Efficient Indexes for Diverse Top-k Range Queries*

Pankaj K. Agarwal
Duke University

Stavros Sintos
Duke University

Alex Steiger
Duke University

ABSTRACT

Let P be a set of n (non-negatively) weighted points in $\mathbb{R}^d$. We consider the problem of computing a subset of (at most) k diverse and high-valued points of P that lie inside a query range, a problem relevant to many areas such as search engines, recommendation systems, and online stores. The *diversity* and *value* of a set of points are measured as functions (say average or minimum) of their pairwise distances and weights, respectively. We study both *bicriteria* and *constrained* optimization problems. In the former, we wish to return a set of k points that maximize a weighted sum of their value and diversity measures, and in the latter, we wish to return a set of at most k points that maximize their value and satisfy a diversity constraint.

We obtain three main types of results in this paper:

(1) Near-linear time $(0.5-\varepsilon)$-approximation algorithms for the bicriteria optimization problem in the offline setting.
(2) Near-linear size indexes for the bicriteria optimization problem that for a query rectangle return a $(0.5-\varepsilon)$-approximate solution in time $O(k\,\text{polylog}(n))$. The indexes can be constructed in $O(n\,\text{polylog}(n))$ time.
(3) Near-linear size indexes for answering constrained optimization range queries. For a query rectangle, a $0.5^{O(d)}$-approximate solution can be computed in $O(k\,\text{polylog}(n))$ time. If we allow some of the returned points to lie at most ε outside of the query rectangle then an $(1-\varepsilon)$-approximate solution can be computed in $O(k\,\text{polylog}(n))$ time. The indexes are constructed in $O(n\,\text{polylog}(n))$ and $n^{O(1/\varepsilon^d)}$ time, respectively.

ACM Reference Format:
Pankaj K. Agarwal, Stavros Sintos, and Alex Steiger. 2020. Efficient Indexes for Diverse Top-k Range Queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3375395.3387667

1 INTRODUCTION

Building an index on a given set of objects so that range queries can be answered efficiently is a fundamental problem in many fields including databases, computational geometry, and GIS. Notwithstanding extensive research over the last four decades, it remains an active research topic. The early work on range searching focused on either returning all points lying in a query range or computing a simple aggregation function (such as sum, max, min) over the values/weights of points lying in a query range [3, 9, 32]. In modern applications, especially as the data sets have become very large, neither of these formulations is satisfactory — there may be too many points lying in a query range, and the simple aggregation function provides very little information about the points in a query range. As a result, there is much work in recent years on returning a summary of points lying in a query range [2, 35, 44].

Of particular interest, motivated by several applications such as online marketing and web keyword search, is to return a set of top/most-relevant/most-interesting k points in a query range [2, 37, 38, 40]. The utility of an object is defined by a weight function on objects. For example, suppose a vendor has a collection of laptops. For simplicity, assume each laptop has two non-negative attributes — RAM size and price. The goal is to build an index so that for a query range ρ, and a parameter k, the top k laptops whose attributes lie in ρ can be returned quickly.

A shortcoming of top-k queries is that the index may return very similar objects in one query. Returning to the laptop example, suppose there are three laptops with specifications (i) 1 GB RAM and \$1000 price, (ii) 1GB RAM and \$1200 price, and (iii) 2GB RAM and \$1800 price, and suppose their weights are 2, 1.9, and 1.8. For $k = 2$, the top-k query will return the first two laptops, while it is desirable to return laptops (i) and (iii) to provide more choices to the user. Thus the goal is to return (at most) k items (i) that have *high values*, and (ii) that are *diverse*. The diversity between a pair of objects can be captured by defining a metric over the set of objects. We can then define the diversity of k items either as the minimum pairwise distance or as the sum of pairwise distances. We now define the problem formally.

Problem statement. We represent an object as a point in $\mathbb{R}^d$ for some constant $d \geq 1$. Let $\mathsf{P} = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$. Each point $p \in \mathsf{P}$ is assigned a weight $w(p) \geq 0$. The goal is to build an index on P that for a query rectangle ρ and a parameter $k \geq 1$, returns (at most) k points of $\mathsf{P} \cap \rho$ maximizing a utility function that captures the total value of these points as well as their diversity. We define two types of query optimization problems: bicriteria optimization and constrained optimization.

Bicriteria optimization: We are given a utility function $f : 2^{\mathsf{P}} \to \mathbb{R}_{\geq 0}$ that captures both the value and the diversity of a subset $\mathsf{S} \subseteq \mathsf{P}$. Our goal is to construct an index that for a query rectangle ρ and a positive integer $k > 0$, returns $\text{argmax}\, f(\mathsf{S})$, where the maximum is taken over all subsets $\mathsf{S} \subseteq \mathsf{P} \cap \rho$ of k points. We study two specific utility functions:

*Work on this paper was supported by NSF under grants CCF-15-13816, CCF-15-46392, IIS-18-14493, and an ARO grant W911NF-15-1-0408.

PODS'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-7108-7/20/06…$15.00
https://doi.org/10.1145/3375395.3387667

Max-sum diversification (MSD) range query: For a given subset $S \subseteq P$ and a parameter $\lambda \geq 0$, we define

$$f(S) = \sum_{p_i, p_j \in S} \|p_i - p_j\| + \lambda \sum_{p_i \in S} w(p_i).$$

For a subset $X \subseteq P$, we define

$$MS(X, k) = \operatorname*{argmax}_{\substack{S \subseteq X \\ |S|=k}} f(S)$$

The problem of computing $MS(P, k)$ in the offline setting is the *max-sum diversification* (MSD) problem, and the special case in which $\lambda = 0$, i.e., we ignore the weights of points, is referred to as the *remote-clique*[1] problem. Our goal is to build an index so that given ρ, k, and λ, $MS(P \cap \rho, k)$ can be reported quickly.

Max-min diversification (MMD) range query: For a given subset $S \subseteq P$ and a parameter $\lambda \geq 0$, we define

$$f(S) = \min_{p_i, p_j \in S} \|p_i - p_j\| + \lambda \min_{p_i \in S} w(p_i).$$

For a subset $X \subseteq P$, we define

$$MM(X, k) = \operatorname*{argmax}_{\substack{S \subseteq X \\ |S|=k}} f(S).$$

The problem of computing $MM(P, k)$ in the offline setting is the *max-min diversification* (MMD) problem, and the special case in which $\lambda = 0$, i.e., we ignore the weights of points, is referred to as the *remote-edge*[1] problem. Our goal is to build an index so that given ρ, k, and λ, $MM(P \cap \rho, k)$ can be reported quickly.

Constrained optimization: In this version, we use diversity as a constraint and return *at most* k points that satisfy the diversity constraint. For a parameter $\delta > 0$, we call a subset $S \subseteq P$ *δ-diverse k-set* if $|S| \leq k$ and the distance between any pair of points of S is at least δ. Here the goal is to build an index on P that for a query rectangle ρ, an integer $k > 0$, and a parameter $\delta \geq 0$, quickly returns a δ-diverse k-set $S \subseteq P \cap \rho$ whose total weight is maximized. That is, return

$$CD(P \cap \rho, k, \delta) = \operatorname{argmax} w(S),$$

where the maximum is taken over all subsets of $P \cap \rho$ that are δ-diverse k-sets. We call such a query a *constrained-diversity top-k (CDT) range query*. The special case where $\delta = 0$ is simply called a *top-k range query* in other literature.

Unlike $MS(X, k)$ and $MM(X, k)$, $CD(X, k, \delta)$ is defined so that its size can be less than k. If we required it to have size exactly k, then reporting a *feasible* solution for a CDT range query would be at least as hard as checking the existence of an independent set of size k in unit disk graphs[2] which is NP-hard [20].

We call an algorithm for any of these three problems an α-approximation algorithm, for some $\alpha \leq 1$, if it returns a subset whose utility is at least α times that of an optimal solution.

Related work. There is extensive work on the MSD and MMD problems as well as their special cases of the remote-clique and remote-edge problems in the offline setting.

Tamir [42] showed that the greedy algorithm for k-center by Gonzalez [23], which takes $O(nk)$ time, is a 0.5-approximation algorithm for the remote-edge problem. A similar greedy algorithm also yields an $O(nk)$-time 0.5-approximation for the remote-clique problem [39] (also see [11, 12]). All of these algorithms work with arbitrary metrics. For certain special distances that include the Euclidean distance there are better approximation algorithms for the remote-clique problem: For distances of negative type (for a formal definition of negative type distances see Subsection 2.1) Cevallos *et al.* [16] propose an $O(nk^2 \log k)$-time $(1 - \frac{4}{k})$-approximation algorithm. The approximation factor is improved to $1 - \frac{4}{k+2}$ in [17]. For distances that are both metrics and of negative type Cevallos *et al.* [17] give a $O(nk^2 \log k)$-time $(1 - \frac{2}{k})$-approximation algorithm. For metrics with doubling dimension q a PTAS with running time $O(n(k+\varepsilon^{-q}) + k\varepsilon^{-1} \log^{O(\varepsilon^{-q})} k)$ is known [17], and the running time can be improved to $O(\varepsilon^{-d} n \log k + \varepsilon^{-d} k^{O(\varepsilon^{-d})})$ for the Euclidean metric [15].

All the algorithms for the remote-clique and remote-edge mentioned above can be adapted to obtain approximation algorithms for the MSD and MMD problem, respectively. For instance, we can either return the solution to the remote-clique/remote-edge problem or return the top-k items, whichever has the higher utility. Hence, any α-approximation for the remote-clique (remote-edge) problem gives an $\alpha/2$-approximation for the MSD (MMD) problem.

The problem of maximizing a linear combination of the remote-clique objective, e.g., the sum of pairwise distances and a submodular function defined on the points, is studied in [12, 16]. Notice that this problem is a generalization of the MSD problem since the sum of weights is a (sub)modular function. For general metrics, [12] give an 0.5 approximation algorithm in $O(nk)$ time; for metrics of negative type, [16] propose a slower algorithm with approximation factor $1 - \frac{4}{k}$. Gollapudi and Sharma [22] described an approximation-preserving reduction from the MSD problem to the remote-clique problem, which also leads to an $O(nk)$-time 0.5-approximation algorithm for MSD using the known results of the remote-clique problem on general metrics. They also give a reduction from MMD to the remote-edge problem, which is unfortunately not correct [41]. An $O(n^2 k)$-time 0.5-approximation for the MMD problem can be obtained by solving $n - 1$ instances of the remote-edge problem (see Subsection 2.2).

Recently the notion of (composable) coresets[3] has been used to get efficient algorithms for the remote-clique and remote-edge problem in the streaming (MapReduce) setting [6, 14, 28]. Using these coresets, one can construct indexes for both remote-clique and remote-edge range queries in $O(k^2 \operatorname{polylog}(n))$ time. A drawback of such an index is that the parameter k is fixed at preprocessing time.

To the best of our knowledge, the only work directly on describing indexes for answering MSD or MMD under range queries is that of Wang *et al.* [43]. They define the RC-index that works with arbitrary metrics and supports range queries in $O(k^2 \gamma^{4(\delta+1)} \log^d n)$ time with an approximation ratio of $(0.5 - 0.5^{\delta-1})$, where δ is an

[1] The remote-clique and remote-edge problems are also called the max-sum and max-min dispersion problems in other literature, respectively.

[2] A *unit disk* graph $G(P)$ defined for a point set P is where the vertices are the points and a pair of points u and v are incident in $G(P)$ if and only if $\|u - v\| \leq 1$.

[3] An α-coreset for a maximization problem is a small subset of the input point set $S \subseteq P$ such that the optimum solution on S approximates the optimal solution on P within a factor of $\alpha < 1$. A collection of α-coresets, where each is for a potentially different point set, are *composable* if their union is a valid α-coreset for the union of the point sets.

Problem	Approx.	Time
MSD	$0.5-\varepsilon$	$O(n+k\log k)$
	$1-\varepsilon$	$O(nk^2)$
MMD	$0.5-\varepsilon$	$O(n+k\log k)$

Table 1. New offline algorithms

Problem	Approx.	Query Time	Space
MSD	$0.5-\varepsilon$	$O(k\log^{d-1} n)$	$O(n\log^d n)$
	$1-\varepsilon$	$O(k^3+k\log^{d-1} n)$	$O(n\log^d n)$
MMD	$0.5-\varepsilon$	$O(k\log^{d-1} n)$	$O(n\log^d n)$
CDT	$(0.5)^{O(d)}$	$O(k\log^d n)$	$O(n\log^{d-1} n)$
(k, ρ part of query, at most $(1+\varepsilon)k$ points)	$1-\varepsilon$	$O(k\log k\log^d n)$	$O(n\log^d n)$

Table 2. New indexes

integral query parameter, and γ is the *expansion constant*. While in some cases γ is small, it can be $O(n)$ even in Euclidean space.

Main results. We present three main results in this paper. We assume $\varepsilon \in (0, 0.5)$ to be a constant. The big-O notation in the running time hide polynomial factors of $1/\varepsilon$.

Offline algorithms (Section 2): We give a $(0.5-\varepsilon)$-approximation algorithm for the MSD problem that runs in $O(n+k\log k)$ time. This is the first algorithm for the MSD problem with approximation factor $0.5-\varepsilon$ with runtime near-linear on both n and k. Notice that this is faster than all previously known algorithms for the remote-clique and MSD problems. In [26] the authors construct a 0.5-approximate solution of the MSD problem in $O(n^2+k^2\log k)$ time by repeatedly taking farthest pairs of remaining points. We derive our result by defining and using a metric function that allows us to compute efficiently approximate farthest pairs of points in weighted Euclidean space. We show that our algorithm can be implemented using an implicit representation of P, which is useful to answer MSD range queries. Finally, we show that the *weighted distance* between two points is of negative type, which allows to use the algorithm in [16] to get an $(1-\varepsilon)$-approximation algorithm that runs in $O(nk^2)$ time, if $\varepsilon \geq 4/k$.

We also present a $(0.5-\varepsilon)$-approximation algorithm for the MMD problem that runs in $O(n\log n)$ time. This is the first algorithm for the MMD problem with a runtime near-linear in n with this approximation factor. It is also faster than all the previously known algorithms for the remote-edge and MMD problems. Our algorithm closely follows the algorithms of Guha [24] and McCutchen and Khuller [34] for the k-center clustering problem in the streaming setting, but new ideas are needed to adapt them to the remote-edge problem and to achieve the $O(n\log n)$ running time. Additionally, we show that using an implicit representation of P, we can improve the running time to $O(n\log k)$. The running time can be improved even further to $O(n+k\log k)$ in expectation if we allow randomization. The implicit representation is also useful to answer MMD range queries. Table 1 summarizes our results in the offline setting.

Range queries (Section 3): We use our offline algorithms and build indexes of $O(n\log^d n)$ size that for a query rectangle ρ and parameters k, λ, ε, return a $(0.5-\varepsilon)$-approximate solution in time $O(k\log^{d-1} n)$. We can improve the query time if ε is fixed in the preprocessing phase. Table 2 summarizes the performance of our indexes.

Constrained optimization (Section 4): The intuition for finding efficient indexes for the CDT problem comes from the known offline algorithms for the weighted independent set problem on unit disk graphs: (1) A greedy $0.5^{O(d)}$-approximation that runs in polynomial time [29, 31]. (2) A PTAS using the shifting grid technique with running time $n^{O(\varepsilon^{-d})}$ [27, 29, 33]. We use the idea in (1) to build an index of size $O(n\log^{d-1} n)$ that returns a $0.5^{O(d)}$-approximate solution in $O(k\log^d n)$ time. All $k, \varepsilon, \delta, \rho$ are part of the query. In order to get this result, we represent the $\mathbb{R}^d$ space in a way that allows us to find the returned set of k points in a greedy fashion inside the query rectangle ρ. Then, we use the shifting grid technique to construct an index of size $O(nk\log^d n)$ such that given a query rectangle ρ it finds a $(1-\varepsilon)$-approximate solution for the CDT problem among the points in ρ in $O(k^2\log^d n)$ time. We note that in this index the parameters k, ε, δ need to be fixed in the preprocessing phase, and the returned set of a CDT range query might contain points that lie at most ε outside of the query rectangle ρ. If we allow returning at most $(1+\varepsilon)k$ points we can modify the second index such that it has $O(n\log^d n)$ space, $O(k\log k\log^d n)$ query time, and k can also be part of the query.

