

# Faster Algorithms for Fair Max-Min Diversification in $\mathbb{R}^d$

## ABSTRACT

The task of extracting a diverse subset from a dataset, often referred to as maximum diversification, plays a pivotal role in various real-world applications that have far-reaching consequences. In this work, we delve into the realm of fairness-aware data subset selection, specifically focusing on the problem of selecting a diverse set from a large collection of data points (FairDiv).

The FairDiv problem is well-studied in the data management and theory community. In this work, we develop the first constant approximation algorithm for FairDiv that runs in near-linear time using only linear space. In contrast, all previously known constant approximation algorithms run in super-linear time (with respect to  $n$  or  $k$ ) and use super-linear space. Our approach achieves this efficiency by employing a novel combination of the Multiplicative Weight Update method and advanced geometric data structures to implicitly and approximately solve a linear program. Furthermore, we improve the efficiency of our techniques by constructing a coresnet. Using our coresnet, we also propose the first efficient streaming algorithm for the FairDiv problem whose efficiency does not depend on the distribution of data points. Empirical evaluation on million-sized datasets demonstrates that our algorithm achieves the best diversity within a minute. All prior techniques are either highly inefficient or do not generate a good solution.

### ACM Reference Format:

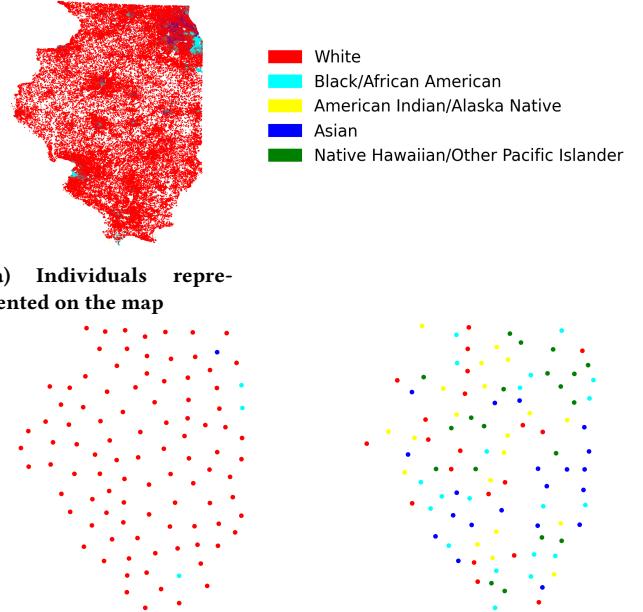
. 2023. Faster Algorithms for Fair Max-Min Diversification in  $\mathbb{R}^d$ . In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

In numerous real-world scenarios, including data summarization, web search, recommendation systems, and feature selection, it is imperative to extract a diverse subset from a dataset (often referred to as maximum diversification). The decisions made in these domains have significant consequences. Therefore, it is crucial to guarantee that the outcomes are not only diverse but also unbiased. For instance, while a primary objective of data summarization is to select a representative sample that encapsulates analogous data points, conventional summarization techniques have been identified to exhibit biases against minority groups, leading to detrimental repercussions. In this work, we study the problem of fairness-aware maximum diversification, where the goal is to choose a diverse set of representative data points, while satisfying a group fairness constraint.

We consider the problem of ensuring group fairness in max-min diversification. We are given a set of items<sup>1</sup>, where each item belongs to one group determined by their sensitive attribute (we refer to it as color). Given a parameter  $k_i$  for each color  $i$ , the goal

<sup>1</sup>The terms item and points are used interchangeably.



(a) Individuals represented on the map  
(b) Max-min diversification ignoring the sensitive attributes  
(c) Representative points with fairness constraints  
Figure 1: Example scenario where each point denotes an individual in the state of Illinois. (b) shows the biased output of max-min diversification technique on this dataset. (c) denotes the fair output of our algorithm.

is to return a subset of items  $S$  such that,  $S$  contains at least  $k_j$  points from each group  $j$ , and the minimum pairwise distance in  $S$  is maximized. We study the problem in the geometric setting where input items are points in  $\mathbb{R}^d$ , for a constant dimension  $d$ . This setting encompasses the majority of realistic scenarios because many datasets are represented as points in the Euclidean space. Even if the input items are not points in  $\mathbb{R}^d$ , it is often the case that the items can be embedded (with low error) in a geometric space with low intrinsic dimension [45, 49, 54]. Although we focus on the Euclidean space, some of our algorithms can be extended to metric spaces with bounded doubling dimension [13, 27, 33, 42].

We motivate the problem with the following example.

*Example 1.1.* Consider a state-court of Illinois, which wants to form a jury consisting of individuals from the state. One of the primary goals of this jury selection task is to identify individuals from neighborhoods that are far apart, i.e. maximize the minimum distance between selected individuals and have representation of people from diverse backgrounds and cultures. This problem has often been studied as a max-min diversification problem [29]. Figure 1 (a) shows the different individuals on the map, where each point is colored based on their sensitive attribute. Running a traditional Max-min diversification algorithm on this dataset returned highly biased results (most of the returned points belong to white, as shown in Figure 1 (b)). Using the fairness constraint, the output contains points from different sensitive groups. The issue of fair

jury selection has been a key focus of courts across the world [2, 3], where many studies have recorded biases in jury and its consequences in their decisions [30].

This example motivates the importance of studying fairness aware variant of Max-Min diversification. We now define the problem formally and then discuss the key contributions.

**Problem Definition.** We are given a set  $P$  of  $n$  points in  $\mathbb{R}^d$  and a set of  $m$  colors  $C = \{c_1, \dots, c_m\}$ . Each point  $p \in P$  is associated with a color  $c(p) \in C$ . For any subset  $S \subseteq P$ , let  $S(c_j) = \{p \in S \mid c(p) = c_j\}$  be the set of points from set  $S$  with color  $c_j$ . We have  $S(c_i) \cap S(c_j) = \emptyset$  for every pair  $i < j$  and  $\bigcup_j S(c_j) = S$ . Let  $\text{div}(S) = \min_{p,q \in S} \|p - q\|_2$  be the diversity function representing the minimum pairwise Euclidean distance among points in  $S$ . For simplicity, throughout the paper we write  $\|p - q\|$  to denote the Euclidean distance of points  $p, q \in \mathbb{R}^d$ .

**DEFINITION 1 (FAIRDIV).** Given a set  $P$  of  $n$  colors in  $\mathbb{R}^d$ , where  $d$  is a constant, a set of colors  $C$ , and integers  $k_1, k_2, \dots, k_m$  such that  $\sum_{j \in [1,m]} k_j = k$ , the goal is to find a set  $S^* \subseteq P$  such that  $\text{div}(S^*)$  is maximized, and for each  $c_j \in C$ , it holds that  $|S^*(c_j)| \geq k_j$ .

Let  $\gamma^* = \text{div}(S^*)$  be the optimum diversity for the FairDiv problem. For a parameter  $\beta > 1$ , we say that an algorithm is a  $\frac{1}{\beta}$ -approximation for the FairDiv problem if it returns a set  $S$  with  $\text{div}(S) \geq \frac{1}{\beta}\gamma^*$ , and  $|S(c_j)| \geq k_j$ , for every  $c_j \in C$ . The approximation ratio is  $\frac{1}{\beta}$ . Finally, we say that an algorithm is a  $\frac{1}{\beta}$ -approximation for the FairDiv problem with  $(1 - \varepsilon)$ -fairness if it returns a set  $S$  with  $\text{div}(S) \geq \frac{1}{\beta}\gamma^*$ , and  $|S(c_j)| \geq \frac{k_j}{1-\varepsilon}$ , for every  $c_j \in C$ .

Sometimes, we might not have access to the entire point set upfront. Instead, we receive the data in a *streaming setting*. This is very common in real applications and modern database systems. By storing only a small number of elements in memory (synopsis), we should maintain the synopsis efficiently under the new insertions, and when needed, we should return a fair and diverse set representing all the items we have encountered in the streaming.

**DEFINITION 2 (SFAIRDIV).** We are given a set of colors  $C$  and integers  $k_1, k_2, \dots, k_m$  such that  $\sum_{j \in [1,m]} k_j = k$ . At a time instance  $t$ , we receive a new point  $p_t$  with color  $c(p_t) \in C$ . Let  $P_t$  be the set of points we have received until time  $t$ . Over any time instance  $t$ , the goal is to maintain a "small" subset of points  $\hat{P}_t \subseteq P_t$ , such that, when needed, a solution for the FairDiv problem in  $P_t$  can be constructed efficiently using only the points stored in  $\hat{P}_t$ .

A data analyst might want to run multiple queries exploring regions of data with fair and diverse representative sets. For example, someone might want to explore neighborhoods in Illinois that are both fair and diverse. We study the *range-query setting*, where the goal is to construct a data structure, such that given a query region, the goal is to return fair and diverse points in the query region in sub-linear time. More formally, we define the next two variations of the FairDiv problem.

**DEFINITION 3. QFairDiv:** Given a set  $P$  of  $n$  colors in  $\mathbb{R}^d$ , and a set of colors  $C$ , the goal is to construct a data structure, such that given a query rectangle  $R$ , and integers  $k_1, k_2, \dots, k_m$  such that  $\sum_{j \in [1,m]} k_j = k$ , return a set  $S^* \subseteq P \cap R$  such that  $\text{div}(S^*)$  is maximized, and for

**Table 1: Table of Notations**

Notation	Meaning
$P$	Point set
$n$	$ P $
$C$	Set of colors
$m$	$ C $
$c(p)$	Color of point $p \in P$
$S(c_j)$	Set of points with color $c_j \in C$ in $S \subseteq P$
$k$	Total output size (lower bound)
$k_j$	Output size having color $c_j$ (lower bound)
$\varepsilon$	approximation error
$S^*$	Optimum solution for the FairDiv problem
$\text{div}(S)$	Diversity of set $S$ (minimum pairwise distance)
$\gamma^*$	$\text{div}(S^*)$
$\mathcal{T}$	BBD-tree
$A$	Matrix representation in MWU method
$h$	probability vector in MWU method

each  $c_j \in C(S^*)$ , it holds that  $|S^*(c_j)| \geq k_j$ . The query time should run in  $o(|P \cap R|)$  time.

## 1.1 Contributions

In Section 3.1, we present the first near-linear time algorithm with respect to all parameters  $n, k$  and  $m$  for the FairDiv problem with constant approximation ratio. Our algorithm is also the first constant approximation algorithm that uses linear space. All previous constant approximation algorithms for FairDiv have super-linear running time and super-linear space with respect to either  $n$  or  $k$ . We formulate a relaxed version of the problem as an LP similar to [6] but instead of constructing and solving the LP explicitly we use the *Multiplicative Weight Update* (MWU) method to find an approximation solution. A straightforward implementation of the MWU approach would still require quadratic time and quadratic space. We combine the MWU approach with advanced geometric data structures in a novel way, taking advantage of the geometry of the input point set to solve the LP in near-linear time and linear space. Furthermore, we apply a rounding technique in near-linear time to derive the final solution. For a constant  $\varepsilon \in (0, 1)$ , we get an  $O(nk)$  time algorithm (ignoring  $\log n$  factors) that returns a  $\frac{1}{2(1+\varepsilon)}$ -approximation for the FairDiv problem. The algorithm uses only  $O(n)$  space. Each fairness constraint is satisfied approximately in expectation: If  $S$  is the returned set then  $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1-\varepsilon}$ , for every  $c_j \in C$ .

If each  $k_j \geq 3(1+\varepsilon)\varepsilon^{-2} \log(2m)$  is sufficiently large, in Section 3.2 we satisfy the fairness constraints approximately with probability  $1 - 1/\delta$ , in time  $O(nk + n \log \frac{1}{\delta})$  and space  $O(n)$ . The approximation factor becomes  $\frac{1}{6(1+\varepsilon)}$ . We note that the fairness and approximation factor are the same as in [6], however our algorithm runs in near-linear time and linear space while the time and space in [6] is super-quadratic.

A set  $G \subseteq P$  is an  $(1 + \varepsilon)$ -coreset if there exists a subset  $\hat{G} \subseteq G$  such that  $\hat{G}$  satisfies all fairness constraints and  $\text{div}(\hat{G}) \geq \gamma^*/(1 + \varepsilon)$ .

It is known how to find an  $(1+\varepsilon)$ -coreset for the FairDiv problem in [6, 52] however the coreset construction and the correctness proof heavily use the greedy Gonzalez [31] algorithm for the  $k$ -center clustering problem. In Section 4, we show a more general result that allows us to construct a coreset faster. In fact, we show that any constant approximation algorithm for the  $k'$ -center algorithm, for  $k' = O(\varepsilon^{-2d}k)$  can be used to construct an  $(1+\varepsilon)$ -coreset for the FairDiv problem. Using our coreset as input to our algorithms, we get a  $\frac{1}{2(1+\varepsilon)}$ -approximation algorithm for the FairDiv problem that runs in  $O(n + mk^2)$  time, for constant  $\varepsilon$  satisfying the fairness constraints approximately in expectation or with high probability.

