^t(i)$:

- for all $i \in [m]$, $gap^t(i) \in [0, W]$ for $W = O(\log(mn))$,
- $\sum_{i=1}^{m} imp(i)\cdot gap^t(i) \leq \sum_{i=1}^{m} imp(i)$

Hence, the guarantees fit exactly the definition of the oracle in the MWU framework, as recapped in Lemma 3.3. Thus, by running the game for $T = \Theta(W\log m/\varepsilon^2)$ rounds and processing all the groups with importance values updated according to Lemma 3.3, we get the following guarantee: for each $i \in [m]$, we have $\sum_{t=1}^{T} gap^t(i) \leq (1+\varepsilon)T$. That is,

$$\sum_{t=1}^{T}\frac{\sum_{\ell=(t-1)(L/T)+1}^{t(L/T)}\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2}{(1+\varepsilon)(1+2\varepsilon)\frac{\Delta}{T}} = \frac{\sum_{\ell=1}^{L}\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2}{(1+\varepsilon)(1+2\varepsilon)\frac{\Delta}{T}} \leq (1+\varepsilon)T,$$

which implies

$$\sum_{\ell=1}^{L}\left(\sum_{j\in P_\ell} a_{ij}\chi_j^\ell\right)^2 \leq (1+\varepsilon)^2(1+2\varepsilon)\Delta \leq \left(1+\frac{1}{10\log^2 n}\right)\Delta.$$

This proves property (A) and thus concludes the proof of the lemma. □

We can now go back to proving Lemma 4.1.

Proof of Lemma 4.1. We present a proof by induction on n (i.e., creating a recursive algorithm as a function of n). Also, we first describe how the algorithm works and provides the desired guarantees, and then in the end come back to bound its computational depth and work.

If $n < c\log^{30} m$ where c is the constant in Lemma 4.2, then we are in the base case. Then, we simply solve the problem by invoking the sequential derandomization of Theorem 3.2, which provides a vector $\chi \in \{-1,1\}^n$ such that that $(\sum_{j=1}^{n} a_{ij}\chi_j)^2 \leq (C\log m)\cdot\Delta$. Here, C is the constant in Theorem 3.2 which satisfies $C(2-1/\log n) \leq C'$

by choosing $C' = 2C$. Otherwise, we are in the case where we solve the problem via recursion, as we discuss next.

By invoking Lemma 4.2, we spend $\tilde{O}(n + m + nnz(A))$ work and $\text{poly}(\log(mn))$ depth and we get a vector $\bar{\chi} \in \{-1, 1\}^n$ and a partition $[n] = P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L$ for some $L \le n/2$ satisfying the following two properties:

(A) $\sum_{\ell \in L} \left(\sum_{j \in P_\ell} a_{ij}\bar{\chi}_j\right)^2 \le \left(1 + \frac{1}{10\log^2 n}\right)\Delta$, and

(B) for every $i \in [m]$ and $\ell \in [L]$, $\left(\sum_{j \in P_\ell} a_{ij}\bar{\chi}_j\right)^2 \le \frac{O(\log(nm))}{L}\Delta$.

Then, the remaining task is to determine a mixture vector $\chi' \in \{-1, 1\}^L$ so that we can mix the $\bar{\chi}$ solutions of parts P_1 to P_L accordingly, i.e., by setting $\chi_j = \bar{\chi}_j \cdot \chi'_\ell$ where $j \in P_\ell$. This problem is similar to the mixture selection in the proofs of Theorem 3.1 and Theorem 3.4. However, unlike those results, which solve the mixture selection via sequential derandomization, we now have a large number of parts L, which can be as large as $n/2$. Thus, it would be too slow to use sequential derandomization here. Instead, we can invoke recursion.

