On Finding Rank Regret Representatives

Selecting the best items in a dataset is a common task in data exploration. However, the concept of "best" lies in the eyes of the beholder: different users may consider different attributes more important and, hence, arrive at different rankings. Nevertheless, one can remove "dominated" items and create a "representative" subset of the data, comprising the "best items" in it. A Pareto-optimal representative is guaranteed to contain the best item of each possible ranking, but it can be a large portion of data. A much smaller representative can be found if we relax the requirement of including the best item for each user and, instead, just limit the users' "regret". Existing work defines regret as the loss in score by limiting consideration to the representative instead of the full dataset, for any chosen ranking function.

However, the score is often not a meaningful number, and users may not understand its absolute value. Sometimes small ranges in score can include large fractions of the dataset. In contrast, users do understand the notion of rank ordering. Therefore, we consider items' positions in the ranked list in defining the regret and propose the *rank-regret representative* as the minimal subset of the data containing at least one of the top-k of any possible ranking function. This problem is polynomial time solvable in 2D space but is NP-hard on 3 or more dimensions. We design a suite of algorithms to fulfill different purposes, such as whether relaxation is permitted on k, the result size, or both, whether a distribution is known, whether theoretical guarantees or practical efficiency is important, etc. Experiments on real datasets demonstrate that we can efficiently find small subsets with small rank-regrets.

CCS Concepts: • **Information systems → Top-k retrieval in databases**.

ACM Reference Format:
. 2021. On Finding Rank Regret Representatives. 1, 1 (March 2021), 34 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Given a dataset with multiple attributes, it is a challenge to combine the values of multiple attributes to arrive at a rank. In many applications, especially in databases with numeric attributes, a weight vector $\boldsymbol{w}$ is used to express user preferences through a linear combination of the attributes (i.e., $\sum_i \boldsymbol{w}[i]A_i$). Finding flights based on a linear combination of criteria such as price and duration [10], diamonds based on depth and carat [31], and houses based on price and floor area [31] are a few examples.

The difficulty is that the concept of "best" lies in the eyes of the beholder. Various users may consider different attributes more important and, hence, arrive at very different rankings. In the absence of explicit user preferences, the system can remove dominated items and offer the remaining Pareto-optimal [9] set as representing the desirable items in the dataset. Such a skyline (or the set of convex hull points) is the smallest subset of the data that is guaranteed to contain the top choice of a user based on any monotonic (or linear, resp.) ranking function. Since the introduction of skylines to the database community [11], a large body of work has been conducted in this area.

Author's address:

XXXX-XXXX/2021/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

A major issue with such representatives is that they can be a large portion of the dataset [7, 33], especially when there are multiple attributes. Hence, several researchers have tackled [14, 54] the challenge of finding a small subset of the data for further consideration.

One elegant way to find a smaller subset is to define the notion of *regret* for any particular user. That is, how much this user loses by restricting consideration only to the subset rather than the whole set. The goal is to find a small subset of the data such that this regret is small for every user, no matter what their preference functions are. There has been considerable attention given to the *regret-ratio minimizing set* [7, 46] problem and its variants [4, 13, 18, 37, 38, 45, 56]. Let m_{all} be the maximum score of the objects in dataset based on a scoring function f. Also, let m_{sub} be the maximum score for a subset of data. The regret-ratio of the subset for f is $(m_{all} - m_{sub})/m_{all}$. The classic regret-ratio minimizing set problem aims to find a subset of size r that minimizes the maximum regret-ratio for any possible function. Other variations of the problem will be pointed out in later sections.

Unfortunately, in most real situations, the actual score is a "made up" number with no direct significance. This is especially true when attribute values are drawn from different domains. In fact, the score itself could also be on a made-up scale. Considering the regret as a ratio helps, but is far from being a complete solution. For example, wine ratings are on a 100 point scale, with the best wines in the high 90s. However, wines rated below 80 almost never make it to a store. Let us say that the best wine in some dataset is 90 points. A regret of 3 points gives a very small regret ratio of 0.03, but actually only promises a wine with a rating of 87, which is below median! In other words, a small value of regret ratio can actually result in a large swing in rank. In the case of wines at least the rating scales see enough use such that most wine-drinkers would have a sense of what a score means. But consider choosing a hotel. If a website takes your preferences into account and scores a hotel at 17.2, do you know what that means? If not, then how can you meaningfully specify a regret ratio?

Although ordinary users may not have a good sense of actual scores, they almost always understand the notion of rank. Therefore, as an alternative to the regret-ratio, we consider items' positions in the ranked list and propose the *rank-regret* measure to quantify an item's distance from the top of the list. We define the *rank-regret* of a subset of the data to be k, if it contains at least one of the top-k objects of any possible ranking function.

Since items in a dataset are usually not uniformly distributed by score, solutions that minimize regret-ratio do not typically minimize rank-regret. In this paper, we seek to find the smallest subset of the given dataset that has a *rank-regret* of k. We call this subset a *k-rank-regret representative* of the database. The 1-rank-regret representative of a database (for linear ranking functions) comprises the points on the convex hull: guaranteed to contain the top choice of any linear ranking function. The number of points on the convex hull is usually very large: almost the entire dataset when there are five or more dimensions [7, 33]. By choosing a value of k larger than 1, we can drastically reduce the size of the rank-regret representative set, while guaranteeing everyone has a choice in their top-k even if not the absolute top choice.

Contributions. The following is a summary of our contributions:

(1) We propose the rank-regret representative as a way of choosing a small subset of the database guaranteed to contain at least one good choice for every user.

dimension	max rank regret	representative size	time	source
any fixed d	k	$O(\frac{n}{k}\log n)$	$O(\frac{n}{k}\log n)$	Theorem 4
2	k	OPT	$\tilde{O}(nk)$	Theorem 7
2	$(2+\delta)k$	$\leq OPT$	$O(n\log n)$	Theorem 12
3	$(1+\delta)k$	$\tilde{O}(OPT)$	$\tilde{O}(nk^2)$	Theorem 13
3	k	$\tilde{O}(OPT)$	$O(nk^{5/2})$	Theorem 15
$d \geq 4$	k	$\tilde{O}(OPT)$	$O(n^{\lfloor d/2\rfloor}k^{\lceil d/2\rceil+1})$	Theorem 15
$d \geq 3$	k	OPT	NP-hard	Section 4.1
$d \geq 4$	$\tilde{O}(k)$	$\tilde{O}(OPT)$	no fixed-para near-linear algorithms (see Section 5.4)	Theorem 14

Note 1: n is the dataset size, d is the dimensionality, and OPT is the minimum size of k-rank-regret representatives.
Note 2: $\tilde{O}(.)$ hides a polylog n factor.

Table 1. Summary of formal results

(2) We establish its connection to the notion of *ϵ-net* in computational geometry and initialize the study of *instance optimal* ϵ-nets.
(3) We give an algorithm to find an optimal k-rank-regret representative in 2D space efficiently.
(4) When the dimensionality is 3 or above, finding an optimal k-rank-regret representative is NP-hard. We present polynomial time algorithms for discovering near-optimal rank regret representatives under different approximation schemes. We also formally separate the case of $d \leq 3$ (where d is the dimensionality) from $d \geq 4$ in terms of what type of polynomial efficiency is achievable.
(5) We design a space partition algorithm that returns a k-rank-regret representative of a small size based on a non-trivial *rank-sum lemma*.
(6) We develop a randomized algorithm that utilizes the knowledge of query distribution to find a k-rank-regret representative with probabilistic guarantees.
(7) We conduct extensive experimental evaluation based on real datasets to verify the effectiveness and efficiency of our techniques.

Table 1 gives an overview of the formal results in this paper.

A short version of this paper appeared in [1]. Compared to that preliminary work, the current paper presents a more comprehensive treatment of the k-rank-regret problem. The new contributions include items (2), (3), new 2D and 3D approximations algorithms in item (4), a more powerful rank-sum lemma in item (5), and experimentation with all the new algorithms.

The rest of the paper is organized as follows. Section 2 formally defines the k-rank-regret problem in the primal and dual spaces. Section 3 clarifies its relevance to ϵ-nets. Section 4 settles the problem optimally in 2D space and proves the NP-hardness on $d \geq 3$. Section 5 presents a systematic study on approximation algorithms. Section 6 introduces our space partitioning algorithm, while Section 7 leverages a query distribution to discover a good solution. Section 8 evaluates our algorithms with extensive experiments. Section 9 reviews the previous work directly related to ours. Finally, Section 10 concludes the paper with a summary of findings.