2 OFFLINE ALGORITHMS

In this section we describe efficient offline algorithms for the MSD and MMD problems we defined above. Later, we use these algorithms to answer range queries efficiently.

2.1 MSD Problem

P is given explicitly. Let $\mathsf{P} \subseteq \mathbb{R}^d$ be the set of points defined above, and let $\varepsilon \in [0, 0.5)$ be an arbitrarily small constant. We describe a $(0.5-\varepsilon)$-approximation algorithm for the MSD problem.

We define a *distance* function $f' : \mathsf{P}\times\mathsf{P} \to \mathbb{R}_{\geq 0}$ as follows. For a pair $p, q \in \mathsf{P}$, $f'(p,q) = 0$ if $p = q$, and $f'(p,q) = \|p-q\| + \frac{\lambda}{k-1}(w(p)+w(q))$ if $p \neq q$. We note that f' is a metric since $w(p) \geq 0$ for all $p \in \mathsf{P}$. For any subset $\mathsf{Q} \subseteq \mathsf{P}$ of k points $f(\mathsf{Q}) = \sum_{p,q\in\mathsf{Q}} f'(p,q)$, so the MSD problem can be formulated as computing $\mathsf{S}^* = \operatorname{argmax}_{\mathsf{S}\subseteq\mathsf{P},|\mathsf{S}|=k} \sum_{p,q\in\mathsf{S}} f'(p,q)$. We use this formulation of the MSD problem.

Algorithm. Let $\mathsf{N} \subseteq \mathbb{S}^{d-1}$ be a centrally symmetric set of $r = O(\frac{1}{\varepsilon^{(d-1)/2}})$ unit vectors in $\mathbb{R}^d$ (i.e., if $u \in \mathsf{N}$ then $-u \in \mathsf{N}$) that is an ε-net, i.e., for any point $v \in \mathbb{S}^{d-1}$ there is a point $u \in \mathsf{N}$ with angle at most $\arccos(\frac{1}{1+\varepsilon}) = O(\sqrt{\varepsilon})$. For a point $p \in \mathsf{P}$ and a vector $u \in \mathsf{N}$, we define the *score* $s(p,u) = \langle p, u\rangle + \frac{\lambda}{k-1}w(p)$. For each $u \in \mathsf{N}$, let P_u denote the set of k points with the highest scores with respect to vector u. We compute the desired set S of k points by repeating the following steps $k/2$ times: For each pair of vectors $u, -u \in \mathsf{N}$, we choose a pair of points ξ_u, ξ_{-u} as described below, compute the vector $\bar{u} = \operatorname{argmax}_u s(\xi_u, u) + s(\xi_{-u}, -u)$, add the pair $\xi_{\bar{u}}, \xi_{-\bar{u}}$ to S, and delete $\xi_{\bar{u}}, \xi_{-\bar{u}}$ from all lists P_u in which they appear[4].

In each step, for a pair $u, -u \in \mathsf{N}$ we compute ξ_u, ξ_{-u} as follows. Let a_u (resp. a_{-u}) be the point of maximum score in P_u (resp. P_{-u}).

[4]If k is odd then in the end of the algorithm we add in the point with the highest weight among the remaining points.

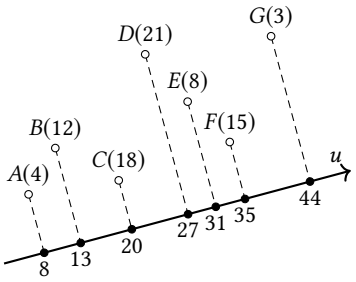

Figure 1. A set of 7 points in the plane (white) and their projections (black) onto a vector u. Each point is labeled with a name and weight in parentheses. When $\lambda = 0$, $a_u = G$ and $a_{-u} = A$ with $s(G, u) = 44$ and $s(A, -u) = -8$. When $\lambda = k - 1$, $a_u = F$ and $a_{-u} = B$ with $s(F, u) = 35 + 15 = 50$ and $s(F, -u) = -13 + 12 = -1$. As $\lambda \to \infty$, $a_u = a_{-u} = D$ since D is the point with the highest weight.

If $a_u \neq a_{-u}$[5], we set $\xi_u = a_u$ and $\xi_{-u} = a_{-u}$ (see Figure 1 for an example). If $a_u = a_{-u}$ then let b_u (resp. b_{-u}) be the point with the second largest score in P_u (resp. P_{-u}). If $s(a_u, u) + s(b_{-u}, -u) \geq s(a_{-u}, -u) + s(b_u, u)$ then we set $\xi_u = a_u$ and $\xi_{-u} = b_{-u}$ otherwise we set $\xi_u = b_u$ and $\xi_{-u} = a_{-u}$.

By maintaining each P_u in a priority queue, a_u and b_u can be reported in $O(\log k)$ time and a point from P_u can be deleted in $O(\log k)$ time. Hence, each round of the algorithm takes $O(|\mathsf{N}| \log k)$ time. Since P_u can be computed in $O(n)$ time and $|\mathsf{N}| = O(1)$, the total time taken by the algorithm is $O(n + k \log k)$.

Analysis. We now prove that $f(\mathsf{S}) \geq (0.5-\varepsilon)f(\mathsf{S}^*)$. Let $\mathrm{d}(p, q) = \max_{u \in \mathsf{N}} \langle p - q, u \rangle$ be a distance function between $p, q \in \mathsf{P}$. Since N is centrally symmetric, $\mathrm{d}(\cdot, \cdot)$ is symmetric. Furthermore, it is easy to see that $\mathrm{d}(\cdot, \cdot)$ satisfies the triangle inequality, so it is a metric. The following observation is straightforward from the definition and [4, 18]: (A) For any two points $x, y \in \mathbb{R}^d$, $\|x - y\| \geq \mathrm{d}(x, y) \geq (1 - \varepsilon)\|x - y\|$.

Next, we define the distance function $f'_{\mathrm{d}} : \mathsf{P} \times \mathsf{P} \to \mathbb{R}_{\geq 0}$ as $f'_{\mathrm{d}}(p, q) = 0$ if $p = q$, and $f'_{\mathrm{d}}(p, q) = \mathrm{d}(p, q) + \frac{\lambda}{k-1}(w(p) + w(q))$. Since d is a metric, f'_{d} is also a metric. From (A), the next observation easily follows: (B) For any $p, q \in \mathsf{P}$, $f'(p, q) \geq f'_{\mathrm{d}}(p, q) \geq (1 - \varepsilon)f'(p, q)$.

We claim the following property.

Lemma 2.1. *In each round, the algorithm computes the farthest pair of points of the current set* P *under the metric* f'_{d}.

Proof. Let S_{l-1} be the set of points that the algorithm added in S at the end of round $l-1$. Let p', q' be the two points added by the algorithm at round l. We show that $f'_{\mathrm{d}}(p', q') = \max_{p,q \in \mathsf{P} \setminus \mathsf{S}_{l-1}} f'_{\mathrm{d}}(p, q)$.

We first show that for any $u \in \mathsf{N}$, ξ_u, ξ_{-u} are the points with the farthest distance with respect to the vector u. Consider the case where $a_u \neq a_{-u}$ and $\xi_u = a_u$, $\xi_{-u} = a_{-u}$. From the definition, for any pair $p, q \in \mathsf{P} \setminus \mathsf{S}_{l-1}$ we have that $s(a_u, u) \geq s(p, u)$ and $s(a_{-u}, -u) \geq s(q, -u)$ so $\langle \xi_u - \xi_{-u}, u \rangle + \frac{\lambda}{k-1}(w(\xi_u) + w(\xi_{-u})) \geq \langle p - q, u \rangle + \frac{\lambda}{k-1}(w(p) + w(q))$. Then we consider the case where $a_u = a_{-u}$ and $\xi_u = a_u$, $\xi_{-u} = b_{-u}$ (the case where $\xi_u = b_u$, $\xi_{-u} = a_{-u}$ is symmetric). Notice that $s(a_u, u) + s(b_{-u}, -u) \geq s(b_u, u) + s(a_{-u}, -u)$. For any pair $p, q \in \mathsf{P} \setminus \mathsf{S}_{l-1}$ we have two cases. If $q \neq a_u = a_{-u}$ then by definition $s(b_{-u}, -u) \geq s(q, -u)$ and $s(a_u, u) \geq s(p, u)$, so $s(a_u, u) + s(b_{-u}, -u) \geq s(p, u) + s(q, -u)$. If $p \neq a_u = a_{-u}$ then by definition $s(b_u, u) \geq s(p, u)$ and $s(a_{-u}, -u) \geq s(q, -u)$, so $s(a_u, u) + s(b_{-u}, -u) \geq s(b_u, u) + s(a_{-u}, -u) \geq s(p, u) + s(q, -u)$. In any case we have that $s(\xi_u, u) + s(\xi_{-u}, -u) \geq s(p, u) + s(q, -u)$ for any $p, q \in \mathsf{P} \setminus \mathsf{S}_{l-1}$.

Assume that the algorithm finds the farthest pair p', q' in the end of round l and consider any pair of points $p, q \in \mathsf{P} \setminus \mathsf{S}_{l-1}$. Let $\bar{u} \in \mathsf{N}$ be a vector such that $\bar{u} = \mathrm{argmax}_{u \in \mathsf{N}} \langle p - q, \bar{u} \rangle$, i.e., $\mathrm{d}(p, q) = \langle p - q, \bar{u} \rangle$. The algorithm considers the vector $\bar{u}$ and as we showed in the previous paragraph it finds a pair $\xi_{\bar{u}}, \xi_{-\bar{u}}$ such that $s(\xi_{\bar{u}}, \bar{u}) + s(\xi_{-\bar{u}}, -\bar{u}) \geq s(p, \bar{u}) + s(q, -\bar{u})$. We conclude that $f'_{\mathrm{d}}(p', q') \geq s(\xi_{\bar{u}}, \bar{u}) + s(\xi_{-\bar{u}}, -\bar{u}) \geq s(p, \bar{u}) + s(q, -\bar{u}) = f'_{\mathrm{d}}(p, q)$. The lemma follows. □

By Lemma 2.1, the algorithm at each step chooses the farthest pair of P under the metric f'_{d} and deletes them from P, and repeats this step $k/2$ times. The argument in [26] implies that $\sum_{p,q \in \mathsf{S}} f'_{\mathrm{d}}(p, q) \geq 0.5 \max_{\mathsf{R} \subset \mathsf{P}, |\mathsf{R}| = k} \sum_{p,q \in \mathsf{R}} f'_{\mathrm{d}}(p, q)$. By combining this inequality with observation (B), we obtain that

$$\sum_{p,q \in \mathsf{S}} f'(p, q) \geq (0.5 - \varepsilon) \sum_{p,q \in \mathsf{S}^*} f'(p, q).$$

Hence, putting everything together we obtain the following result.

Theorem 2.2. *Given a set* P *of* n *non-negatively weighted points in* $\mathbb{R}^d$*, an integer* $k \leq n$*, a constant* $\varepsilon \in (0, 0.5)$*, and a parameter* $\lambda \geq 0$*, a subset* $\mathsf{S} \subseteq \mathsf{P}$ *of* k *points can be computed in* $O(n + k \log k)$ *time such that* $f(\mathsf{S}) \geq (0.5 - \varepsilon) f(MS(\mathsf{P}, k))$.

Cevallos *et al.* [17] presented a $(1 - \frac{2}{k})$-approximation algorithm with $O(nk^2 \log k)$ running time for the remote-clique problem assuming that the distance function is metric and of *negative type*. Equivalently, for constant $\varepsilon \in [\frac{2}{k}, 1)$ their algorithm gives a $(1 - \varepsilon)$-approximation in $O(nk^2)$ time. Negative type distances are defined as follows. Let $D \in \mathbb{R}^{n \times n}$ be the distance metric corresponding to a distance function *dist*, i.e., $D_{a,b} = dist(a, b)$ for two items a, b. Then we say that *dist* is of negative type if $x^T D x \leq 0$, $\forall x \in \mathbb{R}^n$ with $\sum_{i=1}^n x_i = 0$. Some examples of negative type distances are l_p norms, the cosine distance, the Jaccard distance [36]. We get the next theorem by showing that the metric function f' is of negative type.

Theorem 2.3. *Given a set* P *of* n *points in* $\mathbb{R}^d$*, an integer* $k \leq n$*, a constant* $\varepsilon \in [\frac{2}{k}, 1)$*, and a parameter* $\lambda \geq 0$*, a subset* $\mathsf{S} \subseteq \mathsf{P}$ *of* k *points can be computed in* $O(nk^2)$ *time such that* $f(\mathsf{S}) \geq (1 - \varepsilon) f(MS(\mathsf{P}, k))$.

Proof. Here we show that the distance function f' is of negative type, so we can use the algorithm from [16] to get the same approximation for the MSD problem.

Let $D \in \mathbb{R}^{n \times n}$ be the distance matrix, such that $D_{i,j} = f'(p_i, p_j)$ if $i \neq j$ and $D_{ii} = 0$. Furthermore, let $D^1 \in \mathbb{R}^{n \times n}$ be the distance matrix, such that $D^1_{i,j} = \frac{\lambda}{k-1}(w(p_i) + w(p_j))$ if $i \neq j$ and $D^1_{ii} = 0$, and let $D^2 \in \mathbb{R}^{n \times n}$ be the distance matrix, such that $D^2_{i,j} = ||p_i - p_j||$ if $i \neq j$ and $D^2_{ii} = 0$. Notice that $D = D^1 + D^2$. Let $x \in \mathbb{R}^n$ be a vector with $\sum_{i=0}^n x_i = 0$. We need to show that $x^T D x \leq 0$. It is known that $x^T D^2 x \leq 0$ [36], so if we also show that $x^T D^1 x \leq 0$ the lemma

[5] Even if P has more than one point, a_u may be the same as a_{-u} if the weight of a_u is very high.

follows.

$$x^T D^1 x = \sum_{i,j} x_i D^1_{ij} x_j = \sum_{i=1}^{n} x_i \sum_{j=1, j\neq i}^{n} x_j \frac{\lambda}{k-1}(w(p_i) + w(p_j))$$
$$= 2\frac{\lambda}{k-1} \sum_{i=1}^{n} w(p_i) x_i \sum_{j=1, j\neq i}^{n} x_j = -2\frac{\lambda}{k-1} \sum_{i=1}^{n} w(p_i) x_i^2 \leq 0.$$

The last inequality holds because we always consider non-negative weights.

Hence, the distance function f' is of negative type and the theorem follows. □

While the approximation ratio of this algorithm is better than what we presented in Theorem 2.2 (for $k > 4$), the running time is much slower.

P is implicitly represented. Suppose P is implicitly represented by a set $\mathcal{B}$ and an index Φ defined as follows. $\mathcal{B} = \{B_1, \ldots, B_s\}$ is a set of pairwise disjoint orthogonal boxes, where $s = O(k\varepsilon^{-d})$, such that (i) $\mathsf{P} \subseteq \bigcup_{i=1}^{s} B_i$, (ii) for each j, $|\mathsf{P} \cap B_j| \geq 1$, and (iii) for each j, $diam(B_j) \leq \frac{\varepsilon}{4}\gamma^*$ where γ^* is the radius of optimal k-center clustering of P and $diam(B_j)$ is the diameter of box B_j. Φ is such that, for a query box B, the points of $\mathsf{P} \cap B$ are enumerated in decreasing order of their weights one-by-one on demand in $\tau(n)$ per point after an initial query time of $\varphi(n) = \Omega(\log n)$.