The generality of our coreset construction allows us to extend our algorithms in different settings. For example, it is known how to maintain a constant approximation for the  $k$ -center clustering in the streaming model efficiently [8, 32, 39]. Using our coreset construction, the streaming algorithm for the  $k$ -center clustering, and our new offline algorithms for FairDiv our new streaming algorithm stores  $O(mk)$  elements in memory, takes  $O(k \log k)$  update time per element for streaming processing, and  $O(mk^2)$  time for post-processing to return a constant approximation for the FairDiv problem satisfying the fairness constraints approximately in expectation or with high probability. This is the first streaming algorithm for FairDiv that does not depend on the spread (maximum over minimum pairwise distance) of the input set. The best known streaming algorithm [50] for the FairDiv problem stores  $O(mk \log \Delta)$  elements in memory, takes  $O(k \log \Delta)$  time per element for streaming processing, and  $O(m^2k^2 \log \Delta) = O(k^4 \log \Delta)$  time for post-processing to return a constant approximation for the FairDiv problem. The parameter  $\Delta$  is the spread of the input set. In the range-query setting, we can use the near-linear space data structure from [5] and our coreset construction to design a data structure of  $O(n \log^{d-1} n)$  space in  $O(n \log^{d-1} n)$  time, such that given a query rectangle  $R$ , a constant  $\varepsilon$ , and parameters  $k_1, \dots, k_m$ , it returns a set  $S \subseteq P \cap R$  in  $O(mk^2)$  time, ignoring  $\log n$  factors, such that  $S$  is a  $\frac{1}{2(1+\varepsilon)}$ -approximation for the FairDiv problem in  $P \cap R$ , and  $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1+\varepsilon}$ , for every color  $c_j \in C$ .

In Section 6 we run extensive experiments on real datasets showing that our new algorithm returns diverse and fair results faster than the other baselines. More specifically, among algorithms that return fair results with similar diversity, our algorithm is always the fastest one. When another baseline is faster than our method it is always the case that the diversity of the set it returns is significantly worse than the diversity of the results returned by our algorithm.

## 2 PRELIMINARIES

**LP-based algorithm for FairDiv.** We first review the LP-based algorithm presented in [6] to find a solution for the FairDiv problem. They run a binary search over all possible pairwise distances. For a distance  $\gamma$ , they solve the following feasibility problem (LP).

$$(LP1) \quad \sum_{p_i \in P(c_j)} x_i \geq k_j \quad \forall c_j \in C \quad (1)$$

$$\sum_{p_i \in P \cap \mathcal{B}(p, \gamma/2)} x_i \leq 1 \quad \forall p \in P \quad (2)$$

$$1 \geq x_i \geq 0, \quad \forall p_i \in P \quad (3)$$

$\mathcal{B}(p, \gamma/2)$  represents a ball with center  $p$  and radius  $\gamma/2$ . If (LP1) is infeasible, they try smaller values of  $\gamma$ . Otherwise, they try larger values of  $\gamma$ . Then they describe a rounding technique to construct a valid solution for the FairDiv problem. Let  $\hat{x}$  be the solution of (LP1) corresponding to largest  $\hat{\gamma}$  that the LP was feasible. They generate a random ordering  $\sigma$  of  $[n]$  as follows:  $\sigma(t)$  is randomly chosen from  $R_t = [n] \setminus \{\sigma(1), \dots, \sigma(t-1)\}$  such that a number  $b \in R_t$  is chosen with probability  $\Pr[\sigma(t) = b] = \frac{\hat{x}_b}{\sum_{\ell \in R_t} \hat{x}_\ell}$ . After generating the ordering  $\sigma$ , they construct the output set  $S \subseteq P$  including the point  $p_j \in S$  if and only if  $\sigma(j) \leq \sigma(\ell)$  for all  $p_\ell \in P \cap \mathcal{B}(p_j, \gamma/2)$ . They show the following theorem.

**THEOREM 2.1 ([6]).** *The algorithm described above returns a set  $S$  with  $\text{div}(S) \geq \gamma^*/2$  such that for each color  $c_j$ ,  $\mathbb{E}[|S(c_j)|] \geq k_j$ .*

Indeed, this LP-based algorithm returns a constant approximation for the FairDiv problem. However, both the space and the running time of the algorithm is  $\Omega(n^2)$ . Specifically, it needs  $O(n^2)$  space only to represent (LP1) because  $|P \cap \mathcal{B}(p, \gamma/2)| = O(n)$  for every point  $p \in P$ . Solving each instance of (LP1) takes  $O(n^{2.37})$  time [35]. The rounding algorithm is executed in  $O(n^2)$  time. Overall, the running time is  $O(n^{2.37} \log n)$ .

**Coreset.** In [6], the authors also described a coreset for the FairDiv problem. In particular they construct a set  $G \subseteq P$  of size  $O(\varepsilon^{-d}km) = O(\varepsilon^{-d}k^2)$ . The coreset  $G$  can be constructed by running the well known Gonzalez algorithm [31] for the  $k'$ -center clustering problem in each set  $P(c_j)$  independently, for  $k' = \varepsilon^{-d}k$ . Let  $G_j$  be the solution of the Gonzalez's algorithm in  $P(c_j)$ . The coreset  $G$  is defined as  $G = \bigcup_{c_j \in C} G_j$ . It holds that  $|G| = O(\varepsilon^{-d}km)$  and it is constructed in  $O(\varepsilon^{-d}nk)$  time. If we combine the results in Theorem 2.1 with the coreset construction for a constant  $\varepsilon$ , we get, an algorithm that returns a set  $S$  with  $\text{div}(S) \geq \frac{1}{2(1+\varepsilon)}\gamma^*$  in  $O(kn + k^{4.74})$  time such that for each color  $c_j$ ,  $\mathbb{E}[|S(c_j)|] \geq k_j$ .

**Diversity with high probability.** All the results above, return a set  $S$  that satisfies fairness in expectation. The authors in [6] extended the results to hold with probability at least  $1 - 1/n$ , also called with high probability. Given  $\hat{x}$ , the solution from (LP1), they convert it to a solution  $\hat{y}$  for the following (non-linear) feasibility problem.

$$(LP2) \quad \sum_{p_i \in P(c_j)} y_i \geq k_j, \quad \forall c_j \in C \quad (4)$$

$$\sum_{p_i \in P \cap \mathcal{B}(p, \gamma/6)} y_i \leq 1, \quad \forall p \in P \quad (5)$$

$$y_i \geq 0, \quad \forall p_i \in P \quad (6)$$

$$y_i > 0 \text{ and } y_\ell > 0 \Rightarrow \|p_i - p_\ell\| \geq \frac{\gamma}{3}, \quad (7)$$

$$\forall p_i, p_\ell \in P(c_j), \forall c_j \in C$$

If  $k_j \geq 3\varepsilon^{-2} \log(2m)$  for every  $c_j \in C$ , the authors showed that, if they apply the same rounding technique as in the previous case they return a set  $S$  such that  $\text{div}(S) \geq \gamma^*/6$ , where  $|S(c_j)| \geq \frac{k_j}{1-\varepsilon}$  for every  $c_j \in C$ , with probability at least  $1 - 1/n$ . Unfortunately, they still need to solve (LP1) to derive this result. Even without solving (LP1), the algorithm they propose to convert the solution from (LP1)  $\hat{x}$  to a solution for (LP2)  $\hat{y}$  takes  $\Omega(n^2)$  time. Hence, the overall running time is super-quadratic. If the coreset is used, then

we get a  $\frac{1}{6(1+\epsilon)}$ -approximation algorithm in  $O(kn + (mk)^2 \cdot 3^7)$  time satisfying the fairness constraints with high probability.

### Geometric data structures.

*BBD-tree.* The main geometric data structure we use is the *BBD-tree* [10, 11], which is a variant of the quadtree [28]. A BBD-tree  $\mathcal{T}$  on a set  $P$  of  $n$  points in  $\mathbb{R}^d$  is a binary tree of height  $O(\log n)$  with exactly  $n$  leaves. Let  $\square$  be the smallest axis-aligned hypercube containing  $P$ . Each node  $u$  of  $\mathcal{T}$  is associated with a region  $\square_u$ , which is either a rectangle or a region between two nested rectangles, and a subset  $P_u \subseteq P$  of points that lie inside  $\square_u$ . Notice that  $\square_{root} = \square$ . If  $|P_u| = 1$ , then  $u$  is a leaf. If  $|P_u| > 1$ , then  $u$  has two children, say,  $w$  and  $z$ , and  $\square_w$  and  $\square_z$  partition  $\square_u$ . Regions associated with the nodes of  $\mathcal{T}$  induce a hierarchical partition of  $\mathbb{R}^d$ . A BBD tree has  $O(n)$  space and can be constructed in  $O(n \log n)$  time. Given a parameter  $\epsilon \in (0, 1)$  and a ball  $\mathcal{B}(x, r)$  in  $\mathbb{R}^d$ , the BBD-tree returns a set of nodes  $U(x, r) = \{u_1, \dots, u_\kappa\}$  from  $\mathcal{T}$  (also called *canonical nodes*) for  $\kappa = O(\log n + \epsilon^{-d})$  in  $O(\log n + \epsilon^{-d})$  time such that  $\square_{u_i} \cap \square_{u_j} = \emptyset$  for every pair  $1 \leq i < j \leq \kappa$ , and  $\mathcal{B}(x, r) \subseteq \bigcup_{1 \leq i \leq \kappa} \square_{u_i} \subseteq \mathcal{B}(x, (1 + \epsilon)r)$ . In other words, the BBD tree returns a set of rectangles (with or without holes) such that the union of them completely cover the ball  $\mathcal{B}(x, r)$  and might cover some part of  $\mathcal{B}(x, (1 + \epsilon)r) \setminus \mathcal{B}(x, r)$ . By reporting all points  $P_{u_i}$  for  $i \leq \kappa$ , the BBD tree can be used for reporting all points in  $P \cap \mathcal{B}(x, r)$  along with some points from  $P \cap (\mathcal{B}(x, (1 + \epsilon)r) \setminus \mathcal{B}(x, r))$ .

*WSPD.* Using a quadtree [28], someone can get a *Well Separated Pair Decomposition* (WSPD) [16, 33] in  $P \in \mathbb{R}^d$ . In  $O(\epsilon^{-d} n \log n)$  time, we can construct a list  $\mathcal{L} = \{L_1, \dots, L_z\}$  of  $z = O(\epsilon^{-d} n)$  distances, such that for every pair  $p, q \in P$ , there exists  $L_j \in \mathcal{L}$  such that  $(1 - \epsilon)||p - q|| \leq L_j \leq (1 + \epsilon)||p - q||$ .

**Multiplicative Weight Update (MWU) method.** The MWU method is used to solve the following linear feasibility problem.

$$\exists x \in \mathcal{P} : Ax \leq b, \quad (8)$$

where  $A \in \mathbb{R}^{m' \times n'}$ ,  $x \in \mathbb{R}^{n'}$ ,  $b \in \mathbb{R}^{m'}$ ,  $Ax \geq 0$ ,  $b \geq 0$ , and  $\mathcal{P}$  is a convex set in  $\mathbb{R}^{n'}$ . Intuitively,  $\mathcal{P}$  captures the “easy” constraints to satisfy while  $A$  represents the “hard” constraints to satisfy. The authors in [9] describe an iterative algorithm using a simple ORACLE. Let ORACLE be a black-box procedure that solves the following single linear constraint for a probability vector  $h \in \mathbb{R}^{m'}$ .

$$\exists x \in \mathcal{P} : h^\top Ax \leq h^\top b. \quad (9)$$

The ORACLE decides if there exists an  $x$  that satisfies the single linear constraint. Otherwise, it returns that there is no feasible solution. A  $\rho$ -ORACLE is an ORACLE such that whenever ORACLE manages to find a feasible solution  $\hat{x}$  to problem (9), then  $A_i \hat{x} - b_i \in [-1, \rho]$  for each constraint  $i \in [m']$ , where  $A_i$  is the  $i$ -th row of  $A$ .

The algorithm starts by initializing  $h$  to a uniform probability vector with value  $1/m'$ . In each iteration the algorithm solves Equation (9). If (9) is infeasible, we return that the original feasibility problem in Equation (8) is infeasible. Let  $x^{(t)}$  be the solution of the problem in Equation (9) in the  $t$ -th iteration of the algorithm. Let  $\delta_i = \frac{1}{\rho}(A_i x^{(t)} - 1)$ . We update  $h[i] = (\delta_i \cdot \epsilon/4 - 1)h[i]$ , where  $h[i]$  is the  $i$ -th element of vector  $h$ . We continue in the next iteration defining a new feasibility problem with respect to the new probability vector  $h$ . After  $T = O(\rho \log(m')/\epsilon^2)$  iterations, if every oracle was feasible, we return  $x^* = \frac{1}{T} \sum_{t=1}^T x^{(t)}$ . Otherwise, if an oracle was

infeasible, they argue that the initial problem is infeasible. In [9] they prove the following theorem.

**THEOREM 2.2 ([9]).** *Given a feasibility problem as defined above, a parameter  $\epsilon$ , and a  $\rho$ -ORACLE, there is an algorithm which either finds an  $x$  such that  $\forall i, A_i x_i \leq b_i + \epsilon$  or correctly concludes that the system is infeasible. The algorithm makes  $O(\rho \log(m')/\epsilon^2)$  calls to the ORACLE with an additional processing time of  $O(m'n')$  per call.*

## 3 EFFICIENT ALGORITHMS FOR FAIRDIV

In this section we propose two efficient algorithms for the FairDiv problem. The first one guarantees approximate fairness in expectation, while the second one guarantees approximate fairness with high probability.