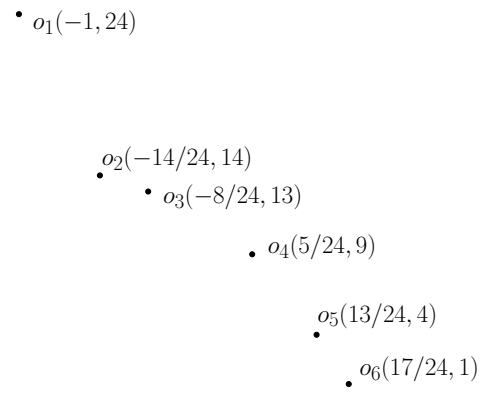

Fig. 1. The input set P for our running example

2 PROBLEM DEFINITIONS

Section 2.1 will first formulate k-rank-regret representative as a problem on multidimensional points. Section 2.2 will then present an equivalent formulation that redefines the problem on multidimensional planes. Our technical discussion will switch between the two formulations, depending on which is easier to work with in a specific context.

2.1 Formulation in the Primal Space

Define P as a set of points in $\mathbb{R}^d$, where $d \geq 2$ is a constant integer. We will refer to each point in P as an *object*, reserving the term "point" for general points in $\mathbb{R}^d$; for the same reason, we reserve the symbol o for objects and p for general points in $\mathbb{R}^d$. For a point $p \in \mathbb{R}^d$, $p[i]$ $(1 \leq i \leq d)$ denotes its coordinate on dimension i. We will sometimes treat p as a d-dimensional vector $\boldsymbol{p} = (\boldsymbol{p}[1], ..., \boldsymbol{p}[d])$ where $\boldsymbol{p}[i] = p[i]$.

A *weight vector* is a d-dimensional vector $\boldsymbol{w} = (\boldsymbol{w}[1], ..., \boldsymbol{w}[d])$ where $\boldsymbol{w}[i] \geq 0$ for each $i \in [1, d]$. The $\boldsymbol{w}$*-score* of an object $o \in P$ is the dot product $\boldsymbol{w} \cdot \boldsymbol{o}$. The $\boldsymbol{w}$*-rank* of o, denoted as $\text{rank}_{\boldsymbol{w}}(o)$, equals r if exactly $r-1$ objects in P have higher $\boldsymbol{w}$-scores than o. The t*-set of* $\boldsymbol{w}$, denoted as $P_{\boldsymbol{w}}(t)$, contains the objects in P with $\boldsymbol{w}$-ranks 1, 2, ..., t, respectively. Denote by $\mathscr{W}$ the set of all possible weight vectors.

For a non-empty subset $S \subseteq P$, define its $\boldsymbol{w}$*-rank regret* under a $\boldsymbol{w} \in \mathscr{W}$ as

$$RR_{\boldsymbol{w}}(S) \quad = \quad \min_{o \in S} \text{rank}_{\boldsymbol{w}}(o)$$

and its *maximum rank regret* as

$$\text{MRR}(S) \quad = \quad \max_{\boldsymbol{w} \in \mathscr{W}} RR_{\boldsymbol{w}}(S). \tag{1}$$

S is a k*-rank-regret representative* of P if $\text{MRR}(S) \leq k$. The problem studied in this paper is:

Problem 1 (k-Rank Regret Representative Problem). *Given a set P of n points and an integer $k \in [1, n]$, find a k-rank-regret representative of P with the smallest size.*

Example: Figure 1 shows a set P of 6 objects in 2D space. Consider the weight vector $\boldsymbol{w} = (20, 1)$. The $\boldsymbol{w}$-ranks of o_6, o_5, o_4, o_3, o_1, and o_2 are 1, 2, 3, 4, 5, and 6, respectively. The 3-set of $\boldsymbol{w}$ is $\{o_6, o_5, o_4\}$. Let $S = \{o_3, o_4\}$. Its $\boldsymbol{w}$-rank regret $RR_{\boldsymbol{w}}(S) = 3$. Later, we will see that $\text{MRR}(S) = 3$, i.e., S is a

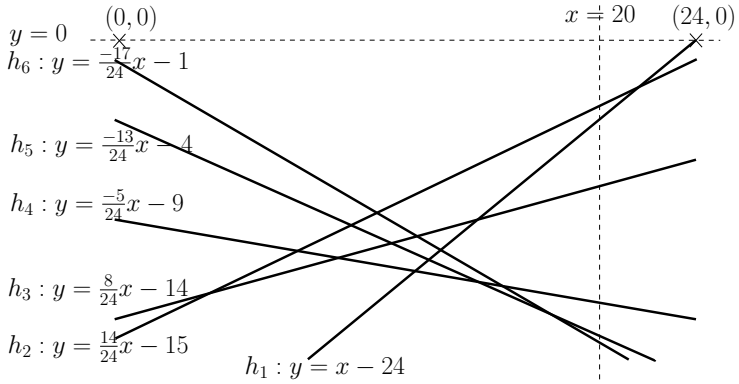

Fig. 2. The dual lines in 2D space for the input P in Figure 1

3-rank-regret representative of P. Furthermore, P admits no smaller 3-rank-regret representatives. □

2.2 Formulation in the Dual Space

Next, we provide another formulation under the *point-plane duality* transformation [21] and establish its equivalence to Problem 1. Define

$$\mathscr{W}_{[d]\neq 0} = \{\boldsymbol{w} \in \mathscr{W} \mid \boldsymbol{w}[d] \neq 0\}.$$

Compared to $\mathscr{W}$, $\mathscr{W}_{[d]\neq 0}$ leaves out the weight vectors $\boldsymbol{w}$ with $\boldsymbol{w}[d] = 0$ that turn out to be unimportant:

Lemma 1. *For any $S \subseteq P$, MRR(S) (defined in (1)) is exactly* $\max_{\boldsymbol{w}\in\mathscr{W}_{[d]\neq 0}} RR_{\boldsymbol{w}}(S)$.

The proof can be found in the appendix. Define

$$\mathscr{W}_{[d]=1} = \{\boldsymbol{w} \in \mathscr{W} \mid \boldsymbol{w}[d] = 1\}.$$

For any object, its $\boldsymbol{w}$-rank remains the same when $\boldsymbol{w}$ is scaled by a positive factor. Thus, every $\boldsymbol{w} \in \mathscr{W}_{[d]\neq 0}$ has the same k-set as $\boldsymbol{w'} = (\frac{\boldsymbol{w}[1]}{\boldsymbol{w}[d]}, ..., \frac{\boldsymbol{w}[d-1]}{\boldsymbol{w}[d]}, 1)$. Hence, it suffices to consider only $\mathscr{W}_{[d]=1}$, where the weight vectors are said to be *canonical* henceforth.

Under point-plane duality, each object $o = (o[1], ..., o[d])$ in P defines a *dual plane*

$$\boldsymbol{x}[d] \quad = \quad \left(\sum_{i=1}^{d-1}(-o[i]) \cdot \boldsymbol{x}[i]\right) - o[d] \tag{2}$$

in the *dual space* $\mathbb{R}^d$ (the plane includes all the points $\boldsymbol{x}$ in $\mathbb{R}^d$ satisfying the equation). Denote by H the set of n dual planes obtained from P in this manner.

Example: Suppose that P is the set of points $o_1, o_2, ..., o_6$ in Figure 1. Figure 2 shows the corresponding dual planes (which are lines in 2D) $h_1, h_2, ...,$ and h_6, which constitute the set H. □

Define the *query space* $\mathscr{Q}$ as the set of all possible $(d-1)$-dimensional vectors $\boldsymbol{q} = (\boldsymbol{q}[1], ..., \boldsymbol{q}[d-1])$ satisfying $\boldsymbol{q}[i] \geq 0$ for all $i \in [1, d-1]$. Each *query* $\boldsymbol{q} \in \mathscr{Q}$ determines a line $\ell_{\boldsymbol{q}}$ in the dual space $\mathbb{R}^d$ that is parallel to dimension d and passes the point $(\boldsymbol{q}[1], ..., \boldsymbol{q}[d-1], -\infty)$. Consider the n intersections between $\ell_{\boldsymbol{q}}$ and the planes in H. For each plane $h \in H$, define its $\boldsymbol{q}$*-rank*, denoted

as $\text{rank}_{\boldsymbol{q}}(h)$, as r if exactly $r-1$ intersections have smaller coordinates on dimension d than the intersection between $\ell_{\boldsymbol{q}}$ and h. The *t-set of $\boldsymbol{q}$*, denoted as $H_{\boldsymbol{q}}(t)$, includes the planes in H with $\boldsymbol{q}$-ranks $1, 2, ..., t$, respectively.