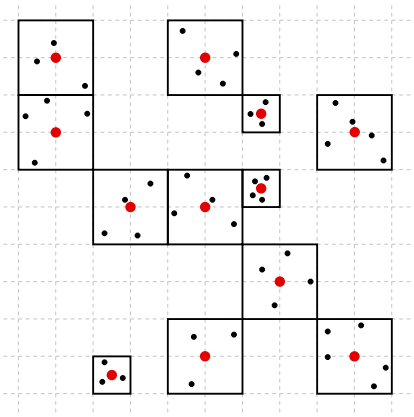

Figure 2. A point set P (in black) contained in boxes $\mathcal{B}$ whose centers are shown in red. For each box $B_i \in \mathcal{B}$, we treat the points $\mathsf{P} \cap B_i$ as if they coincide with the center b_i of B_i.

We now describe how to adapt the above algorithm to work with this representation of P. At a high level, we perturb the locations of all point in $\mathsf{P} \cap B_i$ to the center b_i of B_i (see Figure 2). Let $\widetilde{\mathsf{P}}$ be the resulting set of perturbed points. We run the above algorithm on $\widetilde{\mathsf{P}}$. Since the points in $\widetilde{\mathsf{P}}$ have k distinct locations, the algorithm can be implemented in $O(k(\tau(n) + \varphi(n)))$ time.

Let N be as above except that we ensure that $\mathrm{d}(p, q) \geq (1 - \varepsilon/4)\|p - q\|$. For each vector $u \in \mathsf{N}$, we construct the list $\widetilde{\mathsf{P}}_u$ of k highest score points with respect to u, as follows. By querying Φ with all boxes of $\mathcal{B}$, we choose the point of the maximum weight from each box. (Since all points in $\widetilde{\mathsf{P}} \cap B_i$, for any i, have the same location the ranking of their scores is the same as that of their weights.) We add all these points in a priority queue L_u with the score of the point as the key. We repeat the following k times: We remove the maximum score point $\widetilde{p}$ from L_u and add $\widetilde{p}$ to $\widetilde{\mathsf{P}}_u$. If $\widetilde{p} \in B_i$, we retrieve the point of $\widetilde{\mathsf{P}} \cap B_i$ of the next largest weight and add it to the queue L_u. After k iterations we have the set $\widetilde{\mathsf{P}}_u$. It takes $O(k\varphi(n))$ time to initially query with each box of $\mathcal{B}$, and then we retrieve a total of k additional points in a total time of $O(k\tau(n))$. We also spend $O(k \log k) = O(k\varphi(n))$ time to perform k insert/delete operations on the priority queue L_u. Hence, $\widetilde{\mathsf{P}}_u$ can be computed in $O(k(\tau(n) + \varphi(n)))$ time.

After having computing all $\widetilde{\mathsf{P}}_u$'s, we run the remainder of the algorithm from Theorem 2.2 as above and compute a set $\widetilde{\mathsf{S}}$ of k points such that $f(\widetilde{\mathsf{S}}) \geq (0.5 - \varepsilon/4) f(MS(\widetilde{\mathsf{P}}, k))$. Finally, we perturb the points of $\widetilde{\mathsf{S}}$ back to their original locations and return those points as the desired S. The correctness of the algorithm follows from the following lemma.

Lemma 2.4. $f(MS(\widetilde{\mathsf{P}}, k)) \geq (1 - \varepsilon/2) f(MS(\mathsf{P}, k))$.

Proof. From [14], we have that $\gamma^* \leq \frac{\pi(RC(\mathsf{P},k))}{\binom{k}{2}}$, where γ^* is the radius of optimal k-center clustering of P. Since we have non-negative weights it follows that for $\lambda > 0$, $\pi(RC(\mathsf{P}, k)) \leq f(MS(\mathsf{P}, k))$ and hence $\gamma^* \leq \frac{f(MS(\mathsf{P},k))}{\binom{k}{2}}$. We conclude that,

$$f(MS(\widetilde{\mathsf{P}}, k)) \geq f(MS(\mathsf{P}, k)) - 2\frac{\varepsilon}{4}\gamma^* \binom{k}{2} \geq (1 - \varepsilon/2) f(MS(\mathsf{P}, k)).$$

□

Combining Lemma 2.4 with Theorem 2.2 we obtain the following.

Lemma 2.5. *Assuming the implicit representation of* P *as defined above, a subset* $\mathsf{S} \subseteq \mathsf{P}$ *of size* k *can be computed in* $O(k(\tau(n) + \varphi(n)))$ *such that* $f(\mathsf{S}) \geq (0.5 - \varepsilon) f(MS(\mathsf{P}, k))$.

Remark. Using the implicit representation of P as defined above, we can also run the algorithm from Theorem 2.3. Skipping the details, for a constant $\varepsilon \in [\frac{4}{k}, 1)$, and parameters k, λ a subset $\mathsf{S} \subseteq \mathsf{P}$ of size k can be computed in $O(k^3 + k(\tau(n) + \varphi(n)))$ time such that $f(\mathsf{S}) \geq (1 - \varepsilon) f(MS(\mathsf{P}, k))$.

Dependency on parameter ε. For the algorithm of Theorem 2.2 we have that the number of different unit vectors we consider is $|\mathsf{N}| = O(\varepsilon^{-(d-1)/2})$, hence the total running time of the algorithm is $O((n + k \log k)\varepsilon^{-(d-1)/2})$. It also follows that the runtime in Lemma 2.5 is $O(k\varepsilon^{-d}\varphi(n) + \varepsilon^{-(d-1)/2}(k\varepsilon^{-d} + k(\tau(n) + \log k)))$. Following the analysis of [17] the algorithm in Theorem 2.3 runs in $O(nk^2 \log \varepsilon^{-1})$ time.

2.2 MMD Problem

P is given explicitly. Given P, k and λ, the MMD problem asks to return a subset T^* of k points that maximizes the utility function $f(\mathsf{T}) = \mu(\mathsf{T}) + \lambda \min_{p \in \mathsf{T}} w(p)$, where for a point set X, $\mu(X) = \min_{p,q \in X; p \neq q} \|p - q\|$ is the closest pair distance of X. Let $p_1, \ldots, p_n$ be the points of P sorted in non-increasing order of their weights and let $\mathsf{P}_i = \langle p_1, \ldots, p_i \rangle$. If the minimum weight point of T^* is p_i, then T^* is the optimal solution of the remote-edge (RE) problem for P_i, i.e., it is a subset of P_i of size k with the largest closest-pair distance. Hence, a Δ-approximate solution of the MMD problem on P can be obtained by computing a Δ-approximate solution of

the RE problem for each P_i and returning the one with the maximum utility. Using the 0.5-approximation algorithm for the RE problem by Tamir [42], we can compute a 0.5-approximate solution for the MMD problem in $O(n^2k)$. Instead of solving the RE problem for each P_i separately, we present an algorithm that maintains a $(0.5-\varepsilon)$-solution of the RE problem under insertion of new points. The solution can be updated in $O(\log k)$ amortized time per insertion resulting in an $O(n \log k)$ time algorithm. Our algorithm closely follows an improved version of the so-called "doubling algorithm" for maintaining a k-center solution under the streaming framework [19, 24, 34]. However, these algorithms require $O(k \log k)$ time to update the solution. We therefore need a number of new ideas to improve the update time as well as to adapt the algorithm to the RE problem. We first give an overview of the algorithm, which closely resembles the description in [24], and then describe the details of the update procedure, which is where most of the new ideas are needed.

Overview of the algorithm. Let $RE(\mathsf{P})$ denote an optimal remote-edge solution and $\psi(\mathsf{P}, \mathsf{S})$ denote $\max_{p\in\mathsf{P}} \min_{q\in\mathsf{S}} \|p-q\|$ for any point sets P and S. We define an (r,ε)-*packing* [25] of a point set P to be a subset $\mathsf{S} \subseteq \mathsf{P}$ that meets two properties: (i) *covering*, $\psi(\mathsf{P},\mathsf{S}) \le (1+\varepsilon)r$, and (ii) *separation*, $\mu(\mathsf{S}) \ge r$. We refer to the value r of an (r,ε)-packing as the packing's *radius*.

We fix a constant $\varepsilon \in (0, 0.5)$. Set J to be the smallest integer such that $(1+\varepsilon)^J \ge 1+\varepsilon^{-1}$; $J = O(\varepsilon^{-1}\log\varepsilon^{-1})$. Set $\alpha := (1+\varepsilon)^J$. Denote by P_t the subset of points processed so far. Our algorithm maintains J radii $r_1, r_2, \ldots, r_J$ and, for each radius r_i, both an (r_i,ε)-packing S_i for P_t of size less than k and a subset T_i of size exactly k such that $\mu(\mathsf{T}_i) \ge r_i/\alpha$. Furthermore, the algorithm maintains the invariant that the largest radius r_x and smallest radius r_y are such that $r_x \ge \alpha r_y/(1+\varepsilon)$. It then follows that $\mu(\mathsf{T}_x) \ge 0.5(1+\varepsilon)^{-2}\psi(\mathsf{P}_t,\mathsf{S}_y)$; the second inequality follows from the fact that covering the k points $RE(\mathsf{P}_t)$ with fewer than k points requires radius at least $0.5\mu(RE(\mathsf{P}_t))$. T_x is the desired RE solution for P_t. Setting $\varepsilon := \varepsilon/5$ before running the algorithm, we obtain the approximation guarantee of $(0.5-\varepsilon)$.

Suppose $p_1, \ldots, p_t$ have been processed by the algorithm and the invariants above hold. Let p_{t+1} be the next point to be inserted. We perform the following steps for each $i \in \{1,\ldots,J\}$. If there is a point q in S_i such that $\|p_{t+1}-q\| \le r_i$, no update is needed on $r_i, \mathsf{S}_i, \mathsf{T}_i$. If $\|p_{t+1}-q\| > r_i$, we add p_{t+1} to S_i. If $|\mathsf{S}_i|$ becomes k after inserting p_{t+1} then the radius r_i is increased, T_i is set to S_i, and S_i is set to a maximal subset $\mathsf{S}'_i \subset \mathsf{S}_i$ such that $\mu(\mathsf{S}'_i) \ge r_i$ for the new value of r_i. After all r_i's have been updated, we record $x_{t+1} = \operatorname{argmax}_{i\le J} r_i$ and mark the corresponding $\mathsf{T}_{x_{t+1}}$ as the desired remote-edge solution. We also compute $f(\mathsf{T}_{x_{t+1}})$, the utility of $\mathsf{T}_{x_{t+1}}$. After having processed all points, we compute $i^* = \operatorname{argmax}_{t\le n} f(\mathsf{S}_{x_t})$ and return $\mathsf{T}_{x_{i^*}}$ as the desired subset of k points.

Next we describe how each point can be inserted in $O(\log k)$ amortized time.

Details of the algorithm. To process each point of P efficiently, instead of maintaining $\mathsf{S}_i, \mathsf{T}_i$ explicitly we maintain three subsets $\mathsf{A}_i, \mathsf{N}_i, \mathsf{D}_i$ where $\mathsf{S}_i = \mathsf{A}_i \cup \mathsf{N}_i, \mathsf{T}_i = \mathsf{A}_i \cup \mathsf{D}_i$. Intuitively, N_i and D_i are buffers that store the points recently inserted into A_i and deleted from A_i, respectively; see below for details.

To initialize the algorithm, read the first k points in the stream, P_k. Then set $r_i := (1+\varepsilon)^i\mu(\mathsf{P}_k)/\alpha$, and $\mathsf{A}_i, \mathsf{N}_i, \mathsf{D}_i := \varnothing$ for all $i \in \{1,2,\ldots,J\}$. For each $i \le J$ we construct a dynamic index $\mathcal{ECP}_i$ for maintaining the closest-pair of $\mathsf{A}_i \cup \mathsf{N}_i$ (which is the same as S_i). This concludes the initialization. Continue with the following update procedure from the beginning of the stream. It is implied in the following that $\mathcal{ECP}_i$ is updated accordingly whenever a point is inserted to or removed from A_i or N_i.

Next, we describe how we update $r_i, \mathsf{A}_i, \mathsf{N}_i, \mathsf{D}_i$ when we insert a new point p. First, we check if p is within distance r_i of a point in $\mathsf{A}_i \cup \mathsf{N}_i$ as follows. We insert p to $\mathcal{ECP}_i$, query it to find the closest pair in $\mathsf{A}_i \cup \mathsf{N}_i \cup \{p\}$, then remove p from $\mathcal{ECP}_i$. If p is in the reported closest pair and the distance between the pair is less than r_i, we are done. Otherwise we insert p to N_i. If $|\mathsf{A}_i \cup \mathsf{N}_i| = k$ after the insertion, then we set $r_i := \alpha^m r_i$ where m is the smallest integer such that $\alpha^m r_i > \mu(\mathsf{A}_i \cup \mathsf{N}_i)$. To determine m, we compute $\mu(\mathsf{A}_i \cup \mathsf{N}_i)$ by querying $\mathcal{ECP}_i$. We set $\mathsf{A}_i := \mathsf{A}_i \cup \mathsf{N}_i$ and $\mathsf{D}_i, \mathsf{N}_i := \varnothing$, then *prune* A_i as follows.

Intuitively, we want to find a maximal subset of A_i such that $\mu(\mathsf{A}_i) \ge r_i$. We achieve this by repeatedly querying $\mathcal{ECP}_i$ for the closest pair of points in A_i and carefully removing at least one of the pair's points from A_i until it has the desired properties. At the start of the pruning process, let $I := \emptyset$ be an empty set. Then let p, q be the closest pair in A_i. If $\|p-q\| \ge r_i$, it must be that $\mu(\mathsf{A}_i) \ge r_i$ as desired, and the pruning stops. Otherwise, we delete at least one of p or q as follows. If $p, q \notin I$ then we check if either p or q are covered by I, i.e., there is a point $p' \in I$ or $q' \in I$ such that $\|p-p'\|$ or $\|q-q'\|$ are less than r_i, respectively. If neither such p' or q' exist, then we add p to I, delete q from A_i, then add q to D_i. If at least one of p, q is covered by I then we delete the covered point from A_i, add it to D_i, and add the uncovered point to I. This concludes the case where $p, q \notin I$. Otherwise, without loss of generality, $p \in I$ and $q \notin I$ in which case we delete q from A_i, add q to D_i, and add p to I. This concludes how we handle the closest pair p, q obtained from $\mathcal{ECP}_i$. The above, starting with the closest pair query, is repeated until $\mu(\mathsf{A}_i) \ge r_i$. This concludes the pruning process.

We note that the two cases for the closest pair p, q are exhaustive; the procedure above ensures that not both points p, q of the closest pair can be in I. We also note that we can check if a point p is covered by a point in I by maintaining another index for closest-pair queries on I, and using it similarly as we do $\mathcal{ECP}_i$ to get the minimum distance from p to the points in $\mathsf{A}_i \cup \mathsf{N}_i$ at the beginning of the update procedure.

Finally, we note that we maintain the index of the largest packing radius, x_t, and the corresponding $\mathsf{A}_{x_t}, \mathsf{D}_{x_t}$. To compute x_t after each t-th update, we maintain another index for closest pair queries on $\mathsf{A}_i \cup \mathsf{D}_i$ for all $i \le J$, then query each of them and set x_t to the largest distance returned. Using ideas from persistent indexes, we record the changes to A_i and D_i for all $i \le J$ so that we need $O(1)$ space per change and T_{x_t}, for any t, can be reported in $O(k)$ time. This concludes the entire update procedure.

