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 of size k from a large collection of n 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 coreset. Using our coreset, 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:

. 2024. Faster Algorithms for Fair Max-Min Diversification in $\mathbb{R}^d$. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

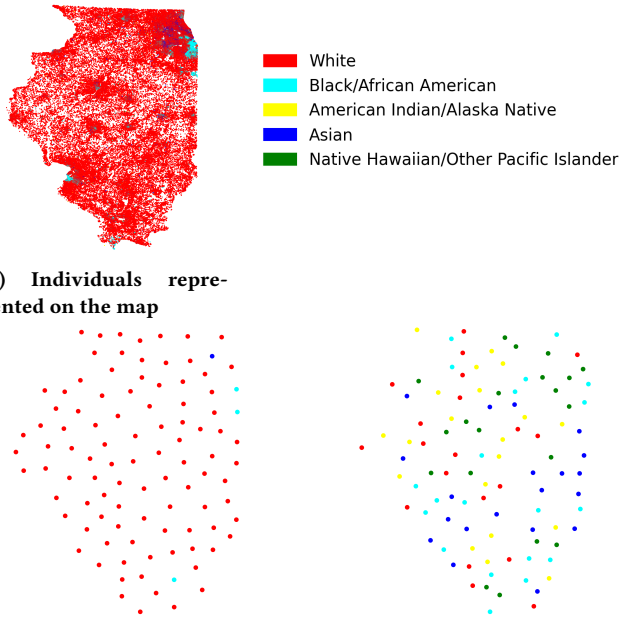
1 INTRODUCTION

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 satisfying a group fairness constraint.

We consider the problem of ensuring group fairness in max-min diversification. We are given a set of items[1], where each item belongs to one group determined by a sensitive attribute (we refer to it as color). Given a parameter k_i for each color i, the goal 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 [44, 48, 53]. Although we focus on the Euclidean space, some of our algorithms can be extended to metric spaces with bounded doubling dimension [14, 28, 33, 41].

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 [30]. 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 [3, 4],



(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.

[1]The terms item and points are used interchangeably.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

where many studies have recorded biases in jury and its consequences in their decisions [31].

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 $C = \{c_1, \ldots, c_m\}$. 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 $k_1, k_2, \ldots, k_m$ 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 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 $c_j \in C$.

Furthermore, we define the notion of *coreset*, which is useful in the next sections. A set $G \subseteq P$ is a $(1+\varepsilon)$-coreset for the FairDiv problem if there exists a subset $\hat{S} \subseteq G$ such that $\hat{S}$ satisfies the fairness constraints and $\text{div}(\hat{S}) \geq \gamma^*/(1+\varepsilon)$.

In many scenarios, the input consists of a stream of data which evolves over time, for example, tweets generated in real time or reviews of restaurants on Google or beer reviews on social media [2]. In the context of Twitter, the goal is to choose a representative subset of tweets in real time originating from various geographic locations (diversity). This selection should ensure that every topic (politics, sports, etc.) is sufficiently represented by related tweets (fairness). In this setting, the above discussed methods would need to be run from scratch for each new tweet. Instead, we study the extension of FairDiv problem when we receive the data in a *streaming setting*. This problem has been studied in [49] with various applications in modern database systems. In another example, an application might handle massive amounts of data that cannot be stored in memory to run an offline algorithm for the FairDiv problem. Instead, a pass is made over the data storing and maintaining only a small subset of elements in memory (synopsis) that is used to get an approximate solution for the FairDiv problem in the full dataset. During this pass, we maintain the synopsis efficiently under new insertions and, when needed, we should return a fair and diverse set representing all the items we have encountered in the stream. We have considered the beer reviews dataset to evaluate our techniques in a streaming setting (Section 6).

Definition 2 (**SFairDiv**). *We are given a set of colors C and integers $k_1, k_2, \ldots, k_m$ 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, a solution for the FairDiv problem in P_t can be constructed efficiently using only the points stored in $\hat{P}_t$.*

Notation	Meaning
P	Point set
n	$\lvert P \rvert$
C	Set of colors
m	$\lvert C \rvert$
$c(p)$	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)
ε	approximation error
S^*	Optimum solution for the FairDiv problem
$\text{div}(S)$	Diversity of set S (minimum pairwise distance)
γ^*	$\text{div}(S^*)$
$\mathcal{T}$	BBD-tree
A	Matrix representation in MWU method
h	probability vector in MWU method

Table 1: Table of Notations

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 variation 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 $k_1, k_2, \ldots, k_m$ 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 each $c_j \in C(S^*)$, it holds that $|S^*(c_j)| \geq k_j$.*

1.1 Contributions

In Table 2 we show our main results and compare them with the state–of–the–art methods.

In Section 3, we present MFD, the first near-linear time algorithm 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. For a constant $\varepsilon \in (0,1)$, we get an $O(nk \log^3 n)$ time algorithm 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 $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1-\varepsilon}, \forall c_j \in C$.

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 at least $1 - 1/\delta$, in time $O(nk \log^3 n + n \log \frac{1}{\delta} \log n)$ and space $O(n)$. The approximation factor becomes $\frac{1}{6(1+\varepsilon)}$.

Problem	Method	Time	Space	Approximation	Fairness
FairDiv/ No Coreset	[7]	$O(n^\lambda)$	$\Omega(n^2)$	$\frac{1}{2(1+\varepsilon)}$	Exact, Randomized
	[7]	$O(nkm^3)$	$O(n+km^2)$	$\frac{1}{(m+1)(1+\varepsilon)}$	Exact, Deterministic
	NEW	$O(nk)$	$O(n)$	$\frac{1}{2(1+\varepsilon)}$	$\frac{1}{1+\varepsilon}$-approx., Randomized
FairDiv/ Coreset	[7]	$O(nk+(mk)^\lambda)$	$\Omega(n+(km)^2)$	$\frac{1}{2(1+\varepsilon)}$	Exact, Randomized
	[7]	$O(nk+k^2m^4)$	$O(n+km^2)$	$\frac{1}{(m+1)(1+\varepsilon)}$	Exact, Deterministic
	NEW	$O(n+mk^2)$	$O(n)$	$\frac{1}{2(1+\varepsilon)}$	$\frac{1}{1+\varepsilon}$-approx., Randomized
		Update / **Post-processing**			
SFairDiv	[49]	$O(k\log\Delta)$ / $O((mk)^2\log\Delta)$	$O(mk\log\Delta)$	$\frac{1-\varepsilon}{3m+2}$	Exact, Deterministic
	[7]	$O(k\log\Delta)$ / $O((mk)^\lambda\log\Delta)$	$O(mk\log\Delta)$	$\frac{1}{2(1+\varepsilon)}$	Exact, Randomized
	NEW	$O(k)$ / $O(mk^2)$	$O(mk)$	$\frac{1}{2(1+\varepsilon)}$	$\frac{1}{1+\varepsilon}$-approx., Randomized
		Construction / **Query**			
QFairDiv	**NEW**	$O(n)$ / $O(mk^2)$	$O(n)$	$\frac{1}{2(1+\varepsilon)}$	$\frac{1}{1+\varepsilon}$-approx., Randomized

Table 2: Comparison of our new algorithms with state–of–the–art. $\Delta = O(2^n)$ is the spread (max. over min. pairwise distance). $\lambda > 2$ is the exponent such that an LP with N constraints can be solved in $O(N^\lambda)$ time. For simplicity, we skip $\log^{O(1)} n$ factors.

In Section 4 we show that any algorithm for the k'-center clustering can be used to derive a $(1+\varepsilon)$-coreset of small size efficiently. By constructing a coreset and then running our MFD algorithm we get a $\frac{1}{2(1+\varepsilon)}$-approximation algorithm for the FairDiv problem that runs in $O(n\log k + mk^2\log^3 k)$ time satisfying the fairness constraints approximately in expectation.

The generality of our coreset construction allows us to extend our algorithms in different settings. In Section 5, we design an efficient streaming algorithm for the SFairDiv problem, called StreamMFD, maintaining a coreset for the FairDiv problem. Our new streaming algorithm stores $O(mk)$ elements, takes $O(k\log k)$ update time per element for streaming processing, and $O(mk^2\log^3 k)$ time for post-processing to return a constant approximation for the SFairDiv problem satisfying the fairness constraints approximately. In the range-query setting, we 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 ε, and parameters $k_1,\ldots,k_m$, it returns a set $S \subseteq P\cap R$ in $O(mk\log^{d-1} n + mk^2\log^3 k)$ time, 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 experiments on real datasets showing that our new algorithms return 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. Overall, MFD provides the best balance between diversity and running time.

2 PRELIMINARIES

Known techniques for FairDiv. We first review the LP-based algorithm presented in [7] to find a solution for the FairDiv problem. They run a binary search over all possible pairwise distances. For a distance γ, they solve the following feasibility problem (LP).

(LP1)
$$\sum_{p_i\in P(c_j)} x_i \geq k_j \qquad \forall c_j \in C \tag{1}$$
$$\sum_{p_i\in P\cap\mathcal{B}(p,\gamma/2)} x_i \leq 1 \qquad \forall p \in P \tag{2}$$
$$1 \geq x_i \geq 0, \qquad \forall p_i \in P \tag{3}$$

$\mathcal{B}(p,\gamma/2)$ represents a ball with center p and radius $\gamma/2$. If (LP1) is infeasible, they try smaller values of γ. Otherwise, they try larger values of γ. 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 σ of $[n]$ as follows: $\sigma(t)$ is randomly chosen from $R_t = [n]\setminus\{\sigma(1),\ldots,\sigma(t-1)\}$ 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 σ, 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 ([7]). *The algorithm described above returns a set S with $\mathrm{div}(S) \geq \gamma^*/2$ such that for each color c_j, $\mathbb{E}[|S(c_j)|] \geq k_j$.*

Notice that 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 $O(n^\lambda)$ time, where λ is the exponent such that an LP with N variables and N constraints can be solved in $O(N^\lambda)$ time[2]. The rounding algorithm is executed in $O(n^2)$ time. Overall, the running time is $O(n^\lambda\log n)$.