Example (cont.) The query space $\mathcal{Q}$ in Figure 2 is one dimensional, because of which we will simplify the vector representation $\boldsymbol{q}$ into a real value q. Consider the query $q = 20$; ℓ_q is the vertical line $x = 20$. The lines in H intersect ℓ_q in the bottom-up order of $h_6, h_5, h_4, h_3, h_1, h_2$. The q-rank of h_1, for example, is 5. □

It is rudimentary to verify the following one-one correspondence between $\mathcal{W}_{[d]=1}$ and $\mathcal{Q}$:

Proposition 1. *Fix any canonical weight vector $\boldsymbol{w} \in \mathcal{W}_{[d]=1}$. Set $\boldsymbol{q} = (\boldsymbol{w}[1], ..., \boldsymbol{w}[d-1])$. For any object $o \in P$ with dual plane $h \in H$, $\text{rank}_{\boldsymbol{w}}(o) = \text{rank}_{\boldsymbol{q}}(h)$.*

Example (cont.): Consider the canonical weight vector $\boldsymbol{w} = (20, 1)$. As mentioned earlier, the $\boldsymbol{w}$-ranks of o_6, o_5, o_4, o_3, o_1, and o_2 are 1, 2, 3, 4, 5, and 6, respectively. These are identical to the q-ranks (where $q = 20$) of their corresponding dual planes. □

Given a non-empty subset $\mathcal{S} \subseteq H$, we define its *$\boldsymbol{q}$-rank regret* as

$$RR_{\boldsymbol{q}}(\mathcal{S}) = \min_{h \in \mathcal{S}} \text{rank}_{\boldsymbol{q}}(h)$$

and accordingly the *maximum rank regret* of $\mathcal{S}$ as

$$\text{MRR}'(\mathcal{S}) = \max_{\boldsymbol{q} \in \mathcal{Q}} RR_{\boldsymbol{q}}(\mathcal{S}). \tag{3}$$

Problem 2 (Dual Version of Problem 1). *Given a set H of n planes in $\mathbb{R}^d$ and an integer $k \in [1, n]$, find a non-empty $\mathcal{S} \subseteq H$ with the smallest size satisfying $MRR'(\mathcal{S}) \leq k$.*

Problems 1 and 2 are equivalent:

Lemma 2. *For any $S \subseteq P$, $MRR(S) = MRR'(\mathcal{S})$, where $\mathcal{S} = \{\text{dual plane of } o \mid o \in S\}$.*

Proof. Let us first prove $\text{MRR}(S) \leq \text{MRR}'(\mathcal{S})$. Consider any $\boldsymbol{w} \in \mathcal{W}$ achieving $\text{rank}_{\boldsymbol{w}}(S) = \text{MRR}(S)$. By Lemma 1, there is a $\boldsymbol{w}^+ \in \mathcal{W}_{[d]\neq 0}$ satisfying $\text{rank}_{\boldsymbol{w}}(S) = \text{rank}_{\boldsymbol{w}^+}(S)$, which implies a canonical $\boldsymbol{w'} \in \mathcal{W}_{[d]=1}$ satisfying $\text{rank}_{\boldsymbol{w'}}(S) = \text{rank}_{\boldsymbol{w}^+}(S) = \text{MRR}(S)$. By Proposition 1, there is a $\boldsymbol{q} \in \mathcal{Q}$ such that $\text{rank}_{\boldsymbol{q}}(\mathcal{S}) = \text{rank}_{\boldsymbol{w'}}(S) = \text{MRR}(S)$. It thus follows that $\text{MRR}'(\mathcal{S}) \geq \text{rank}_{\boldsymbol{q}}(\mathcal{S}) \geq \text{MRR}(S)$.

Reversing the above proves $\text{MRR}'(\mathcal{S}) \leq \text{MRR}(S)$. □

3 EQUIVALENCE TO EPSILON-NETS

A *halfspace* in $\mathbb{R}^d$ is the set of points $p \in \mathbb{R}^d$ satisfying $\sum_{i=1}^{d} c_i \cdot p[i] \geq c_{d+1}$, where $c_1, ..., c_{d+1}$ are real-valued coefficients. The halfspace is *non-negative* if $c_1, c_2, ..., c_d$ are non-negative (note: there are no constraints on c_{d+1}).

Given a real value $\epsilon \in (0, 1]$, we call a subset $S \subseteq P$ an *ϵ-net* if every non-negative halfspace covering at least ϵn objects in P must cover at least one object in S. The lemma below reveals a connection between ϵ-nets and rank regret representatives:

Lemma 3. *A subset S of P is a k-rank-regret representative of P if and only if S is a (k/n)-net of P.*

Proof. The IF direction: Consider an arbitrary weight vector $\boldsymbol{w} = (\boldsymbol{w}[1], ..., \boldsymbol{w}[d])$. Let $o \in P$ be an object with $\boldsymbol{w}$-rank k; specially, if no such objects exist (due to ties in scores), define o as an object that has the largest $\boldsymbol{w}$-rank among all the objects with $\boldsymbol{w}$-ranks at most k. Set $\tau = \boldsymbol{w} \cdot \boldsymbol{o}$. At least k objects $\boldsymbol{o'} \in P$ satisfy $\boldsymbol{w} \cdot \boldsymbol{o'} \geq \tau$. Thus, the halfplane $\boldsymbol{w} \cdot \boldsymbol{p} \geq \tau$ covers at least k objects of P and, by definition of (k/n)-net, must contain an object $\boldsymbol{o''} \in S$. The $\boldsymbol{w}$-rank of $\boldsymbol{o''}$, therefore, is at most k. This means that S is a k-rank-regret representative.

The ONLY-IF direction: Consider any halfspace h covering at least k objects of P. Let $\sum_{i=1}^{d} c_i \cdot p[i] \geq c_{d+1}$ be the inequality of h. Set $\boldsymbol{w} = (c[1], ..., c[d])$. Being a k-rank-regret representative, S must contain an object o in the k-set of $\boldsymbol{w}$. The $\boldsymbol{w}$-score of o must be at least c_{d+1} (otherwise, the at least k objects covered by h all have scores strictly higher than o, giving a contradiction). It thus follows that o is covered by h. This means that S is a (k/n)-net. □

As proved in [34], by random sampling $O(\frac{n}{k} \log n)$ points from P with replacement (assuming that d is a fixed constant), we obtain a set S_{sam} of points that is a (k/n)-net with probability at least $1 - 1/n^2$. Combining this with Lemma 3 yields:

Theorem 4. *We can compute in $O(\frac{n}{k} \log n)$ time a subset $S \subseteq P$ of size $O(\frac{n}{k} \log n)$ that is a k-rank-regret representative of P with probability at least $1 - 1/n^2$.*

Instance optimal ϵ-nets. Lemma 3 gives an alternative interpretation of Problem 1: its goal is to find the *smallest* ϵ-net (where $\epsilon = k/n$) on the *given* P, namely, an *instance optimal* ϵ-net of P. Even at $d = 2$, $1/\epsilon$ is a known *worst-case* lower bound on the ϵ-net size (a higher lower bound of $\Omega(\frac{n}{k} \log \frac{n}{k})$ holds for $d \geq 4$; see [39]). Hence, when measured by the *worst-case* quality, n/k is the best possible, and Theorem 4 is already near optimal. However, n/k is a pessimistic estimate on the size of the smallest (k/n)-net for *every* P. As shown in the experiments, we can find (k/n)-nets whose sizes are considerably smaller than n/k on real-world data.

Heuristics. In practice, we may shrink the sample set S_{sam} described earlier to produce a smaller k-rank-regret representative. The first idea is to keep only the *extreme set* of S_{sam} — denoted as $EXT(S_{sam})$ — namely, the set of objects on the convex hull boundary of S_{sam}. We can shrink S_{sam} even further by resorting to *skylines* [11]. Given two distinct objects $o, o' \in EXT(S_{sam})$, we say that o *dominates* o' if $o[i] \geq o'[i]$ on all $i \in [1, d]$. The *skyline* of $EXT(S_{sam})$ is the set of objects in $EXT(S_{sam})$ that are not dominated by other objects in $EXT(S_{sam})$. The skyline serves as a k-rank-regret representative. We refer to the above method as *net-extreme-skyline* (NES).

4 EXACT ALGORITHMS

Section 4.1 will point out the relationships between the proposed k-rank-regret problem and the existing k-regret minimizing set problem, and establish the former's NP-hardness for $d \geq 3$. Section 4.2 will explain how to solve Problem 1 in polynomial time for $d = 2$.