In particular, for each $i \in [m]$ and each $\ell \in [L]$, define $a'_{i\ell} = \left(\sum_{j \in P_\ell} a_{ij}\bar{\chi}_j\right)$, and define $\Delta' = \left(1 + \frac{1}{10\log^2 n}\right)\Delta$. These satisfy the conditions of our inductive lemma (Lemma 4.1) for $n' = L \le n/2$, in the sense that for each $i \in [m]$, we have $\sum_{\ell=1}^{n'} (a'_{i\ell})^2 \le \Delta'$ and $\max_{i \in [m], \ell \in [L]} (a'_{i\ell})^2 \le \frac{\log^5(n'm)}{n'}\Delta'$. Thus, by invoking Lemma 4.1 recursively/inductively on this instance with $n' \le n/2$ variables, we get a vector $\chi' \in \{-1, 1\}^L$ with the guarantee that for each set $i \in [m]$, we have $(\sum_{\ell=1}^{n'} a'_{i\ell}\chi'_j)^2 = (2 - 1/\log n') \cdot (C' \log m) \cdot \Delta'$. Hence, we can conclude that for every set $i \in [m]$, we have

$$\begin{aligned}
(\sum_{j=1}^{n} a_{ij}\chi_j)^2 &= (\sum_{\ell=1}^{n'} \sum_{j \in P_\ell} a_{ij}\chi_j)^2 = (\sum_{\ell=1}^{n'} \sum_{j \in P_\ell} a_{ij}\bar{\chi}_j\chi'_\ell)^2 \\
&= (\sum_{\ell=1}^{n'} \chi'_\ell \sum_{j \in P_\ell} a_{ij}\bar{\chi}_j)^2 = (\sum_{\ell=1}^{n'} \chi'_\ell a'_{i\ell})^2 \\
&\le (2 - \frac{1}{\log n'}) \cdot (C' \log m) \cdot \Delta' \\
&= (2 - \frac{1}{\log n'})(1 + \frac{1}{10\log^2 n}) \cdot (C' \log m) \cdot \Delta \\
&\le \left((2 - \frac{1}{\log n - 1})(1 + \frac{1}{10\log^2 n})\right) \cdot (C' \log m) \cdot \Delta \\
&\le (2 - \frac{1}{\log n}) \cdot (C' \log m),
\end{aligned}$$

which satisfies the desired output guarantee.

Finally, we discuss the computational depth and work of this algorithm. If we are in the base case of $n = O(\log^{30} m)$, the algorithm follows just by invoking the sequential derandomization of Theorem 3.2, which has $\text{poly}(\log(nm))$ depth and $\tilde{O}(n + m + nnz(A))$ work. Let us now examine the depth in the recursive case. For larger n, we invoked Lemma 4.2, which has depth $\text{poly}(\log(nm))$, and then we solved the remaining (mixture) problem by applying a recursion on an instance with $L \le n/2$ variables. Hence, the depth of the instance with n variables and m sets satisfies the recursion $D(n, m) \le \text{poly}(\log(nm)) + D(n/2, m)$ with the base case of $D(n, m) \le \text{poly}(\log(nm))$ if $n = O(\log^{30} m)$. Hence, $D(n, m) \le \text{poly}(\log(nm))$. A similar argument shows that the work $W(n, m)$ of the instance with n variables and m satisfies $W(n, m) \le \tilde{O}(n+m+nnz(A)) + W(n/2, m)$, which thus shows that the work is bounded by $\tilde{O}(n + m + nnz(A))$. □

4.2 A Polylogarithmic-Depth Algorithm for $O(\sqrt{s \log m})$ Discrepancy

In this subsection, we prove Theorem 1.1, using Lemma 4.1 developed in the previous subsection. For that, we will need an additional helper lemma about the derandomization of pairwise analysis.

Lemma 4.3. *Let $n, m \in \mathbb{N}$ with $m \ge 2$, and let $\{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $[n]$ such that $|S_i| \le k$ for all $i \in [m]$. Also, suppose that for each $i \in [m]$ we are given an importance value $imp(i) \le \mathbb{R}^+$. There exists a deterministic parallel algorithm that can compute a vector $\chi \in \{-1, 1\}^n$, using $\tilde{O}(n + m + \sum_{i=1}^m |S_i|) \cdot \text{poly}(k)$ work and $\text{poly}(k \log(mn))$ depth such that we have $\sum_{i=1}^m imp(i) \cdot (\sum_{j \in S_i} \chi_j)^2 \le \sum_{i=1}^m imp(i) \cdot |S_i|$.*