In the same paper they also propose the Fair-Greedy-Flow algorithm that returns a $\frac{1}{(m+1)(1+\varepsilon)}$-approximation in $O(nkm^3\log n)$ time, for constant ε. The algorithm maps the FairDiv problem to a max-flow instance with $O(km)$ nodes and $O(mk^2)$ edges.

The authors in [7] also described a $(1+\varepsilon)$-coreset for the FairDiv problem. They run the well known Gonzalez algorithm [32] for the k'-center clustering problem in each set $P(c_j)$ independently,

[2]In any case, $\lambda \geq 2$. Currently, the best theoretical algorithm solving an LP has $\lambda \geq 2.37$ [35].

for $k' = \varepsilon^{-d}k$. Let G_j be the solution of the Gonzalez's algorithm in $P(c_j)$. Then, $G = \bigcup_{c_j \in C} G_j$. 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 ε, we get, an algorithm that returns a set S with $\text{div}(S) \geq \frac{1}{2(1+\varepsilon)}\gamma^*$ in $O(kn + (km)^\lambda)$ time such that for each color c_j, $\mathbb{E}[|S(c_j)|] \geq k_j$. To satisfy fairness exactly, we can combine the coreset with the Fair-Greedy-flow algorithm [7] to get a $\frac{1}{(m+1)(1+\varepsilon)}$-approximation in $O(kn + k^2m^4 \log k)$ time.

Diversity with high probability. All the results above, return a set S that satisfies fairness in expectation. The authors in [7] 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.

(FP1)
$$\sum_{p_i \in P(c_j)} y_i \geq k_j, \qquad \forall c_j \in C \tag{4}$$
$$\sum_{p_i \in P \cap \mathcal{B}(p,\gamma/6)} y_i \leq 1, \qquad \forall p \in P \tag{5}$$
$$y_i \geq 0, \qquad \forall p_i \in P \tag{6}$$
$$y_i > 0 \text{ and } y_\ell > 0 \Rightarrow ||p_i - p_\ell|| \geq \frac{\gamma}{3}, \quad \forall p_i, p_\ell \in P(c_j), \forall c_j \in C \tag{7}$$

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 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 (FP1) $\hat{y}$ takes $\Omega(n^2)$ time. Hence, the overall running time is super-quadratic. If the coreset is used, then we get a $\frac{1}{6(1+\varepsilon)}$-approximation algorithm in $O(kn + (mk)^{2.37})$ time satisfying the fairness constraints with high probability.

Geometric data structures.

BBD-tree. The main geometric data structure we use is the *BBD-tree* [11, 12], which is a variant of the quadtree [29]. 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 $\varepsilon \in (0, 1)$ and a ball $\mathcal{B}(x, r)$ in $\mathbb{R}^d$, the BBD-tree runs the query procedure $\mathcal{T}(x, r)$ that returns a set of nodes $\mathcal{U}(x, r) = \{u_1, \ldots, u_\kappa\}$ from $\mathcal{T}$ (also called *canonical nodes*) for $\kappa = O(\log n + \varepsilon^{-d})$ in $O(\log n + \varepsilon^{-d})$ time such that $\square_{u_i} \cap \square_{u_j} = \emptyset$ for every pair $1 \leq i < j \leq \kappa$, and $\mathcal{B}(x, r) \subseteq \bigcup_{1 \leq i \leq \kappa} \square_{u_i} \subseteq \mathcal{B}(x, (1 + \varepsilon)r)$. 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 $P \cap (\mathcal{B}(x, (1+\varepsilon)r) \setminus \mathcal{B}(x, r))$.

WSPD. Using a quadtree [29], someone can get a *Well Separated Pair Decomposition* (WSPD) [17, 33] in $P \in \mathbb{R}^d$. In $O(\varepsilon^{-d}n \log n)$ time, we can construct a list $\mathcal{L} = \{L_1, \ldots, L_z\}$ of $z = O(\varepsilon^{-d}n)$ distances, such that for every pair $p, q \in P$, there exists $L_j \in \mathcal{L}$ such that $(1 - \varepsilon)||p - q|| \leq L_j \leq (1 + \varepsilon)||p - q||$.

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

$$\exists x \in \mathcal{P} : Ax \leq b, \tag{8}$$

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 [10] 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 $h \in \mathbb{R}^{m'}$.

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

The ORACLE decides if there exists an x that satisfies the single linear constraint. Otherwise, it returns that there is no feasible solution. A ρ-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 $h[i] = (\delta_i \cdot \varepsilon/4 - 1)h[i]$, 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 $T = O(\rho \log(m')/\varepsilon^2)$ iterations, if every oracle was feasible, they 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. Overall, every algorithm using the MWU method to solve a problem in the form of Equation (8) should implement two procedures: Oracle$(\cdot)$ that implements a ρ-ORACLE and Update$(\cdot)$ that updates the probability vector h. In [10] they prove the following theorem.

Theorem 2.2 ([10]). *Given a feasibility problem as defined above, a parameter ε, a ρ-ORACLE implemented in procedure* Oracle$(\cdot)$*, and an update procedure* Update$(\cdot)$*, there is an algorithm which either finds an x such that $\forall i, A_i x_i \leq b_i + \varepsilon$ or correctly concludes that the system is infeasible. The algorithm makes $O(\rho \log(m')/\varepsilon^2)$ calls to procedures* Oracle$(\cdot)$ *and* Update$(\cdot)$.

3 EFFICIENT ALGORITHM 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. Recall the LP-based algorithm proposed in [7]: solve (LP1) using an LP solver and then round the solution as described in Section 2. We design a new algorithm that uses the MWU approach to solve a modified feasibility problem. While the

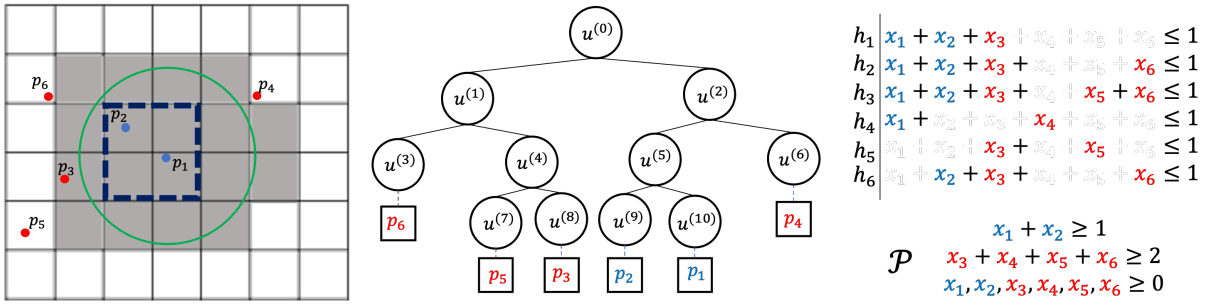


Figure 2: Left: Input set of points. Middle: Simplified BBD tree, Right: The decision problem $\exists x \in \mathcal{P}$ s.t. $h^\top Ax \leq b$.

MWU approach can work directly on (LP1) it takes $\Omega(n^2)$ time to run. Instead, we define a new linear feasibility problem, called (LP2) and we use the MWU method to approximately solve (LP2) in near-linear time. Finally, we round its fractional solution to return a valid solution for the FairDiv problem in near-linear time using advanced geometric data structures.

Assume that γ is a pairwise distance among the items in P. Our algorithm checks whether there exists a set $S \subseteq P$ that satisfies the fairness constraints such that $\text{div}(S) \geq \frac{\gamma}{2(1+\varepsilon)}$. We map this decision problem to a new linear feasibility problem (LP2). The Constraints (1) and Constraints (3) from (LP1) remain the same. However, we slightly modify Constraints (2). For a point p we define a set $S_p^\varepsilon \subseteq P$ denoting its "neighboring" points, with a definition of neighboring which is convenient for the data structure we use. The set S_p^ε contains all points within distance $\frac{\gamma}{2(1+\varepsilon)}$ from p, might contain some points within distance $\gamma/2$, and no point with distance more than $\gamma/2$. The properties of the BBD tree are used to formally define S_p^ε. We define $S_p^\varepsilon = \{p \in \square_{u_i} \cap P \mid u_i \in \mathcal{U}(p, \frac{\gamma}{2(1+\varepsilon)})\}$, i.e., the set of points in the canonical nodes returned by query $\mathcal{T}(p, \frac{\gamma}{2(1+\varepsilon)})$. We replace Constraints (2) with $\sum_{p_i \in S_p^\varepsilon} x_i \leq 1, \forall p \in P$. Overall, the new feasibility problem is:

(LP2)
$$\sum_{p_i \in P(c_j)} x_i \geq k_j \qquad \forall c_j \in C \tag{10}$$
$$\sum_{p_i \in S_p^\varepsilon} x_i \leq 1 \qquad \forall p \in P \tag{11}$$
$$1 \geq x_i \geq 0, \qquad \forall p_i \in P \tag{12}$$

We use the MWU method to compute a feasible solution for (LP2). Recall that the MWU method solves feasibility problems in the form of Equation (8), $\exists x \in \mathcal{P} : Ax \leq b$. Next, we show that (LP2) can be written in this form by defining $\mathcal{P}$, A, and b.