4.1 Connections to Regret-Ratio Minimizing Sets

In this subsection, each object $o \in P$ is assumed to have positive coordinates $o[1], ..., o[d]$, which can be achieved by shifting the coordinate system appropriately. Accordingly, the score of an object is always non-negative under any weight vector $\boldsymbol{w}$.

Define $\text{gain}_{\boldsymbol{w}}(P, k)$ as the lowest $\boldsymbol{w}$-score of the objects in the k-set of $\boldsymbol{w}$. Given a subset $S \subseteq P$, Chester et al. [18] defined its *k-regret ratio* under $\boldsymbol{w}$ as

$$k\text{-regratio}_{\boldsymbol{w}}(S) = \frac{max\{0, \text{gain}_{\boldsymbol{w}}(P, k) - \text{gain}_{\boldsymbol{w}}(S, 1)\}}{\text{gain}_{\boldsymbol{w}}(P, k)}$$

and its *maximum k-regret ratio* as

$$k\text{-regratio}(S) \quad = \quad \max_{\boldsymbol{w} \in \mathscr{W}} k\text{-regratio}_{\boldsymbol{w}}(S).$$

In the *k-regret minimizing set problem*, given an integer $k \in [1, n]$ and a size threshold s, we want to find a subset S with $|S| = s$ to minimize k-regratio(S).

MRR(S) (see (1)) has a connection to k-regratio(S):

Lemma 5. *For any $S \subseteq P$ and any $k \in [1, n]$, MRR$(S) \leq k$ if and only if k-regratio$(S) = 0$.*

Proof. If k-regratio$(S) = 0$, $\text{gain}_{\boldsymbol{w}}(P, k) \leq \text{gain}_{\boldsymbol{w}}(S, 1)$ holds for any weight vector $\boldsymbol{w}$, implying that S contains at least one object in the k-set of $\boldsymbol{w}$. Hence, MRR$(S) \leq k$. Reversing the argument proves the only-if direction. □

In 2D space, the k-regret minimizing set problem can be settled in $\tilde{O}(n^2)$ time [13] (where $\tilde{O}(.)$ hides a polylog n factor). Problem 1 can then be settled in $\tilde{O}(n^2)$ time. By Lemma 5, it suffices to find the smallest $s \in [1, n]$ such that some subset $S \subseteq P$ of size s achieves k-regratio$(S) = 0$. Since k-regratio(S) monotonically decreases when $|S|$ increases, we can discover the desired s with binary search, which requires solving $O(\log n)$ instances of the k-regret minimizing set problem. In the next subsection, we will present an algorithm with a more appealing time complexity of $\tilde{O}(nk)$.

When $d = 3$, Agarwal et al. [4] proved the NP-hardness of the following problem: given a size threshold $s \in [1, n]$, decide whether there is an S with $|S| = s$ and 2-regratio$(S) = 0$. This implies that the 3D version Problem 1 is NP-hard even when $k = 2$. To see why, if we could find in polynomial time an $S \subseteq P$ satisfying MRR$(S) \leq k$ with the smallest $|S|$, we could settle the above decision problem by comparing $|S|$ to s (by Lemma 5). The NP-hardness at $d = 3$ indicates that Problem 1 is NP-hard for all $d \geq 3$.

4.2 A Faster 2D Algorithm

4.2.1 Levels. In general, let H be a set of n planes in $\mathbb{R}^d$. Fix an arbitrary point $p \in \mathbb{R}^d$ and a plane h that does not pass p. We say that p is *above* h if we must move p towards the negative direction of dimension d for p to touch h; otherwise, p is *below* h. The *level* of p is the number of planes in H below p. The *l-level* ($l \in [0, n]$) is the set of all points in $\mathbb{R}^d$ whose levels are exactly l, and the *($\leq l$)-level* is the set of all points in $\mathbb{R}^d$ whose levels are at most l.

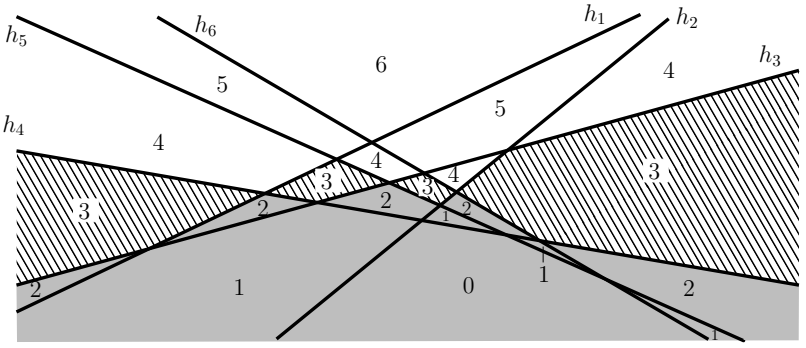

Fig. 3. Illustration of levels. Each number indicates the level of the corresponding cell. The gray area is the (≤ 2)-level, while the striped area is the 3-level.

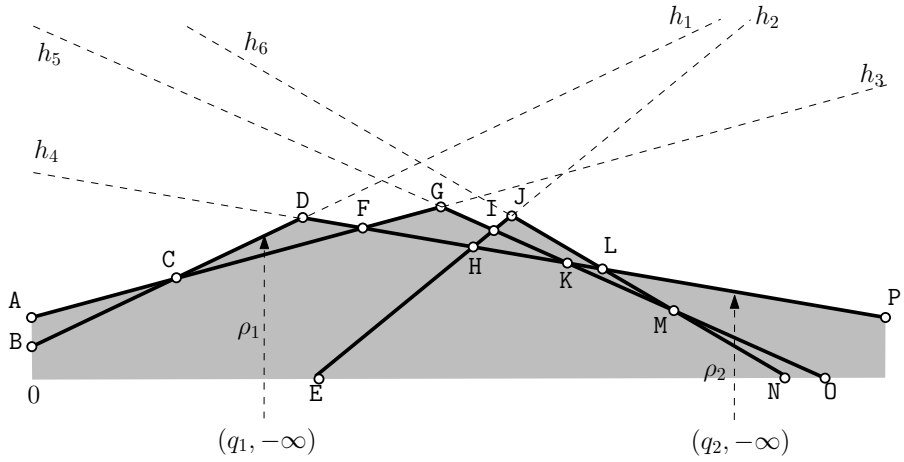

Fig. 4. The boundary edges in the (≤ 2)-level (taken from Figure 3) are shown in bold segments.

Example: Figure 3 shows a set H of 2D planes, i.e., lines, which are taken from Figure 2. Each number indicates the level of the point where the number is placed. The gray area is the (≤ 2)-level, the striped area is the 3-level, and their union is the (≤ 3)-level. □

In 2D space, the $(\leq k)$-level induced by H consists of non-overlapping polygons (see Figure 3) whose edges we refer to as *boundary edges*. There are $O(nk)$ boundary edges [19] and they can be computed in $\tilde{O}(nk)$ time [28]. See Figure 4 for an illustration.

4.2.2 Algorithm. We can rephrase (the 2D version of) Problem 2 in terms of levels. Recall that the input is a set H of lines in the dual space $\mathbb{R}^2$ and the query space $\mathcal{Q}$ is the interval $[0, \infty)$. Each query $q \in \mathcal{Q}$ corresponds to a point $(q, -\infty)$ at the bottom of the dual space. Imaging shooting a ray ρ from $(q, -\infty)$ upwards, which stops right before leaving the $(\leq k-1)$-level. $H_q(k)$ (the k-set of q) includes exactly those lines of H intersecting with ρ. A subset $\mathcal{S} \subseteq H$ *hits* q if $\mathcal{S}$ has at least one line intersecting with ρ. The goal of Problem 2 is to find the smallest $\mathcal{S}$ that hits all queries in $\mathcal{Q}$.

Example (cont.): Recall that the gray area of Figure 4 corresponds to the (≤ 2)-level defined by the set of lines in Figure 3. Assuming $k = 3$, the rays shot from points $(q_1, -\infty)$ and $(q_2, -\infty)$ are ρ_1 and ρ_2, respectively. $\mathcal{S} = \{h_1, h_2, h_3\}$ hits q_1, but does not hit q_2, meaning that $\mathcal{S}$ contains at least a line in the 2-set of q_1, but nothing in that of q_2. Hence, $\mathcal{S}$ is not a solution to Problem 2. An optimal solution is $\mathcal{S} = \{h_3, h_4\}$ (it hits all queries). Optimal solutions are not unique; e.g., $\mathcal{S} = \{h_2, h_5\}$ is another example. □

Next, we explain how to solve Problem 2 in $\tilde{O}(nk)$ time. We define an *envelop chain* as a sequence C of line segments $\sigma_1, \sigma_2, ..., \sigma_{|C|}$ such that:

- every segment of C is in the $(\leq k-1)$-level and is part of a line in H, i.e., the segment's *support line*;
- C is connected, namely, σ_i and σ_{i+1} share an endpoint for all $i \in [1, |C| - 1]$;
- C is x-monotone, namely, any vertical line in $\mathbb{R}^2$ can intersect with at most one segment in C;
- C is concave, namely, the support line of σ_i has a larger slope than that of σ_{i+1} (equivalently, we need to make a right turn in walking from σ_i onto σ_{i+1}).