Analysis. We describe a charging scheme so that each point $p \in \mathsf{P}$ is charged at most $O(1)$ operations. Fix a point $p \in \mathsf{P}$ and an integer $i \in \{1,2,\ldots,J\}$. Consider the update performed after reading p from the stream. If r_i is not increased in the update, then it is easy to verify only $O(1)$ operations are done. If r_i is increased, then A_i is pruned. In each iteration of the pruning step, $O(1)$ operations

are done, and at least one of them is the deletion of a point from A_i. Since any point is inserted to and deleted from each subset A_i, N_i, D_i at most once, any point is charged $O(1)$ total operations by this scheme. We maintain the invariant that $|A_i \cup N_i| \leq k$ throughout the algorithm, so each operation involving $\mathcal{ECP}_i$ takes $O(\log |A_i \cup D_i|) = O(\log k)$ time using the index of Bespamyatnikh [10]. All other operations take constant time. Thus, $O(\log k)$ time is charged to every point for each $i \in \{1, 2, \ldots, J\}$. We have $J = O(1)$, so the algorithm takes $O(n \log k)$ assuming P is sorted by weight. Sorting P takes $O(n \log n)$ time, so the overall runtime is $O(n \log n)$.

Since our algorithm is primarily a fast implementation of the algorithms of Guha [24] and McCutchen and Khuller [34], the correctness of it is implied by their analyses. For purposes of self-containment, we include a full analysis in Appendix A. Putting everything together, we obtain the following.

Theorem 2.6. *Given a set* P *of* n *non-negatively weighted points in* $\mathbb{R}^d$*, an integer* $k \leq n$*, a constant* $\varepsilon \in (0, 0.5)$*, and a parameter* $\lambda \geq 0$*, a subset* $S \subseteq P$ *of* k *points can be computed in* $O(n \log n)$ *time such that* $f(S) \geq (0.5 - \varepsilon) f(MM(P, k))$.

P is implicitly represented. We use the following implicit representation of P which is the same as the one described in Subsection 2.1 for the MSD problem. Let $\mathcal{B} = \{B_1, \ldots, B_s\}$ be a set of $s = O(k\varepsilon^{-d})$ boxes with the same properties as in the implicit representation of the MSD problem, and let Φ be an index that, for a query box B, can enumerate the points of $P \cap B$ in decreasing order of their weights one-by-one on demand in $\tau(n)$ per point, after an initial query time of $\varphi(n) = \Omega(\log n)$.

We now describe how to adapt the above algorithm to work with this representation of P. For each box $B_i \in \mathcal{B}$, we obtain the heaviest point $p_i \in P \cap B_i$ by querying Φ. Let H_1 be the set of these $O(k)$ points, and let H_2 be the set of k heaviest points in P by computing the bounding box $\widehat{B}$ of $\bigcup_i B_i$ and querying Φ with $\widehat{B}$. We run the above algorithm from Theorem 2.6 on $H := H_1 \cup H_2$ in $O(|H| \log |H|) = O(k \log k)$ time. It takes $O(k(\varphi(n) + \tau(n)))$ time to compute H_1 and $O(\varphi(n) + k\tau(n))$ time to compute H_2, hence $H_1 \cup H_2$ can be computed in $O(k(\varphi(n) + \tau(n))$ time. The correctness of the algorithm follows from the following lemma.

Lemma 2.7. $f(MM(H_1 \cup H_2, k)) \geq (1 - \varepsilon/2) f(MM(P, k))$.

Proof. From [14], we have that $\gamma_k^* \leq \mu(RE(P, k))$ where γ_k^* is the radius of optimal k-center clustering of P. Since we have non-negative weights it follows that for $\lambda \geq 0$, $\mu(RE(P, k)) \leq f(MM(P, k))$ and hence $\gamma_k^* \leq f(MM(P, k))$.

Consider an optimum solution $O = MM(P, k)$. If $\mu(O) \leq \frac{\varepsilon}{4}\gamma_k^*$, then $\lambda \min_{p \in O} w(p) \geq (1 - \frac{\varepsilon}{4}) f(O)$. In this case, we have $\min_{p \in H_2} w(p) \geq \min_{p \in O} w(p)$, so $f(MM(H_2, k)) \geq (1 - \frac{\varepsilon}{4}) f(O)$.

Otherwise, $\mu(O) > \frac{\varepsilon}{4}\gamma_k^* \geq diam(B_i)$ for all $B_i \in \mathcal{B}$ where the second inequality follows from the properties of $\mathcal{B}$. It follows that $|O \cap B_i| \leq 1$ for all $B_i \in \mathcal{B}$. Let p_i be the point in $B_i \cap O$ (if any) for each $B_i \in \mathcal{B}$. By construction, there is a unique point $q_i \in B_i \cap H_1$ with $w(q_i) \geq w(p_i)$. It follows that

$$f(MM(H_1, k)) \geq f(O) - 2\frac{\varepsilon}{4}\gamma_k^* \geq (1 - \varepsilon/2) f(O).$$

In either case above, we have $f(MM(H_i, k)) \geq (1 - \frac{\varepsilon}{2}) f(O)$ for some $i \in \{1, 2\}$. Since $f(MM(H_1 \cup H_2, k)) \geq f(MM(H_1, k))$, $f(MM(H_2, k))$ always, we are done. □

Combining Lemma 2.7 with Theorem 2.6 we obtain the following.

Lemma 2.8. *Assuming the implicit representation of* P *is as defined above, a subset* $S \subseteq P$ *of size* k *can be computed in* $O(k(\tau(n) + \varphi(n)))$ *time such that* $f(S) \geq (0.5 - \varepsilon) f(MM(P, k))$.

Remark. When P is given explicitly, the construction of $H_1 \cup H_2$ can be improved to only $O(n)$ expected time, which in turn improves the total runtime to $O(n + k \log k)$ in expectation as follows[6]. First, we compute a 2-approximate k-center radius γ for P using the (expected) linear-time algorithm of Har-Peled and Raichel [25]; that is, $\gamma_k^* \leq \gamma \leq 2\gamma_k^*$ where γ_k^* is the optimal k-center radius for P. Set $\delta := \varepsilon\gamma/4\sqrt{d}$. Consider the axis-aligned grid in $\mathbb{R}^d$ with sidelength δ. For each point $p \in P$, compute the grid cell containing it, and let $\mathcal{B}$ be the set of all such grid cells that contain the points in P. It follows that $|\mathcal{B}| = O(k\varepsilon^{-d}) = O(k)$ [5]. Note that $\mathcal{B}$ satisfies the properties of the implicit representation described in Subsection 2.1. After computing $P \cap B_i$ for each $B_i \in P$ by bucketing, the subsets H_1 (the subset of heaviest points in each $B_i \in \mathcal{B}$) and H_2 (the k heaviest points in P) required can be easily computed in $O(n)$ time overall using a selection algorithm. As above, we then run the algorithm from Theorem 2.6 on input $H_1 \cup H_2$ in $O(k \log k)$ time to obtain the solution. By Lemma 2.7 and Theorem 2.6 we conclude to the following theorem.

Theorem 2.9. *Given a set* P *of* n *non-negatively weighted points in* $\mathbb{R}^d$*, an integer* $k \leq n$*, a constant* $\varepsilon \in (0, 0.5)$*, and a parameter* $\lambda \geq 0$*, a subset* $S \subseteq P$ *of* k *points can be computed in* $O(n \log k)$ *time or* $O(n + k \log k)$ *expected time such that* $f(S) \geq (0.5 - \varepsilon) f(MM(P, k))$.

Dependency on parameter ε. When ε is not a constant, the algorithm of Theorem 2.6 takes $O(n\varepsilon^{-1} \log k \log \varepsilon^{-1})$ time to stream over the points after they have been sorted in $O(n \log n)$ time. It follows that the runtime in Lemma 2.8 is $O(k\varepsilon^{-d}\tau(n) + k\varphi(n) + k\varepsilon^{-d-1} \log k \log \varepsilon^{-1})$, and the runtime of the algorithm from Theorem 2.9 is $O(n + k\varepsilon^{-d-1} \log k \log \varepsilon^{-1})$.

3 INDEXES FOR MSD AND MMD RANGE QUERIES

In this section we describe indexes that answer MSD and MMD range queries efficiently with good approximation ratios. As mentioned in the Introduction our main idea is that given a query rectangle ρ, we obtain an implicit representation of $P \cap \rho$ and we use the algorithms described in Section 2 that work with an implicit representation of $P \cap \rho$. We use the following two indexes to compute the implicit representation of $P \cap \rho$:

Abrahamsen *et al.* [1] describe an index to answer a range Euclidean k-center query. We use the following property of their index. Given a set P of n points in $\mathbb{R}^d$ an index $\Psi(P)$ of size $O(n \log^{d-1} n)$ can be constructed in $O(n \log^{d-1} n)$ time such that given a rectangle ρ, an integer $k \geq 1$, and a constant $\varepsilon \in (0, 1)$, a collection of interior-disjoint rectangles $\mathcal{B}$ can be computed in $O(k \log^{d-1} n)$ time such that $P \cap \rho \subseteq \mathcal{B}$ and $|\mathcal{B}| = O(k\varepsilon^{-d})$, and for all $B \in \mathcal{B}$, $|P \cap B| \geq 1$ and $diam(B) \leq \frac{\varepsilon}{4}\gamma_k^*(P \cap \rho)$, where $\gamma_k^*(X)$ is the radius of the optimum k-center of X.

[6] Using the deterministic 2-approximation algorithm of Feder and Greene [21] instead to construct $H_1 \cup H_2$ results in an overall deterministic runtime of $O(n \log k)$.

The second index we use is by Rahul *et al.* [37] for answering top-k range queries. Given a set P of n weighted points in $\mathbb{R}^d$ an index $\Phi(\mathsf{P})$ of size $O(n \log^d n)$ can be constructed in $O(n \log^d n)$ time such that for a rectangle ρ, a query is initialized in $O(\log^{d-1} n)$ time and the points of $\mathsf{P} \cap \rho$ can be reported sequentially in an non-increasing order of their weights, in an on-demand fashion, and in $O(\log \log n)$ time per point.

MSD range queries. Given $\mathsf{P} \subset \mathbb{R}^d$, we construct an index for the range MSD problem that returns a $(0.5 - \varepsilon)$-approximation of the optimum answer for a query rectangle ρ. In this index, the parameters k, ε, and λ can be specified by a user as part of the query. More precisely, we build the index $\Psi := \Psi(\mathsf{P})$ on P and the index $\Phi := \Phi(\mathsf{P})$ on P. The size of the overall index is $O(n \log^d n)$, and it takes $O(n \log^d n)$ time to build Ψ and Φ.

Given a query rectangle ρ, and parameters k, λ, ε, we first query Ψ with ρ, k, and ε to obtain the rectangles $\mathcal{B} = \{B_1, \ldots, B_s\}$, where $s = O(k\varepsilon^{-d})$, that cover all points in $\mathsf{P} \cap \rho$. Next, for each $B_i \in \mathcal{B}$, we initiate a query on Φ with B_i and enumerate the points of $\mathsf{P} \cap B_i$ in a non-increasing order of their weights as needed. Finally, we run the MSD algorithm with this implicit representation of $\mathsf{P} \cap \rho$. Since $\tau(n) = O(\log \log n)$ and $\varphi(n) = O(\log^{d-1} n)$ by Lemma 2.5 we obtain the following.

Theorem 3.1. *Given a set* P *of* n *non-negatively weighted points in* $\mathbb{R}^d$*, an index can be constructed in* $O(n \log^d n)$ *time with* $O(n \log^d n)$ *space such that for a query rectangle* ρ*, an integer* $k \leq n$*, a constant* $\varepsilon \in (0, 0.5)$*, and a parameter* $\lambda \geq 0$*, a subset* $\mathsf{S} \subseteq \mathsf{P} \cap \rho$ *of* k *points can be computed in* $O(k \log^{d-1} n)$ *time such that* $f(\mathsf{S}) \geq (0.5 - \varepsilon) f(MS(\mathsf{P} \cap \rho, k))$.

Remarks. i) If the constant ε is fixed then we can construct copies of the index $\Phi(\mathsf{P})$ from [37] for each direction $u \in \mathsf{N}$, where the weight of a point $p \in \mathsf{P}$ is the inner product $\langle p, u \rangle$ and answer a range MSD query in time $O(\log^{d-1} n + k \log \log n)$. We describe the full details of the index in Appendix B.

ii) Using the same indexes above, we can use Theorem 2.3 and the remark in the end of Subsection 2.1 to construct an index of size $O(n \log^d n)$ that computes an $(1 - \varepsilon)$-approximation for MSD range queries in $O(k^3 + k \log^{d-1} n)$ time.

MMD range queries. Using the same indexes as in the MSD range query, and the result of Lemma 2.8, we obtain the following.

Theorem 3.2. *Given a set* P *of* n *non-negatively weighted points in* $\mathbb{R}^d$*, an index can be constructed in* $O(n \log^d n)$ *time with* $O(n \log^d n)$ *space such that given a rectangle* ρ*, an integer* $k \leq n$*, a constant* $\varepsilon \in (0, 0.5)$*, and a parameter* $\lambda \geq 0$*, a subset* $\mathsf{S} \subseteq \mathsf{P} \cap \rho$ *of* k *points can be computed in* $O(k \log^{d-1} n)$ *time such that* $f(\mathsf{S}) \geq (0.5 - \varepsilon) f(MM(\mathsf{P} \cap \rho, k))$.

Dependency on parameter ε. When ε is not a constant, the index of Abrahamsem *et al.* takes $O(k\varepsilon^{-d+1} \log^{d-1} n + k\varepsilon^{-d})$ time to report the $O(k\varepsilon^{-d})$ boxes. From Lemma 2.5 and the discussion at the end section Subsection 2.1, it follows that MSD range queries take $O(k\varepsilon^{-d}(\log^{d-1} n + \varepsilon^{-(d-1)/2}))$. From Lemma 2.8 and the discussion at the end section Subsection 2.2, it follows that MMD range queries take $O(k\varepsilon^{-d} \log^{d-1} n + k\varepsilon^{-d-1}(\log^d n + \log k \log \varepsilon^{-1}))$ time.

4 CONSTRAINED OPTIMIZATION

In this section we present efficient indexes for the constrained diverse top-k (CDT) problem.

4.1 A coarse-approximation index

We first describe a greedy algorithm for the offline version of the CDT problem, which will be used by the query procedure of our index. We run the following operations k times or until there is no point left in P: Find the heaviest point p in P. Remove all points of P within distance δ from p, including p itself. This procedure returns a set S of at most k points. It is trivial to observe that $\min_{p,q \in \mathsf{S}} ||p - q|| \geq \delta$.

Let C^* be an optimal CDT solution for P. Let $p_i \in \mathsf{P}$ be the point selected by the greedy algorithm in the i-th iteration, and let $C_i \subseteq C^*$ be the set of points in the optimal solution that are removed in iteration i. By construction,

$$w(p_i) \geq \max_{p \in C_i} w(p) \geq \frac{1}{|C_i|} \cdot w(C_i).$$

A packing argument shows that $|C_i| \leq 2^{O(d)}$. Hence, $w(p_i) \geq 0.5^{O(d)} w(C_i)$. We claim that $C^* = \bigcup C_i$. Indeed, if there is a point $p \in C^* \setminus \bigcup C_i$, then $|\mathsf{S}| = k$, there is an i such that $C_i = \varnothing$ and $w(()p) < w(()p_i)$. We can obtain a better CDT solution of P by replacing p with p_i in C^*, which contradicts the optimality of C^*. Hence, $\sum w(C_i) = w(C^*) = w(CD(\mathsf{P}, k, \delta))$. Thus,

$$w(\mathsf{S}) = \sum w(p_i) \geq 0.5^{0(d)} \sum w(C_i) = 0.5^{O(d)} w(CD(\mathsf{P}, k, \delta)).$$

Next, we describe an index that, for a query rectangle ρ, enables us to run in $O(k \log^d n)$ time a slightly relaxed version of the above procedure in which δ is a soft constraint, see below.