Instead of considering that the trivial constraints $\mathcal{P}$ contains only the inequalities $1 \geq x_i \geq 0$, we assume that Constraints (10) are also trivial and contained in $\mathcal{P}$. This will allow us later to design a k-ORACLE. 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. The new Constraints (11) define the binary square matrix A, 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. Finally, b is defined as a vector in $\mathbb{R}^d$ with all elements being 1. From Theorem 2.2, we know that the MWU method returns a solution that satisfies the constraints in $\mathcal{P}$ (Constraints (10) and (12)) exactly, while the Constraints (11) are satisfied with an ε additive error.

A straightforward implementation of the MWU method over (LP2) would still take super-quadratic time to run; even the computation of A takes $O(n^2)$ time. Our new algorithm does not construct the matrix A explicitly. Likewise, it does not construct the sets S_p^ε explicitly. Instead, we use geometric data structures to implicitly represent A and S_p^ε and execute our algorithm in near-linear time.

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 [21, 23, 26]. 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 [23] 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 [21, 23] 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.

Example. We use a toy example to describe our new algorithm in the next paragraphs. In Figure 2 (Left) we show the input points that consists of two blue points p_1, p_2 and four red points p_3, p_4, p_5, p_6. Let blue color be c_1 and red color be c_2. The goal is to solve the FairDiv problem among the five points with $k_1 = 1$ and $k_2 = 2$, i.e., our solution should contain at least one blue and at least two red points. Throughout the example, we assume that $\gamma = 5$ and $\varepsilon = 1$. In Figure 2 (Middle) we show a simplified version of the BBD tree constructed on the input set. Each node corresponds to a rectangle in $\mathbb{R}^d$. For example, the node $u^{(5)}$ corresponds to the dashed rectangle that contains p_1 and p_2 in Figure 2 (Left). The leaf nodes in the BBD tree correspond to the non-empty cells in Figure 2 (Left). For simplicity, we assume that each grid cell has diagonal 1

Algorithm 1: MFD$(P, \varepsilon, k_1, \ldots, k_m)$

1 $\Gamma \leftarrow$ Sorted array of pairwise distances in P;
2 $M_l \leftarrow 0,\ M_u \leftarrow |\Gamma| - 1,\ k \leftarrow k_1 + \ldots + k_m$;
3 $T \leftarrow O(\varepsilon^{-2}\rho \log n)$;
4 **while** $M_l \neq M_u$ **do**
5 $\quad M \leftarrow \lceil (M_l + M_u)/2 \rceil,\ \ \gamma \leftarrow \Gamma[M]$;
6 $\quad \hat{x} \leftarrow (0, \ldots, 0)^\top \in \mathbb{R}^n$;
7 $\quad h \leftarrow (\frac{1}{n}, \ldots, \frac{1}{n})^\top \in \mathbb{R}^n$;
8 $\quad$ **for** $1, \ldots, T$ **do**
9 $\quad\quad \bar{x} \leftarrow$Oracle$(P, h, \gamma, \varepsilon, k_1, \ldots, k_m)$;
10 $\quad\quad$ **if** $\bar{x} \neq \emptyset$ **then**
11 $\quad\quad\quad \hat{x} \leftarrow \hat{x} + \bar{x}$;
12 $\quad\quad\quad h \leftarrow$Update$(P, \bar{x}, \gamma, \varepsilon)$;
13 $\quad\quad$ **else**
14 $\quad\quad\quad M_u \leftarrow M - 1$, Go to Line 4;
15 $\quad \hat{x} \leftarrow \hat{x}/T$;
16 $\quad M_l \leftarrow M$;
17 $S \leftarrow$Round$(P, \varepsilon, \hat{x})$;
18 **return** S;

Algorithm 2: Oracle$(P, h, \gamma, \varepsilon, k_1, \ldots, k_m)$

1 $\mathcal{T} \leftarrow$ BBD tree on P;
2 **foreach** $u \in \mathcal{T}$ **do** $u_s \leftarrow 0$;
3 **foreach** $p_\ell \in P$ **do**
4 $\quad \mathcal{U}_{p_\ell} \leftarrow \mathcal{T}(p_\ell, \frac{\gamma}{2(1+\varepsilon)})$;
5 $\quad$ **foreach** $u \in \mathcal{U}_{p_\ell}$ **do** $u_s \leftarrow u_s + h[\ell]$;
6 **foreach** $p_i \in P$ **do**
7 $\quad w_i \leftarrow 0$;
8 $\quad v \leftarrow$ leaf node of $\mathcal{T}$ such that $p_i \in \square_v \cap P$;
9 $\quad$ **foreach** *u in the path from v to the root of $\mathcal{T}$* **do**
10 $\quad\quad w_i \leftarrow w_i + u_s$
11 $\bar{x} = (0, \ldots, 0) \in \mathbb{R}^n$;
12 **foreach** $c_j \in C$ **do**
13 $\quad W_j \leftarrow k_j$-th smallest weight in $\{w_i \mid p_i \in P(c_j)\}$;
14 $\quad P_j \leftarrow \{p_i \in P(c_j) \mid w_i \leq W_j\}$;
15 $\quad$ **foreach** $p_i \in P_j$ **do** $\bar{x}_i = 1$;
16 **if** $\sum_{p_i \in P} \bar{x}_i w_i \leq 1$ **then return** $\bar{x}$;
17 **else return** $\emptyset$ (Infeasible) ;

(which is equal to the maximum error ε). Consider the blue point p_1. The green circle with center p_1 has radius $\gamma/(2(1+\varepsilon)) = 1.25$. By definition, all points that lie in grid cells that are intersected by the green circle (the gray cells in Figure 2 (Left)) belong in $S^\varepsilon_{p_1}$, i.e. $S^\varepsilon_{p_1} = \{p_1, p_2, p_3\}$. Notice that $p_3 \in S^\varepsilon_{p_1}$ because the green circle around p_1 intersects the grid cell (associated with the node $u^{(8)}$) that p_3 belongs to. On the other hand $p_4 \notin S^\varepsilon_{p_1}$. Interestingly, $S^\varepsilon_{p_4} = \{p_1, p_4\}$ because a ball of radius 1.25 and center p_4 intersects the cell that p_1 belongs to. We also have $S^\varepsilon_{p_2} = \{p_1, p_2, p_3, p_6\}$, $S^\varepsilon_{p_3} = \{p_1, p_2, p_3, p_5, p_6\}$, $S^\varepsilon_{p_5} = \{p_3, p_5\}$, and $S^\varepsilon_{p_6} = \{p_2, p_3, p_6\}$ Using the sets S^ε_p, in Figure 2 (Right), we show the (LP2) for our instance. The trivial constraints $\mathcal{P}$ are shown in the bottom, while the main constraints are shown at the top.

New Algorithm. Our new algorithm is called **M**ultiplicative weight update method for **F**air **D**iversification (MFD) and it is described in Algorithm 1. First it computes a sorted array Γ of (candidates of) pairwise distances in P. Then it runs a binary search on Γ. Lines 2, 4, 5, 13, 14, 16 are all trivial executing the binary search on Γ. Each time we find a feasible (infeasible) solution for our optimization problem (LP2) we try larger (smaller) values of γ. For a particular $\gamma \in \Gamma$, in lines 4–16 we use the MWU method to solve (LP2). The algorithm follows the MWU method as described in the end of Section 2. In particular, for at most $T = O(\varepsilon^{-2}\rho \log n)$ iterations, it calls Oracle$(\cdot)$ in line 9 to decide whether there exists x such that $h^\top Ax \leq h^\top b$ and $x \in \mathcal{P}$. Recall that in our case $b \in \mathbb{R}^n$ and $b = \{1, \ldots, 1\}$, so it is sufficient to decide whether there exists x such that $h^\top Ax \leq \sum_\ell h[\ell] \Leftrightarrow h^\top Ax \leq 1$ and $x \in \mathcal{P}$. If Oracle() returns a feasible solution $\bar{x}$, in line 11 it updates the final solution $\hat{x}$ and uses the Update$(\cdot)$ procedure to update the vector h based on $\bar{x}$. If all T iterations return feasible solutions, in line 15 it computes the final solution of (LP2) for a given γ. In the end, in line 17 we run a rounding procedure to derive the final set of points S. Overall, the algorithm follows the high level idea of the LP-based algorithm in [7], using the MWU method [10] instead of an LP solver. If Q_Γ, Q_O, Q_U, Q_R is the running time to compute Γ, and run Oracle$(\cdot)$ Update$(\cdot)$, and Round$(\cdot)$, respectively, then the overall running time of Algorithm 1 is $O\left(Q_\Gamma + (\varepsilon^{-2}\rho \log n)(Q_O + Q_U + n) \log |\Gamma| + Q_R\right)$. Using the results in [10] and [7], $Q_\Gamma, Q_O, Q_U, Q_R = \Omega(n^2)$, leading to a super-quadratic time algorithm. In the next paragraphs, we use geometric tools to show how we can find a set Γ and run all the procedures Oracle, Update, Round only in near-linear time using only linear space with respect to n. We also show that $\rho = k$.

The Oracle$(\cdot)$ **procedure.** We design a k-ORACLE procedure as defined in Section 2. The goal is to decide whether there exists x such that $h^\top Ax \leq 1$ and $x \in \mathcal{P}$. We note that Oracle$(\cdot)$ does not compute the matrix A explicitly. In fact, $h^\top Ax$ can be written as $\sum_{p_i \in P} \alpha_i x_i$, for some real coefficients α_i. Intuitively, for every color c_j, our goal is to find the k_j points with color c_j, having the smallest coefficients α_i. Our algorithm first finds all coefficients α_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 slightly modified 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 initialize a weight $u_s = 0$. The data structure $\mathcal{T}$ has $O(n)$ space and can be constructed in $O(n \log n)$ time.