Example (cont.): In Figure 4. The sequence AC, CD, DF is connected, x-monotone, but not concave (we make a left turn in walking from AC to CD). Two envelop examples are AF, FK, KO and AG, GO. □

The *length* of an envelop chain $C = \sigma_1, \sigma_2, ..., \sigma_{|C|}$ is $|C|$. The projection of C onto the x-axis gives an interval $[x_1, x_2]$ (specifically, x_1 and x_2 are the x-coordinates of the left endpoint and right endpoint of σ_1 and $\sigma_{|C|}$, respectively). Let $H[C]$ be the set of support lines of $\sigma_1, \sigma_2, ..., \sigma_{|C|}$.

Lemma 6. *Let C^* be an envelop chain of the minimum length whose x-projection covers the entire $\mathscr{Q} = [0, \infty)$. Then, $H[C^*]$ is an optimal solution to Problem 2.*

The proof is shown in the appendix.

Example (cont.): In Figure 4, no envelop chains of length 1 have an x-projection covering $\mathscr{Q}$. On the other hand, the x-projection of $C =$ AF, FP covers $\mathscr{Q}$. Hence, $H[C] = \{h_3, h_4\}$ must be an optimal solution (this corresponds to $\{p_3, p_4\}$ in Figure 1). □

We can find C^* by adapting a dynamic programming strategy in [18] originally designed for the regret-ratio minimizing set problem. Let e be a boundary edge in the $(\leq k-1)$-level of H, p be the right endpoint of e, and $p[1]$ be the x-coordinate of p. Define $minlen(e)$ as the smallest length of all envelop chains C such that

- the last segment of C contains e;
- the x-projection of C covers $[0, p[1]]$.

Call e *terminal* if its right endpoint falls on the dual space's right boundary. *OPT* (i.e., $|C^*|$) must be equal to the $minlen(e)$ of some terminal boundary edge e.

Example (cont.): Set $k = 3$. For the boundary edge CD in Figure 4, we have $minlen(\text{CD}) = 1$ because there is a monotone chain C with only one segment BD such that (i) BD contains CD and (ii) the x-projection of C covers $[0, \text{D}[1]]$. Similarly, $minlen(\text{LP}) = 2$, evidenced by the monotone chain $C =$ AF, FP; this C is an optimal solution to Problem 2. □

To describe the computation of $minlen(e)$, let us view each boundary edge e in the $(\leq k-1)$-level as a directed edge pointing from the left endpoint to the right endpoint. Trivially, $minlen(e) = 1$ if the x-projection of e covers the coordinate 0. Otherwise, let p be the left endpoint of e and $\text{IN}(p)$ be the set of incoming edges of p in the $(\leq k-1)$-level. For each $e' \in \text{IN}(p)$, define its *contribution* as:

- $minlen(e')$ if e' and e are on the same line in H;
- $1 + minlen(e')$ if e' has a greater slope than e;
- ∞ otherwise.

Then, $minlen(e)$ equals the minimum contribution of all $e' \in \text{IN}(p)$.

If m is the total number of boundary edges in the $(\leq k-1)$-level, it is now straightforward to compute the $minlen(e)$ of all those edges e in $\tilde{O}(m)$ time by dynamic programming. This produces the value of *OPT*. It is standard to construct an optimal solution C^* from the dynamic programming process with the same time complexity. As all the boundary edges can be found in $\tilde{O}(m) = \tilde{O}(nk)$ time (Section 4.2.1), we have arrived at:

Theorem 7. *When $d = 2$, Problem 2 (hence, Problem 1) can be settled in $\tilde{O}(nk)$ time.*

5 A THEORY ON APPROXIMATION ALGORITHMS

In this section, we will explain how to solve Problem 1 with strong approximation guarantees in $O(n \log n)$ time for $d = 2$ and in polynomial time for $d \geq 3$. Denote by *OPT* the size of an optimal

k-rank-regret representative of the input P. Our aim is to return a c_1k-rank-regret representative of size $c_2 \cdot OPT$ where $1 \leq c_1 = \tilde{O}(1)$ and $1 \leq c_2 = \tilde{O}(1)$. A *size-approximation algorithm* fulfills the purpose with $c_1 = 1$ and $c_2 \geq 1$, a *regret-approximation algorithm* achieves $c_1 \geq 1$ and $c_2 = 1$, and a *bi-criteria approximation algorithm* yields $c_1 > 1$ and $c_2 > 1$.

Section 5.1 will introduce the *shallow cutting technique*, which we apply to design our 2D and 3D algorithms in Sections 5.2 and 5.3, respectively. Both algorithms achieve a time complexity that is "fixed-parameter near-linear" (formally defined in Section 5.4). In the scenario of $d \geq 4$, Section 5.4 will establish a hardness result to exclude such time complexities for bi-criteria approximation algorithms (hence, also for size/regret-approximation algorithms). Nevertheless, Section 5.5 proves that the $d \geq 4$ problem still admits polynomial-time solutions.

5.1 Shallow Cutting

A *simplex* in $\mathbb{R}^d$ is a d-dimensional convex polytope with $d+1$ vertices. A 1D simplex is an interval, a 2D simplex is a triangle, a 3D simplex is a tetrahedron, etc. A *prism* in $\mathbb{R}^d$ is a special d-dimensional simplex that has a vertex at the bottom of $\mathbb{R}^d$, namely, the vertex's d-th coordinate is $-\infty$. Figure 5 shows an example for $d = 2$ and 3, respectively. A 2D prism has the shape of an infinitely extending trapezoid, which can be thought of as a triangle whose lower vertex is at the bottom of $\mathbb{R}^2$. Likewise, a 3D prism can be thought of as a tetrahedron whose lower vertex has z-coordinate $-\infty$.

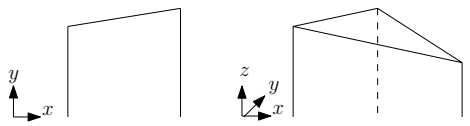

Fig. 5. 2D and 3D prisms

Let H be a set of n planes in $\mathbb{R}^d$. Fix some integer $k \in [0, n]$ and a constant $\lambda > 0$. A $(\lambda, k/n)$-*shallow-cutting* of H is a set Ξ of prisms satisfying:

- Every prism in Ξ is covered by the $(\leq (1+\lambda)k)$-level;
- The union of all prisms in Ξ covers the $(\leq k)$-level;
- Each prism $\Delta \in \Xi$ intersects with $O(1+k)$ planes in H, which constitute the *conflict set* of Δ, denoted as H_Δ.

Example: Figure 6 illustrates a $(\lambda, k/n)$-shallow cutting Ξ where $n = 6$, $k = 2$, and $\lambda = 1$. Ξ contains the four prisms $\Delta_1, ..., \Delta_4$ shown. The union of all the prisms covers the (≤ 2)-level, but is contained in the (≤ 4)-level. The conflict set H_{Δ_4}, for example, consists of h_4, h_5, and h_6. □

Lemma 8 ([2, 16]). *When $d = 2$ and 3, for any $\lambda > 0$ and $k \in [0, n]$, we can compute a $(\lambda, k/n)$-shallow cutting of $O(n/(1+k))$ non-overlapping prisms and all the conflict sets in $O(n \log n)$ time.*

5.2 A 2D Algorithm

We will describe a regret-approximation algorithm to solve Problem 1 with $d = 2$. Our goal is to find a small subset $S \subseteq P$ that is a ck-rank-regret representative where $c = 2 + \delta$ and $\delta > 0$ can be an arbitrarily small constant.

We will work on the corresponding instance of Problem 2. Here, the input is a set H of n lines in $\mathbb{R}^2$, from which we want to extract a subset $\mathcal{S}$ to make sure $RR_{\boldsymbol{q}}(\mathcal{S}) \leq ck$ for every query $\boldsymbol{q} \in \mathcal{Q}$.