Proof Sketch. Notice that under a random selection of χ with merely pairwise independence—i.e., for each $j, j' \in [n]$, where $j \ne j'$, and each $(a, b) \in \{-1, 1\}^2$ we have $P[\chi_j = a] = 1/2$ and $P[(\chi_j, \chi_{j'}) = (a, b)] = 1/4$—we have

$$\begin{aligned}
\mathbb{E}[\sum_{i=1}^m imp(i) \cdot (\sum_{j \in S_i} \chi_j)^2] &= \sum_{i=1}^m imp(i) \cdot \mathbb{E}[(\sum_{j \in S_i} \chi_j)^2] \\
&= \sum_{i=1}^m imp(i) \cdot \sum_{j \in S_i} E[(\chi_j)^2] \\
&= \sum_{i=1}^m imp(i) \cdot |S_i|,
\end{aligned}$$

where the penultimate equality relied on the pairwise independence of χ_j and $\chi_{j'}$ for $j \ne j'$. Given this, such a vector χ can be computed deterministically and in parallel, using Luby's method for work-efficient parallel derandomization of pairwise independent analysis [21]. Following this method, the statement follows as a black-box application of Lemma 3.4 in [12]. □

Finally, we prove our unweighted set balancing result.

Theorem 4.4. *Consider $m \ge 2$ subsets $S_1, S_2, \ldots, S_m \subseteq [n]$ and suppose $|S_i| \le s$ for each $i \in [m]$. There is a deterministic parallel algorithm, with $\tilde{O}(n+m+\sum_{i=1}^m |S_i|)$ work and $\text{poly}(\log(mn))$ depth, that computes a vector $\chi \in \{-1, 1\}^n$ such that, for each $i \in [m]$, we have $disc(S_i) = |\sum_{j \in S_i} \chi_j| = O(\sqrt{s \log m})$.*

Proof. If $s \le \text{poly}(\log(mn))$, the result follows from Theorem 2.1. Let us assume that s is larger. Set $\varepsilon = 0.01$. First, we partition the n variables into $L = \frac{s}{\Theta(\log m/\varepsilon^2)}$ parts $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_L = [n]$, where L is a power of two, with the following two properties:

(I) For each part $\ell \in [L]$ and each set $i \in [m]$, we have $|S_i \cap P_\ell| \le (1 + \varepsilon)s/L$.

This can be computed using Lemma 2.3 using work $\tilde{O}(nnz(A) + n + m))$ and depth $\text{poly}(\log(mn))$. We bundle the parts into $T = \Theta(W \log m/\varepsilon^2)$ groups, by viewing parts $(t - 1)(L/T) + 1$ to tL/T as group t, for each $t \in [T]$. Again, T is a power of two, so we have $T|L$. We will process the groups sequentially, each as one round of

MWU. Initially, we set the importance value of each set $i \in [m]$ as $imp(i) = 1$. Then, we process the groups one by one and adjust the importance values as we will describe.

Let us zoom in on one round. Consider round $t \in [T]$ and the corresponding group of L/T parts $P_{(t-1)(L/T)+1}, P_{(t-1)(L/T)+2}, \ldots, P_{t(L/T)}$. We invoke the pairwise derandomization of Lemma 4.3 on the family of sets $S_i \cap P_\ell$ for all $i \in [m]$ and $\ell \in [(t-1)(L/T)+1, t(L/T)]$. Moreover, all subsets of set S_i are given importance value $imp(i)$ inherited from set S_i where $i \in [m]$.

Since in each part $\ell \in [L]$, each set $S_i \cap P_\ell$ has size at most $k = (1+\varepsilon)s/L = O(\log m)$, applying Lemma 4.3 takes depth poly$(\log(mn))$. From Lemma 4.3, we get the following property:

$$\sum_{i=1}^{m} imp(i) \cdot \left(\sum_{\ell=(t-1)(L/T)+1}^{t(L/T)} \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2 \right) \le (1+\varepsilon) \sum_{i=1}^{m} imp(i) \cdot \frac{s}{T}. \tag{1}$$

Let us define

$$gap^t(i) = \frac{\left(\sum_{\ell=(t-1)(L/T)+1}^{t(L/T)} \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2 \right)}{(1+\varepsilon)\frac{s}{T}}$$

From Equation (1), we can conclude that $\sum_{i=1}^m imp(i) \cdot gap^t(i) \le \sum_{i=1}^m imp(i)$. Furthermore, we have that $gap^t(i) \le O(\log m)$. The reason is that $\left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2 \le (|S_i \cap P_\ell|)^2 \le (1+\varepsilon)k(s/L)$, and thus

$$\left(\sum_{\ell=(t-1)(L/T)+1}^{t(L/T)} \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2 \right) \le (1+\varepsilon)k \cdot (s/L) \cdot (L/T)$$
$$= k \cdot (1+\varepsilon)(s/T).$$

So, $gap^t(i) \le k$, which means we have $gap^t(i) \in [0, W]$ for $W = O(\log m)$.