Using our modified BBD tree $\mathcal{T}$, in Algorithm 2 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 $p_\ell \in P$ we run the query $\mathcal{T}(p_\ell, \frac{\gamma}{2(1+\varepsilon)})$ and we get the set of canonical nodes $\mathcal{U}_{p_\ell} := \mathcal{U}(p_\ell, \frac{\gamma}{2(1+\varepsilon)})$. For each node $u \in \mathcal{U}_{p_\ell}$, we update $u_s \leftarrow u_s + h[\ell]$. 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 $h^\top Ax = \sum_{p_i \in P} w_i x_i$. Recall that $h^\top Ax = \sum_{p_i \in P} \alpha_i x_i$, for the real coefficients α_i. By definition, each α_i is defined as $\alpha_i = \sum_{p_\ell \in P} h[\ell] \cdot \mathcal{I}(p_i \in S^\varepsilon_{p_\ell})$, where $\mathcal{I}(p_i \in S^\varepsilon_{p_\ell}) = 1$ if $p_i \in S^\varepsilon_{p_\ell}$ and 0 otherwise. If $p_i \in S^\varepsilon_{p_\ell}$ then by definition α_i contains a term $h[\ell]$ in the sum. There exists also 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.

Let $\bar{x}$ be the feasible solution returned by Oracle$(\cdot)$. Notice that by definition, $\bar{x}$ sets k variables to 1. Hence, for each Constraint (11), it holds that $A_i \bar{x} - b_i \leq k - 1$ and $A_i \bar{x} - b_i \geq -1$, where A_i is the i-th row of A. Similarly, we can write $A_i \bar{x} \leq k$ and $A_i \bar{x} \geq 0$ since $b_i = 1$. We conclude that our Oracle procedure computes a k-ORACLE as defined in [10], so $\rho = k$.

Example (cont). We show the execution of Algorithm 2 in our example. Assume that $h^\top = [.1, .1, .1, .1, .4, .2]$, with $h_1 + h_2 + h_3 + h_4 + h_5 + h_6 = 1$. By the definition of canonical subsets in the BBD tree, we have $\mathcal{U}_{p_1} = \{u^{(5)}, u^{(8)}\}$, $\mathcal{U}_{p_2} = \{u^{(5)}, u^{(8)}, u^{(3)}\}$, $\mathcal{U}_{p_3} = \{u^{(1)}, u^{(5)}\}$, $\mathcal{U}_{p_4} = \{u^{(6)}, u^{(10)}\}$, $\mathcal{U}_{p_5} = \{u^{(4)}\}$, and $\mathcal{U}_{p_6} = \{u^{(3)}, u^{(8)}, u^{(9)}\}$. Hence, from lines 3–5 we get $u_s^{(1)} = h_2 + h_3 = 0.2$, $u_s^{(3)} = h_1 = 0.3$, $u_s^{(4)} = h_6 = 0.1$, $u_s^{(5)} = h_4 + h_5 = 0.4$, $u_s^{(6)} = h_4 = 0.05$, $u_s^{(7)} = h_6 = 0.1$, $u_s^{(8)} = h_1 = 0.3$, and all the rest $u_s^{(i)} = 0$. Then in lines 6–10 we compute the coefficients w_i. For example consider p_1. We compute $w_1 = u_s^{(10)} + u_s^{(5)} + u_s^{(2)} + u_s^{(0)} = 0.4$. Similarly, we compute $w_2 = 0.5$, $w_3 = 0.9$, $w_4 = 0.1$, $w_5 = 0.5$, and $w_6 = 0.4$. We observe that these are indeed the correct coefficients in the inequality $h^\top Ax \leq 1$. For instance, the coefficient of x_1 is $h_1 + h_2 + h_3 + h_4 = 0.4 = w_1$. Overall, we have $h^\top Ax = 0.4 \cdot x_1 + 0.5 \cdot x_2 + 0.9 \cdot x_3 + 0.1 \cdot x_4 + 0.5 \cdot x_5 + 0.4 \cdot x_6$ and we correctly identified all coefficients. Then in line 13 among the blue points we choose the smallest weight, $W_1 = w_1 = 0.4$ and among the red points we choose the second smallest weight $W_2 = w_6 = 0.4$. Hence, $P_1 = \{p_1\}$ and $P_2 = \{p_4, p_6\}$ and the algorithm sets $\bar{x}^\top = [1, 0, 0, 1, 0, 1]$. Finally, in line 16 the algorithm computes $w_1 + w_4 + w_6 = 0.4 + 0.4 + 0.1 = 0.9 < 1$ so $\bar{x}$ is a feasible solution.

Algorithm 3: Update$(P, \bar{x}, \gamma, \varepsilon)$

```
1  T ← BBD tree on P;
2  foreach u ∈ T do u_w ← 0;
3  foreach p_i ∈ P such that x̄_i > 0 do
4      v ← leaf node of T such that p_i ∈ □_v ∩ P;
5      foreach u in the path from v to the root of T do
6          u_w ← u_w + x̄_i;
7  foreach p_ℓ ∈ P do
8      U_{p_ℓ} ← T(p_ℓ, γ/(2(1+ε)));
9      R_ℓ ← Σ_{u∈U_{p_ℓ}} u_w;
10     δ_ℓ = |R_ℓ − 1|;
11 Update h using δ_ℓ's as described in [10];
12 return h;
```

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. Overall, $Q_O = O(n \log n + n\varepsilon^{-d})$.

The Update$(\cdot)$ **procedure.** 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_ℓ is the ℓ-th row of A (ℓ-th constraint in (11)). In [10] they update each $h[\ell]$ in constant time after computing δ_ℓ. In our case, if we try to calculate all δ_ℓ's with a trivial way we would need $\Omega(nk)$ time leading to a $\Omega(nk^2)$ time overall algorithm. In Algorithm 3 we show a faster way to calculate all δ_ℓ's. Our Oracle method sets k variables $\bar{x}$ to 1. For each $p_\ell \in P$ the goal is to find $A_\ell \bar{x} = \sum_{p_i \in S^\varepsilon_{p_\ell}} \bar{x}_i = \sum_{p_i \in S^\varepsilon_{p_\ell}, \bar{x}_i > 0} \bar{x}_i$. 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 $\bar{x}_i > 0$ we start from the leaf containing p_i, and we visit the tree bottom up until we reach the root. For each node u we encounter, we update $u_w = u_w + \bar{x}_i$. After the modification of $\mathcal{T}$, for each constraint/point p_ℓ we run the query $\mathcal{T}(p_\ell, \frac{\gamma}{2(1+\varepsilon)})$ and we get the set of canonical nodes $\mathcal{U}_{p_\ell}$. We compute $R_\ell \leftarrow \sum_{u \in \mathcal{U}_{p_\ell}} u_w = \sum_{p_i \in S^\varepsilon_{p_\ell}} \bar{x}_i = A_\ell \bar{x}$. We return $\delta_\ell = |R_\ell - 1|$. The correctness follows by observing that the coefficient of p_i's variable in the ℓ-th row of A is 1 if and only if $p_i \in S^\varepsilon_{p_\ell}$. We need $O(n)$ time to compute all values v_w by traversing the tree $\mathcal{T}$ bottom up. Then for each p_ℓ we run a range query on $\mathcal{T}$ so we need $O(\log n + \varepsilon^{-d})$. Overall, $Q_U = O(n \log n + n\varepsilon^{-d})$.

Example (cont). We show the execution of the Update procedure using our example in Figure 2, assuming that $\bar{x}^\top = [1, 0, 1, 0, 1, 0]$. By definition, $\delta_1 = 1$, $\delta_2 = 1$, $\delta_3 = 2$, $\delta_4 = 0$, $\delta_5 = 1$, and $\delta_6 = 0$. In lines 4–6, for p_1 we start from $u^{(10)}$ and we traverse the tree bottom-up. We set $u_w^{(10)} = u_w^{(5)} = u_w^{(2)} = u_w^{(0)} = \bar{x}[1] = 1$. After traversing all points we have, $u_w^{(10)} = 1$, $u_w^{(5)} = 1$, $u_w^{(2)} = 1$, $u_w^{(7)} = 1$, $u_w^{(8)} = 1$, $u_w^{(4)} = 2$, $u_w^{(1)} = 2$, $u_w^{(0)} = 3$, and all the other nodes have weight 0. Then in line 10, we compute δ_ℓ for each p_ℓ. For p_1, we get

Algorithm 4: Round$(P, \varepsilon, \hat{x})$

```
1  F ← P;
2  Ξ ← BBD tree for sampling on F;
3  foreach u ∈ Ξ do u_b ← 1 ;
4  while F ≠ ∅ do
5      p_i ← Ξ.sample(),   F ← F \ {p_i};
6      U_{p_i} ← Ξ(p_i, γ/(2(1+ε)));
7      if u_b == 1 for every u ∈ U_{p_i} then
8          S ← S ∪ {p_i};
9      v ← leaf node of Ξ such that p_i ∈ □_v ∩ P;
10     foreach u in the path from v to the root of Ξ do u_b ← 0 ;
11 return S;
```

$\mathcal{U}_{p_1} = \{u^{(5)}, u^{(8)}\}$, so $R_1 = u_w^{(5)} + u_w^{(8)} = 2$, and $\delta_1 = |2-1| = 1$. Similarly, we compute the other δ_ℓ's.