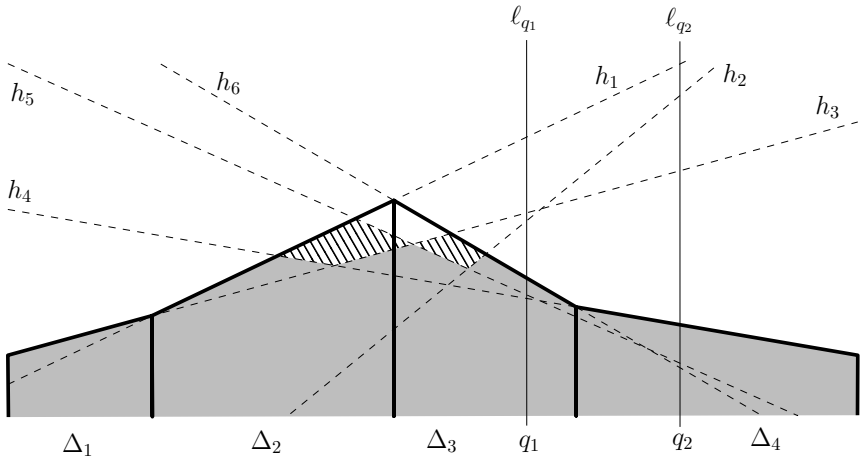

Fig. 6. A $(1, 1/3)$-shallow cutting in 2D space. All the lines come from Figure 4. The gray area is the (≤ 2)-level, the striped area is a part of the 3-level, and the closed white region is a part of the 4-level.

The query space $\mathcal{Q}$ is $[0, \infty)$. Accordingly, we will represent each query simply as a real-value $q \geq 0$.

A rank-sum lemma. Consider any query $q \in [0, \infty)$ and the vertical line ℓ_q passing the point $(q, -\infty)$. Let h be a line in H and p be the intersection point between h and ℓ_q. Recall that $\text{rank}_q(h)$ (i.e., the q-rank of h) equals 1 plus the number of lines below p in H. The lemma states an important property about $\text{rank}_q(h)$:

Lemma 9. *Consider any line $h \in H$. Let q_1 and q_2 be queries satisfying $q_1 \leq q_2$. For any $q \in [q_1, q_2]$, $rank_q(h) \leq rank_{q_1}(h) + rank_{q_2}(h) - 1$.*

The above will be subsumed by Lemma 16 which, however, requires a more sophisticated argument. Understanding the proof of Lemma 9 will make it easier to follow that of Lemma 16. Let us first see an illustration in Figure 6. Line h_5 has q_1-rank 2 and q_2-rank 2 (see q_1 and q_2 in the figure). The lemma assures us that h_5 has q-rank at most 3 for any $q \in [q_1, q_2]$.

Proof. Given a query q, define (i) p_q as the intersection point between h and ℓ_q, and (ii) ρ_q as the downward-shooting ray that emanates from but does not include p_q. Let S_q be the set of lines in H intersecting with ρ_q. $|S_{q_1}| = \text{rank}_{q_1}(h) - 1$ and $|S_{q_2}| = \text{rank}_{q_2}(h) - 1$. We will prove that, for any $q \in [q_1, q_2]$, any line $h' \in S_q$ must belong to either S_{q_1} or S_{q_2}. This indicates $|S_q| \leq |S_{q_1}| + |S_{q_2}|$, from which the lemma follows.

Let p'_{q_1} (or p'_{q_2}) be the intersection point between h' and ℓ_{q_1} (or ℓ_{q_2}, resp.), as shown in Figure 7. Assume that h' belongs to neither S_{q_1} nor S_{q_2}, which means that p'_{q_1} and p'_{q_2} must be on or above h. Hence, the entire segment $p'_{q_1}p'_{q_2}$ must be on or above h. Therefore, the q-rank of h' cannot be lower than that of h, contradicting $h' \in S_q$. □

The algorithm. We start by using Lemma 8 to obtain a $(\delta/2, (k-1)/n)$-shallow cutting Ξ on H. Remember that Lemma 8 also produces the conflict set of each prism Δ, namely, the set $H_\Delta \subseteq H$ of lines intersecting with Δ.

For each line $h \in H$, we generate an interval $\mathcal{I}_h$ as follows. First, identify the *leftmost* (or *rightmost*) prism Δ_1 (or Δ_2, resp.) intersecting h. Let σ_1 be the part of h that appears in Δ_1; note that σ_1 is a

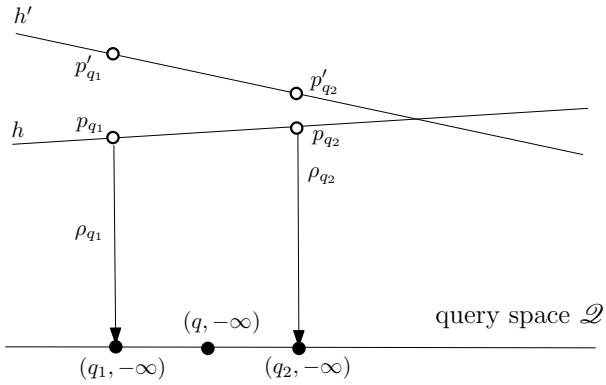

Fig. 7. Proof of Lemma 9

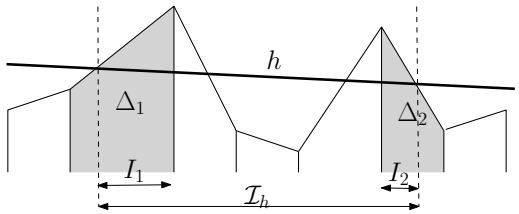

Fig. 8. Illustration of $\mathcal{I}_h$

segment. Obtain similarly a segment σ_2 with respect to Δ_2. Define I_1 (or I_2) as the x-projection of σ_1 (or σ_2, resp.). The interval $\mathcal{I}_h$ is the minimum bounding interval of I_1 and I_2. See Figure 8 for an illustration. In the special case where h intersects with no prisms of Ξ, define $\mathcal{I}_h$ as the empty interval.

Lemma 10. *For any line $h \in H$ whose $\mathcal{I}_h$ is not empty, $rank_q(h) \leq (2+\delta)k$ for any $q \in \mathcal{I}_h \cap \mathcal{Q}$.*

Proof. Let Δ_1 and Δ_2 be the two prisms that define $\mathcal{I}_h$. By the fact that $q \in \mathcal{I}_h$, we can find queries $q_1, q_2 \in \mathcal{I}_h$ such that (i) $0 < q_1 \leq q \leq q_2$, and (ii) the x-projection of Δ_1 (or Δ_2) covers q_1 (or q_2, resp.). Suppose that ℓ_{q_1} intersects h at point p_1. Since p_1 is inside Δ_1, the level of p_1 must be at most $(1+\delta/2)(k-1)$ because the entire Δ_1 is in the $(\leq (1+\delta/2)(k-1))$-level of H (definition of shallow cutting; see Section 5.1). Hence, $\text{rank}_{q_1}(h) \leq (1+\delta/2)(k-1)+1 < (1+\delta/2)k$. Similarly, $\text{rank}_{q_2}(h) < (1+\delta/2)k$. The claim then follows from Lemma 9. □

Denote by $\mathcal{S}^*$ an optimal solution to Problem 2. We observe:

Lemma 11. *The union of $\mathcal{I}_h$ for all the $h \in \mathcal{S}^*$ covers $\mathcal{Q}$.*

Proof. Assume, on the contrary, that the union fails to include a query $q \geq 0$. By definition of $\mathcal{S}^*$, there exists a line $h \in \mathcal{S}^*$ whose q-rank is at most k. Let p be the intersection point between h and ℓ_q. The fact $\text{rank}_q(h) \leq k$ indicates that the level of p is at most $k-1$. Now, consider the prism $\Delta \in \Xi$ whose x-projection covers q. We assert that p must fall inside Δ; otherwise, p falls outside the union of all the prisms of Ξ, contradicting the fact that the union must contain the $(\leq k-1)$-level of H. However, Δ covering p implies that $\mathcal{I}_h$ must contain q, giving a contradiction. □

Define $\mathcal{I} = \{\mathcal{I}_h \mid h \in H\}$. We find a subset $\mathcal{J} \subseteq \mathcal{I}$ of the smallest size having the property that the union of all the intervals in $\mathcal{J}$ covers $\mathcal{Q}$. The existence of $\mathcal{J}$ is guaranteed by Lemma 11. Finally, we return $\mathcal{S} = \{h \in H \mid \mathcal{I}_h \in \mathcal{J}\}$ as our final result.