### 3.1 Diversity in expectation

**High-level idea.** We use the LP defined by [6] ((LP1) in Section 2) to represent the FairDiv instance. Let  $y$  be a parameter which is close to the optimum  $y^*$ . We explain later how we can find such a parameter running binary search efficiently without computing all possible distances. Instead of constructing and using an LP solver to solve the LP explicitly, we use the Multiplicative Weight Update (MWU) method to find an approximation solution to it. A straightforward implementation of the MWU approach would still require quadratic time and quadratic space. We combine the MWU approach with advanced geometric data structures (BBD tree) in a novel way taking advantage of the geometry of the input point set that allows us to solve the LP in near-linear time and linear space with respect to both  $n$  and  $k$ . Furthermore, we apply a rounding technique in near-linear time to derive the final solution.

Assume that we want to solve (LP1) using the MWU method as described in Section 2. A straightforward mapping is the following:  $\mathcal{P}$  consists of the convex set of  $x \geq 0$  constraints (3) (trivial constraints), while  $A$  is the matrix with  $m' = m + n = O(n)$  and  $n' = n = O(n)$  including all constraints from (1) and (2). For our problem definition the vector  $b$  is the vector having 1 in each coordinate. A straightforward implementation of the ORACLE requires  $O(n^2)$  time because every point might belong to  $O(n)$  balls of radius  $y/2$ . In fact, even to represent matrix  $A$  we might need  $O(n^2)$  time and space. Overall, the multiplicative weights algorithm runs in  $O(n^2 k^2 \log(m)/\epsilon^2)$  time. While the running time is asymptotically faster than solving the LP exactly using an LP solver, it is still quadratic on  $n$ . Next we show how to use geometric tools to improve the running time to near-linear.

Before we continue with our method, we note that there has been a lot of work related to implicitly solving special classes of LPs in computational geometry using the MWU method [20, 22, 25]. Despite the geometric nature of our problem, to the best of our knowledge, all previous (geometric) techniques that implicitly solve an LP do not extend to our problem. For example, in [22] they describe algorithms for geometric problems using the MWU method with running time that depends linearly on the number of non-zero numbers in matrix  $A$ . In our problem definition there might be  $O(n^2)$  non-zero numbers in  $A$ , hence these algorithms cannot be used to derive near-linear time algorithms for our problem. In [20, 22] they also provide near-linear time constant approximation algorithms for some geometric problems such as the geometric set

cover, covering points by disks, or the more relevant to our problem, maximum independent set of disks. However there are three major differences with our problem definition: i) These algorithms work only in  $\mathbb{R}^2$  or  $\mathbb{R}^3$ . We propose approximation algorithms for any constant dimension  $d$ . ii) They do not consider fairness constraints and it is not clear how to extend their methods to satisfy fairness constraints. iii) The optimization problem in FairDiv is different from the optimization problems they study.

**Intuition and definitions for new algorithm.** There are three key ideas for our new algorithm. i) Instead of considering that the only trivial constraints are  $x_i \geq 0$ , we assume that Constraints (1)  $\sum_{p_i \in P(c_j)} x_i \geq k_j$  are also trivial. ii) We slightly change the Constraints (2) that define the matrix  $A$  in the MWU. In particular, for a point  $p$  we define a set  $S_p^\varepsilon \subseteq P$  denoting its “neighboring” points. In particular, for a point  $p$ ,  $S_p^\varepsilon$  contains all points within distance  $\frac{\gamma}{2(1+\varepsilon)}$  from  $p$  and might contain some points within distance  $\gamma/2$  from  $p$ . We define  $S_p^\varepsilon$  more formally later. Instead of having  $\sum_{p_i \in P \cap \mathcal{B}(p, \gamma/2)} x_i \leq 1$  in Constraints (2) we have:

$$\sum_{p_i \in S_p^\varepsilon} x_i \leq 1 \quad \forall p \in P. \quad (10)$$

Matrix  $A$  is a binary square matrix, having one row for every point  $p \in P$ . The value  $A[\ell, i] = 1$  if  $p_i \in S_{p_\ell}^\varepsilon$ , otherwise it is 0. iii) We do not construct the matrix  $A$  explicitly. Likewise, we do not construct the sets  $S_p^\varepsilon$  explicitly. Instead, we use geometric data structures to implicitly represent  $A$  and execute ORACLE in near-linear time.

Let  $\mathcal{P}$  be the set that consists of  $x_i \geq 0$  for each  $p_i \in P$  in Constraints (3) and  $\sum_{p_i \in P(c_j)} x_i \geq k_j$  for each  $j \in [m]$  in Constraints (1). The set  $\mathcal{P}$  is convex because it is defined as the intersection of  $m+n$  halfspaces in  $\mathbb{R}^n$ . Hence, it is valid to use the MWU method. Next we focus on the main problem we have to solve, i.e., describe ORACLE. More specifically, we are given the feasibility problem  $h^\top Ax \leq h^\top b$ , where  $x \in \mathcal{P}$  as defined above. The matrix  $A$  is defined over the group of Constraints (10). Recall that  $b \in \mathbb{R}^n$  and  $b = \{1, \dots, 1\}$ . Hence it turns out that the feasibility problem we need to solve is  $h^\top Ax \leq \sum_\ell h[\ell] \Leftrightarrow h^\top Ax \leq 1$  such that  $x \in \mathcal{P}$ .

**Efficient algorithm for ORACLE.** Notice that  $h^\top Ax$  can be written as  $\sum_{p_i \in P} \alpha_i x_i$ , for some real coefficients  $\alpha_i$ . Intuitively, for every color  $c_j$ , our goal is to find the  $k_j$  points with color  $c_j$ , having the smallest coefficients  $\alpha_i$ . Our algorithm first finds all coefficients  $\alpha_i$  and then it chooses the  $k_j$  smallest from each color  $c_j$ . In that way, we find the solution  $\bar{x}$  that minimizes  $h^\top A\bar{x}$  for  $\bar{x} \in \mathcal{P}$ . Finally, we check whether  $h^\top A\bar{x} \leq 1$ .

We are given a probability vector  $h \in \mathbb{R}^n$ . Each value  $h[i] \in h$  corresponds to the *weight* of the  $i$ -th row of matrix  $A$ . In other words each point  $p_\ell \in P$  is associated with a weight  $h[\ell]$ . We build a modified version of the BBD tree  $\mathcal{T}$  over the weighted points  $P$ . Let  $\mathcal{T}$  be the tree constructed as described in Section 2 over the set of points  $P$ . For every node  $u$  of  $\mathcal{T}$ , we store  $u_s = \sum_{p_\ell \in \square_u \cap P} h[\ell]$ , i.e.,  $u_s$  is the sum of all weights of points stored in the subtree rooted at  $u$ . For a point  $p \in P$  and a radius  $r$ , let  $\mathcal{T}(p, r)$  be a range query returning  $\kappa = O(\log n + \varepsilon^{-d})$  canonical nodes  $\mathcal{U}(p, r) = \{u_1, \dots, u_\kappa\}$  of  $\mathcal{T}$  such that, i) for a point  $q \in P$ , if  $\|p - q\| \leq r$ , then  $\exists u_i \in \mathcal{U}(p, r)$  such that  $q \in \square_{u_i} \cap P$ , and ii) if  $q \in \square_{u_i} \cap P$  then  $\|p - q\| \leq (1 + \varepsilon)r$ . The data structure  $\mathcal{T}$  has  $O(n)$  space and can be constructed in  $O(n \log n)$  time.

The properties of the BBD can be used to formally define  $S_p^\varepsilon$ . Recall that we do not construct explicitly  $S_p^\varepsilon$ , otherwise we would need quadratic time. The set  $S_p^\varepsilon$  is defined as  $S_p^\varepsilon = \{p \in \square_{u_i} \cap P \mid u_i \in \mathcal{U}(p, \frac{\gamma}{2(1+\varepsilon)})\}$ , i.e., the set of points in the canonical nodes returned by query  $\mathcal{T}(p, \frac{\gamma}{2(1+\varepsilon)})$ .

Using our modified BBD tree  $\mathcal{T}$ , we show how to check whether there exists  $\bar{x}$  such that  $h^\top A\bar{x} \leq 1$  and  $\bar{x} \in \mathcal{P}$ . For each  $p_i \in P$  we run the query  $\mathcal{T}(p_i, \frac{\gamma}{2(1+\varepsilon)})$  and we get the set of canonical nodes  $\mathcal{U}_{p_i} := \mathcal{U}(p_i, \frac{\gamma}{2(1+\varepsilon)})$ . For each node  $u \in \mathcal{U}_{p_i}$ , we update  $u_s \leftarrow u_s + h[i]$ . After we consider all points, we revisit each point  $p_i \in P$  and continue as follows: We initialize a weight  $w_i = 0$ . We start from the leaf node that contains  $p_i$  and we traverse  $\mathcal{T}$  bottom up until we reach the root of the BBD tree. Let  $v$  be a node we traverse; we update  $w_i = w_i + v_s$ . After computing all values  $w_i$ , for each color  $c_j$  we find the  $k_j$  points from  $P(c_j)$  with the smallest weights  $w_i$ . Let  $P_j$  be these points. For each  $p_i \in P_j$  we set  $\bar{x}_i = 1$ . Otherwise, if  $p_i \notin P_j$  and  $c(p_i) = c_j$ , we set  $\bar{x}_i = 0$ . If  $\sum_{p_i \in P} w_i \bar{x}_i \leq 1$  the oracle returns  $\bar{x}$  as a feasible solution. Otherwise, it returns that there is no feasible solution.

*Proof of correctness.* We show that the ORACLE we design is correct. We first show that  $\alpha_i = w_i$  so  $h^\top Ax = \sum_{p_i \in P} w_i x_i$ . Assume that  $h^\top Ax = \sum_{p_i \in P} \alpha_i x_i$ , for the real coefficients  $\alpha_i$ . By definition, each  $\alpha_i$  is defined as  $\alpha_i = \sum_{p_\ell \in P} h[\ell] \cdot I(p_i \in S_{p_\ell}^\varepsilon)$ , where  $I(p_i \in S_{p_\ell}^\varepsilon) = 1$  if  $p_i \in S_{p_\ell}^\varepsilon$  and 0 otherwise. If  $p_i \in S_{p_\ell}^\varepsilon$  then by definition  $\alpha_i$  contains a term  $h[\ell]$  and there exists a node  $u \in \mathcal{U}_{p_\ell}$  such that  $p_i \in \square_u \cap P$ , i.e.,  $p_i$  lies in a leaf node of the subtree rooted at  $u$ . Starting from the leaf containing  $p_i$ , our algorithm will always visit the node  $u$ , and since  $u \in \mathcal{U}_{p_\ell}$  we have that  $h[\ell]$  is a term in the sum  $u_s$  so by updating  $w_i = w_i + u_s$  we include the term  $h[\ell]$  in the weight  $w_i$ . Overall, we have that  $w_i = \alpha_i$  and our algorithm finds all the correct coefficients in the linear function  $h^\top Ax$ . Then we focus on minimizing the sum  $\sum_{p_i \in P} w_i x_i$  satisfying  $x \in \mathcal{P}$ . For each color  $c_j$  we should satisfy  $\sum_{p_i \in P(c_j)} x_i \geq k_j$ . We can re-write  $\sum_{p_i \in P} w_i x_i = \sum_{c_j \in C} \sum_{p_i \in P(c_j)} w_i x_i$ . Notice that the partial sum  $\sum_{p_i \in P(c_j)} w_i x_i$  is minimized for  $x \in \mathcal{P}$  setting  $\bar{x}_i = 1$  for the  $k_j$  smallest factors  $w_i$  in the partial sum. Every point has a unique color, so there is no point  $p_i$  that belongs in two different partial sums. Repeating the same argument for each color  $c_j \in C$ , we conclude that indeed our algorithm finds the minimum value of  $\sum_{p_i \in P} w_i x_i$  satisfying  $x \in \mathcal{P}$ . Overall, our algorithm correctly returns whether the feasibility problem  $h^\top Ax \leq 1$  for  $x \in \mathcal{P}$  is feasible or infeasible.

*Running time.* For each new probability vector  $h$  we construct  $\mathcal{T}$  in  $O(n \log n)$  time. For each point  $p_i$  we find  $\mathcal{U}_{p_i}$  in  $O(\log n + \varepsilon^{-d})$  time. Furthermore, the height of  $\mathcal{T}$  is  $O(\log n)$  so for each point  $p_i$  we need additional  $O(\log n)$  time to compute  $w_i$ . After computing the weights, we find the smallest  $k_j$  of them of each color in linear time. Hence, our oracle runs in  $O(n \log n + n \varepsilon^{-d})$  time.