The Round$(\cdot)$ **procedure.** The real vector $\hat{x}$ we get satisfies (LP2) approximately. From the MWU method (see Theorem 2.2) the Constraints in $\mathcal{P}$, (Constraints (10) and (12)) are satisfied exactly, however the Constraints (11) are satisfied approximately. In fact, it holds that

$$\sum_{p_i \in S_p^\varepsilon} \hat{x}_i \leq 1 + \varepsilon, \qquad \forall p \in P \tag{13}$$

We follow a modified version of the randomized rounding from [7] 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 [7] (also briefly described in Section 2) because, i) the Constraints (2) are quite different from the Constraints (13) since the latter are satisfied with an additive error ε, and their sum is over the set S_p^ε, and ii) the rounding technique in [7] 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 Ξ be a BBD tree constructed over a weighted set P. We modify Ξ so that we can sample a point p_i with probability $\frac{\hat{x}_i}{\sum_{j \in F} \hat{x}_j}$, where $F \subseteq P$ is a subset of P. For each node u of Ξ, 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 u having two children v, e. We visit v with probability $v_s/(v_s + e_s)$ and we visit e with probability $e_s/(u_s + e_s)$. 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 $u_s \leftarrow u_s - \hat{x}_i$ of the nodes u we traverse. 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 Ξ 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 Ξ 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 $\Xi(p_i, \frac{\gamma}{2(1+\varepsilon)})$ and we get a set of $O(\log n + \varepsilon^{-d})$ canonical nodes $\mathcal{U}_{p_i}$. 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 $S_{p_i}^\varepsilon$ 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.

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

Proof. First, we show that the minimum pairwise distance in S is $\frac{\gamma}{2(1+\varepsilon)}$. Let p_i, p_ℓ be a pair of points with distance less than $\frac{\gamma}{2(1+\varepsilon)}$. Without loss of generality assume that p_i was added to S first. Assume that p_ℓ is selected as a sample in a subsequent iteration. Since $||p_i - p_\ell|| < \frac{\gamma}{2(1+\varepsilon)}$, 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_ℓ is not inserted in S.

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

$$\Pr[p_\ell \in S] = \sum_{t=1}^{n} \Pr[\sigma(t) = p_\ell \mid V_t] \Pr[V_t] = \sum_{t=1}^{n} \frac{\hat{x}_\ell}{\sum_{p_i \in S_{p_\ell}^\varepsilon} \hat{x}_i} \Pr[V_t]$$

$$= \frac{\hat{x}_\ell}{\sum_{p_i \in S_{p_\ell}^\varepsilon} \hat{x}_i} \sum_{t=1}^{n} \Pr[V_t] = \frac{\hat{x}_\ell}{\sum_{p_i \in S_{p_\ell}^\varepsilon} \hat{x}_i} \geq \frac{\hat{x}_\ell}{1+\varepsilon}$$

The last inequality holds because $\sum_{p_i \in S_{p_\ell}^\varepsilon} \hat{x}_i \leq 1 + \varepsilon$ from Constraints (13). For $c_j \in C$, $\mathbb{E}[|S(c_j)|] \geq \sum_{p_i \in P(c_j)} \frac{\hat{x}_i}{1+\varepsilon} \geq \frac{k_j}{1+\varepsilon}$. □

The rounding is executed in $Q_R = O(n(\log n + \varepsilon^{-d}))$ time because Ξ has $O(\log n)$ height and each query takes $O(\log n + \varepsilon^{-d})$ time.

Example (cont). We show the execution of the Round procedure using our example in Figure 2. Let $\hat{x}^\top = [.2, .2, .05, .15, .25, .15]$. Initially, for every node $u^{(i)}$ in Figure 2 (Middle), we have $u_b^{(i)} = 1$. In the while loop (lines 4–10) we first sample a point from P. Let p_2 be the first point we sample. We get $\mathcal{U}_{p_2} = \{u^{(5)}, u^{(3)}, u^{(8)}\}$. All nodes in $\mathcal{U}_{p_2}$ have weight 1 so in line 8, we add p_2 in S. Then, starting from $u^{(9)}$, we set all the weights of the nodes to 0 until we reach the root. Hence, we set $u_b^{(9)} = u_b^{(5)} = u_b^{(2)} = u_b^{(0)} = 1$. In the next iteration of the while loop, assume that we sample the point p_6. We have $\mathcal{U}_{p_6} = \{u^{(3)}, u^{(8)}, u^{(9)}\}$. We observe that $u_b^{(9)} = 0$ so we do not add p_6 in S. Intuitively, p_6 is very close to p_2 that we have already added. Starting from $u^{(3)}$, we set $u_b^{(3)} = u_b^{(1)} = 0$. In the next iteration, assume that we sample p_5. We get $\mathcal{U}_{p_5} = \{u^{(4)}\}$. In line 7, we observe that $u_b^{(4)} = 1$ so we add p_5 in S. We also set $u_b^{(7)} = u_b^{(4)} = 0$ (all the other nodes above $u^{(4)}$ have already weight equal to 0). Next, assume that we sample p_3. We have $\mathcal{U}_{p_3} = \{u^{(5)}, u^{(1)}\}$. However, $u_b^{(5)} = u_b^{(1)} = 0$ so we do not

add p_3 in S. We also set $u_b^{(8)} = 0$. Next, assume that we sample p_4. We have $\mathcal{U}_{p_4} = \{u^{(6)}, u^{(10)}\}$ with $u_b^{(6)} = u_b^{(10)} = 1$. So we add p_4 in S and we set $u_b^{(6)} = 0$. In the final iteration, we sample p_1. We get $\mathcal{U}_{p_1} = \{u^{(5)}, u^{(8)}\}$ and we observe that $u_b^{(5)} = 0$ so we do not add p_1 in S. We set $u_b^{(10)} = 0$. Our algorithm returns $S = \{p_2, p_5, p_4\}$.

Compute the set Γ. We note that so far we assumed that γ is any distance and we tried to find a solution S with $\text{div}(S) \geq \frac{\gamma}{2(1+\varepsilon)}$. In order to find a good approximation of the optimum diversity γ^*, we use the notion of the Well Separated Pair Decomposition (WSPD) [17, 33] briefly described in Section 2. Let Γ be the sorted array of $O(n/\varepsilon^d)$ distances from WSPD. Any pairwise distance in P can be approximated by a distance in the array Γ within a factor $1+\varepsilon$, hence, we might not get the optimum γ^* exactly. In the worst case, we might get a smaller value which is at least $\frac{\gamma^*}{1+\varepsilon}$. We need $O(n\varepsilon^{-d}\log n)$ time to compute and sort the WSPD distances, so $Q_\Gamma = O(n\varepsilon^{-d}\log n)$.

Putting everything together, 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 $k_1, \ldots, k_m$ be m integer parameters with $k_1 + \ldots + k_m = k$. Let $\varepsilon \in (0,1)$ be a constant. 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 $\text{div}(S) \geq \frac{\gamma^*}{2(1+\varepsilon)}$ 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 [7], 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 $S_p^\varepsilon = S_p(\frac{\gamma}{2(1+\varepsilon)})$.

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

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

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

$$y_i \geq 0, \qquad \forall p_i \in P \tag{16}$$

$$y_i > 0 \text{ and } y_\ell > 0 \Rightarrow ||p_i - p_\ell|| \geq \frac{\gamma}{3(1+\varepsilon)^2}, \quad \forall p_i, p_\ell \in P(c_j), \forall j \in [m] \tag{17}$$

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 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 subrtee 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 another variable $\hat{y}_\ell$ in the next iterations. For Constraints (15), for every point $p \in P$, we have,

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

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_ℓ 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 Ξ 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 $\Xi(p_i, \frac{\gamma}{6(1+\varepsilon)^3})$[3]. 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

[3]For technical reasons we set the denominator to $(1+\varepsilon)^3$ instead of $(1+\varepsilon)^2$.

of points of each color with high probability. In particular, if $k_j \geq 3\frac{1+\varepsilon}{\varepsilon^2}\log(2m)$, 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 γ such that $\gamma^* \geq \gamma \geq \gamma^*/(1+\varepsilon)$.

Putting everything together, and setting $\varepsilon \leftarrow \varepsilon/6$,[4] 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 $k_1, \ldots, k_m$ be m integer parameters with $k_1 + \ldots + k_m = k$ and $k_j \geq 3(1+\varepsilon)\varepsilon^{-2}\log(2m)$ for each $c_j \in C$. Let ε be a constant parameter and δ 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 $\mathrm{div}(S) \geq \frac{\gamma}{6(1+\varepsilon)}$ 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 [7] they describe a $(1+\varepsilon)$-coreset for the FairDiv problem. In particular, using the Gonzalez algorithm [32] 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 coreset construction. We show that if $k' = O(\varepsilon^{-2d}k)$ points are chosen for each color $c_j \in C$ using any constant approximation algorithm for the k'-center clustering problem in the Euclidean space, then their union is a valid coreset for the FairDiv problem.

Let Alg be an α-approximation for the k'-center clustering problem that runs in $O(T(n,k'))$ time. For every color $c_j \in C$, we run Alg on $P(c_j)$ for $k' = O(\varepsilon^{-2d}k)$ and we get the set of centers G'_j. We return the coreset $G' = \bigcup_{c_j \in C} G'_j$. The coreset 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 $(1+\varepsilon)$-coreset for the FairDiv problem.*

Proof. We fix a color $c_j \in C$. For any k, let μ_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 ξ be a constant number. Let D be a grid in $\mathbb{R}^d$ such that each grid cell has side length $\frac{\varepsilon \cdot \mu_{\hat{k}}}{4\alpha \cdot \xi}$. 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 [6, 8] 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 $\frac{\varepsilon}{4\alpha}\mu_{\hat{k}}$. Hence, $\mu(A) \leq \frac{\varepsilon}{4\alpha}\mu_{\hat{k}}$. By definition it also holds that $\mu_{k'} \leq \mu(A)$. It follows that $\mu_{k'} \leq \frac{\varepsilon}{4\alpha}\mu_{\hat{k}}$. We have, $\mu(G'_j) \leq \alpha\mu_{k'} \leq \frac{\varepsilon}{4}\mu_{\hat{k}}$. For any k, let σ_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 $\sigma_k \geq \mu_k$ [47]. It is also known that the Gonzalez algorithm for k iterations returns a solution with diversity at least $\frac{1}{2}\sigma_k$ [47].