Theorem 12. *In 2D space, the above algorithm runs in $O(n \log n)$ time and returns a set $\mathcal{S}$ of size at most OPT whose maximum rank regret is at most $(2+\delta)k$, where $\delta > 0$ can be an arbitrarily small constant.*

Proof. Lemma 11 and the minimality of $\mathcal{J}$ together imply $|\mathcal{S}| = |\mathcal{J}| \leq |\mathcal{S}^*| = OPT$. $\text{MRR}(\mathcal{S}) \leq (2+\delta)k$ follows from Lemma 10 and the fact that every query is covered by the $\mathcal{I}_h$ of at least one $h \in \mathcal{J}$.

Regarding the running time, Lemma 8 itself costs $O(n \log n)$ time. For each line $h \in H$, its Δ_1 and Δ_2 can be found in time proportional to the number of prisms intersecting h. Hence, the total time spent for this purpose is proportional to the total size of all the prisms' conflict sets, which is $O(\frac{n}{1+k} \cdot (1+k)) = O(n)$. The problem of discovering $\mathcal{J}$ from $\mathcal{I}$ is known as the *interval covering problem*, which can be optimally settled in $O(n \log n)$ time (see, e.g., [1]). □

5.3 A 3D Algorithm

The 3D space can also be dealt with using shallow cutting in a manner similar to what was described in the 2D algorithm. We move the proof of the following theorem to the appendix because the details are somewhat repetitive.

Theorem 13. *For $d = 3$, we can compute in $\tilde{O}(nk^2)$ time a subset $\mathcal{S} \subseteq H$ of size at most $OPT \cdot O(\log n)$ whose maximum rank regret is at most $(1+\delta)k$, where $\delta > 0$ can be an arbitrarily small constant.*

5.4 Hardness of Dimensions $d \geq 4$

For $k = O(\text{polylog}\, n)$, our 2D and 3D algorithms in Theorems 7, 12, and 13 all finish in $\tilde{O}(n)$ time. In particular, their time complexities have the form $f(k) \cdot \tilde{O}(n)$ where $f(k)$ is a monotonic function depending only on k and satisfying $f(k) = \tilde{O}(1)$ for $k = \text{polylog}\, n$. This is a nice feature because users prefer small k in practice. Can we hope to retain this feature when the dimensionality is 4 or above?

Let $\mathscr{A}$ be a bi-criteria approximation algorithm for Problem 1. We say that $\mathscr{A}$ is *fixed-parameter near-linear* if its running time is bounded by $f(k) \cdot \tilde{O}(n)$ where $f(k)$ is a monotonic function depending only on k and satisfying $f(1) = \tilde{O}(1)$. Our algorithms in Theorems 7, 12, and 13 are all fixed-parameter near-linear. This subsection will prove that no fixed-parameter near-linear algorithms exist for $d \geq 4$, unless major breakthroughs could be made in computational geometry.

An object $o \in P$ is an *extreme point* of P if it is a vertex of the convex hull of P. The *extreme point problem*, where the goal is to report all the extreme points of P, has been extensively studied. The first major result was a 1993 algorithm due to Matousek [43] that has running time $O(n^{2-2/(\lceil d/2 \rceil + 1) + \epsilon})$. In his 1996 paper [15], Chan pointed out that the time can be improved to $O(n^{2-2/(\lceil d/2 \rceil + 1)})$. In the same paper, Chan gave an *output-sensitive* algorithm with time $\tilde{O}(n + (n \cdot OUT)^{1-1/(\lceil d/2 \rceil + 1)})$, where OUT is the number of extreme points. For $d = 4$, the bound is $\tilde{O}(n + (n \cdot OUT)^{2/3})$, which still remains the best today.

In this subsection, we will prove:

Theorem 14. *Let OPT be the size of an optimal k-rank-regret representative of P. Suppose that there exists an algorithm $\mathscr{A}$ that, for some value $c = O(\text{polylog}\, n)$, can compute a (ck)-rank-regret*

representative of size $\tilde{O}(OPT)$ in $f(k)\cdot\tilde{O}(n)$ time in 4D space where function $f(k)$ satisfies $f(1) = \tilde{O}(1)$. Then, there exists an algorithm solving the 4D extreme point problem in $\tilde{O}(n + OUT^{4/3})$ time.

$\tilde{O}(n + OUT^{4/3})$ compares more favorably with Chan's bound $\tilde{O}(n + (n \cdot OUT)^{2/3})$ and would make an exciting result. An impossibility result in 4D space trivially holds for $d \geq 5$ as well.

Proof for $c = 1$. Denote by X the set of extreme points of P. The optimal 1-rank-regret representative is the set S^* of objects each maximizing the score of at least one weight vector. Thus, $S^* \subseteq X$.

S^* is only a *subset* of X because we have restricted each weight vector $\boldsymbol{w}$ to take non-negative components $w[1], ..., w[4]$. By requiring each $w[i]$ ($i \in [1, 4]$) to be positive or negative independently, we obtain $2^4 = 16$ instances of Problem 1, all on the same P. Denote by S^*_j ($1 \leq j \leq 16$) the optimal 1-rank-regret representative of the j-th instance. X must be the union of $S^*_1, ..., S^*_{16}$.

Let us run $\mathscr{A}$ on each of the 16 instances on P, by forcing the input k to 1. Denote by S_j the output of $\mathscr{A}$ for the j-th instance. Since $c = 1$, it must hold that $S^*_j \subseteq S_j$ for all $j \in [1, 16]$. Furthermore, the $\tilde{O}(OPT)$-output-size requirement of $\mathscr{A}$ guarantees $|S_j| = |S^*_j| \cdot \tilde{O}(1)$. The total running time of $\mathscr{A}$ in solving all the instances is $f(1) \cdot \tilde{O}(n) = \tilde{O}(n)$.

Set $S = S_1 \cup S_2 \cup ... \cup S_{16}$. Because

$$|S| \leq 16 \cdot \max_{j=1}^{16} |S_j| = \tilde{O}\left(\max_{j=1}^{16} |S^*_j|\right) = \tilde{O}(|X|) = \tilde{O}(OUT)$$

we can find X in $\tilde{O}(OUT^{4/3})$ time by running Chan's algorithm on S. This gives an overall algorithm to compute X in $\tilde{O}(n + OUT^{4/3})$ time.

Proof for $c > 1$. We can extend the argument to any $c = O(\text{polylog}\, n)$. The crucial idea is to create a new dataset P' by duplicating each object (of P) c times. The argument then proceeds as before except that $\mathscr{A}$ should be applied to the 16 instances on P'. The property $S^*_j \subseteq S_j$ is now rephrased as: for every object $o \in S^*_j$, at least one of its copies exists in S_j. To understand why, note that there must be a weight vector $\boldsymbol{w}$ such that o has $\boldsymbol{w}$-rank 1 in P. Thus, if none of the c copies of o is in S_j, the best $\boldsymbol{w}$-rank (in P') of the objects in S_j under $\boldsymbol{w}$ is at least $c + 1$, contradicting the fact that the algorithm must have a maximum rank regret $c \cdot k = c$. The rest of the argument then runs through with no difficulty. This completes the proof of Theorem 14.

5.5 Algorithms for Dimensions $d \geq 4$

Next, we explain how to obtain a polynomial-time size-approximation algorithm for Problem 1 when $d \geq 4$.

A hitting-set approach. Recall that the k-set $P_{\boldsymbol{w}}(k)$ of a weight vector $\boldsymbol{w}$ contains the objects in P with ranks at most k. A subset $S \subseteq P$ *hits* a k-set $P_{\boldsymbol{w}}(k)$ if $S \cap P_{\boldsymbol{w}}(k) \neq \emptyset$. Problem 1 essentially aims to find the smallest subset hitting all the k-sets.

Although the number of weight vectors is infinite, the number of *distinct* k-sets is finite because different weight vectors may end up having the same k-set. Let $\mathcal{K}$ be the collection of all the (distinct) k-sets $\{K_1, ..., K_s\}$ where $s = |\mathcal{K}|$. Problem 1 then becomes a *hitting set* problem: find the smallest subset $S \subseteq \bigcup_{i=1}^{s} K_i$ such that $S \cap K_i \neq \emptyset$ for every $i \in [1, s]$. The standard greedy algorithm runs in time $\tilde{O}(sk)$ and guarantees a subset S of size at most $OPT \cdot (1 + \ln n)$.

There is considerable work on bounding the number s of k-sets. Currently, the best bound is $O(nk^{1/3})$ [22] for $d = 2$, $O(nk^{3/2})$ for $d = 3$ [51], and in general $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil - c_d})$ for fixed $d \geq 4$ [5], where c_d is a tiny constant that tends to 0 very quickly as d grows. We must also account for the time to *enumerate* all the k-sets. Enumeration can be done in $\tilde{O}(nk + sk)$ expected time [28] in 2D, $\tilde{O}(nk^2 + sk)$ expected time [3] in 3D, and $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil} + sk)$ expected time [44] in fixed d-dimensional space with $d \geq 4$.