**Update  $h$  efficiently.** Next, we describe how we can update  $h$  efficiently at the beginning of each iteration. Let  $\bar{x}$  be the solution of the oracle in the previous iteration. Let  $\delta_\ell = |A_\ell \bar{x} - b_\ell| = |A_\ell \bar{x} - 1|$ , where  $A_\ell$  is the  $\ell$ -th row of  $A$  ( $\ell$ -th constraint in (10)). In [9] they update each  $h[\ell]$  in constant time after computing  $\delta_\ell$ . In our case, if we try to calculate all  $\delta_\ell$ 's with a trivial way we would need  $\Omega(nk)$  time leading to a  $\Omega(nk^2)$  time overall algorithm. We show a faster way to calculate all  $\delta_\ell$ 's. Our method for ORACLE sets  $k$  variables

$\bar{x}$  to 1. For each  $p_\ell \in P$  the goal is to find  $A_\ell \bar{x} = \sum_{p_i \in S_{p_\ell}^\epsilon} \hat{x}_i = \sum_{p_i \in S_{p_\ell}^\epsilon, \hat{x}_i > 0} \hat{x}_i$ . We modify  $\mathcal{T}$  as follows. For each node  $u \in \mathcal{T}$  we define the variable  $u_w = 0$ . For each  $p_i$  with  $\hat{x}_i > 0$  we start from the leaf containing  $p_i$ , and we visit the tree bottom up until we reach the root. For each node  $v$  we encounter, we update  $v_w = v_w + \hat{x}_i$ . After the modification of  $\mathcal{T}$ , for each constraint/point  $p_\ell$  we run the query  $\mathcal{T}(p_\ell, \frac{\gamma}{2(1+\epsilon)})$  and we get the set of canonical nodes  $\mathcal{U}_{p_\ell}$ . We compute  $R_\ell \leftarrow \sum_{u \in \mathcal{U}_{p_\ell}} u_w = \sum_{p_i \in S_{p_\ell}^\epsilon} \hat{x}_i = A_\ell \bar{x}$ . We return  $\delta_\ell = |R_\ell - 1|$ . The correctness follows by observing that the coefficient of  $p_i$ 's variable in the  $\ell$ -th row of  $A$  is 1 if and only if  $p_i \in S_{p_\ell}^\epsilon$ . We need  $O(n)$  time to compute all values  $v_w$  by traversing the tree  $\mathcal{T}$  bottom up. Then for each  $p_\ell$  we run a range query on  $\mathcal{T}$  so we need  $O(\log n + \epsilon^{-d})$ . Overall, we can update the vector  $h$  in  $O(n(\log n + \epsilon^{-d}))$  time.

**Overall algorithm.** Using our ORACLE we can run the algorithm of Theorem 2.2 to solve the modified linear feasibility problem. Each solution  $\bar{x}$  of our ORACLE sets  $k$  variables to 1. Hence, for each inequality it holds that  $A_i \bar{x} - b_i \leq k - 1$  and  $A_i \bar{x} - b_i \geq -1$ . Similarly, we can write  $A_i \bar{x} \leq k$  and  $A_i \bar{x} \geq 0$  since  $b_i = 1$ . So we can set  $\rho = k$ . Overall, our algorithm finds a solution  $\hat{x}$  such that  $\hat{x} \in \mathcal{P}$  and  $A_i \hat{x} \leq 1 + \epsilon$  for each  $i \leq n$ . We apply  $O(\frac{k \log n}{\epsilon^2})$  calls to the oracle and each oracle takes  $O(n(\log n + \epsilon^{-d}))$  time. In the end of each iteration we spend  $O(n(\log n + \epsilon^{-d}))$  time to update the probability vector  $h$ . Hence, the overall running time is  $O\left(n \left(\frac{k \log n}{\epsilon^2}\right) (\log n + \epsilon^{-d})\right)$ .

**Rounding.** The real vector  $\hat{x}$  we get satisfies the following constraints.

$$\sum_{p_i \in P(c_j)} \hat{x}_i \geq k_j \quad \forall c_j \in C \quad (11)$$

$$\sum_{p_i \in S_p^\epsilon} \hat{x}_i \leq 1 + \epsilon \quad \forall p \in P \quad (12)$$

$$1 \geq \hat{x}_i \geq 0, \quad \forall p_i \in P \quad (13)$$

We follow a modified version of the randomized rounding from [6] to round  $\hat{x}$  and return a set  $S \subseteq P$  as the solution to the FairDiv problem. Our rounding method has major differences from the rounding in [6] (also briefly described in Section 2) because, i) the Constraints (2) are quite different from the Constraints (12) since the latter are satisfied with an additive error  $\epsilon$ , and their sum is over the set  $S_p^\epsilon$ , and ii) the rounding technique in [6] is executed in quadratic time with respect to the number of points. We propose a near-linear time algorithm to execute the rounding.

Before we describe the actual rounding algorithm we describe a modified BBD tree that we are going to use to sample efficiently. For every point  $p_i \in P$ , we define its weight  $\hat{x}_i$ . Let  $\Xi$  be a BBD tree constructed over a weighted set  $P$ . We modify  $\Xi$  so that we can sample a point  $p_i$  with probability  $\frac{\hat{x}_i}{\sum_{j \in F} \hat{x}_j}$ , where  $F \subset P$  is a subset of  $P$ . For each node  $u$  of  $\Xi$ , we store the value  $u_s = \sum_{p_\ell \in \square_u \cap P} \hat{x}_\ell$ , i.e., sum of  $\hat{x}_i$ 's of all points stored in the subtree of  $u$ . Initially  $F = P$ . We sample as follows: Assume any node (initially the root)  $u$  having two children  $v, e$ . We visit  $v$  with probability  $v_s / (v_s + e_s)$  and we visit  $e$  with probability  $e_s / (v_s + e_s)$ . When we insert a point  $p_i$  in  $F$ , we start from the leaf node that stores  $p_i$  and we visit the tree bottom up updating the values  $u_s$  of the nodes  $u$  we traverse.

In particular, we set  $u_s \leftarrow u_s - \hat{x}_i$ . In this way, we "remove" the weight of point  $p_i$  from the tree and it is not considered in the next iterations of sampling. It is straightforward to see that this procedure guarantees that each point  $p_i$  is selected with probability  $\frac{\hat{x}_i}{\sum_{j \in F} \hat{x}_j}$ . In order to make sure that we do not select two nearby points, for every node  $u$  of  $\Xi$  we also set a boolean variable  $u_b = 1$ . If  $u_b = 1$  it means that our algorithm has not sampled a point that lies in  $\square_u \cap P$ . If  $u_b = 0$  it means that we have already sampled a point in  $\square_u \cap P$  in a previous iteration so we should not re-consider node  $u$  to get a new sample.

Let  $S = \emptyset$  be the set of points that we return for the FairDiv problem. Let also  $F = P$  represent the set of points that we can sample from, as described in the previous paragraph. Using  $\Xi$  we sample a point  $p_i$  with probability  $\frac{\hat{x}_i}{\sum_{p_\ell \in F} \hat{x}_j}$ . We update  $F \leftarrow F \setminus \{p_i\}$ . Next, we run a query  $\Xi(p_i, \frac{\gamma}{2(1+\epsilon)})$  and we get a set of  $O(\log n + \epsilon^{-d})$  canonical nodes  $\mathcal{U}_{p_i}$ . For each node  $u \in \mathcal{U}_{p_i}$ , we check whether  $u_b = 1$ . If  $u_b = 1$  for every  $u \in \mathcal{U}_{p_i}$ , then we insert  $p_i$  in  $S$ . Otherwise, we do not insert it in  $S$ , i.e., we have already sampled a point from  $S_{p_i}^\epsilon$  in a previous iteration. Next, starting from the leaf node that contains  $p_i$  we traverse the tree bottom up until we reach the root node. For each node  $v$  we encounter we set  $v_q = 0$ . After we sample all points in  $P$ , we return the set  $S$ . With this implementation we create an ordering  $\sigma$  of the points in  $P$ , where  $\sigma(t)$  denotes the point sampled in the  $t$ -th iteration. Notice that we do not actually add in  $S$  one point in every iteration.

**LEMMA 3.1.** *The minimum pairwise distance in  $S$  is  $\frac{\gamma}{2(1+\epsilon)}$  and for each color  $c_j \in C$  it holds that  $\mathbb{E}[|S(c_j)|] \geq \frac{k_j}{1+\epsilon}$ .*

**PROOF.** First, we show that the minimum pairwise distance in  $S$  is  $\frac{\gamma}{2(1+\epsilon)}$ . Let  $p_i, p_\ell$  be a pair of points with distance less than  $\frac{\gamma}{2(1+\epsilon)}$ . Without loss of generality assume that  $p_i$  was added to  $S$  first. Assume that  $p_\ell$  is selected as a sample in a subsequent iteration. Since  $\|p_i - p_\ell\| < \frac{\gamma}{2(1+\epsilon)}$ , by definition, there exists a unique node  $u \in \mathcal{U}_{p_\ell}$  such that  $p_i \in \square_u \cap P$ . Hence,  $p_i$  lies in a leaf node of the subtree rooted at  $u$ . Since  $p_i$  has already been selected we have that  $u_b = 0$  (because  $u$  is an ancestor of the leaf node of  $p_i$ ), so  $p_\ell$  is not inserted in  $S$ .

Next, we argue about the fairness requirement. Let  $p_\ell$  be a point with  $\hat{x}_\ell > 0$ . Let  $V_t$  be the event that the first point included in  $S$  from the set  $S_{p_\ell}^\epsilon$  is the point from the  $t$ -th step. Then

$$\begin{aligned} \Pr[p_\ell \in S] &= \sum_{t=1}^n \Pr[\sigma(t) = p_\ell \mid V_t] \Pr[V_t] = \sum_{t=1}^n \frac{\hat{x}_\ell}{\sum_{p_i \in S_{p_\ell}^\epsilon} \hat{x}_i} \Pr[V_t] \\ &= \frac{\hat{x}_\ell}{\sum_{p_i \in S_{p_\ell}^\epsilon} \hat{x}_i} \sum_{t=1}^n \Pr[V_t] = \frac{\hat{x}_\ell}{\sum_{p_i \in S_{p_\ell}^\epsilon} \hat{x}_i} \geq \frac{\hat{x}_\ell}{1 + \epsilon} \end{aligned}$$

The last inequality holds because  $\sum_{p_i \in S_{p_\ell}^\epsilon} \hat{x}_i \leq 1 + \epsilon$  from Constraints (12). For  $c_j \in C$ ,  $\mathbb{E}[|S(c_j)|] \geq \sum_{p_i \in P(c_j)} \frac{\hat{x}_i}{1 + \epsilon} \geq \frac{k_j}{1 + \epsilon}$ .  $\square$

The rounding can be executed in  $O(n(\log n + \epsilon^{-d}))$  time because  $\Xi$  has  $O(\log n)$  height and each query takes  $O(\log n + \epsilon^{-d})$  time.

**Selection of  $\gamma$ .** We note that so far we assume that  $\gamma$  is any value and we try to find a solution  $S$  with  $\text{div}(S) \geq \frac{\gamma}{2(1+\epsilon)}$ . In order to find a good approximation of the optimum diversity  $\gamma^*$ , we use the

notion of the Well Separated Pair Decomposition (WSPD) [16, 33] briefly described in Section 2. Let  $\mathcal{L}$  be the sorted list of  $O(n/\varepsilon^d)$  distances from WSPD. We run a binary search on  $\mathcal{L}$ . For each distance  $L \in \mathcal{L}$  we set  $y \leftarrow L$ . If the problem is infeasible, we continue the binary search with smaller values of  $L$ . If the problem is feasible then we continue the binary search with larger values of  $L$ . In the end, we return the feasible solution corresponding to the largest  $L$ . We need  $O(ne^{-d})$  time to find the WSPD distances and  $O(ne^{-d} \log n)$  to sort the distances. The binary search executes  $O(\log n)$  iterations.<sup>2</sup> Hence, the running time of the overall algorithm increases by a  $\log n$  factor. Any pairwise distance in  $P$  can be approximated by a distance in the list  $\mathcal{L}$  within a factor  $1 + \varepsilon$ , hence, we might not get the optimum  $y^*$  exactly. In the worst case, we might get a smaller value which is at least  $\frac{y^*}{1+\varepsilon}$ . Putting everything together, and setting  $\varepsilon \leftarrow \varepsilon/4$ ,<sup>3</sup> we get the next theorem.

**THEOREM 3.2.** *Let  $C$  be a set of  $m$  colors,  $P$  be a set of  $n$  points in  $\mathbb{R}^d$  for a constant  $d$ , where each point  $p \in P$  is associated with a color  $c(p) \in C$  and let  $k_1, \dots, k_m$  be  $m$  integer parameters with  $k_1 + \dots + k_m = k$ . Let  $\varepsilon$  be a constant parameter. There exists an algorithm for the FairDiv problem that returns a set  $S \subseteq P$  such that  $\mathbb{E}[|S(c_j)|] \geq \frac{k_j}{1+\varepsilon}$  for each color  $c_j \in C$ , and  $\text{div}(S) \geq \frac{y^*}{2(1+\varepsilon)}$  in  $O(nk \log^3 n)$  time and  $O(n)$  space.*

### 3.2 Diversity with high probability

In this subsection, we describe how to use the result of the previous subsection to get a solution where the fairness constraints hold with probability at least  $1 - \delta$  for any parameter  $\delta \in (0, 1)$ . We get the intuition from [6], however the implementation of our algorithm has significant changes to achieve the near-linear time execution.

We transform the near-optimal fractional solution  $\hat{x}$  we derived from the MWU method to a new fractional solution  $\hat{y} \in \mathbb{R}^n$ . Recall that  $\mathcal{T}$  is a BBD tree constructed on  $P$ . Recall that  $\mathcal{U}(p, \beta)$  is the set of canonical nodes returned by query  $\mathcal{T}(p, \beta)$ . Let  $S_p(\beta) \subseteq P$  be the set of points covered by the rectangles associated with the canonical nodes  $\mathcal{U}(p, \beta)$ . Notice that  $S_p^\varepsilon = S_p\left(\frac{\gamma}{2(1+\varepsilon)}\right)$ .