Let $O = \{o_1, \ldots, o_{\hat{k}}\}$ 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 $P' \subseteq G'$ is a valid $(1+\varepsilon)$-coreset. Let $r = ||o_i - o_\ell||$ for any pair $o_i, o_\ell \in O$ We have $||p_i - p_\ell|| \geq r - ||o_i - p_i|| - ||o_\ell - p_\ell|| \geq r - \frac{\varepsilon}{2}\mu_{\hat{k}} \geq r - \frac{\varepsilon}{2}\sigma_{\hat{k}} \geq (1-\varepsilon)r$. 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 [7] are satisfied within a $(1-\varepsilon)$ (or $(1+\varepsilon)$) factor, so by setting $\varepsilon \leftarrow \varepsilon/\zeta$, for a sufficiently large constant ζ that depends on d, we conclude that G' is an $(1+\varepsilon)$-coreset for FairDiv. □

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 $(1+\varepsilon)$-coreset 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 result we fix Alg to be either the Feder and Greene [27] algorithm or the Har-Pelled 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 coreset 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 $\mathrm{div}(S) \geq \frac{\gamma^*}{2(1+\varepsilon)}$ in $O(n\log k + mk^2\log^3(k))$ time and $O(n)$ space. The same algorithm can be executed in $O(n + mk^2\log^3(k))$ expected time.*

Using our coreset 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 $\mathrm{div}(S) \geq \frac{\gamma^*}{6(1+\varepsilon)}$ 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.*

We notice that the running time of the algorithm in Corollary 4.3 or Corollary 4.4 is not always faster than the algorithm in Theorem 3.2 or Theorem 3.3, respectively. While for small values of k, an algorithm using the coreset is faster, when k is large the asymptotic running time of the algorithm without using the coreset is faster.

[4] 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 EXTENSIONS

In this section, we show how our coreset 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.

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.

There are two known algorithms for the SFairDiv problem. In [49], they describe a streaming algorithm that stores $O(mk \log \Delta)$ elements in memory, takes $O(k \log \Delta)$ time per element for streaming processing, and $O(m^2k^2 \log \Delta)$ time for post-processing that returns a $\frac{1-\varepsilon}{3m+2}$-approximation solution. In [7], they improved the approximation factor to a constant, however all asymptotic complexities still depend on $\log \Delta$. Notice that Δ 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, called StreamMFD, for the FairDiv problem whose space, update, and post-processing time are independent of Δ.

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 [22] 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 doubling algorithm to maintain a constant approximation for the k-center clustering, our new coreset 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 ε, 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 $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1+\varepsilon}$ for each color $c_j \in C$, and $\text{div}(S) \geq \frac{\gamma^*}{2(1+\varepsilon)}$.*

In the streaming setting, our new algorithm is called StreamMFD.

5.2 Range-query setting

Given a set of n points $P \in \mathbb{R}^d$, in [6, 43] 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 $(2+\varepsilon)$-approximation of the k-center clustering in $P \cap R$ is returned in $O(k \log^{d-1} n + \varepsilon^{-d} k \log \frac{k}{\varepsilon})$ time. Using this k-center data structure, we construct our coreset 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 ε, and parameters $k_1, \ldots, k_m$ with $k_1 + \ldots k_m = k$, it returns a set $S \subseteq P \cap R$ in $O(mk \log^{d-1} n + mk^2 \log^3 k)$ time such that $\mathbb{E}[S(c_j)] \geq \frac{k_j}{1+\varepsilon}$ for each color $c_j \in C$, and $\text{div}(S) \geq \frac{\gamma^*}{2(1+\varepsilon)}$.*

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 MFD behavior change with varying parameters? Is the fairness requirement violated by MFD? This RQ identifies the best parameters for MFD to be used for comparison with baselines.

RQ2: How does MFD result compare against other baselines? Is it efficient to identify a fair and diverse solution?

RQ3: How efficient is our new algorithm StreamMFD in the streaming setting?

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
Beer reviews	Style of beer	3	6	1,518,829

Table 3: Dataset Statistics

Datasets We consider the following datasets.

(1) The Adult dataset[13] 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 [39] 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 [42] 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 [42]. Another version of the same semi-synthetic dataset containing almost 1 million individuals.
(6) We use Beer reviews dataset for experiments in the streaming setting. The dataset contains a large number of reviews over different beers. We categorize the reviews into three groups: reviews related to Lager, Ale, and Other beers.

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

First, we discuss how we implemented our new MFD algorithm. We implemented our main algorithm from Corollary 4.3 combining the algorithm from Theorem 3.2 with the coreset construction in Theorem 4.2. For coreset construction, we modified a python implementation for constructing coresets [46]. For every color $c_j \in C$, we run the Gonzalez algorithm for k iterations. In the end, we have a coreset of size $m \cdot k$. Then, the coreset is given as input to

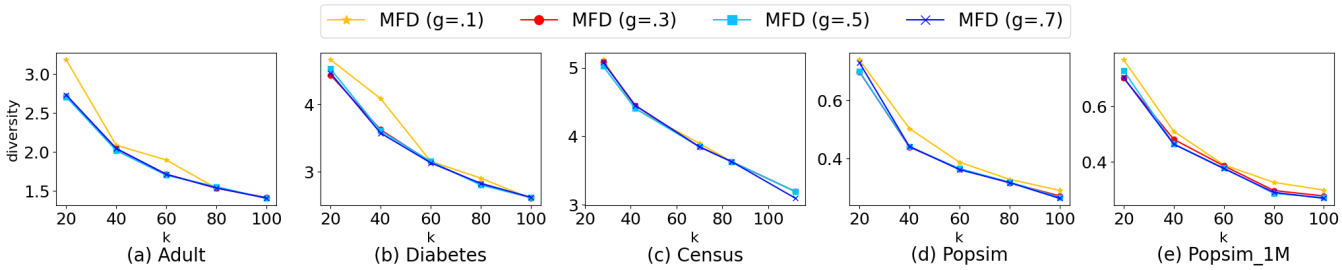


Figure 3: Comparison of diversity vs k for MFD with different early stopping parameters.

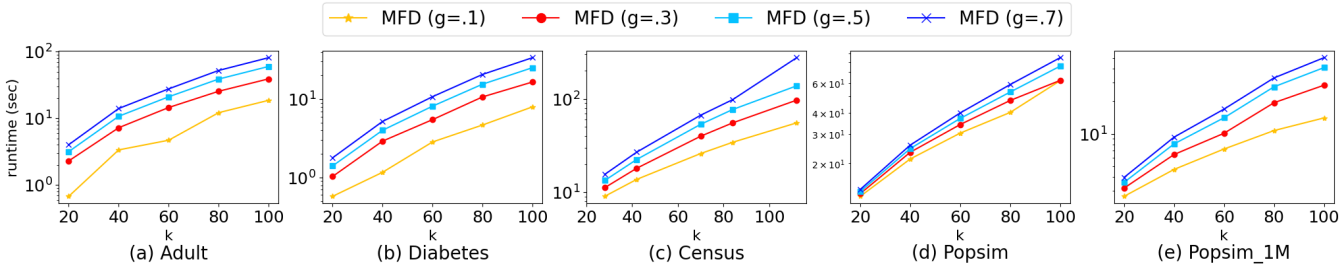


Figure 4: Comparison of running time vs k for MFD with different early stopping parameters.

	$g = 0.1$				$g = 0.3$			
k	FN	FY	MN	MY	FN	FY	MN	MY
20	0	0	0	0	0	0	0	0
40	0.2	0	1.2	0	0	0	0	0
60	0	0	0	0	0	0	0	0
80	0	0	0.4	0	0	0	0	0
100	0	0	0	0	0	0	0	0

	$g = 0.1$					$g = 0.3$				
Am Ind	As	Afr Am	Nat Haw	Wh	Am Ind	As	Afr Am	Nat Haw	Wh
0	0.2	0.2	0.4	0.4	0.4	0.2	1	0	0.2
0.6	1.4	1.2	0	0.4	0.2	0	0.2	0	0
0.4	0.4	1.2	0	0.4	0	0	0.8	0	0
1.4	0.4	1	0	0	0.6	0	1.4	0	0
0	0.4	0.4	1.4	0	0	0	0	0.8	0

Table 4: Number of points (on average) per color missed by MFD-0.1 and MFD-0.3 in Diabetes (left) and Popsim (right) datasets.

our MFD method. We note that we include the coreset construction time in the total running time of MFD.

Next, we describe some differences between the theoretical MFD algorithm and our implementation. Instead of constructing a WSPD to run a binary search on the distances, we get as an upper bound γ 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 the selected points is an upper bound on the diversity of FairDiv. If MFD does not find a feasible solution for diversity γ, we set $\gamma \leftarrow (1 - 0.15)\gamma$, and re-run the algorithm, stopping at the first feasible solution. Additionally, instead of using a BBD tree we use ParGeo's KD tree [52] with modifications to support sum queries. In practice, we observe that sometimes the MFD 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. We modified the algorithm to allow for *early stopping*. We introduce a parameter $g \in (0, 1]$ such that the MFD 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 MFD algorithm's running time. Since MFD is a randomized algorithm, in each case, we run the algorithm five times and we report the average diversity and running time.