Index. The main index that we build is the balanced box decomposition tree (or *BBD* tree for short) [7, 8], a variant of the *quadtree*. A BBD-tree 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 T is associated with a region $\square_u$, which is either a rectangle or a region between two nested rectangles, and a subset $\mathsf{P}_u \subseteq \mathsf{P}$ of points that lie inside $\square_u$. Notice that $\square_{\text{root}} = \square$. If $|\mathsf{P}_u| = 1$, then u is a leaf. If $|\mathsf{P}_u| > 1$, then u has two children, say, w and z, and $\square_w$ and $\square_z$ partition $\square_u$ (see Figure 3). Regions associated with the nodes of T induce a hierarchical partition of $\mathbb{R}^d$.[7]

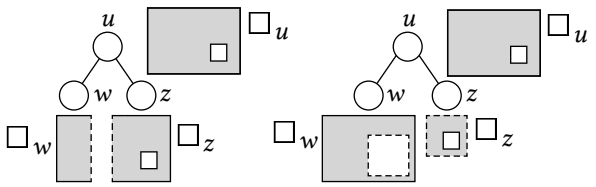

Figure 3. Two example BBD subtrees with identical regions $\square_u$ at the root nodes. On the left, $\square_u$ is split by a vertical line. On the right, $\square_u$ is split into $\square_w$, $\square_z$ by a rectangle.

[7] A BBD-tree can be viewed as a generalization of a kd-tree that combines the features of kd-trees and quadtrees and is similar to hB-trees [30] used for answering range queries.

For each node u, we also store a bool variable b_u. Initially, $b_u = 0$ for all nodes of T. We use b_u to implicitly delete the points from P. For a node u, let $\mathsf{P}_u^* \subseteq \mathsf{P}_u$ be the subset of points that have not been deleted. That is, if $b_u = 1$ then $\mathsf{P}_u^* = \emptyset$. If $b_u = 0$ then $\mathsf{P}_u^* = \mathsf{P}_u$ for a leaf u and $\mathsf{P}_u^* = \mathsf{P}_w^* \cup \mathsf{P}_z^*$ for an interior node u with w and z as its children. The BBD-tree takes $O(n)$ space and can be constructed in $O(n \log n)$ time.

Additionally, we construct a d-dimensional range tree Λ for answering range-max queries, i.e., for a query rectangle R, it returns the point of maximum weight in $R \cap \mathsf{P}$ in $O(\log^{d-1} n)$ time. The construction time and the size of our index is dominated that of Λ which is constructed in $O(n \log^{d-1} n)$ time and has size $O(n \log^{d-1} n)$.

For a set Z of nodes of T, let $\square(Z) = \bigcup_{z \in Z} \square_z$. For a subset $X \subset \mathbb{R}^d$ and for a parameter $\varepsilon > 0$, let $X^\varepsilon = X \oplus B(0, \varepsilon \cdot diam(X)) = \{y \in \mathbb{R}^d \mid \exists x \in X, \|y - x\| \leq \varepsilon \cdot diam(X)\}$.

A crucial property of a BBD-tree, which we will be using repeatedly, is stated in the following lemma:

Lemma 4.1. *Let $C \subseteq \mathbb{R}^d$ be a convex region and let $\varepsilon > 0$ be a parameter.*

(i) There exists a set Z_C of $O(\log n + \varepsilon^{-(d-1)})$ nodes of T such that $C \subseteq \square(Z_C) \subseteq C^\varepsilon$, the (interiors of the) regions associated with the nodes of Z_C are pairwise disjoint, and there are $O(|Z_C|)$ distinct ancestors of the nodes in Z_C. There is an algorithm to compute Z_C in $O(\log n + \varepsilon^{-(d-1)})$ time.

(ii) There is also a set $\overline{Z}_C$ of $O(\log n + \varepsilon^{-(d-1)})$ nodes such that $\square(\overline{Z}_C) = \square \setminus \square(Z_C)$.

The proof of part (i) can be found in [7], and the proof of (ii) follows from the construction of T. Let CoverRegion(C, ε) denote the procedure that computes Z_C.

Next, we define the query procedure. Let R be a query rectangle, and let δ, ε, k be the parameters as described above. The query procedure runs the above offline greedy algorithm efficiently using T and treating δ as a soft constraint. In the initialization phase, the algorithm computes Z_R using CoverRegion(R, ε). Next, it sets $b_v = 1$ for all $v \in \overline{Z}_R$. For each node $v \in Z_R$, by querying Λ, it computes the heaviest point $\xi_v^* \in \mathsf{P}_v \cap R$ (note that $\mathsf{P}_v \subseteq R^\varepsilon$ but a point of P_v may lie outside R). Next, by proceeding in a bottom-up manner, for all ancestors w of nodes in Z_R, we compute ξ_w^*, the maximum-weight point of $\mathsf{P}_w^* \cap R$ in $O(1)$ time per node. This completes the initialization phase.

The query procedure maintains a subset $Q \subseteq \mathsf{P} \cap R$ of at most k points. Initially, $Q = \emptyset$. It also maintains two sets Z and $\overline{Z}$ of nodes with pairwise-disjoint regions such that $R \setminus \bigcup_{q \in Q} B(q, (1+\varepsilon)\delta)) \subseteq \square(Z) \subseteq R^\varepsilon \setminus \bigcup_{q \in Q} B(q, \delta)$ and $\square(\overline{Z}) = \square \setminus \square(Z)$; see Figure 4. Initially $Z = Z_R$ and $\overline{Z} = \overline{Z}_R$. If a node v is an ancestor of a node in Z, then it also stores the point ξ_v^* of $\mathsf{P}_v^* \cap R$. The initialization phase has computed initial ξ_v^*'s. The procedure terminates when $|Q| = k$ or $\mathsf{P}_{\text{root}}^* = \emptyset$.

In each iteration, the procedure performs the following steps:

(i) Add ξ_{root}^* to Q.

(ii) Using the CoverRegion$(B(\xi_{\text{root}}^*, \delta))$ procedure, compute a set U of nodes such that $B(\xi_{\text{root}}^*, \delta) \subseteq \square(U) \subseteq B(\xi_{\text{root}}^*, (1+\varepsilon)\delta)$.

(iii) Update the sets Z and $\overline{Z}$ of nodes to satisfy the properties above for the new value of Q.

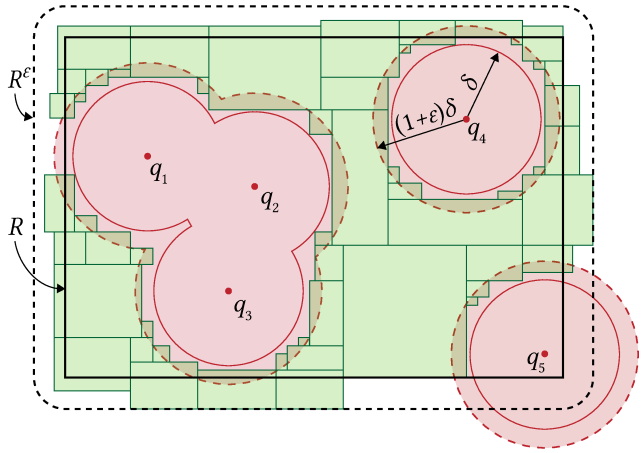

Figure 4. An example after five iterations of the algorithm for a query rectangle R in black with $\square(Z)$ in green and $\bigcup_j B(q_j, \delta)$ in red.

The only non-trivial step is (iii). We update Z and $\overline{Z}$ as follows. Let u be a node of the set U from step (ii). If u is a descendant of a node $\overline{Z}$, then there is nothing to do. Otherwise we add u to $\overline{Z}$, set $b_u = 1$, and remove all descendants of u from Z and $\overline{Z}$. Next, if an ancestor w of u is in Z (there is at most one such node), then we remove w from Z and add the siblings of nodes on the path from w to u in T to Z (i.e. we shrink $\square(Z)$ from $\square_w$ to $\square_w \setminus \square_u$). For each node v added to Z, we compute the heaviest point in $\mathsf{P} \cap (R \cap \square_v)$ using Λ. Then we update the ξ_v^*'s at all ancestors of u. We repeat this step for all nodes of U.

When the procedure terminates, it returns Q. It is easy to verify that Q is a δ-diverse k-set, i.e., $|Q| \leq k$ and the pairwise distances between any pair of points of Q is at least δ. If the procedure terminates with $|Q| < k$, then $R \subseteq \bigcup_{q \in Q} B(q, (1+\varepsilon)\delta)$. The analysis of the greedy algorithm implies that $w(Q) \geq 0.5^{O(d)} w(CD(\mathsf{P} \cap \rho, k, \delta))$.

By Lemma 4.1, $O(\log n + \varepsilon^{-(d-1)})$ nodes are added to or deleted from Z or $\overline{Z}$ in each iteration. The total number of insertions and deletions in $Z, \overline{Z}$ is $O(k(\log n + \varepsilon^{-(d-1)}))$. Each time a node is added to Z a range-max query on Λ is performed in $O(\log^{d-1} n)$ time. Finally, there are $O(\log n)$ ancestors of a node, so we spend $O(\log n(\log n + \varepsilon^{-(d-1)}))$ time to update the ξ_v^*'s in each iteration. Hence, each iteration takes $O(\log^{d-1} n(\log n + \varepsilon^{d-1}))$ time. The time taken by the query procedure is $O(k \log^{d-1} n(\log n + \varepsilon^{-d-1}))$. We thus obtain the following:

Theorem 4.2. *Given a set P of n points in $\mathbb{R}^d$, an index can be constructed in $O(n \log^{d-1} n)$ time with $O(n \log^{d-1} n)$ space such that given a rectangle ρ, an integer $k \leq n$, a constant $\varepsilon \in (0, 1)$, and a parameter $\delta > 0$, a δ-diverse k-set $\mathsf{S} \subseteq \mathsf{P} \cap \rho$ with $w(\mathsf{S}) \geq 0.5^{O(d)} w(CD(\mathsf{P} \cap \rho, k, \delta))$ can be reported in $O(k \log^d n)$ time.*

4.2 An ε-approximation index

Let P be a set of n points $\mathbb{R}^d$, and let $k \geq 1, \delta > 0, \varepsilon \in (0, 1)$ be parameters that are fixed during the preprocessing phase. We describe an index to preprocess P that for a query rectangle ρ, returns a subset $\mathsf{S} \subseteq \mathsf{P}$ of at most k points such that $|\mathsf{S}| \leq k$,

$S \subseteq \rho \oplus [0, \varepsilon\delta]^d$, $\mu(S) \geq \delta$, and $w(S) \geq (1-\varepsilon)CD(P, k, \delta)$. We note that unlike the previous index, some points of S may lie outside ρ (but within distance $\varepsilon\delta$ from ρ). Since δ is fixed in the beginning, without loss of generality, we assume $\delta = 1$.[8] Furthermore, for any set $A \subseteq \mathbb{R}^d$ and for an integer $i > 0$, we use $CD(A, i)$ to denote $CD(A, i, 1)$. Finally, let $T_i(A)$ denote the pair $(CD(A, i), w(CD(A, i)))$ and $\mathcal{T}(A) = \{T_1(A), \ldots, T_k(A)\}$.

We observe that the problem is *decomposable* in the following sense: We call point sets A, B *well-separated* if the closest distance between A and B is greater than 1 (recall $\delta = 1$), i.e., $\|p - q\| > 1$ for all $p \in A$ and $q \in B$ (by definition, $A \cap B = \varnothing$). Then, for any i and well-separated subsets A, B, we have

$$CD(A \cup B, i) = CD(A, j) \cup CD(B, i - j)$$

for some $j \leq i$. Now let S be a set of points that can be partitioned into pairwise well-separated subsets $S_1, S_2, \ldots, S_m$. For $t \leq m$, set $X_t = \bigcup_{j \leq t} S_t$. The previous observation implies that for any $t \leq m$,

$$w(CD(X_t, i)) = \max_{0 \leq j \leq i} w(CD(X_{t-1}, i - j)) + w(CD(S_t, j))$$

where $X_0 = \emptyset$. Using the recurrence and dynamic programming, $\mathcal{T}(X_1), \ldots, \mathcal{T}(X_m) = \mathcal{T}(S)$ can be computed in a total of $O(mk^2)$ time assuming we have $\mathcal{T}(S_1), \ldots, \mathcal{T}(S_m)$ at our disposal.

We are now able to describe our index, which critically uses the above observation. For sake of exposition, we begin by describing an offline algorithm in 1D similar to that of Hunt *et al.* [27] and Matsui [33]. Then we describe the index for 1D, and finally describe the index in higher dimensions.

1D offline algorithm. Let P be a set of n points in $\mathbb{R}^1$. Without loss of generality, assume that no point in P has an integer value. We describe a $(1-\varepsilon)$-approximation algorithm to compute $CD(P, k)$ using the shifted grid technique [27, 29, 33].

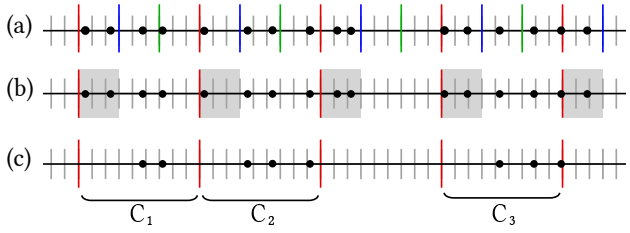

Figure 5. An example with $r = 3$. (a) The grid points of G_1, G_2, G_3 are shown in red, blue, and green, respectively, and the grid points of F not in any G_i are shown in grey. (b) The darkened intervals contain the points in $P \setminus P^{(1)}$. (c) $P^{(1)}$ and the non-empty cells C_1, C_2, and C_3 of G_1.

Set $r = \lceil 1/\varepsilon \rceil$. We create r grids $G_0, G_1, \ldots, G_{r-1}$ over $\mathbb{R}^1$ where G_i has grid points $\{i + ra \mid a \in \mathbb{Z}\}$. Each grid cell has length $r = O(\varepsilon^{-1})$. Since no point in P has integer coordinates, each point in P lies in exactly one grid cell for each G_i.

For each grid G_i, we remove the points of P that are within distance 1 (recall that $\delta = 1$) from the left endpoint of their containing grid cells in G_i. Let $P^{(i)}$ be the set of remaining points. Each point $p \in P$ is within distance 1 from the left endpoint of its containing grid cell in exactly one grid and thus it remains in all but one grid. See Figure 5. By a simple packing argument,

$$\max_{i<r} w(CD(P_i, k)) \geq (1-\varepsilon) w(CD(P, k)).$$

Thus, the goal is to compute $CD(P^{(i)}, k)$ for all i and then report the best among them.

Fix a value of i. We compute $CD(P^{(i)}, k)$ as follows. Let $C_1, \ldots, C_m$ be the non-empty cells of G_i, and let $S_t = C_t \cap P^{(i)}$ for each $t \leq m$. We describe below how we compute $\mathcal{T}(S_t)$ for each $t \leq m$. For now, assume that we have $\mathcal{T}(S_t)$ at our disposal. For $1 \leq t < t' \leq m$, we note that $S_t, S_{t'}$ are well-separated because points of $P \cap C_{t'}$ lying within distance 1 from the left endpoint of $C_{t'}$ have been deleted. Using the dynamic programming approach outlined above, we can compute $\mathcal{T}(P^{(i)})$ and thus $CD(P^{(i)}, k)$ in $O(mk^2)$ time.