The feasible solution  $\hat{y}$  we will construct is the solution for the following set of (non-linear) constraints.

$$\sum_{p_i \in P(c_j)} y_i \geq k_j, \quad \forall j \in [m] \quad (14)$$

$$\sum_{p_i \in S_p\left(\frac{\gamma}{6(1+\varepsilon)^2}\right)} y_i \leq 1 + \varepsilon, \quad \forall p \in P \quad (15)$$

$$y_i \geq 0, \quad \forall p_i \in P \quad (16)$$

$$y_i > 0 \text{ and } y_\ell > 0 \Rightarrow \|p_i - p_\ell\| \geq \frac{\gamma}{3(1+\varepsilon)^2}, \quad (17)$$

$$\forall p_i, p_\ell \in P(c_j), \forall j \in [m]$$

Next, we describe the algorithm to get a solution  $\hat{y}$  for the new set of constraints. For each color  $c_j \in C$  we construct a BBD tree  $\mathcal{T}_j$  on  $P(c_j)$ . For a query  $\mathcal{T}_j(p, \beta)$ , we define  $\mathcal{U}_j(p, \beta)$  and  $S_{p,j}(\beta)$  as we had before, considering only the points of color  $c_j$ . Each point

<sup>2</sup>For simplicity we consider  $\varepsilon \geq 1/n$ .

<sup>3</sup>Notice that  $\frac{\gamma}{2(1+\varepsilon/4)^2} \geq \frac{\gamma}{2(1+\varepsilon)}$  for any  $\varepsilon \in (0, 1)$ .

$p_i$  in  $\mathcal{T}_j$  is associated with its weight  $\hat{x}_i$ . In each node  $u$  of every tree  $\mathcal{T}_j$  we store the sum  $u.s$  of the weights of the points stored in the subtree rooted at  $u$ , i.e.,  $u.s = \sum_{p_i \in \square_u \cap P(c_j)} \hat{x}_i$ . Given a query ball  $\mathcal{B}(p, r)$  we can run a weighted range sum query in  $\mathcal{T}_j$  and return  $M_j(p, r) = \sum_{p_i \in S_{p,j}(r)} \hat{x}_i$  in  $O(\log n + \varepsilon^{-d})$  time. If  $u.s > 0$ , then we call node  $u$  *active*. Otherwise, if  $u.s = 0$ , node  $u$  is *inactive*.

For each point  $p_i$  with  $\hat{x}_i > 0$  and no inactive ancestor, we perform the following steps. Let  $c_j$  be the color of  $p_i$ . We run a sum query on  $\mathcal{T}_j$  and we get  $\hat{y}_i = M_j(p_i, \frac{\gamma}{3(1+\varepsilon)^2})$ . During the search procedure if we visit a node  $u$  with  $u.s = 0$ , we stop traversing the subtree rooted at  $u$ . For each node  $u \in \mathcal{U}_j(p_i, \frac{\gamma}{3(1+\varepsilon)^2})$ , we traverse the tree from  $u$  to the root of the tree and for each node  $v$  we visit, we update  $v.s \leftarrow v.s - u.s$ . Then we set  $u.s = 0$ . In the end of this process, if there are  $\hat{y}_i$  values that are not defined, we set  $\hat{y}_i = 0$ .

**Correctness.** We show that our method finds a solution  $\hat{y}$  that satisfies the new constraints. Constraints (14)  $\sum_{p_i \in P(c_j)} \hat{y}_i \geq k_j$  are satisfied because for each color  $c_j \in C$ ,  $\sum_{p_i \in P(c_j)} \hat{y}_i = \sum_{p_i \in P(c_j)} \hat{x}_i$ . This equality holds because when we find node  $u$  as canonical node for a query  $\mathcal{T}_j(p, \frac{\gamma}{3(1+\varepsilon)^2})$ , we set  $u$  to an inactive node so no point  $q \in \square_u \cap P(c_j)$  contributes to another variable  $\hat{y}_\ell$  in the next iterations. For Constraints (15), for every point  $p \in P$ , we have,

$$\begin{aligned} \sum_{p_i \in S_p\left(\frac{\gamma}{6(1+\varepsilon)^2}\right)} \hat{y}_i &\leq \sum_{p_\ell \in P: \exists p_i \in S_p\left(\frac{\gamma}{6(1+\varepsilon)^2}\right), p_\ell \in S_{p_i}\left(\frac{\gamma}{3(1+\varepsilon)^2}\right)} \hat{x}_\ell \\ &\leq \sum_{p_\ell \in P: \|p - p_\ell\| \leq \frac{\gamma}{2(1+\varepsilon)}} \hat{x}_\ell \leq \sum_{p_\ell \in S_p\left(\frac{\gamma}{2(1+\varepsilon)}\right)} \hat{x}_\ell \\ &= \sum_{p_\ell \in S_p^\varepsilon} \hat{x}_\ell \leq 1 + \varepsilon. \end{aligned}$$

Constraints (16) are satisfied by definition. Finally, for Constraints (17), notice that when our algorithm sets some positive value  $\hat{y}_i$  then all canonical nodes in  $\mathcal{U}_j(p_i, \frac{\gamma}{3(1+\varepsilon)^2})$  are set to inactive. Hence, any point  $p_\ell$  with  $\|p_i - p_\ell\| < \frac{\gamma}{3(1+\varepsilon)^2}$ , has at least one inactive ancestor and  $\hat{y}_i$  is set to 0.

**Running time.** We construct all BBD trees  $\mathcal{T}_j$  in  $O(n \log n)$  time. For each point  $p_i$  we check if there is no inactive ancestor in  $O(\log n)$  time since  $\mathcal{T}_j$  has depth  $O(\log n)$ . Each weighted range sum query on  $\mathcal{T}_j$  takes  $O(\log n + \varepsilon^{-d})$  time. The  $u.s$  variables are also updated in  $O(\log n + \varepsilon^{-d})$  time. Overall, given the values  $\hat{x}$ , we can compute  $\hat{y}$  in  $O(n(\log n + \varepsilon^{-d}))$  time.

**Rounding.** The rounding procedure is similar to the rounding in Subsection 3.1. Again, we construct a BBD tree  $\Xi$  to sample points from  $P$ . Each time we choose a point  $p_i$  we check if another point has already been selected in a previous iteration in the area of  $\Xi(p_i, \frac{\gamma}{6(1+\varepsilon)^3})$ <sup>4</sup>. Using the proof of Lemma 3.1, we get that for any pair  $p_i, p_j \in S$  in the set we return, we have  $\min_{p, q \in S} \|p - q\|_2 \geq \frac{\gamma}{6(1+\varepsilon)^3}$ . Furthermore, let  $Y_i$  be the random variable which is 1 if  $p_i \in S$  and 0 otherwise. Using the same proof, we get  $\mathbb{E} \left[ \sum_{p_i \in S(c_j)} Y_i \right] \geq \frac{k_j}{1+\varepsilon}$  for each color  $c_j$ . By definition, all variables  $Y_i$  are independent, so we can apply the Chernoff bound to get a bound on the number of points of each color with high probability. In particular, if  $k_j \geq$

<sup>4</sup>For technical reasons we set the denominator to  $(1 + \varepsilon)^3$  instead of  $(1 + \varepsilon)^2$ .

$\frac{3}{\varepsilon^2} \log(2m)$ , we get  $\Pr[\sum_{p_i \in S(c_j)} Y_i \leq (1 - \varepsilon) \frac{k_j}{1+\varepsilon}] \leq \frac{1}{2m}$ . If we apply the union bound we have that  $\sum_{p_i \in S(c_j)} Y_i \geq (1 - \varepsilon) \frac{k_j}{1+\varepsilon}$  for all colors  $c_j \in C$  with probability at least  $1/2$ . If we repeat the process  $\log \frac{1}{\delta}$  iterations, then with probability at least  $1 - \delta$  we find a solution  $S$  with  $\sum_{p_i \in S(c_j)} Y_i \geq (1 - \varepsilon) \frac{k_j}{1+\varepsilon}$  for all colors  $c_j \in C$ . In each repetition we spend  $O(n(\log n + \varepsilon^{-d}))$  as in Subsection 3.1. The total running time of the rounding procedure is  $O(n \log(\frac{1}{\delta})(\log n + \varepsilon^{-d}))$ .

Finally, as we had in Subsection 3.1, recall that we run a binary search on the WSPD set  $L$ , finding a value  $\gamma$  such that  $\gamma^* \geq \gamma \geq \gamma^*/(1 + \varepsilon)$ .

Putting everything together, and setting  $\varepsilon \leftarrow \varepsilon/6$ ,<sup>5</sup> we get the next theorem.

**THEOREM 3.3.** *Let  $C$  be a set of  $m$  colors,  $P$  be a set of  $n$  points in  $\mathbb{R}^d$  for a constant dimension  $d$ , where each point  $p \in P$  is associated with a color  $c(p) \in C$  and let  $k_1, \dots, k_m$  be  $m$  integer parameters with  $k_1 + \dots + k_m = k$  and  $k_j \geq 3(1 + \varepsilon)\varepsilon^{-2} \log(2m)$  for each  $c_j \in C$ . Let  $\varepsilon$  be a constant parameter and  $\delta$  be a parameter. There exists an algorithm that returns a set  $S \subseteq P$  such that  $|S(c_j)| \geq \frac{k_j}{1+\varepsilon}$  for each color  $c_j \in C$  with probability at least  $1 - \delta$ , and  $\text{div}(S) \geq \frac{\gamma^*}{6(1+\varepsilon)}$  in  $O\left(nk \log^3 n + n \log(\frac{1}{\delta}) \log n\right)$  time and  $O(n)$  space.*

## 4 CORESET

As shown in Section 2, in [6] they describe a coresnet for the FairDiv problem. In particular, using the Gonzalez algorithm [31] for the  $k$ -center clustering problem, they get a set  $G \subseteq P$  of  $O(mk\varepsilon^{-d})$  points in  $O(nk\varepsilon^{-d})$  time, such that  $G$  contains a solution for the FairDiv problem with diversity at least  $\gamma^*/(1 + \varepsilon)$ . Their proof of correctness relies on the execution of Gonzalez algorithm choosing the furthest point from the set of centers that have been already selected in each iteration. Unfortunately, Gonzalez algorithm takes  $O(nk)$  time. Ideally, we would like to use other faster constant approximation algorithms for the  $k$ -center problem in the Euclidean case. In this section, we show a more general coresnet construction. We show that if  $k' = O(\varepsilon^{-2d}k)$  points are chosen for each color  $c_j \in C$  using any constant approximation for the  $k'$ -center clustering problem in the Euclidean space, then their union is a valid coresnet for the FairDiv problem.

Let  $\text{Alg}$  be an  $\alpha$ -approximation for the  $k'$ -center clustering problem that runs in  $O(T(n, k'))$  time. For every color  $c_j \in C$ , we run  $\text{Alg}$  on  $P(c_j)$  for  $k' = O(\varepsilon^{-2d}k)$  and we get the set of centers  $G'_j$ . We return the coresnet  $G' = \bigcup_{c_j \in C} G'_j$ . The coresnet  $G'$  is constructed in  $O(\sum_{c_j \in C} T(|P(c_j)|, k'))$  time and has cardinality  $|G'| = O(\varepsilon^{-2d}km)$ .

**LEMMA 4.1.** *The set  $G'$  is a  $(1 + \varepsilon)$ -coresnet for the FairDiv problem.*

**PROOF.** We fix a color  $c_j \in C$ . For any  $k$ , let  $\mu_k$  be the optimum radius for the  $k$ -center clustering problem in  $P(c_j)$ . Let  $\hat{k} = O(\varepsilon^{-d}k)$  and  $k' = O(\varepsilon^{-2d}k)$ . For a subset  $Q \subseteq P$ , we define  $\mu(Q) = \max_{p \in P(c_j)} \min_{q \in Q} \|p - q\|$ , i.e., the value of the  $|Q|$ -center solution  $Q$  on  $P(c_j)$ . Let  $\xi$  be a constant number. Let  $D$  be a grid in  $\mathbb{R}^d$  such that each grid cell has side length  $\frac{\varepsilon \cdot \mu_k}{4\alpha \cdot \xi}$ . Let  $A = \emptyset$ . For every

cell  $g \in D$ , if  $|g \cap P(c_j)| > 0$ , then we insert a representative point  $p_g \in D \cap P(c_j)$  in  $A$ . It is known [5, 7] that  $|A| = O(\varepsilon^{-2d}k)$  and for every point  $p \in P(c_j)$  there exists a point  $p_a \in A$  such that  $\frac{\varepsilon}{4\alpha} \mu_k$ . Hence,  $\mu(A) \leq \frac{\varepsilon}{4\alpha} \mu_k$ . By definition it also holds that  $\mu_{k'} \leq \mu(A)$ . It follows that  $\mu_{k'} \leq \frac{\varepsilon}{4\alpha} \mu_k$ . We have,  $\mu(G'_j) \leq \alpha \mu_{k'} \leq \frac{\varepsilon}{4} \mu_k$ . For any  $k$ , let  $\sigma_k$  be the minimum pairwise distance of the optimum solution of the (unfair)  $k$ -MaxMin diversification problem in  $P(c_j)$  (i.e., choose a set of  $k$  points in  $P(c_j)$  that maximize the minimum pairwise distance). It is always true that  $\sigma_k \geq \mu_k$  [48]. It is also known that the Gonzalez algorithm for  $k$  iterations returns a solution with diversity at least  $\frac{1}{2}\sigma_k$  [48].