- MFD: Our new implementation as described above.
- SFDM-2: It is a streaming algorithm designed and implemented in [49] [5]. The algorithm uses a parameter ε to control the error in the solution. We tried two representative errors $\varepsilon = 0.15$ and $\varepsilon = 0.75$. We call them SFDM-2 ($e = .15$) and SFDM-2 ($e = .75$), respectively.
- FMMD-S: The algorihtm presented and implemented in [51].
- FairFlow [40] as implemented in [51].
- FairGreedyFlow [7] as implemented in [51].

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.

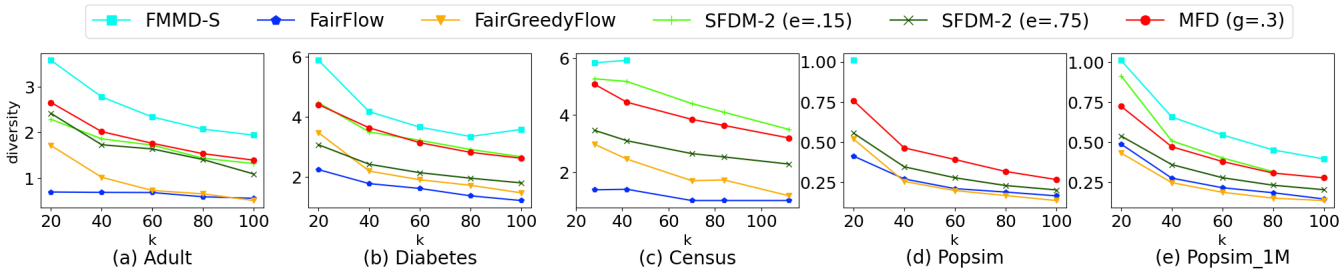


Figure 5: Comparison of diversity vs k for MFD and baselines. Higher diversity is better. Equal k_j.

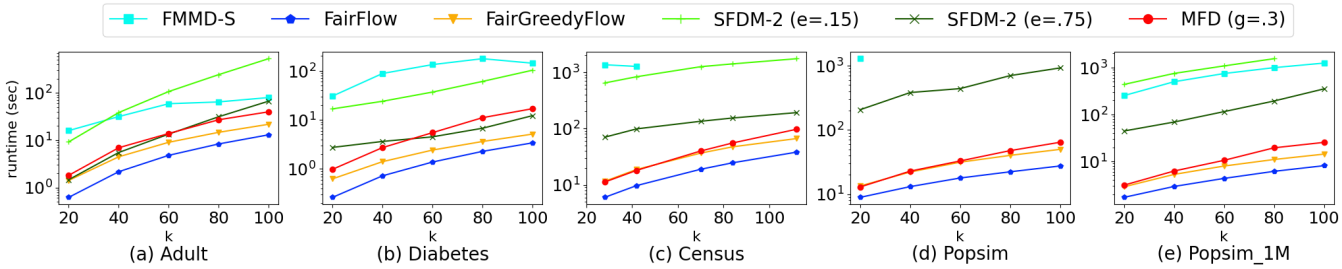


Figure 6: Comparison of running time vs k for MFD and baselines. Equal k_j.

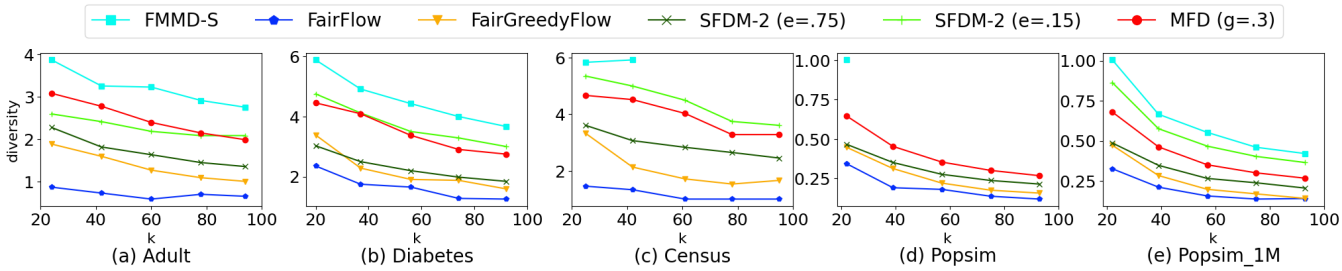


Figure 7: Comparison of diversity vs k for MFD and baselines. Higher diversity is better. Proportional k_j.

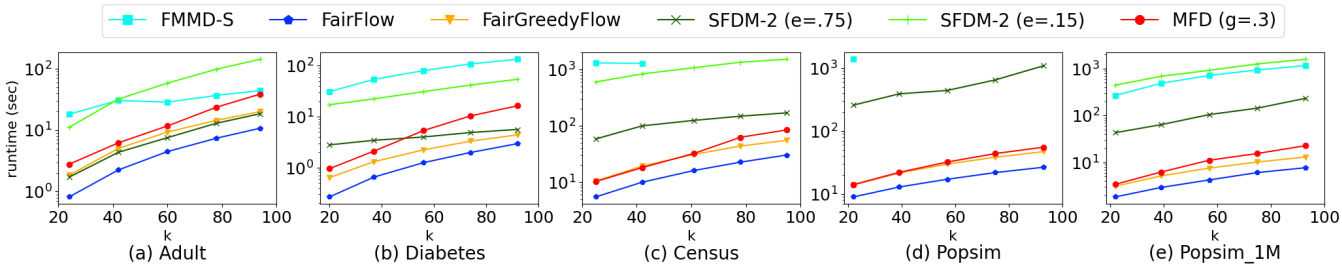


Figure 8: Comparison of running time vs k for MFD and baselines. Proportional k_j.

6.1 Micro-benchmark experiments

In this section we compare the diversity and the running time of our MFD 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 3 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 4 presents the running time over different values of k. It shows that the smaller the g is the faster the MFD is.

[5]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 MFD. As a lower bound we use the minimum non-zero pairwise distance in the coreset of size $m \cdot k$ we construct.

Before we conclude that a small value of g is always sufficient, 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 Table 4 we show two representative results using the Diabetes (left table) and Popsim (right table) dataset for MFD with $g = 0.1$ and $g = 0.3$. For each k, we assume that for every color we should take the same number of points, i.e., in Diabetes (Popsim) dataset we should take at least $k/4$ ($k/5$) points per color. We run our algorithm five times and we show the number of missing points on average for each color. The groups in the second row of Diabetes represent sex and Y or N (if meds prescribed), while the groups in the second row of Popsim represent race. When $g = 0.1$ MFD usually misses some points from each group. However for $g = 0.3$ in most cases, MFD-0.3 does not miss any point. For Diabetes, MFD-0.3 never misses a point. For Popsim, the maximum average number of points it misses for a group is 1.4. For each k it misses on average 1.16 points in total, over all groups. Generally, we observed that in all datasets, for every $g \geq 0.3$ it is rare to miss more than 2 points. We conclude that MFD for $g \geq 0.3$ successfully satisfies the fairness requirements. In the next sections, we fix g=0.3 for MFD.

> Key Takeaways: MFD 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.

6.2 End-to-end results

To compare the quality of MFD 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: **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. **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 MFD 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 MFD 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 is not scalable to million scale datasets. Furthermore, it takes at least 50× the time taken by MFD. Among all other techniques, MFD 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 MFD 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 MFD. Among the techniques that ran for all datasets and k, MFD has the best diversity. All other techniques SFDM-2 ($e = 0.75$), FairFlow and FairGreedyFlow achieve poorer diversity than MFD. 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 MFD and baselines FairFlow, FairGreedyFlow) require less than 40 seconds to identify a diverse set of 100 points for more than 1M records. In fact, MFD 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 MFD to identify a fair and diverse solution.

Overall, FMMD-S and SFDM-2 return more diverse results, however for some datasets they do not even finish, or they take even 50X more time than our algorithm. At the same time, FairFlow and FairGreedyFlow are faster, but the sets they return have a much worse diversity. Although our algorithm is slightly slower than the fastest algorithms for FairDiv – it scales to large datasets, and returns sets with high diversity in reasonable time. Overall, it achieves the best tradeoff between diversity and time. Every other algorithm is either too slow (no result within 100 sec) or the diversity is too low (5 times worse than the best one).

In order to further show that MFD provides the best balance between diversity and running time, in Figure 9, we fix $k = 100$ (similar results hold for any other k), and for each algorithm we show the running time to compute a fair and diverse set along with the diversity of the returned set. We represent each algorithm as a point in the (runtime, diversity) plane. An algorithm returns a *pareto-optimal solution* if and only if there is no other algorithm that computes a more diverse (fair) set faster. We observe that MFD and FairFlow are the only algorithms that always return a pareto-optimal solution. FairFlow always returns a pareto-optimal solution because it is the fastest algorithm, however it returns sets with arbitrarily low diversity (Figure 5 (c)). Overall, MFD achieves the best equilibrium between diversity and running time.

Proportional size. Figure 7 and 8 compares the diversity and running time of MFD and other techniques for the setting where 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 MFD but did not finish for Popsim dataset. Among all other baselines, MFD achieves the best diversity while identifying the solution in less than a minute, even for million sized datasets.