We now describe the procedure for computing $\mathcal{T}(S_t)$ for $1 \leq t \leq m$. Notice that each grid cell C_t has length r, so $CD(S_t, j)$ can contain at most r points, even for $j > r$. To compute $CD(S_t, j)$, we consider all subsets of S_t of size at most $\min\{j, r\}$. For each subset X, we compute $\mu(X)$. If $\mu(X) \leq 1$ we set $w(X) = -\infty$, otherwise we compute $w(X)$. We set $CD(S_t, j)$ to be the subset X^* with the highest weight, and set $T_j(S_t) = (X^*, w(X^*))$. The total time spent is $|S_t|^{O(r)} = |S_t|^{O(\varepsilon^{-1})}$. Summing this cost over all non-empty grid cells of G_i and adding the cost of dynamic programming, the cost of computing $\mathcal{T}(P^{(i)})$ is $n^{O(\varepsilon^{-1})}$. Putting everything together, we obtain the following:

Lemma 4.3. *Let P be a set of n points in $\mathbb{R}^1$ and let k, ε, δ be parameters. There exists an $n^{O(\varepsilon^{-1})}$-time $(1-\varepsilon)$-approximation algorithm to compute $CD(P, k, \delta)$.*

Index. Next, we describe an index that is used to implement the above procedure efficiently in the range query setting.

Let $r, G_0, \ldots, G_{r-1}$ and $P^{(0)}, \ldots, P^{(r-1)}$ be the same as above. We construct a fine grid F of size $1/r$, i.e., $F = \{a/r \mid a \in \mathbb{Z}\}$. We note that F is a refinement of G_i. We call an interval *F-aligned* if its endpoints are grid points of F. For each $i \leq t$, we build an index $\Psi^{(i)}$ on $P^{(i)}$ that for a F-aligned query interval I, returns $CD(P^{(i)} \cap I, k)$.

The index $\Psi^{(i)}$ consists of two parts. First, we build a 1D range tree $\Lambda^{(i)}$ on the non-empty grid cells $C_1, \ldots, C_m$ of G_i. That is, $\Lambda^{(i)}$ is a height-balanced binary tree. Each node $u \in \Lambda^{(i)}$ is associated with an interval σ_u and a subset $P_u = P^{(i)} \cap \sigma_u$ of points. If u is the i-th leftmost leaf then $\sigma_u = C_i$, and if u is an interior node with children w and z then σ_u is the smallest interval containing σ_w and σ_z. Each node u of $\Lambda^{(i)}$ stores $\mathcal{T}(P_u^{(i)})$. If u is a leaf, we compute $\mathcal{T}(P_u^{(i)})$ using the brute-force approach described above. If u is an interior node with children w and z, then we compute $\mathcal{T}(P_u^{(i)})$ from $\mathcal{T}(P_w^{(i)})$ and $\mathcal{T}(P_z^{(i)})$ using dynamic programming, as outlined above.

The total time spent constructing $\Lambda^{(i)}$ is $n^{O(\varepsilon^{-1})}$. $\Lambda^{(i)}$ has $O(n)$ nodes and each node requires $O(k^2)$ space to store $\mathcal{T}(P_u^{(i)})$. On the other hand, a point of $P^{(i)}$ is stored in at most one node for any fixed level, so it appears at most k times in a given $\mathcal{T}(P_u^{(i)})$, and $\Lambda^{(i)}$ has $O(\log n)$ levels. Putting these two bounds together, we conclude that the size of $\Lambda^{(i)}$ is $O(nk \min\{k, \log n\})$.

The second part of the index $\Psi^{(i)}$ consists of a set of tables. In particular, let C_t be a non-empty grid cell of G_i, and let $S_t = C_t \cap$

[8] If $\delta \neq 1$, we can scale P as well as query rectangles by the factor $1/\delta$ to ensure $\delta = 1$

$\mathsf{P}^{(i)}$. For every pair a, b of grid points of F within C_t, we compute $\mathcal{T}(\mathsf{P}^{(i)} \cap [a, b])$ using the brute-force approach. The total size is $O(\varepsilon^{-4}k^2)$ for each C_t, and it takes $|\mathsf{S}_t|^{O(\varepsilon^{-1})}$ time to construct these tables. The total size of these tables is $O(nk)$, and we spend $n^{O(\varepsilon^{-1})}$ time to compute them. Hence, the size of $\Psi^{(i)}$ is $O(nk \min\{k, \log n\})$, and the preprocessing time is $n^{O(\varepsilon^{-1})}$.

Finally, we describe the query procedure. Let I be a query interval. If I is not F-aligned, we extend I to the smallest F-aligned interval containing I, so let us assume that I is F-aligned. Let C_L (resp. C_R) be the cell of G_i containing the left (resp. right) endpoint of I. We split I into three sub-intervals: left interval $I_L = I \cap \mathsf{C}_L$, right interval $I_R = I \cap \mathsf{C}_R$, and middle interval $I_M = I \setminus (I_L \cup I_R)$. Since I is F-aligned, I_L (resp. I_R) spans a contiguous sequence of refined grid cells within a single cell of G_i, so we already have computed $\mathcal{T}(\mathsf{P}^{(i)} \cap I_L)$ (resp. $\mathcal{T}(\mathsf{P}^{(i)} \cap I_R)$) during preprocessing. If $I_M \neq \varnothing$, we compute $\mathcal{T}(\mathsf{P}^{(i)} \cap I_M)$ as follows: We query $\Lambda^{(i)}$ with I_M and identify $O(\log n)$ nodes $u_1, \ldots, u_s$ such that $\sigma_{u_1}, \ldots, \sigma_{u_s}$ partition I_M. Since $\mathsf{P}^{(i)}_{u_1}, \ldots, \mathsf{P}^{(i)}_{u_s}$ are well-separated and we have precomputed $\mathcal{T}(\mathsf{P}^{(i)}_{u_j})$ for each j, we can compute $\mathcal{T}(\mathsf{P}^{(i)} \cap I_M)$ in $O(k^2 \log n)$ time using dynamic programming. Finally, we compute $\mathcal{T}(\mathsf{P}^{(i)} \cap I)$ from $\mathcal{T}(\mathsf{P}^{(i)} \cap I_L)$, $\mathcal{T}(\mathsf{P}^{(i)} \cap I_R)$, and $\mathcal{T}(\mathsf{P}^{(i)} \cap I_M)$, again using dynamic programming. The total query time is $O(k^2 \log n)$.

Repeating this for all r grids and returning the best among them, we obtain a feasible subset of at most k points whose weight is at least $(1-\varepsilon)w(CD(\mathsf{P} \cap I, k))$. Putting everything together, we obtain the following:

Theorem 4.4. *Given a set P of n points in $\mathbb{R}^1$, an integer $k \leq n$, a constant $\varepsilon \in (0,1)$, and a parameter $\delta > 0$, an index can be constructed in $n^{O(\varepsilon^{-1})}$ time with $O(nk \min\{k, \log n\})$ space such that given an interval ρ, a δ-diverse k-set $\mathsf{S} \subseteq \mathsf{P} \cap (\rho \oplus [0, \varepsilon])$ with $w(\mathsf{S}) \geq (1-\varepsilon)w(CD(\mathsf{P} \cap \rho, k, \delta))$ can be reported in $O(k^2 \log n)$ time.*

A modified index. We can get an index with faster query time and smaller size that allows k to be provided at query time at the cost of allowing the index to answer queries with εk additional points. For any set $\mathsf{A} \subseteq \mathbb{R}^d$, let $\mathcal{T}^{\varepsilon}(\mathsf{A})$ denote the subset $\{T_{\lfloor (1+\varepsilon)^i \rfloor}(\mathsf{A}) \mid i \in \mathbb{Z}.\, 0 \leq i \leq \lceil \log_{1+\varepsilon} |\mathsf{A}| \rceil\}$ of $\mathcal{T}(\mathsf{A})$. To obtain the new index, we first build the previous index for $k = n$. Then, for any set A for which $\mathcal{T}(\mathsf{A})$ was stored in the index, we keep only $\mathcal{T}^{\varepsilon}(\mathsf{A})$ instead. The size of the resulting index is $O((n/\varepsilon)\log n) = O(n \log n)$. For a query interval I, we can compute a $(1-\varepsilon)$-approximate solution S for $\mathsf{P} \cap \rho$ where $|\mathsf{S}| \leq (1+\varepsilon)k$ and $\mathsf{S} \subseteq \mathsf{P} \cap (I \oplus [0, \varepsilon\delta])$ in $O(k \log k \log n)$ time. Note that query time is better than that of the previous index when $k > \sqrt{\log n}$. We refer the reader to Appendix C for the full details and conclude with the following theorem.

Theorem 4.5. *Given a set P of n points in $\mathbb{R}^1$, a constant $\varepsilon \in (0,1)$, and a parameter $\delta > 0$, an index can be constructed in $n^{O(\varepsilon^{-1})}$ time with $O(n \log n)$ space such that given an interval ρ and integer $k \leq n$, a δ-diverse k-set $\mathsf{S} \subseteq \mathsf{P} \cap (\rho \oplus [0, \varepsilon])$ with $w(\mathsf{S}) \geq (1-\varepsilon)w(CD(\mathsf{P} \cap \rho, k, \delta))$ can be reported in $O(k \log k \log n)$ time.*

Higher dimensions. Now, we discuss how to extend our index to higher dimensions. We describe the index in $\mathbb{R}^2$, which extends to $d > 2$ in a straightforward manner.

Set $r = \lceil 2\varepsilon^{-1} \rceil$ and $s = r^2$. We construct s 2-dimensional grids $G_0, \ldots, G_{s-1}$. For $i < s$, if i is of the form $i = \alpha r + \beta$, where $\alpha, \beta \in [0 : r-1]$, then

$$G_i = \{(\alpha + ar, \beta + br) \mid a, b \in \mathbb{Z}^d\}.$$

We index a grid cell of G_i by its bottom-left vertex, i.e., by grid cell (α, β) we mean the square $[\alpha, \alpha+r] \times [\beta, \beta+r]$. Similarly by row (resp. column) α we mean the strip $\mathbb{R} \times [\alpha, \alpha+r]$ (resp. $[\alpha, \alpha+r] \times \mathbb{R}$). For each cell of G_i, we remove the points of P that lie inside it and within distance 1 from its bottom or left boundary. Let $\mathsf{P}^{(i)}$ denote the set of remaining points. We also construct a 2-dimensional fine grid

$$F = \{(a/r, b/r) \mid a, b \in \mathbb{Z}\}.$$

F is a refinement of every G_i. We call a rectangle *F-aligned* if its vertices are grid points of F. For each $i < s$, we build an index $\Psi^{(i)}$ on $\mathsf{P}^{(i)}$ that for a F-aligned rectangle ρ, returns $CD(\mathsf{P}^{(i)} \cap \rho, k)$. For a query rectangle ρ, we compute $CD(\mathsf{P}^{(i)} \cap \rho, k)$ for all $i < s$ and return the one with the highest weight. Generalizing the 1D argument, we can show that

$$\max_{i<s} w(CD(\mathsf{P}^{(i)} \cap \rho, k)) \geq (1-\varepsilon)w(CD(\mathsf{P} \cap \rho, k)).$$

We now describe the index $\Psi^{(i)}$. As in 1D, $\Psi^{(i)}$ consists of two parts. First, we build a 2D range tree $\Lambda^{(i)}$ on the non-empty grid cells $\mathsf{C}_1, \ldots, \mathsf{C}_m$ of G_i. The first level of $\Lambda^{(i)}$, built on the x-projections of $\mathsf{C}_1, \ldots, \mathsf{C}_m$, is identical to the 1D range tree described above. Each node u of the first level is associated with an x-interval I_x and a collection C_u of non-empty grid cells. Next, we build a 1D range tree $\Lambda^{(i)}_u$ on the y-projections of the grid cells of C_u and attach it as a second-level tree at u. If a node $w \in \Lambda^{(i)}_u$ is associated with the y-interval I_w, then we also associate it with the rectangle $\square_w = I_u \times I_w$ and the subset $\mathsf{P}^{(i)}_w = \mathsf{P}^{(i)} \cap \square_w$. We compute and store $\mathcal{T}(\mathsf{P}^{(i)}_w)$ at w. Again, $\mathcal{T}(\mathsf{P}^{(i)}_w)$ is computed by brute-force if w is a leaf and using dynamic programming otherwise. The total size of $\Lambda^{(i)}$ is $O(nk \log n \min\{k, \log n\})$, and the time taken to build $\Lambda^{(i)}$ is $n^{O(\varepsilon^{-2})}$. A useful property of $\Lambda^{(i)}$ is that for a rectangle ρ aligned with grid cells of G_i, there are $O(\log^2 n)$ second level nodes $u_1, \ldots, u_t$ in $\Lambda^{(i)}$ such that $\square_{u_1}, \ldots, \square_{u_t}$ are pairwise disjoint and contain all non-empty grid cells of $G_i \cap \rho$, and they can be computed in $O(\log^2 n)$ time.

The second part of the index is a collection of 1D indexes. Fix a non-empty row of G_i, i.e., the strip $\Sigma_\alpha = \mathbb{R} \times [\alpha, \alpha+r]$. For a pair of integers $a < b$ such that $[a/r, b/r] \subseteq [\alpha, \alpha+r]$, i.e., $\mathbb{R} \times [a/r, b/r]$ is the union of a contiguous set of rows of F that lie inside row α of G_i, let $\mathsf{P}^x_{a,b}$ be the x-projection of points in $\mathsf{P}^{(i)}_{a,b} = \mathsf{P}^{(i)} \cap \mathbb{R} \times [a/r, b/r]$, i.e., $\mathsf{P}^x_{a,b} = \{x_q \mid (x_q, y_q) \in \mathsf{P}^{(i)}, y_q \in [a/r, b/r]\}$. For an interval I, let $\mathsf{P}^{(i)}_{a,b}[I] = \{(x_q, y_q) \in \mathsf{P}^{(i)}_{a,b} \mid x_q \in \mathsf{P}^x_{a,b} \cap I\}$. By slightly adapting the 1D index described above we build a 1D index $\Psi^x_{a,b}$ on $\mathsf{P}^x_{a,b}$ so that for a F-aligned interval I, $\mathcal{T}(\mathsf{P}^{(i)}_{a,b}[I])$ can be reported quickly. We build $O(\varepsilon^{-4})$ indexes for the row α of G_i, and repeat this step for all non-empty rows of G_i. Next, we do the same for each non-empty column of G_i, the only difference is that we work with the y-projections of points. Summing over all rows and columns of G_i, the total size of these indexes is $O(nk \min\{k, \log n\})$, and they can be constructed in $n^{O(\varepsilon^{-2})}$ time.

We now describe the query procedure. Let $\rho = I_x \times I_y$, where $I_x = [\alpha_L, \alpha_R]$ and $I_y = [\beta_L, \beta_R]$ be a query rectangle. If ρ is not F-aligned, we set ρ to be the smallest F-aligned rectangle containing ρ. So assume ρ is F-aligned. Suppose the bottom-left vertex (α_L, β_L) of ρ lies in the cell (a_L, b_L) of G_i, and the top-right vertex (α_R, β_R) of ρ lies in the cell (a_R, b_R) of G_i. We partition ρ into (at most) four boundary rectangles $\rho_x^-, \rho_x^+, \rho_y^-, \rho_y^+$ and (at most) one center rectangle ρ_c as follows (see Figure 6): Assume that $a_R > a_L + r$ and $b_R > b_L + r$; the other cases can be handled similarly. Set $\rho_c = [a_L + r, a_R] \times [b_L + r, b_R]$, $\rho_x^- = [\alpha_L, a_L + r] \times [b_L + r, b_R]$, $\rho_x^+ = [a_R, \alpha_R] \times [b_L + r, \beta_R]$, $\rho_y^- = [\alpha_L, \alpha_R] \times [\beta_L, b_L + r]$, $\rho_y^+ = [\alpha_L, \alpha_R] \times [b_R, \beta_R]$.