Let  $O = \{o_1, \dots, o_{\hat{k}}\}$  be the list of  $\hat{k}$  centers returned by the Gonzalez algorithm (in order) on  $P(c_j)$ . For any  $o_i \in O$ , let  $p_i \in G'_j$  be its closest point in  $G'_j$  and let  $P'_j = \bigcup_{i \in [\hat{k}]} p_i$ , and  $P' = \bigcup_{c_j \in C} P'_j$ . We show that  $P' \subseteq C'$  is a valid  $(1 + \varepsilon)$ -coresnet. Let  $r = \|o_i - o_\ell\|$  for any pair  $o_i, o_\ell \in O$ . We have  $\|p_i - p_\ell\| \geq r - \|o_i - p_i\| - \|o_\ell - p_\ell\| \geq r - \frac{\varepsilon}{2} \mu_k \geq r - \frac{\varepsilon}{2} \sigma_k \geq (1 - \varepsilon)r$ . The last inequality holds because  $r \geq \frac{1}{2}\sigma_k$  (recall that Gonzalez algorithm returns a set of points with diversity at least  $\frac{1}{2}\sigma_k$ ). Similarly,  $\|p_i - p_{i+1}\| \leq (1 + \varepsilon)r$ . Hence,  $(1 - \varepsilon)r \leq \|p_i - p_{i+1}\| \leq (1 + \varepsilon)r$ . All inequalities from Theorem 5 in [6] are satisfied within a  $(1 - \varepsilon)$  (or  $(1 + \varepsilon)$ ) factor, so by setting  $\varepsilon \leftarrow \varepsilon/\zeta$ , for a sufficiently large constant  $\zeta$  that depends on  $d$ , we conclude that  $G'$  is an  $(1 + \varepsilon)$ -coresnet for FairDiv.  $\square$

Overall, we state our new result.

**THEOREM 4.2.** *In the Euclidean space, any constant approximation algorithm for the  $k$ -center clustering with running time  $O(T(n, k))$  can be used to derive a coresnet for the FairDiv problem of size  $O(\varepsilon^{-2d}mk)$  in  $O(\sum_{c_j \in C} T(|P(c_j)|, \varepsilon^{-2d}k))$  time.*

In the next results we fix  $\text{Alg}$  to be either the Feder and Greene [26] algorithm or the Har-Peled and Raichel algorithm [34] to return a 2-approximation for the  $k$ -center clustering in  $O(n \log k)$  time or  $O(n)$  expected time, respectively. Using our coresnet  $G'$  as input to the algorithm in Theorem 3.2 we get the next corollary.

**COROLLARY 4.3.** *There exists an algorithm that returns a set  $S \subseteq P$  such that  $\mathbb{E}[|S(c_j)|] \geq \frac{k_j}{1+\varepsilon}$  for each color  $c_j \in C$ , and  $\text{div}(S) \geq \frac{\gamma^*}{2(1+\varepsilon)}$  in  $O(n \log k + mk^2 \log^3(k))$  time and  $O(mk)$  additional space. The same algorithm can be executed in  $O(n + mk^2 \log^3(k))$  expected time.*

Using our coresnet  $G'$  as input to the algorithm in Theorem 3.3 we get the next corollary.

**COROLLARY 4.4.** *There exists an algorithm that returns a set  $S \subseteq P$  such that  $|S(c_j)| \geq \frac{k_j}{1+\varepsilon}$  with probability at least  $1 - \delta$  for each color  $c_j \in C$ , and  $\text{div}(S) \geq \frac{\gamma^*}{6(1+\varepsilon)}$  in  $O(n \log k + mk^2 \log^3 k + mk \log k \log \frac{1}{\delta})$  time and  $O(mk)$  additional space. The same algorithm can be executed in  $O(n + mk^2 \log^3 k + mk \log k \log \frac{1}{\delta})$  expected time.*

## 5 EXTENSIONS

In this section, we show how our coresnet construction and our new algorithms for the FairDiv problem can be used to get a faster algorithm in the streaming setting, SFairDiv problem. We also show how our results can be used to design the first efficient data structure for the QFairDiv problem.

<sup>5</sup>Notice that  $\frac{\gamma}{6(1+\varepsilon/6)^4} \geq \frac{\gamma}{6(1+\varepsilon)}$  and  $\frac{1-\varepsilon/6}{1+\varepsilon/6} k_j \geq \frac{k_j}{1+\varepsilon}$  for  $\varepsilon \in (0, 1)$ .

## 5.1 Streaming setting

In the streaming setting we care about three quantities: The number of elements that the algorithm needs to store in each iteration, the update time for any new item we get, and the post-processing time we need to create an actual solution for the FairDiv problem.

The current best streaming algorithm for the SFairDiv problem is presented in [50]. For a constant  $\epsilon$ , it stores  $O(mk \log \Delta)$  items, it has  $O(k \log \Delta)$  update time per element, and it has  $O(m^2 k^2 \log \Delta)$  post-processing time to construct a  $\frac{1-\epsilon}{3m+2}$ -approximation for the FairDiv problem, where  $\Delta$  is the spread of the input set. Notice that  $\Delta$  can be exponential with respect to  $n$  even in  $\mathbb{R}^d$  making all quantities linear on  $n$  in the worst case. We present the first constant-approximation streaming algorithm for the FairDiv problem whose space, update time, and post-processing time are independent of  $\Delta$ .

Because of Lemma 4.1 it is sufficient to maintain any constant approximation of the  $k$ -center clustering solution over the set of items we have seen so far. It is known that the doubling algorithm [21] maintains an 8-approximation for the  $k$ -center problem in  $\mathbb{R}^d$  with  $O(k \log k)$  update time storing  $O(k)$  elements. Using the well known doubling algorithm to maintain a constant  $k$ -center clustering solution, our new coresets construction in Theorem 4.2, and our new near-linear time algorithm in Theorem 3.2 for the FairDiv problem, we give the following result for the SFairDiv problem.

**THEOREM 5.1.** *For a constant parameter  $\epsilon$ , there exists a streaming algorithm for the SFairDiv problem that stores  $O(mk)$  items, has  $O(k \log k)$  update time, and has  $O(mk^2 \log^3 k)$  post-processing time. After the post-processing the algorithm returns a set of points  $S$  such that  $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1+\epsilon}$  for each color  $c_j \in C$ , and  $\text{div}(S) \geq \frac{\gamma^*}{2(1+\epsilon)}$ .*

Using the result from Theorem 3.3 (instead of Theorem 3.2) we can also satisfy the fairness constraints with high probability.

## 5.2 Range-query setting

Using our new results we show how we design the first efficient data structure for the QFairDiv problem. Given a set of  $n$  points  $P \in \mathbb{R}^d$ , in [5, 44] they show that there exists a data structure of  $O(n \log^{d-1} n)$  space that can be constructed in  $O(n \log^{d-1} n)$  time, such that, given any query hyper-rectangle  $R$  and any query parameter  $k$ , a  $(2 + \epsilon)$ -approximation of the  $k$ -center clustering in  $P \cap R$  is returned in  $O(k \log^{d-1} n + \epsilon^{-d} k \log \frac{k}{\epsilon})$  time. Using this  $k$ -center data structure, we construct our coresets from Theorem 4.2 in  $P \cap R$  efficiently, and then using our new near-linear time algorithm in Theorem 3.2 for the FairDiv problem, we give the following result for the QFairDiv problem.

**THEOREM 5.2.** *For the QFairDiv problem, a data structure of size  $O(n \log^{d-1} n)$  can be constructed in  $O(n \log^{d-1} n)$  time, such that given a query rectangle  $R$ , a constant parameter  $\epsilon$ , and parameters  $k_1, \dots, k_m$  with  $k_1 + \dots + k_m = k$ , it returns a set  $S \subseteq P \cap R$  in  $O(mk \log^{d-1} n + mk^2 \log^3 k)$  time such that  $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1+\epsilon}$  for each color  $c_j \in C$ , and  $\text{div}(S) \geq \frac{\gamma^*}{2(1+\epsilon)}$ .*

Using the result from Theorem 3.3 (instead of Theorem 3.2) we can also satisfy the fairness constraints with high probability.

## 6 EXPERIMENTS

In this section, we evaluate the effectiveness of our algorithm to identify a diverse and fair set of points. Specifically, we answer the following research questions:

- RQ1:** How does MWU behavior change with varying parameters? Is the fairness requirement violated by MWU? This RQ identifies the best parameters for MWU to be used for comparison with baselines.
- RQ2:** How does MWU result compare against other baselines? Is it efficient to identify a fair and diverse solution?

**Table 2: Dataset Statistics**

Dataset	Groups	m	d	n
Adult	Race, Sex	10	6	32,561
Diabetes	Sex, Meds Prescribed-Y/N	4	8	101,763
Census	Sex, Age	14	6	2,426,116
Popsim	Race	5	2	4,110,608
Popsim_1M	Race	5	2	821,804

**Datasets** We consider the following datasets.

- (1) The Adult dataset [12] contains around 32K records of individuals describing their income, and education details. Race and Sex are considered as sensitive attributes to generate 10 colors (protected groups).
- (2) The Diabetes dataset [37] contains health statistics of around 100K patients, where Sex and medical prescriptions are used to generate colors.
- (3) The Census dataset [40] contains around 2M records of individuals. Age and Sex are used as sensitive attributes to consider 14 colors. We choose 6 numerical attributes to represent the points in the dataset.
- (4) Popsim [43] is a semi-synthetic dataset that combined population statistics along with a geo-database. It is used to represent individual-level data with demographic information for the state of Illinois. Race is used as sensitive attribute to consider 5 colors. The parameters longitude, latitude are used to convert the locations into 4,110,608 points in  $\mathbb{R}^2$ .
- (5) Popsim\_1M [43]. Another version of the same semi-synthetic dataset containing almost 1 million individuals.

**Baselines** We compare our algorithm with the state of the art algorithms on the FairDiv problem.

First, we discuss how we implemented our new MWU algorithm. We implemented our main algorithm from Corollary 4.3 combining the algorithm from Theorem 3.2 with the coresets construction in Theorem 4.2. For coresets construction, we modified a python implementation for constructing coresets [47]. For every color  $c_j \in C$ , we run Gonzalez algorithm for  $k$  iterations. In the end, we have a coresets of size  $m \cdot k$ . Then the coresets is given as input to our MWU method. We note that we include the coresets construction time in the total running time of MWU.

Next, we describe some differences between the theoretical MWU algorithm and our implementation. Instead of constructing a WSPD to run a binary search on the distances, we get as an upper bound  $\gamma$  on the maximum possible diversity: We run Gonzalez algorithm for  $k$  iterations in the entire point set  $P$  without considering the colors. It is known that the minimum pairwise distance among

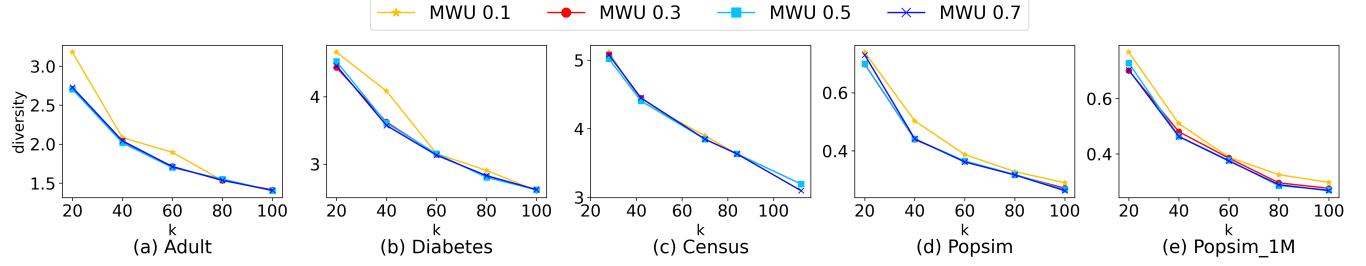
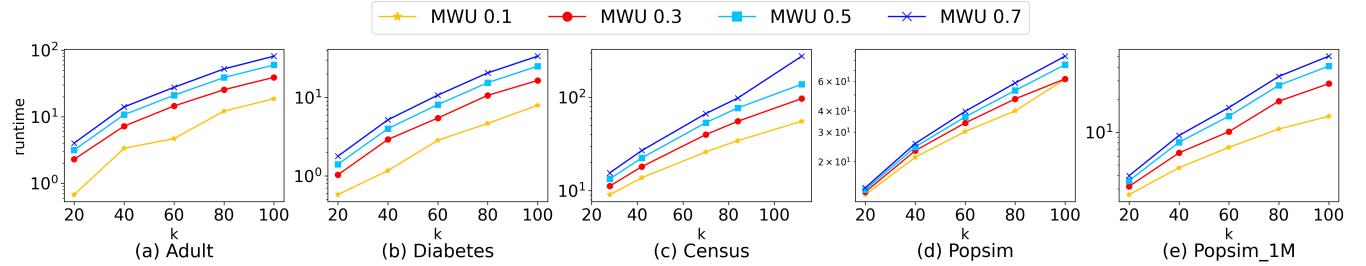
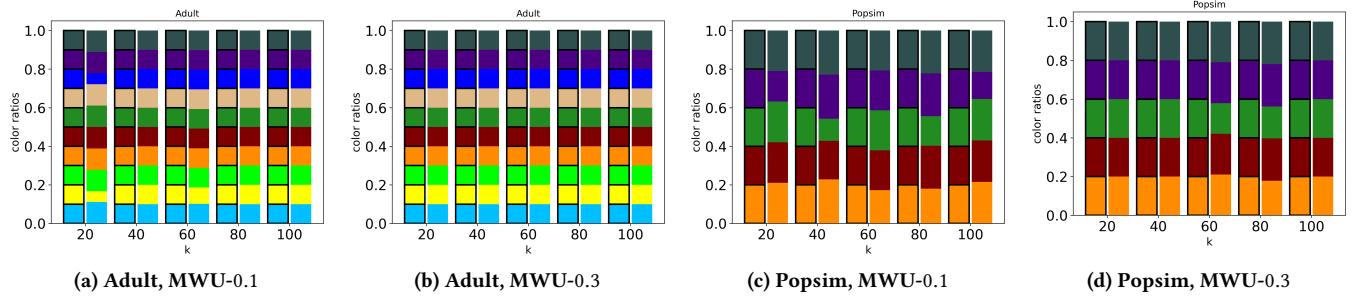
Figure 2: Comparison of diversity vs  $k$  for MWU with different early stopping parameters.Figure 3: Comparison of running time vs  $k$  for MWU with different early stopping parameters.