Hence, the guarantees fit exactly the definition of the oracle in the MWU framework, as recapped in Lemma 3.3. Thus, by running the game for $T = \Theta(W \log m/\varepsilon^2)$ rounds and processing all the groups with importance values updated according to Lemma 3.3, we get the following guarantee: for each $i \in [m]$, we have $\sum_{t=1}^T gap^t(i) \le (1+\varepsilon)T$. That is,

$$\sum_{t=1}^{T} \frac{\sum_{\ell=(t-1)(L/T)+1}^{t(L/T)} \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2}{(1+\varepsilon)\frac{s}{T}} = \frac{\sum_{\ell=1}^{L} \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2}{(1+\varepsilon)\frac{s}{T}}$$
$$\le (1+\varepsilon)T,$$

which implies

$$\sum_{\ell=1}^{L} \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2 \le (1+\varepsilon)^2 s \le (1+3\varepsilon)s.$$

There is also the trivial bound that for each $i \in [m]$ and $\ell \in [L]$, we have

$$\left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_j^\ell \right)^2 \le (1+\varepsilon)ks/L \le O(\log m) \cdot s/L.$$

These two conditions prepare us to invoke Lemma 4.1. In particular, for each $i \in [m]$ and each $\ell \in [L]$, define $a'_{i\ell} = \left(\sum_{j \in S_i \cap P_\ell} \bar{\chi}_\ell \right)$, and define $\Delta' = (1+3\varepsilon)\, s$. These satisfy the condition of Lemma 4.1 for $n' = L$, in the sense that for each $i \in [m]$, we have $\sum_{\ell=1}^{n'} (a'_{i\ell})^2 \le \Delta'$ and $\max_{i \in [m], \ell \in [L]} (a'_{i\ell})^2 \le \frac{\log^5(n'm)}{n'} \Delta'$. Thus, by invoking Lemma 4.1, we get a vector $\chi' \in \{-1, 1\}^L$ with the guarantee that for each set $i \in [m]$, we have $(\sum_{\ell=1}^{n'} a'_{i\ell} \chi'_\ell)^2 = 2(C' \log m) \cdot \Delta'$. Hence, we can conclude that for every set $i \in [m]$, we have

$$(\sum_{j \in S_i} \chi_j)^2 = (\sum_{\ell=1}^{n'} \sum_{j \in S_i \cap P_\ell} \chi_j)^2 = (\sum_{\ell=1}^{n'} \sum_{j \in S_i \cap P_\ell} \bar{\chi}_j \chi'_\ell)^2$$
$$= (\sum_{\ell=1}^{n'} \chi'_\ell \sum_{j \in S_i \cap P_\ell} \bar{\chi}_j)^2 = (\sum_{\ell=1}^{n'} \chi'_\ell a'_{i\ell})^2$$
$$\le 2(C' \log m) \cdot \Delta' \le (3C' \log m) \cdot s$$

which satisfies the desired output guarantee. □

REFERENCES

[1] Daniel Anderson and Guy E Blelloch. 2021. Parallel Minimum Cuts in $O(mlog^2n)$ Work and Low Depth. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA).*

[2] Alexandr Andoni, Clifford Stein, and Peilin Zhong. 2020. Parallel approximate undirected shortest paths via low hop emulators. In *ACM SIGACT Symposium on Theory of Computing (STOC).* 322–335.

[3] Sanjeev Arora, Elad Hazan, and Satyen Kale. 2012. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing* 8, 1 (2012), 121–164.

[4] Nikhil Bansal. 2010. Constructive algorithms for discrepancy minimization. In *IEEE Symposium on Foundations of Computer Science (FOCS).* 3–10.

[5] Bonnie Berger and John Rompel. 1989. Simulating $(\log^c n)$-wise independence in NC. In *IEEE Symposium on Foundations of Computer Science (FOCS).* 2–7.