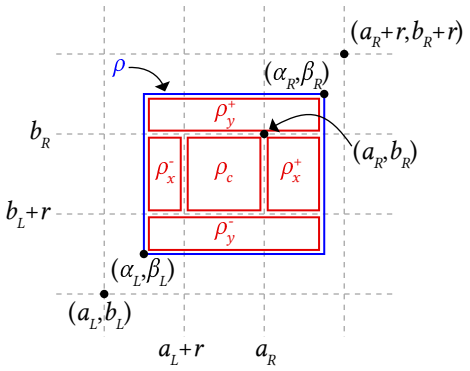

Figure 6. An example of partitioning ρ into four boundary rectangles $\rho_x^-, \rho_x^+, \rho_y^-, \rho_y^+$ and a center rectangle ρ_c.

If ρ_c contains a grid cell of G_i, we query $\Lambda^{(i)}$ with ρ_c and identify $t = O(\log^2 n)$ nodes of the second level trees $u_1, \ldots, u_t$ such that $\square_{u_1}, \ldots, \square_{u_t}$ induce a partition of the non-empty grid cells of ρ_c. Using $\mathcal{T}(\mathsf{P}^{(i)}_{u_1}), \ldots, \mathcal{T}(\mathsf{P}^{(i)}_{u_t})$, we can compute $\mathcal{T}(\mathsf{P}^{(i)} \cap \rho_c)$ in $O(k^2 \log^2 n)$ time. Next, by querying appropriate 1D indexes, we compute $\mathcal{T}(\mathsf{P}^{(i)} \cap \rho_x^-), \mathcal{T}(\mathsf{P}^{(i)} \cap \rho_x^+), \mathcal{T}(\mathsf{P}^{(i)} \cap \rho_y^-), \mathcal{T}(\mathsf{P}^{(i)} \cap \rho_y^+)$ in $O(k^2 \log n)$ time. Indeed, suppose we want to compute $\mathcal{T}(\mathsf{P}^{(i)} \cap \rho_y^-)$. Since ρ is F-aligned, so is ρ_y^-. Let $g = \beta_L \cdot r$ (i.e., $\beta_L = g/r$) and $h = (b_L + r) \cdot r$. Then $\mathsf{P}^{(i)}_{g,h} = \mathsf{P}^{(i)} \cap \mathbb{R} \times [\beta_L, b_L + r]$. Therefore $\mathcal{T}(\mathsf{P}^{(i)} \cap \rho_y^-)$ can be obtained by querying $\mathsf{P}^{x}_{g,h}$ with the interval $I_x = [\alpha_L, \alpha_R]$, which is F-aligned. Using the index $\Psi^{x}_{g,h}$, we obtain $\mathcal{T}(\mathsf{P}^{(i)}_{g,h}[I_x])$ in $O(k^2 \log n)$ time. We can handle other boundary rectangles in a similar manner. Putting everything together, the overall query time is $O(k^2 \log^2 n)$.

Finally, we remark that this index can be extended to $d > 2$ by constructing a d-dimensional grid, a d-dimensional range tree, and a family of $(d-1)$-dimensional range trees. A query rectangle is partitioned into (at most) one center rectangle and $2d$ boundary rectangles – one for each facet of the query rectangle. The preprocessing time, size, and query time in $\mathbb{R}^d$ are $n^{O(\varepsilon^{-d})}$, $O(nk \log^{d-1} n \min\{k, \log n\})$, and $O(k^2 \log^d n)$, respectively.

The modified index can also be easily extended to higher dimensions. Hence, we conclude the following:

Theorem 4.6. *Given a set P of n points in $\mathbb{R}^d$, an integer $k \le n$, a constant $\varepsilon \in (0,1)$, and a parameter $\delta > 0$, an index can be constructed in $n^{O(\varepsilon^{-d})}$ time with $O(nk \log^{d-1} n \min\{k, \log n\})$ space such that for a query rectangle ρ, a δ-diverse k-set $\mathsf{S} \subseteq \mathsf{P} \cap (\rho \oplus [0, \varepsilon\delta]^d)$ with $w(\mathsf{S}) \ge (1-\varepsilon) w(CD(\mathsf{P} \cap \rho, k, \delta))$ can be reported in $O(k^2 \log^d n)$ time.*

Alternatively, an index can be constructed in $n^{O(\varepsilon^{-d})}$ time with $O(nk \log^d n)$ space such that for a query rectangle ρ and integer $k \le n$, a δ-diverse $((1+\varepsilon)k)$-set $\mathsf{S} \subseteq \mathsf{P} \cap (\rho \oplus [0, \varepsilon\delta]^d)$ with $w(\mathsf{S}) \ge (1-\varepsilon) w(CD(\mathsf{P} \cap \rho, k, \delta))$ can be reported in $O(k \log k \log^d n)$ time.

Dependency on parameter ε. For $d = 1$, we have $O(\varepsilon^{-1})$ grids, and the index $\Psi^{(i)}$ constructed for each grid is size $O(nk \min\{k, \log n\} + nk\varepsilon^{-4})$ so the total space is $O(nk\varepsilon^{-1}(\min\{k, \log n\} + \varepsilon^{-4}))$. For $d = 2$, we have $O(\varepsilon^{-2})$ grids. For each grid, we construct the 2D range tree $\Lambda^{(i)}$ of size $O(nk \log n \min\{k, \log n\})$ and $O(\varepsilon^{-4})$ 1D indexes of size $O(nk(\min\{k, \log n\} + \varepsilon^{-4}))$. Hence, the total space is

$$O(nk\varepsilon^{-2}(\log n \min\{k, \log n\} + \min\{k, \log n\}\varepsilon^{-4} + \varepsilon^{-8}))$$

which is bounded by $O(nk\varepsilon^{-2}(\log n \min\{k, \log n\} + \varepsilon^{-8}))$. It can be shown by induction that for any d, our d-dimensional index has size

$$O(nk\varepsilon^{-d}(\log^{d-1} n \min\{k, \log n\} + \varepsilon^{-4d})).$$

The query time can be easily bounded in $O(k^2\varepsilon^{-d} \log^d n)$ time.

5 FUTURE WORK

There are still many interesting open problems to consider for finding diverse high-valued subsets. For example: i) Can we get (near-)linear time $(1-\varepsilon)$-approximation algorithms for the MSD or the MMD problem? Notice that all our algorithms have approximation ratio $0.5 - \varepsilon$, while all previous algorithms with approximation ratio $1 - \varepsilon$ have superlinear running time. Similarly, we can ask if there are near-linear size indexes that return $(1-\varepsilon)$-approximation solutions efficiently. ii) Can we extend our methods from the Euclidean space to metrics with constant doubling dimension? iii) Finally, it remains an open question if the MSD problem is NP-hard in the Euclidean space for constant dimension d.

Acknowledgments. We thank Jie Xue for helpful discussions about the MMD problem.

REFERENCES

[1] M. Abrahamsen, M. de Berg, K. Buchin, M. Mehr, and A. D. Mehrabi. Range-clustering queries. In B. Aronov and M. J. Katz, editors, *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, volume 77 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[2] P. Afshani, G. S. Brodal, and N. Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 390–400, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics.

[3] P. K. Agarwal, J. Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

[4] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry*, 1(4):189–201, 1992.

[5] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[6] S. Aghamolaei, M. Farhadi, and H. Zarrabi-Zadeh. Diversity maximization via composable coresets. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*. Queen's University, Ontario, Canada, 2015.

[7] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry*, 17(3-4):135–152, 2000.

[8] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[9] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[10] S. N. Bespamyatnikh. An optimal algorithm for closest-pair maintenance. *Discrete & Computational Geometry*, 19(2):175–195, 1998.

[11] B. E. Birnbaum and K. J. Goldman. An improved analysis for a greedy remote-clique algorithm using factor-revealing lps. *Algorithmica*, 55(1):42–59, 2009.

[12] A. Borodin, A. Jain, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Trans. Algorithms*, 13(3):41:1–41:25, 2017.

[13] 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.

[14] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. Mapreduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *Proceedings of the VLDB Endowment*, 10(5):469–480, 2017.

[15] A. Cevallos. Approximation algorithms for geometric dispersion. Technical report, EPFL, 2016.

[16] 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. Society for Industrial and Applied Mathematics, 2017.

[17] A. Cevallos, F. Eisenbrand, and R. Zenklusen. An improved analysis of local search for max-sum diversification. *Mathematics of Operations Research*, 44(4):1494–1509, 2019.

[18] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 300–309. ACM, 2000.

[19] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[20] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, Jan. 1991.

[21] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 434–444. ACM, 1988.

[22] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th international conference on World wide web*, pages 381–390. ACM, 2009.

[23] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[24] S. Guha. Tight results for clustering and summarizing data streams. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 268–275, New York, NY, USA, 2009. ACM.

[25] S. Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

[26] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3):133–137, 1997.

[27] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. NC-approximation schemes for NP-and PSPACE-hard problems for geometric graphs. *Journal of algorithms*, 26(2):238–274, 1998.

[28] P. Indyk, S. Mahabadi, M. Mahdian, and V. S. Mirrokni. Composable core-sets for diversity and coverage maximization. In R. Hull and M. Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 100–108. ACM, 2014.

[29] S. Kahruman-Anderoglu. *Optimization in geometric graphs: Complexity and approximation*. Texas A&M University, 2009.

[30] D. B. Lomet and B. Salzberg. The hB-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems (TODS)*, 15(4):625–658, 1990.

[31] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995.

[32] J. Matoušek. Geometric range searching. *ACM Comput. Surv.*, 26(4):422–461, Dec. 1994.

[33] T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Japanese Conference on Discrete and Computational Geometry*, pages 194–200. Springer, 1998.

[34] R. Matthew Mccutchen and S. Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proceedings of the 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*, APPROX '08 / RANDOM '08, pages 165–178, Berlin, Heidelberg, 2008. Springer-Verlag.

[35] Y. Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9:1–9:21, Jan. 2014.

[36] D. R. PW and P. Elzbieta. *Dissimilarity Representation For Pattern Recognition, The: Foundations And Applications*, volume 64. World scientific, 2005.

[37] S. Rahul, P. Gupta, R. Janardan, and K. Rajan. Efficient top-k queries for orthogonal ranges. In *International Workshop on Algorithms and Computation*, pages 110–121. Springer, 2011.

[38] S. Rahul and Y. Tao. On top-k range reporting in 2d space. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '15, pages 265–275, New York, NY, USA, 2015. ACM.

[39] 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.

[40] C. Sheng and Y. Tao. Dynamic top-k range reporting in external memory. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '12, pages 121–130, New York, NY, USA, 2012. ACM.

[41] M. Sydow. Approximation guarantees for max sum and max min facility dispersion with parameterised triangle inequality and applications in result diversification. 2014.

[42] A. Tamir. Obnoxious facility location on graphs. *SIAM J. Discrete Math.*, 4(4):550–567, 1991.

[43] Y. Wang, A. Meliou, and G. Miklau. Rc-index: Diversifying answers to range queries. *Proceedings of the VLDB Endowment*, 11(7):773–786, 2018.

[44] K. Yi, L. Wang, and Z. Wei. Indexing for summary queries: Theory and practice. *ACM Trans. Database Syst.*, 39(1):2:1–2:39, Jan. 2014.

A MISSING PROOFS FROM SUBSECTION 2.2

Correctness of our algorithm. We will prove our algorithm maintains the following invariants immediately before reading the t-th point in the stream for each $i \in \{1, 2, \ldots, J\}$:

(1) $r_i = \alpha^{m_i}(1+\varepsilon)^i \mu(\mathsf{P}_k)$ for some integer m_i.
(2) $(\mathsf{A}_i \cup \mathsf{N}_i)$ is a (r_i, ε)-packing of P_{t-1} such that $|\mathsf{A}_i \cup \mathsf{N}_i| < k$.
(3) If $t > k$, then $|\mathsf{A}_i \cup \mathsf{D}_i| = k$ and $\mu(\mathsf{A}_i \cup \mathsf{D}_i) \geq r_i/\alpha$.

Suppose $\mathsf{P}_{t-1} = \langle p_1, \ldots, p_{t-1} \rangle$ have been processed by the algorithm and the invariants above hold. Let $p = p_t$ be the next point to be inserted, and fix $i \in \{1, 2, \ldots, J\}$. The first step of the update procedure is to compute a closest pair x, y in $\mathsf{A}_i \cup \mathsf{N}_i \cup \{p\}$. If $p \in \{x, y\}$ and $\|x - y\| \leq r_i$, then the update procedure terminates with all invariants held. Otherwise, $p \notin \{x, y\}$, or $p = x$, without loss of generality, and $\|p - y\| > r_i$. In the former case, $\min_{q \in \mathsf{A}_i \cup \mathsf{D}_i} \|p - q\| \geq \|x - y\| \geq r_i$ by the second invariant. Thus, in either case, we have $\mu(\mathsf{A}_i \cup \mathsf{N}_i \cup \{p\}) \geq r_i$, so N_i is set to $\mathsf{N}_i := \mathsf{N}_i \cup \{p\}$ during the update. For the new value of N_i, $\mathsf{A}_i \cup \mathsf{N}_i$ is a (r_i, ε)-packing of $\mathsf{P}_t = \mathsf{P}_{t-1} \cup \{p\}$. In the case that $|\mathsf{A}_i \cup \mathsf{N}_i| < k$, the procedure then stops with all invariants held.

When $|\mathsf{A}_i \cup \mathsf{N}_i| = k$ after inserting p to N_i, the update proceeds by increasing r_i by a factor of α, after which the the first invariant holds. For purposes of analysis, denote by $\widehat{r_i}$ value of r_i before the increase; that is, the value of r_i at the beginning of the entire update procedure for point p. After increasing r_i , we have $\widehat{r_i} \leq r_i/\alpha \leq \mu(\mathsf{A}_i \cup \mathsf{N}_i) < r_i$. Then A_i is set to $\mathsf{A}_i \cup \mathsf{N}_i$, D_i is emptied, and the pruning begins. In each iteration, a closest pair x, y in A_i is found and at least one of x, y is deleted from A_i. Since every point deleted from A_i is added to D_i, during and after the pruning $\mathsf{A}_i \cup \mathsf{D}_i$ is equal to the value of $\mathsf{A}_i \cup \mathsf{N}_i$ immediately after r_i was increased. It follows that $\mu(\mathsf{A}_i \cup \mathsf{D}_i) \geq r_i/\alpha$ and thus the third invariant holds when the pruning is complete. It remains to show the second invariant holds when the pruning is complete.

Since r_i is increased by the smallest factor of α such that $\mu(\mathsf{A}_i) < r_i$, at least one iteration of the pruning is performed and thus at least one point is removed from A_i. This ensures that $|\mathsf{A}_i| < k$ after the first iteration, and thus it holds when the pruning is complete. For the second invariant to hold, it only remains to show A_i is a (r_i, ε)-packing of P_t in the end. Recall that a set $I \subseteq \mathsf{A}_i$ is maintained during the pruning which is initially set to $\varnothing$. See that any point q deleted from A_i during the pruning is found to be within r_i distance of a point $p \in \mathsf{A}_i \cap I$ and that no point in I is deleted from A_i. Thus, any deleted point is within r_i distance of a point in A_i when the pruning is complete; that is, $\psi(\mathsf{D}_i, \mathsf{A}_i) \leq r_i$. As mentioned earlier, $\mathsf{A}_i \cup \mathsf{D}_i$ immediately after the pruning is the same as $\mathsf{A}_i \cup \mathsf{N}_i$ immediately before r_i was increased (from value $\widehat{r_i}$), and at that time, $\mathsf{A}_i \cup \mathsf{N}_i$ was a $(\widehat{r_i}, \varepsilon)$-packing of P_t. Since $\widehat{r_i} \leq r_i/\alpha$, we have $\psi(\mathsf{P}_t, \mathsf{A}_i \cup \mathsf{D}_i) \leq (1+\varepsilon) r_i/\alpha$ when the pruning stops. It follows that $\psi(\mathsf{P}_t, \mathsf{A}_i) \leq \psi(\mathsf{P}_t, \mathsf{A}_i \cup \mathsf{D}_i) + \psi(\mathsf{D}_i, \mathsf{A}_i) \leq (1+\varepsilon) r_i$ by the triangle inequality and fact that $1/\alpha < \varepsilon/(1+\varepsilon)$. Finally, since $\mu(\mathsf{A}_i) \geq r_i$ when the pruning is complete, we conclude $\mathsf{A}_i = \mathsf{A}_i \cup \mathsf{N}_i$ is a (r_i, ε)-packing of P_t of size less than k. Thus, the second invariant is met. We conclude all invariants are held when the update is complete.