Figure 4: Number of points per color returned by MWU-0.1 and MWU-0.3.

the selected points is an upper bound on the diversity of FairDiv. If MWU does not find a feasible solution for diversity  $\gamma$ , we set  $\gamma \leftarrow (1 - 0.15)\gamma$ , and re-run the MWU algorithm, stopping at the first feasible solution. Additionally, instead of using a BBD tree we use ParGeo's KD tree [53] with modifications to support sum queries. In practice, we observe that sometimes the MWU algorithm runs all  $T = O(\varepsilon^{-2}k \log n)$  iterations only to find an unrounded solution which is very similar to ones found by stopping at earlier iterations. Hence, we modified the algorithm to allow for *early stopping*. We introduce a parameter  $g \in (0, 1]$  such that the MWU algorithm runs for at most  $g \cdot T$  iterations rather than the full  $T$  iterations from the theory. We settled on a default value of  $g$  in our experiments of 0.3. As we will show, this does not affect the quality of the results yet significantly improves the MWU algorithm's running time. Since MWU is a randomized algorithm, in each case, we run the algorithm five times and we report the average diversity and average running time.

- MWU: Our new implementation as described above.

- SFDM-2: It is a streaming algorithm designed and implemented in [50]. The algorithm uses the minimum and maximum pairwise distance to define a range on the diversity. In the original implementation the authors selected this range manually. In order to be fair with our implementation, we use the same upper bound used in MWU. As a lower bound we use the minimum non-zero pairwise distance in the core-set of size  $m \cdot k$  we construct. The algorithm uses a parameter  $\varepsilon$  to control the error in the final solution. We tried two representative errors  $\varepsilon = 0.15$  and  $\varepsilon = 0.75$ . We call them SFDM-2 ( $\varepsilon = .15$ ) and SFDM-2 ( $\varepsilon = .75$ ), respectively.
- FMMMD-S: The algorithm presented and implemented in [52]. The algorithm creates its own coresset.
- FairFlow: The algorithm presented in [41] and implemented in [52].
- FairGreedyFlow: The algorithm presented in [6] and implemented in [52]. It takes as input the coresset we construct to improve its running time.

All algorithms are implemented in python, except for the kd-tree implementation in ParGeo which is implemented in C++. If an algorithm takes more than 30 minutes to finish, we stop its execution and we do not show the results in the figures. All datasets and our code can be found in [1].

*Setup.* We run all our experiments on a e2-standard-16 Google Cloud VM with 16 vCPUs (8 cores) and 64 GB of memory running Debian 11 Bullseye v20231004.

## 6.1 Micro-benchmark experiments

In this section we compare the diversity and the running time of our MWU algorithm for different early stopping parameters  $g$ . We test  $g = 0.1, 0.3, 0.5, 0.7$ . In all cases, we set  $k_j = k/m$  (equal  $k_j$ ). We also run the same experiments with proportional  $k_j = k \frac{|P(c_j)|}{n}$ , but we skip them from this version because all observations are identical to the equal case. In Figure 2 we show the diversity over different values of  $k$  for all datasets. We observe that the early stopping does not affect the diversity a lot. Figure 3 presents the running time over different values of  $k$ . It shows that the smaller the  $g$  is the faster the MWU is.

Before we conclude that a small value of  $g$  is good for our algorithm, recall that Theorem 3.2 (and Corollary 4.3) does not always guarantee at least  $k_j$  points in the final solution from every color  $c_j \in C$ . We show the number of missing points from each color for different parameters  $g$ . In Figure 4 we show two representative results using the Adult and Popsim dataset for MWU with  $g = 0.1$  and  $g = 0.3$ . For every  $k$ , the left bar shows the real number (as a ratio of  $k$ ) of items we should return from each color, while the right bar contains the number (as a ratio of the solution size) of the points returned by the MWU algorithm. When  $g = 0.1$  indeed MWU misses some points from each color especially when  $k$  is small. However as it is shown in Figures 4b and 4d for  $g = 0.3$  MWU returns all the necessary points to satisfy fairness in almost all cases. More generally, we observed that in all datasets, for every  $g \geq 0.3$  it is very rare to miss some points. In the worst case we observed that in some rare cases we miss a couple of points when  $k$  is small (less than 20) or at most 7 points when  $k$  is 100. Hence, we conclude that MWU for  $g \geq 0.3$  successfully satisfies the fairness requirements. In the next experiments we always use  $g = 0.3$  for the MWU algorithm.

**Key Takeaways:** MWU with  $g = 0.3$  satisfies the fairness criterion for most settings, has comparable diversity to other values of  $g$  and is highly efficient as compared to higher values of  $g$ . Using smaller  $g$  may violate fairness criterion but can be used for increased efficiency.

## 6.2 End-to-end results

To compare the quality of MWU with state-of-the-art, we considered two groups of experiments. In the first group, the fairness constraint ensures that the number of points returned for each color are equal, i.e.,  $k_j = \frac{k}{m}, \forall j$ . In the second group, we choose proportional  $k_j$ 's, i.e.,  $k_j = k \frac{|P(c_j)|}{n}$ . Generally, the first group of equal  $k_j$  leads to more fair solutions that are more difficult to satisfy.

The main goal of this comparison is to identify techniques that maximize diversity within a reasonable amount of time. Two extreme algorithms are:

1. **Random selection** would choose  $k_j$  points randomly from each color. This approach is expected to be highly efficient but would have very poor diversity
2. **Exhaustive search** would exhaustively consider all possible subsets to calculate diversity and return the best solution. This approach may return a highly accurate solution but would be highly inefficient. This approach would not scale to millions sized datasets.

The key goal of this experiment is to identify a technique that returns the best solution within a reasonable amount of time.

**Equal number of points from each color.** Figure 5 compares the diversity of MWU and baselines for this setting where equal points from each color are returned (higher diversity is better). We observe that the diversity of the returned solution decreases with increasing  $k$ . For example, diversity of MWU reduces from 5 for  $k = 20$  to 3.5 for  $k = 100$ .

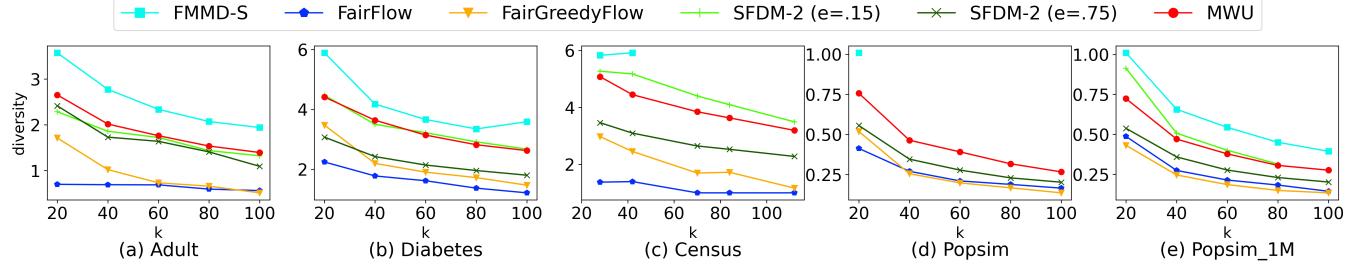
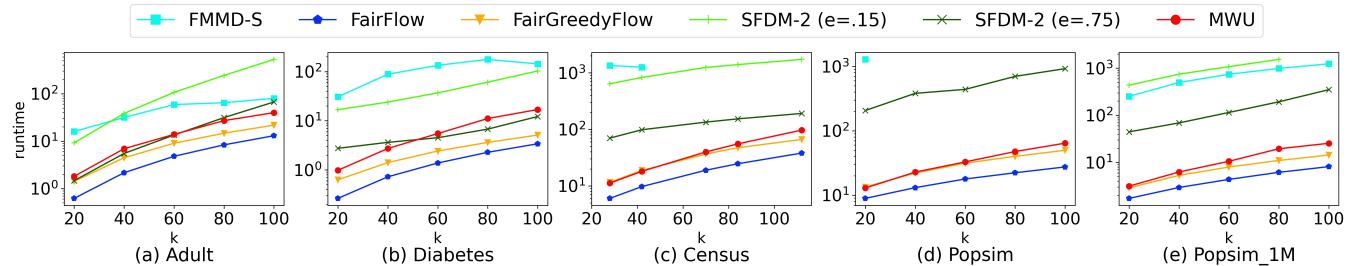
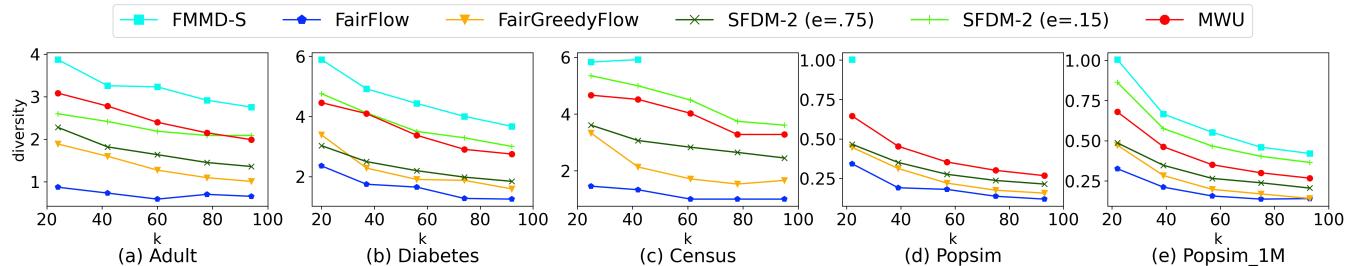
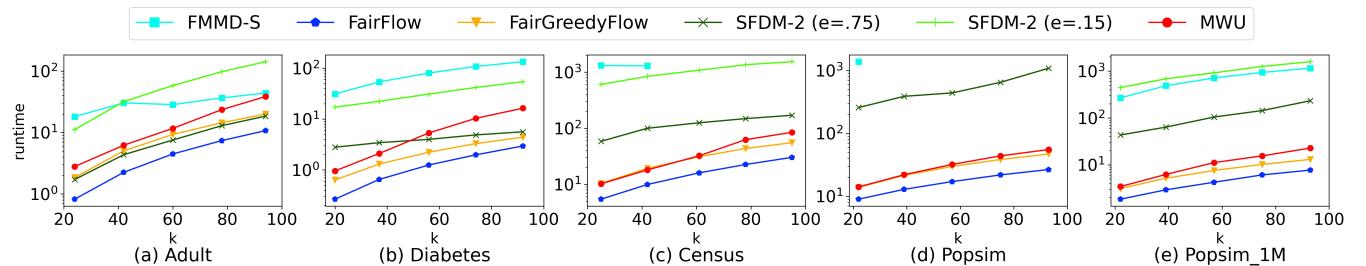
FMMD-S achieves the highest diversity for Adult, Diabetes and Popsim\_1M dataset, but it did not run for Census and Popsim datasets for  $k > 40$ . Therefore, FMMD-S is not scalable to million scale datasets. Furthermore, it takes at least 50× the time taken by MWU. Among all other techniques, MWU achieves the best diversity for most values of  $k$  and datasets. SFDM-2 ( $\epsilon = 0.15$ ) baseline achieves comparable (or slightly better in some cases) diversity as that of MWU for datasets and  $k$  whenever it ran. However, it did not finish for Popsim dataset for  $k \geq 20$  and took at least 50× the time taken by MWU. Among the techniques that ran for all datasets and  $k$ , MWU has the best diversity. All other techniques SFDM-2 ( $\epsilon = 0.75$ ), FairFlow and FairGreedyFlow achieve poorer diversity than MWU. Among these techniques, FairFlow and FairGreedyFlow achieve the lowest diversity across all datasets.

Figure 6 compares the running time of different techniques for varying  $k$  and datasets. We observe that the running time increases sub-linearly with  $k$  for all techniques. FMMD-S and SFDM-2 are generally considerably slower than all other techniques. All other techniques (our approach MWU and baselines FairFlow, FairGreedyFlow) require less than 40 seconds to identify a diverse set of 100 points for more than 1M records. In fact, MWU runs for  $k = 100$  on 4M records within less than 80 seconds. In contrast to few baselines that take more than 1000 seconds for a dataset with less than 1M points. This validates the efficiency of MWU to identify a fair and diverse solution.