[6] Guy E Blelloch. 1996. Programming parallel algorithms. *Commun. ACM* 39, 3 (1996), 85–97.

[7] Guy E Blelloch, Yan Gu, Julian Shun, and Yihan Sun. 2020. Parallelism in randomized incremental algorithms. *Journal of the ACM (JACM)* 67, 5 (2020), 1–27.

[8] Richard P Brent. 1974. The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM)* 21, 2 (1974), 201–206.

[9] Nairen Cao, Jeremy T Fineman, and Katina Russell. 2020. Efficient construction of directed hopsets and parallel approximate shortest paths. In *ACM SIGACT Symposium on Theory of Computing (STOC).* 336–349.

[10] Laxman Dhulipala, Guy E Blelloch, and Julian Shun. 2021. Theoretically efficient parallel graph algorithms can be fast and scalable. *ACM Transactions on Parallel Computing (TOPC)* 8, 1 (2021), 1–70.

[11] Jeremy T Fineman. 2018. Nearly work-efficient parallel algorithm for digraph reachability. In *ACM Symposium on Theory of Computing (STOC).* 457–470.

[12] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhoň. 2023. Work-Efficient Parallel Derandomization I: Chernoff-like Concentrations via Pairwise Independence. In *IEEE Symposium on Foundations of Computer Science (FOCS).* to appear.

[13] David G Harris. 2019. Deterministic parallel algorithms for bilinear objective functions. *Algorithmica* 81 (2019), 1288–1318.

[14] Joseph JáJá. 1992. An introduction to parallel algorithms. *Reading, MA: Addison-Wesley* 10 (1992), 133889.

[15] Arun Jambulapati, Yang P Liu, and Aaron Sidford. 2019. Parallel reachability in almost linear work and square root depth. In *IEEE Symposium on Foundations of Computer Science (FOCS).* IEEE, 1664–1686.

[16] DR Karger and D Koller. 1994. (De) randomized construction of small sample spaces in NC. In *IEEE Symposium on Foundations of Computer Science (FOCS).* 252–263.

[17] Howard J Karloff and David B Shmoys. 1987. Efficient parallel algorithms for edge coloring problems. *Journal of Algorithms* 8, 1 (1987), 39–52.

[18] Gavriela Freund Lev, Nicholas Pippenger, and Leslie G Valiant. 1981. A fast parallel algorithm for routing in permutation networks. *IEEE transactions on Computers* 100, 2 (1981), 93–100.

[19] Jason Li. 2020. Faster parallel algorithm for approximate shortest path. In *ACM SIGACT Symposium on Theory of Computing (STOC).* 308–321.

[20] Shachar Lovett and Raghu Meka. 2012. Constructive Discrepancy Minimization by Walking on the Edges. In *IEEE Symposium on Foundations of Computer Science (FOCS).* 61–67.

[21] Michael Luby. 1988. Removing randomness in parallel computation without a processor penalty. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 162–173.

[22] Sanjeev Mahajan, Edgar A Ramos, and KV Subrahmanyam. 2001. Solving some discrepancy problems in NC. *Algorithmica* 29, 3 (2001), 371–395.

[23] Rajeev Motwani, Joseph Naor, and Moni Naor. 1989. The probabilistic method yields deterministic parallel algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 8–13.

[24] Rajeev Motwani, Joseph Seffi Naor, and Moni Naor. 1994. The probabilistic method yields deterministic parallel algorithms. *J. Comput. System Sci.* 49, 3 (1994), 478–516.

[25] Prabhakar Raghavan. 1986. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. In *IEEE Symposium on Foundations of Computer Science (FOCS)*.

[26] Václav Rozhoň, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. 2022. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. 1114–1121.

[27] Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. 2022. Undirected (1+ ε)-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms. In *ACM Symposium on Theory of Computing (STOC)*. 478–487.

[28] Joel Spencer. 1977. Balancing games. *Journal of Combinatorial Theory, Series B* 23, 1 (1977), 68–74.

[29] J. Spencer. 1985. Six standard deviations suffice. *Trans. of the American Mathematical Society* 289, 2 (1985), 679–706.

Received 13-NOV-2023; accepted 2024-02-11