Next we prove the approximation guarantee of $0.5 - \varepsilon$. Consider any $t \leq n$ and the values of the radii r_i and subsets $\mathsf{S}_t, \mathsf{T}_t$ after processing the t-th point. First, see that $r_x := \max_i r_i$ and $r_y := \min_i r_i$ are such that $r_x \geq (1+\varepsilon)^{J-1} r_y = \alpha r_y/(1+\varepsilon)$ by the first invariant and facts that $\alpha = (1+\varepsilon)^J$ and $i \in \{1, 2, \ldots, J\}$ for all radii r_i. By the third invariant, $\mathsf{T}_x = \mathsf{A}_x \cup \mathsf{D}_x$ is such that $\mu(\mathsf{T}_x) \geq r_x/\alpha \geq r_y/(1+\varepsilon)$. Furthermore, by the second invariant, $\mathsf{S}_y = \mathsf{A}_y \cup \mathsf{N}_y$ is so that $\psi(\mathsf{S}_y, \mathsf{P}_t) \leq (1+\varepsilon) r_y$, and thus we conclude $\mu(\mathsf{T}_x) \geq (1+\varepsilon)^{-2} \psi(\mathsf{P}_t, \mathsf{T}_x) \geq 0.5(1+\varepsilon)^{-2} \mu(RE(\mathsf{P}_t))$. Choosing $\varepsilon := \varepsilon/5$ at the beginning of the algorithm results in an approximation factor of $(0.5 - \varepsilon)$ as desired.

B FASTER INDEXES FOR FIXED ε

We discuss the index for the remote-clique range queries for fixed ε. In the end we extend it to handle MSD range queries.

Preprocessing. We construct an ε-net N. For each $u \in \mathsf{N}$ we compute the projection of each point $p \in \mathsf{P}$ on u, $\langle p, u \rangle$ and we set $w(p) = \langle p, u \rangle$. Then we construct the index $\Phi_u(\mathsf{P})$ on the weighted point set P , where the weights are the inner products we computed. In the end of the preprocessing phase we have built $O(1)$ indexes Φ. In total, our index uses $O(n \log^d n)$ space and can be constructed in $O(n \log^d n)$ time.

Query algorithm. The query procedure is similar to the algorithm in Theorem 2.2. The only difference is that we query Φ_u to get the next point in $\mathsf{P} \cap \rho$ with the largest score (projection) on a direction u and Φ_{-u} to get the point in $\mathsf{P} \cap \rho$ with the largest score (projection) on direction $-u$. Each time that we get the point with the next largest score on a direction u we check if it is already added in S and if this is true then we ask for the next point with the largest score on u.

Analysis. In order to initialize the points with the largest scores on each direction in N we spend $O(\log^{d-1} n)$ time. Then notice that we may ask at most $O(k)$ times for the point with the next largest score, so using the index from [1] we know that we spend $O(k \log\log n)$ in total to get the points with the largest scores after the initialization. Finally, notice that we may visit all vectors in N at most k times so in total our query procedure takes $O(\log^{d-1} n + k \log\log n)$ time.

The correctness follows from the correctness proof of Theorem 2.2, since at each iteration we find the farthest pair using the polygonal metric.

We can also use that index for the MSD range queries. Unfortunately, we cannot compute the scores of points if we do not know k, λ in the preprocessing phase. However, we can get a set S_1 using the procedure above to approximate the remote-clique and then by constructing an index Φ over the weighted points we can find the k points S_2 with the largest weights among the points in $\mathsf{P} \cap \rho$. Then we compare $\pi(\mathsf{S}_1)$ with $\lambda w(\mathsf{S}_2)$ and we return S_1 if the former is larger than the latter, and we return S_2, otherwise. We can observe that in any case if S is the set we return then $f(\mathsf{S}) \geq (0.25 - \varepsilon) f(MS(\mathsf{P} \cap \rho, k))$. It remains to show how to compute $\pi(\mathsf{S}_1)$ and $\lambda w(\mathsf{S}_2)$, given the sets $\mathsf{S}_1, \mathsf{S}_2$, in order to compare them. It is trivial to compute $\lambda w(\mathsf{S}_2)$ in $O(k)$ time by taking the sum of the weights of the points in S_2. We can also trivially compute $\pi(\mathsf{S}_1)$ by summing up all the pairwise distances in $O(k^2)$ time. However, we would like to have query time which is linear on k. We approximate $\pi(\mathsf{S}_1)$ within an ε factor using WSPD's.

We first give the definition of the WPSD's. Given a set of n points P and a separator parameter ε, in [13] the authors show that there exists a set $W = \{(A_1, B_1), \ldots, (A_s, B_s))\}$ of $s = O(\frac{n}{\varepsilon^d})$

pairs such that $A_i, B_i \subset \mathsf{P}$, $A_i \cap B_i = \emptyset$, $\bigcup_{i=1}^{s} A_i \times B_i = \mathsf{P} \times \mathsf{P}$, $\max\{diam(A_i), diam(B_i)\} \leq \varepsilon \min_{p\in A_i, q\in B_i} ||p-q||$, and for any pair of points $p, q \in \mathsf{P}$ there exists a unique pair A_j, B_j such that $p \in A_j$ and $q \in B_j$ (or $p \in B_j$ and $q \in A_j$). Such a pair decomposition is called $\frac{1}{\varepsilon}$-WSPD and an implicit representation of it can be constructed in $O(\frac{n}{\varepsilon^d} + n \log n)$ time [13, 25]. In particular, a set of pair of representative points (a_i, b_i) where $a_i \in A_i$, and $b_i \in B_i$ for each i, can be computed, such that for any pair of points $p \in A_i$ and $q \in B_i$, $(1-\varepsilon)||p-q|| \leq ||a_i - b_i|| \leq \frac{||p-q||}{1-\varepsilon}$. In addition the cardinality of the sets A_i, B_i can also be derived using the construction algorithm in [13, 25].

By constructing an $\frac{1}{\varepsilon}$-WSPD $W = \{(a_1, b_1), \ldots, (a_s, b_s))\}$ over the set S_1 we compute $\tau = \sum_{i=1}^{s} ||a_i - b_i|| \cdot |A_i| \cdot |B_i|$. We can easily see that $(1-\varepsilon)\pi(\mathsf{S}_1) \leq \tau \leq \frac{1}{1-\varepsilon}\pi(\mathsf{S}_1)$. Hence, by comparing τ with $\lambda w(\mathsf{S}_2)$ and by setting $\varepsilon \leftarrow \varepsilon/c$ for a sufficiently large constant c we get that, if $\tau \geq \lambda w(\mathsf{S}_2)$ then $f(\mathsf{S}_1) \geq (0.25-\varepsilon) f(MS(\mathsf{P} \cap \rho, k))$, and if $\tau < \lambda w(\mathsf{S}_2)$ then $f(\mathsf{S}_2) \geq (0.25-\varepsilon) f(MS(\mathsf{P} \cap \rho, k))$.

We conclude with the next theorem.

Theorem B.1. *Given a set P of n points in $\mathbb{R}^d$ and a constant $\varepsilon \in (0, 0.25)$, an index can be constructed in $O(n \log^d n)$ time with $O(n \log^d n)$ space such that given the parameters k and λ, and a rectangle ρ, a set $\mathsf{S} \subseteq \mathsf{P} \cap \rho$ of k points can be found in $O(\log^{d-1} + k \log\log n)$ time such that $f(\mathsf{S}) \geq (0.25-\varepsilon) f(MS(\mathsf{P} \cap \rho, k))$. The same index can also guarantee a set $\mathsf{S} \subseteq \mathsf{P} \cap \rho$ of k points such that $f(\mathsf{S}) \geq (0.5-\varepsilon) f(RC(\mathsf{P} \cap \rho))$, where $RC(\mathsf{P} \cap \rho)$ is the optimum solution of the MSD problem with $w(p) = 0$, for each $p \in \mathsf{P}$.*

C A MODIFIED INDEX

Here we prove Theorem 4.5. Recall the construction of the index described in Theorem 4.4. Let $r, G_0, \ldots, G_{r-1}, F$, and $\mathsf{P}^{(0)}, \ldots, \mathsf{P}^{(r-1)}$ be defined as before. For each $i \leq t$, let $\Psi^{(i)}$ be the index on $\mathsf{P}^{(i)}$. These indexes are composed of two parts. The first part is a 1D range tree $\Lambda^{(i)}$ that stores $\mathcal{T}(\mathsf{P}_u^{(i)})$ for each node of u where $\mathsf{P}_u^{(i)}$ is a subset of $\mathsf{P}^{(i)}$ associated with node u. The second part is a collection of tables constructed for each non-empty grid cell C_t of G_i: for each pair of integers $a, b \in \mathsf{C}_t$, we store $\mathcal{T}(\mathsf{P}^{(i)} \cap [a, b])$.

To modify the index, we first build $\Psi^{(i)}$ as before but with n as the role of k. Then, for any table $\mathcal{T}(\mathsf{A})$ stored for a set $\mathsf{A} \subseteq \mathsf{P}^{(i)}$, we instead store the subset $\mathcal{T}^{\varepsilon}(\mathsf{A}) = \{T_{\lfloor(1+\varepsilon)^i\rfloor}(\mathsf{A}) \mid i \in \mathbb{Z}. 0 \leq i \leq \lceil \log_{1+\varepsilon} |\mathsf{A}| \rceil\}$ of $\mathcal{T}(\mathsf{A})$.

The preprocessing time for $\Psi^{(i)}$ remains the same, $n^{O(\varepsilon^{-1})}$. By definition, the size of $\mathcal{T}^{\varepsilon}(\mathsf{A})$ is $\sum_{i=0}^{\lceil \log_{1+\varepsilon} |\mathsf{A}| \rceil} (1+\varepsilon)^i = O(|\mathsf{A}|/\varepsilon) = O(|\mathsf{A}|)$. A point of $\mathsf{P}^{(i)}$ is stored in at most one node for any fixed level of $\Lambda^{(i)}$, so the size of all tables associated with nodes of a fixed level of $\Lambda^{(i)}$ is $O(n)$. $\Lambda^{(i)}$ has $O(\log n)$ levels, so the size of $\Lambda^{(i)}$ is $O(n \log n)$. Now consider a non-empty grid cell C_t of G_i. The size of each table $\mathcal{T}^{\varepsilon}(\mathsf{P}^{(i)} \cap [a, b])$ for each pair of integers $a, b \in \mathsf{C}_t$ is bounded by $O(\varepsilon^{-4} |\mathsf{P}^{(i)} \cap \mathsf{C}_t|)$. The total size of these tables is $O(n)$. Hence, the size of $\Psi^{(i)}$ is $O(n \log n)$.

Next, we describe the modified query procedure. Let I be a query interval and $k \leq n$ be an integer. If I is not F-aligned, we extend I to the smallest F-aligned interval containing I, so let us assume that I is F-aligned. We do the following for each grid G_i. As before, we partition I into three sub-intervals: $I_L \cup I_M \cup I_R$. For I_M, we identify $s = O(\log n)$ nodes $u_1, \ldots, u_s$ of $\Lambda^{(i)}$ such that $\sigma_{u_1}, \ldots, \sigma_{u_s}$ partition I_M. From these nodes we obtain $\mathcal{T}^{\varepsilon}(\mathsf{P}_{u_t}^{(i)})$ for $t \leq s$. From $\Psi^{(i)}$ we also obtain $\mathcal{T}^{\varepsilon}(\mathsf{P}^{(i)} \cap I_L)$ and $\mathcal{T}^{\varepsilon}(\mathsf{P}^{(i)} \cap I_R)$. These intervals with I_L and I_R make $m = O(\log n)$ intervals $I_1, \ldots, I_m$ which partition I. For each interval I_t, we have $\mathcal{T}^{\varepsilon}(\mathsf{P}^{(i)} \cap I_t)$. Like before, we use dynamic programming to compute a CDT solution S using these tables. The details are as follows.

Define the recurrence $R(\ell, t)$ for non-negative integers ℓ, t where $\ell \leq n$ and $0 < t \leq m$ as

$$R(\ell, t) = \max \begin{cases} R(\ell, t-1) \\ \max\limits_{0 \leq j \leq \log_{1+\varepsilon} \ell} \; R\big(\ell - \lfloor (1+\varepsilon)^j \rfloor, t-1\big) + \\ \qquad\qquad w(CD\big(\mathsf{P}^{(i)} \cap I_t, \lfloor (1+\varepsilon)^j \rfloor\big)) \end{cases}$$

and $R(\ell, 0) = 0$. A simple inductive argument shows that $R(\ell, t)$ is the weight of the best CDT solution S in $\mathsf{P}^{(i)} \cap \bigcup_{x \leq t} I_x$ of size at most ℓ where $\mathsf{S} \cap I_x$ is $\varnothing$ or $CD\big(\mathsf{P}^{(i)} \cap I_x, \lfloor (1+\varepsilon)^{j_x} \rfloor\big)$ for some integer j_x. $R(\ell, t)$ can be computed in $O(\ell t \varepsilon^{-1} \log \ell) = (\ell t \log \ell)$ time since the tables $\mathcal{T}^{\varepsilon}(\mathsf{P}^{(i)} \cap I_t)$ are at our disposal.

Using the above, we compute $R(\lfloor (1+\varepsilon)k \rfloor, m)$ in $O(k \log n \log k)$ time. With standard backtracking techniques, we also identify the corresponding solution S with $w(\mathsf{S}) = R(\lfloor (1+\varepsilon)k \rfloor, m)$ in the same time.

We claim $w(\mathsf{S}) \geq w(CD(\mathsf{P}^{(i)} \cap I, k))$. First, let $\mathsf{S}_t^* = I_t \cap CD(\mathsf{P}^{(i)} \cap I)$ and let $k_t = |\mathsf{S}_t^*|$ for all $t \leq m$. Let $\widehat{k_t} = \left\lfloor (1+\varepsilon)^{\lceil \log_{1+\varepsilon} k_t \rceil} \right\rfloor$. Clearly $k_t \leq \widehat{k_t} \leq \lfloor (1+\varepsilon)k_t \rfloor$, and hence $\sum_t \widehat{k_t} \leq \lfloor (1+\varepsilon)k \rfloor$. By the properties of $R(\ell, t)$ above, we have

$$\begin{aligned} w(S) &= R(\lfloor (1+\varepsilon)k \rfloor, m) \\ &\geq \sum_t w(CD(\mathsf{P}^{(i)} \cap I_t, \widehat{k_t})) \\ &\geq \sum_t w(\mathsf{S}_t^*) \\ &\geq w(CD(\mathsf{P}^{(i)} \cap I, k)). \end{aligned}$$

We conclude S is such that $|\mathsf{S}| \leq (1+\varepsilon)k$ and $w(\mathsf{S}) \geq w(CD(\mathsf{P}^{(i)} \cap I, k))$.

Repeating this for all r grids and returning the best solution among them, we obtain a feasible subset of at most $(1+\varepsilon)k$ points whose weight is at least $(1-\varepsilon)w(CD(\mathsf{P} \cap I, k))$. Putting everything together, we conclude with Theorem 4.5.