**Key Takeaways:**

- (1) MWU achieves the best diversity among the techniques that run across all datasets.
- (2) SFDM-2 ( $\epsilon = 0.15$ ) and FMMD-S achieve higher diversity for some settings but take 40× more time than MWU.
- (3) All other baselines achieve worse diversity than MWU but take almost the same time to run.

**Proportional size.** Figure 7 and 8 compares the diversity and running time of MWU and other techniques for the setting where

Figure 5: Comparison of diversity vs  $k$  for MWU and baselines. Higher diversity is better. Equal  $k_j$ .Figure 6: Comparison of running time vs  $k$  for MWU and baselines. Equal  $k_j$ .Figure 7: Comparison of diversity vs  $k$  for MWU and baselines. Higher diversity is better. Proportional  $k_j$ .Figure 8: Comparison of running time vs  $k$  for MWU and baselines. Proportional  $k_j$ .

the number of points returned for each color is proportional to their color size in the dataset. The results for this case are similar to that of equal size, where FMMD-S achieves the highest diversity but did not scale to million sized datasets. Similar was the case for SFDM-2 ( $\epsilon = 0.15$ ) which performed as good as MWU but did not finish for Popsim dataset. Among all other baselines, MWU achieves the best diversity while identifying the solution in less than a minute, even for million sized datasets.

**Key Takeaways:** MWU achieves the best quality result within less than a minute for a million scale datasets. Therefore, MWU provides the best balance between diversity and running time.

## 7 RELATED WORK

There is a lot of recent work on the FairDiv problem. In [41] the authors define the FairDiv problem and propose algorithms for general metric spaces. In particular, when  $m = 2$ , they showed an  $O(nk)$  time algorithm with  $1/4$ -approximation factor using linear space. For any number of colors  $m$ , they propose FairFlow that runs in  $O(kn + m^2 k^2 \log k)$  time and returns a  $\frac{1}{3m-1}$ -approximation. When  $k$  is small, they also propose a  $1/5$ -approximation algorithm with exponential running time with respect to  $k$ ,  $O(kn + k^2 m^k)$ . [6] improved some of the results from the previous paper. For any number of colors  $m$ , they design an LP-based relaxation algorithm to get a  $1/2$ -approximation satisfying the fairness constraints in expectation, i.e., the expected number of points from color  $c_j$  is at least  $k_j$ . The running time of the algorithm is  $O(n^{2.37})$  using the best known polynomial-time LP solver [35], and it uses  $\Omega(n^2)$  space. The same algorithm is extended to return a  $1/6$ -approximation such that the number of points of a color  $c_j$  is at least  $k_j/1 - \epsilon$  with high probability. The running time and space requirement remains the same. In the same paper they also propose a greedy algorithm, called Fair-Greedy-Flow, that returns an  $\frac{1}{(m+1)(1+\epsilon)}$ -approximation in  $O(nkm^3)$  time, for constant  $\epsilon$ , skipping  $\log n$  factors.

In the Euclidean metric (and metric spaces with bounded doubling dimension), [6] constructed a coreset of size  $O(\epsilon^{-d} mk)$  for the FairDiv problem in  $O(nk)$  time. By first constructing the coreset and then running the LP-based algorithm [6] gets a constant approximation for the FairDiv problem in  $O(nk + (mk)^{2.37})$  time. Similarly, Fair-Greedy-Flow with a coreset runs in  $O(nk + k^2 k^4)$  time. Using the coreset construction, they also design a Fair-Euclidean algorithm that returns a constant approximation in  $O(kn + m^{d+2} k \prod_{c_j \in C} k_j^2)$  time. Recently, [52] used a coreset construction to propose the FFMD-S algorithm that returns a  $\frac{1-\epsilon}{5}$ -approximation in  $O(mkn + m^k)$  time for constant  $\epsilon$ . Notice that all known constant-approximation algorithms are super-linear on  $n$  or/and  $k$ .

In the streaming setting, the authors in [50, 51] presented the SFDM-2 algorithm that stores  $O(k \log \Delta)$  items in memory, it takes  $O(k \log \Delta)$  time per item for streaming processing, and it requires  $O(k^2 \log \Delta)$  post-processing time to output a  $\frac{1-\epsilon}{3m+2}$ -approximation for the FairDiv problem, for constant  $\epsilon$ . The quantity  $\Delta$  is defined as the spread of the input set,  $\Delta = \frac{\max_{p,q \in P} ||p-q||}{\min_{p,q \in P} ||p-q||} = O(2^n)$ .

Fairness has been studied under different diversity definitions. In fair Max-Sum diversification the goal is to select a set  $S \subseteq P$  such that  $S$  contains at least  $k_j$  points from color  $c_j$ , and the sum of pairwise distances is maximized, i.e.,  $\frac{1}{2} \sum_{p,q \in S} ||p-q||$  is maximized. There are a few efficient constant approximation algorithms for this problem [4, 14, 15, 17–19]. The objective function for the fair Max-Sum problem is quite different from FairDiv so the techniques from these papers cannot be used in our problem.

The Max-Min diversification has a strong connection with the  $k$ -center clustering. For example, the same greedy Gonzalez [31] algorithm returns a  $\frac{1}{2}$ -approximation for the Max-Min diversification [46, 48] and 2-approximation for the  $k$ -center clustering [31]. Kleindessner et al. [38] defined the fair  $k$ -center problem where the goal is to find  $k_j$  centers with color  $c_j$  minimizing the  $k$ -center objective. They proposed a  $(3 \cdot 2^{m-1} - 1)$ -approximation algorithm in  $O(km^2 n + km^4)$  time. There are many improvements over this

algorithm, such as [23, 24, 36] leading to a 3-approximation algorithm for the fair  $k$ -center that runs in  $O(nk)$  time. The analysis of the algorithms for  $k$ -center is different than the algorithms for the  $k$ -Max-Min diversification problem. In some cases, an optimum solution for  $k$ -center can be arbitrary bad for  $k$ -Max-Min diversification and vice versa. Hence, it is unclear if algorithms for the fair  $k$ -center problem can be used for the FairDiv problem.

## 8 CONCLUSION

We studied the FairDiv problem when the input is a set of points in  $\mathbb{R}^d$ . We proposed the first near-linear time algorithm with constant approximation ratio for the FairDiv problem. We also showed a more general way to construct a coreset for the FairDiv problem that allows us to accelerate the running time even further. Our coreset construction also allowed us to design an efficient streaming algorithm that does not depend on the distribution of the data points for the FairDiv in the streaming setting, and the first data structure for the FairDiv in the range-query setting.

There are multiple interesting open problems derived from our work. Is it possible to get a constant approximation for the FairDiv problem satisfying the fairness exactly? The goal is to get at least  $k_j$  points instead of at least  $\frac{k_j}{1-\epsilon}$  points, from each color  $c_j \in C$ . It will also be interesting to check whether the new techniques from this paper can be used to accelerate the other version of the FairDiv problem [52] where we have both a lower bound  $k_j^-$  and an upper bound  $k_j^+$  for each color  $c_j \in C$ . Finally, it is interesting to study whether other data structures can be used, such that a cover-tree or a net-tree, to get constant approximation for the FairDiv problem in metrics with bounded doubling dimension.

## REFERENCES

- [1] <https://anonymous.4open.science/r/FairDiversityandClustering/>.
- [2] Courts seek to increase jury diversity <https://ejr.org/report/race-and-the-jury-why-representative-juries-are-necessary/#chapter-2>.
- [3] Courts seek to increase jury diversity <https://www.uscourts.gov/news/2019/05/09/courts-seek-increase-jury-diversity>.
- [4] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–40, 2013.
- [5] M. Abrahamsen, M. de Berg, K. Buchin, M. Mehrab, and A. D. Mehrabi. Range-clustering queries. In *33rd International Symposium on Computational Geometry (SoCG 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- [6] R. Addanki, A. McGregor, A. Meliou, and Z. Moumoulidou. Improved approximation and scalability for fair max-min diversification. In *25th International Conference on Database Theory (ICDT 2022)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2022.
- [7] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33:201–226, 2002.
- [8] P. K. Agarwal, S. Sintos, and A. Steiger. Efficient indexes for diverse top-k range queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 213–227, 2020.
- [9] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012.
- [10] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry*, 17(3-4):135–152, 2000.
- [11] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.
- [12] B. Becker and R. Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: <https://doi.org/10.24432/C5XW20>.
- [13] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104, 2006.
- [14] A. Borodin, A. Jain, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Transactions on Algorithms*

- (TALG), 13(3):1–25, 2017.
- [15] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166, 2012.
- [16] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.
- [17] M. Ceccarello, A. Pietracaprina, and G. Pucci. Fast coresets-based diversity maximization under matroid constraints. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 81–89, 2018.
- [18] M. Ceccarello, A. Pietracaprina, and G. Pucci. A general coresets-based approach to diversity maximization under matroid constraints. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(5):1–27, 2020.
- [19] A. Cevallos, F. Eisenbrand, and R. Zenklusen. Local search for max-sum diversification. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 130–142. SIAM, 2017.
- [20] T. M. Chan and Q. He. Faster approximation algorithms for geometric set cover. In *36th International Symposium on Computational Geometry (SoCG 2020)*, 2020.
- [21] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamical information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635, 1997.
- [22] C. Chekuri, S. Har-Peled, and K. Quanrud. Fast lp-based approximations for geometric packing and covering problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1019–1038. SIAM, 2020.
- [23] D. Z. Chen, J. Li, H. Liang, and H. Wang. Matroid and knapsack center problems. *Algorithmica*, 75:27–52, 2016.
- [24] A. Chiplunkar, S. Kale, and S. N. Ramamoorthy. How to solve fair k-center in massive data models. In *International Conference on Machine Learning*, pages 1877–1886. PMLR, 2020.
- [25] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 135–141, 2005.
- [26] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, 1988.
- [27] A. E. Feldmann and D. Marx. The parameterized hardness of the k-center problem in transportation networks. *Algorithmica*, 82:1899–2005, 2020.
- [28] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4:1–9, 1974.
- [29] H. Fukurai, E. W. Butler, and R. Krooth. Cross-sectional jury representation or systematic jury representation? simple random and cluster sampling strategies in jury selection. *Journal of Criminal Justice*, 19(1):31–48, 1991.
- [30] J. M. Gau. A jury of whose peers? the impact of selection procedures on racial composition and the prevalence of majority-white juries. *Journal of Crime and Justice*, 39(1):75–87, 2016.
- [31] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.
- [32] S. Guha. Tight results for clustering and summarizing data streams. In *Proceedings of the 12th International Conference on Database Theory*, pages 268–275, 2009.
- [33] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 150–158, 2005.
- [34] S. Har-Peled and B. Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *Journal of the ACM (JACM)*, 62(6):1–35, 2015.
- [35] S. Jiang, Z. Song, O. Weinstein, and H. Zhang. A faster algorithm for solving general lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 823–832, 2021.
- [36] M. Jones, H. Nguyen, and T. Nguyen. Fair k-centers via maximum matching. In *International conference on machine learning*, pages 4940–4949. PMLR, 2020.
- [37] M. Kahn. Diabetes. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5T59G>.
- [38] M. Kleinmuntz, P. Awasthi, and J. Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning*, pages 3448–3457. PMLR, 2019.
- [39] R. Matthew McCutchen and S. Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques: 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25–27, 2008. Proceedings*, pages 165–178. Springer, 2008.
- [40] M. Meek, T. Thiesson, and H. Heckerman. US Census Data (1990). UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5VP42>.
- [41] Z. Moumoulidou, A. McGregor, and A. Meliou. Diverse data selection under fairness constraints. *ICDT 2021*, 2021.
- [42] T. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179. IEEE, 2002.
- [43] K. D. Nguyen, N. Shahbazi, and A. Asudeh. Popsim: An individual-level population simulator for equitable allocation of city resources. *arXiv preprint arXiv:2305.02204*, 2023.
- [44] E. Oh and H.-K. Ahn. Approximate range queries for clustering. In *34th International Symposium on Computational Geometry (SoCG 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [45] P. Pope, C. Zhu, A. Abdulkader, M. Goldblum, and T. Goldstein. The intrinsic dimension of images and its impact on learning. *arXiv preprint arXiv:2104.08894*, 2021.
- [46] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations research*, 42(2):299–310, 1994.
- [47] A. Roberts, K. Vombatkere, and B. Suwal. GitHub - kvombatkere/CoreSets-algorithms: Python implementation of coresets algorithms for clustering and streaming.
- [48] A. Tamir. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4(4):550–567, 1991.
- [49] K. Verbeek and S. Suri. Metric embedding, hyperbolic space, and social networks. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 501–510, 2014.
- [50] Y. Wang, F. Fabbri, and M. Mathioudakis. Streaming algorithms for diversity maximization with fairness constraints. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 41–53. IEEE, 2022.
- [51] Y. Wang, F. Fabbri, M. Mathioudakis, and J. Li. Fair max-min diversity maximization in streaming and sliding-window models. *Entropy*, 25(7):1066, 2023.
- [52] Y. Wang, M. Mathioudakis, J. Li, and F. Fabbri. Max-min diversification with fairness constraints: Exact and approximation algorithms. *arXiv preprint arXiv:2301.02053*, 2023.
- [53] Y. Wang, S. Yu, L. Dhulipala, Y. Gu, and J. Shun. Pargeo: a library for parallel computational geometry. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 450–452, 2022.
- [54] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pages 1–8, 2012.