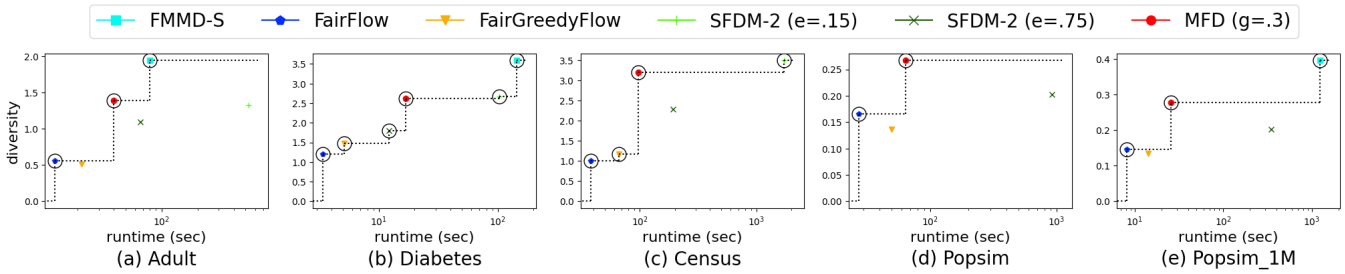


Figure 9: For every algorithm we show the running time to derive a diverse set for $k = 100$ along with the diversity of the returned set. Only MFD and FairFlow always return a pareto-optimal solution.

Key Takeaways:

(1) MFD achieves the best diversity among the techniques that run across all datasets.
(2) SFDM-2 ($\epsilon = 0.15$) and FMMD-S achieve higher diversity but they usually take 20× or even 60× more time than MFD for some datasets. All other baselines achieve worse diversity than MFD but take almost the same time to run.
(3) MFD achieves the best quality result within less than a minute for a million scale datasets.
(4) MFD provides the best balance between diversity and running time.
(5) MFD always returns a pareto-optimal solution.

Comparison of MFD and FairGreedyFlow. Given that the same coreset is given as input, FairGreedyFlow runs in $O(k^2m^4\log(k))$ time, while MFD runs in $O(k^2m\log^3(k))$ time, so in theory MFD is faster than FairGreedyFlow. However, in practice, as shown in Figures 6, 8, FairGreedyFlow runs faster than MFD. There are two reasons explaining this. i) The running time of MFD actually depends on $1/\varepsilon^{d+2}$ (please check the analysis in Section 3). The parameters ε, d are constants so we do not include this factor in the final asymptotic complexity. Our experiments show that for larger d (Diabetes dataset, $d = 8$) FairGreedyFlow is faster than MFD, whereas for smaller d (Popsim dataset, $d = 2$), the running time is almost identical. ii) FairGreedyFlow, maps the FairDiv problem to the max-flow problem. In theory, solving an instance of the max-flow is slow. However in practice, they used an optimized implementation of the Ford-Fulkerson algorithm from python's networkx library.

6.3 Streaming setting

Finally, we show experiments for the SFairDiv problem. We implement our algorithm from Theorem 5.1, called StreamMFD, and compare its efficiency and efficacy with SFDM-2, which is a streaming implementation for the SFairDiv problem[6]. At each step of the streaming phase, our algorithm stores $O(km)$ points, while the SFDM-2 stores $O(km\log\Delta)$ points. As we had in the offline case, we run two versions of the SFDM-2 algorithm, the SFDM-2 ($e = .15$) and the SFDM-2 ($e = .75$). For different values of k, in Figure 10 we show the average update time (average time to insert a new point), the post-processing time (time to construct the final solution after the end of the stream), and the diversity of the sets returned by StreamMFD, and SFDM-2. StreamMFD has the fastest update time, it has the fastest post-processing time, and it returns sets with diversity close to the diversity of the sets returned by SFDM-2 ($e = .15$). On the other hand, SFDM-2 ($e = .15$) has an expensive update time (sometimes 30× slower than StreamMFD), while SFDM-2 ($e = .15$) returns sets with very low diversity and it has 2.5× slower update time than StreamMFD. Overall, StreamMFD is the best algorithm in the streaming setting because it provides the best balance between update time, post-processing time, and diversity.

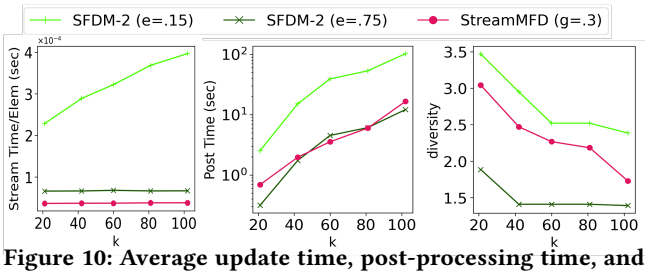


Figure 10: Average update time, post-processing time, and diversity in the streaming setting for Beer reviews.

[6]Recall that in the offline setting we used the coreset to define a range of diversities for SFDM-2. In the streaming setting, no coreset can be computed before someone visits the entire input set. As described in [49], that first introduced SFDM-2, we use the minimum (nonzero) and maximum distance of the entire point set to define the range of diversities (they assume the two quantities are known before the beginning of the stream).

7 RELATED WORK

In [40] 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 $O(kn + m^2k^2\log k)$ 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. In [7] they 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 $O(n^\lambda)$ it uses $\Omega(n^2)$ space, where λ is the exponent to solve an LP. 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 $k_j/1-\varepsilon$ 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 $\frac{1}{(m+1)(1+\varepsilon)}$-approximation in $O(nkm^3)$ time, for constant ε, skipping $\log n$ factors.

In the Euclidean metric, in [7] they constructed a $(1+\varepsilon)$-coreset of size $O(\varepsilon^{-d}mk)$ for the FairDiv problem in $O(nk)$ time. Using the coreset, 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, [51] used a coreset construction to propose the FFMD-S algorithm that returns a $\frac{1-\varepsilon}{5}$-approximation in $O(mkn + m^k)$ time for constant ε.

In the streaming setting, the authors in [49, 50] 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 $\frac{1-\varepsilon}{3m+2}$-approximation for the FairDiv problem, for constant ε. The quantity Δ is defined as the *spread* of the input set, $\Delta = \frac{\max_{p,q \in P} ||p-q||}{\min_{p,q \in P} ||p-q||} = O(2^n)$. In [7], they give a streaming algorithm with constant approximation factor, however the space, update time, and post-processing time still depends on $\log \Delta$.

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 [5, 15, 16, 18–20]. 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, [7, 9, 40].

The Max-Min diversification has a strong connection with the k-center clustering. For example, the same greedy Gonzalez [32] algorithm returns a $\frac{1}{2}$-approximation for the Max-Min diversification [45, 47] and 2-approximation for the k-center clustering [32]. 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 $O(km^2n + km^4)$ time. There are many improvements over this algorithm, such as [24, 25, 36]. 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, as shown in [7]. For example, assume there are two blue points with coordinates 0 and $5 - \varepsilon/2$ and two red points with coordinates $5 + \varepsilon/2$ and 10, for an arbitrary small value $\varepsilon > 0$. Selecting the blue point with coordinate $5 - \varepsilon/2$ and the red point with coordinate $5 + \varepsilon/2$ constructs an optimum solution for the fair 2-center problem. However the diversity is equal to ε, while the optimum diversity for the FairDiv problem is $10 \gg \varepsilon$. Hence, it is unclear if algorithms for the fair k-center problem can be used for the FairDiv problem.

8 FUTURE WORK

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 $\frac{k_j}{1-\varepsilon}$ 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 [51] 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, to get constant approximation in metrics with bounded doubling dimension.

REFERENCES

[1] https://anonymous.4open.science/r/FairDiversityandClustering/.
[2] Beer review https://snap.stanford.edu/data/web-BeerAdvocate.html.
[3] Courts seek to increase jury diversity https://eji.org/report/race-and-the-jury/why-representative-juries-are-necessary/#chapter-2.
[4] Courts seek to increase jury diversity https://www.uscourts.gov/news/2019/05/09/courts-seek-increase-jury-diversity.
[5] 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.
[6] M. Abrahamsen, M. de Berg, K. Buchin, M. Mehr, and A. D. Mehrabi. Range-clustering queries. In *33rd International Symposium on Computational Geometry (SoCG 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[7] 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 für Informatik, 2022.
[8] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33:201–226, 2002.
[9] 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.
[10] 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.
[11] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry*, 17(3-4):135–152, 2000.
[12] 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.
[13] B. Becker and R. Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] M. Ceccarello, A. Pietracaprina, and G. Pucci. Fast coreset-based diversity maximization under matroid constraints. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 81–89, 2018.
[19] M. Ceccarello, A. Pietracaprina, and G. Pucci. A general coreset-based approach to diversity maximization under matroid constraints. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(5):1–27, 2020.
[20] 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.
[21] T. M. Chan and Q. He. Faster approximation algorithms for geometric set cover. In *36th International Symposium on Computational Geometry (SoCG 2020)*, 2020.
[22] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635, 1997.
[23] 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.
[24] D. Z. Chen, J. Li, H. Liang, and H. Wang. Matroid and knapsack center problems. *Algorithmica*, 75:27–52, 2016.
[25] 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.
[26] 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.
[27] 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.
[28] A. E. Feldmann and D. Marx. The parameterized hardness of the k-center problem in transportation networks. *Algorithmica*, 82:1989–2005, 2020.

[29] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4:1–9, 1974.

[30] 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.

[31] 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.

[32] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[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. Kleindessner, P. Awasthi, and J. Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning*, pages 3448–3457. PMLR, 2019.

[39] M. Meek, T. Thiesson, and H. Heckerman. US Census Data (1990). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5VP42.

[40] Z. Moumoulidou, A. McGregor, and A. Meliou. Diverse data selection under fairness constraints. *ICDT 2021*, 2021.

[41] 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.

[42] 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.

[43] 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.

[44] P. Pope, C. Zhu, A. Abdelkader, M. Goldblum, and T. Goldstein. The intrinsic dimension of images and its impact on learning. *arXiv preprint arXiv:2104.08894*, 2021.

[45] 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.

[46] A. Roberts, K. Vombatkere, and B. Suwal. GitHub - kvombatkere/CoreSets-algorithms: Python implementation of coreset algorithms for clustering and streaming.

[47] A. Tamir. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4(4):550–567, 1991.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.