The above discussion gives:

Theorem 15. *For any constant $d \geq 2$, we can compute in $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil} + sk)$ expected time a subset $\mathcal{S} \subseteq H$ of size at most $OPT \cdot (1 + \ln n)$ whose maximum rank regret is at most k, where s is the number of distinct k-sets which is bounded by $O(nk^{3/2})$ for $d = 3$ and by $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ for $d \geq 4$.*

It is interesting to compare the 3D result of Theorem 15 to that of Theorem 13. The time complexity of Theorem 15 is $O(nk^{5/2})$ at $d = 3$, worse than the bound in Theorem 13. On the other hand, Theorem 15 produces size-approximation algorithms, whereas Theorem 13 gives a bi-criteria algorithm.

A remark is in order. The hitting set instance mentioned earlier is actually an instance of the *geometric hitting set* problem. By replacing the standard greedy algorithm with the re-weighting algorithm of [32] for geometric hitting set, we can reduce the approximation ratio from $1 + \ln n$ to $O(\log OPT)$ at the cost of slightly higher computation time.

k-set enumeration. Theorem 15 requires enumerating all possible k-sets, for which purpose the algorithms in [5, 6, 22, 51] are rather complicated. To alleviate the issue, we describe a practical method which does not improve the complexities in Theorem 15, but is much easier to implement. Our method is adapted from an algorithm in [6], which, however, requires the sophisticated concept of *k-set polytope* (see [6] for details). Instead, we will adopt an intuitive graph perspective.

We call a k-set *clean* if it has size k.[1] As far as the hitting set approach is concerned, it suffices to consider the set $\mathcal{K}_{clean}$ of clean k-sets [5, 6, 22, 51]. Let us introduce the *k-set graph* $G(V, E)$ where:

- $V = \mathcal{K}_{clean}$;
- E has an edge between two k-sets (a.k.a. vertices) K_1 and K_2 if and only if $|K_1 \cap K_2| = k - 1$.

$G(V, E)$ is connected, namely, it has a single connected component.

Figure 9 shows an algorithm for generating $\mathcal{K}_{clean}$ incrementally by performing a BFS (breaths first search) on G. After finding an arbitrary clean k-set (Line 1), the algorithm adds it to a queue (Line 2) and continues the traversal until the queue is empty (Line 3). At every iteration, the algorithm removes a k-set K (i.e., a vertex in G) from the queue (Line 4) and generates another set K' of size k by replacing exactly one object $o \in K$ with an object $o' \in P \setminus K$ (Lines 5-7). If K' does not belong to $\mathcal{K}_{clean}$, the algorithm checks whether K' is a valid k-set. If so, K' is a neighbor vertex of K in G and, hence, is added to $\mathcal{K}_{clean}$ and to the queue (Line 8-9). After the BFS finishes, the final $\mathcal{K}_{clean}$ is returned (Line 10).

Deciding if K' is a k-set can be done through linear programming. Specifically, the answer is yes if and only if there exist a weight vector $\boldsymbol{w}$ and a real value τ such that

- $\boldsymbol{o} \cdot \boldsymbol{w} > \tau$ for every object $\boldsymbol{o} \in K'$;
- $\boldsymbol{o} \cdot \boldsymbol{w} < \tau$ for every object $\boldsymbol{o} \in P \setminus K'$.

[1] The size may be greater than k due to a tie in score among multiple objects.

algorithm enum-kset (P)
1. $\mathcal{K}_{clean} = \{K\}$, where K is an arbitrary clean k-set
2. *Enqueue*(K)
3. **while** *queue* is not empty **do**
4. K = *Dequeue*( )
5. **for** $o \in K$ **do**
6. **for** $o' \in P \setminus K$ **do**
7. $K' = K \cup \{o'\} \setminus \{o\}$
8. **if** $K' \notin \mathcal{K}_{clean}$ and K' is a valid k-set **then**
9. add K' to $\mathcal{K}_{clean}$; *Enqueue*(K')
10. return $\mathcal{K}_{clean}$

Fig. 9. The k-set enumeration algorithm

Motivated by this, we construct a linear program that has $d + 2$ variables: $\boldsymbol{w}[1], ..., \boldsymbol{w}[d]$, τ, and g:

maximize g subject to:
1. $g \geq 0$
2. $\forall \boldsymbol{o} \in K'$: $\quad \boldsymbol{o} \cdot \boldsymbol{w} - \tau \geq g$
3. $\forall \boldsymbol{o} \in P \setminus K'$: $\quad \boldsymbol{o} \cdot \boldsymbol{w} - \tau \leq -g$

The above program never returns a negative g (because setting $\boldsymbol{w} = \boldsymbol{0}$ and $\tau = g = 0$ gives a feasible solution). The required $\boldsymbol{w}$ and τ exist if and only if the returned g is positive.

6 A SPACE PARTITION ALGORITHM

This section will present a heuristic algorithm for finding a k-rank-regret representative in any dimensionality. The algorithm is built on a *rank sum lemma* that generalizes Lemma 9 and is interesting in its own right. We will first present this lemma in Section 6.1 and then explain the algorithm in Section 6.2.

6.1 A Rank Sum Lemma in Arbitrary Dimensions

We will consider Problem 2. Recall that the input is a set H of planes in $\mathbb{R}^d$; and the query space $\mathcal{Q}$ includes all $(d-1)$-dimensional vectors $\boldsymbol{q}$ such that $\boldsymbol{q}[i] \geq 0$ for every $i \in [1, d-1]$. We want to find an $\mathcal{S} \subseteq H$ such that, for every query $\boldsymbol{q}$, $\mathcal{S}$ contains a plane h satisfying $\text{rank}_{\boldsymbol{q}}(h) \leq k$.

The rest of this subsection serves as a proof of:

Lemma 16. *Fix an arbitrary plane $h \in H$. Consider a simplex Δ in the $(d-1)$-dimensional query space $\mathcal{Q}$. Let $\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_d$ be the query vectors at the d vertices of Δ. Then, for any query $\boldsymbol{q}$ in Δ:*

$$rank_{\boldsymbol{q}}(h) \quad \leq \quad \left(\sum_{i=1}^{d} rank_{\boldsymbol{q}_i}(h) \right) - (d-1). \tag{4}$$

Note how Lemma 16 generalizes Lemma 9: the interval $[q_1, q_2]$ in Lemma 9 is a simplex Δ in one-dimensional space.

Proposition 2. *Fix an arbitrary plane $h \in H$. Consider a segment $\boldsymbol{q}_1\boldsymbol{q}_2$ in the $(d-1)$-dimensional query space $\mathcal{Q}$. Then, for any query $\boldsymbol{q}$ on the segment, $rank_{\boldsymbol{q}}(h) \leq rank_{\boldsymbol{q}_1}(h) + rank_{\boldsymbol{q}_2}(h) - 1$.*

Proof. The proof uses the same ideas as in the proof of Lemma 9. Each query $\boldsymbol{q}$ corresponds to the point $(\boldsymbol{q}[1], ..., \boldsymbol{q}[d-1], -\infty)$ at the bottom of the dual space $\mathbb{R}^d$. Denote by $\ell_{\boldsymbol{q}}$ the line

parallel to dimension d and passing $(\boldsymbol{q}[1], ..., \boldsymbol{q}[d-1], -\infty)$. Define (i) $p_{\boldsymbol{q}}$ as the intersection point between h and $\ell_{\boldsymbol{q}}$, and (ii) $\rho_{\boldsymbol{q}}$ as the open downward-shooting ray that emanates from $p_{\boldsymbol{q}}$ but does not include $p_{\boldsymbol{q}}$. Let $S_{\boldsymbol{q}}$ be the set of lines in H intersecting with $\rho_{\boldsymbol{q}}$. $|S_{\boldsymbol{q}_1}| = \text{rank}_{\boldsymbol{q}_1}(h) - 1$ and $|S_{\boldsymbol{q}_2}| = \text{rank}_{\boldsymbol{q}_2}(h) - 1$. We will prove that, for any $\boldsymbol{q}$ on the segment $\boldsymbol{q}_1\boldsymbol{q}_2$, any plane $h' \in S_{\boldsymbol{q}}$ must belong to either $S_{\boldsymbol{q}_1}$ or $S_{\boldsymbol{q}_2}$. This indicates $|S_{\boldsymbol{q}}| \leq |S_{\boldsymbol{q}_1}| + |S_{\boldsymbol{q}_2}|$, from which the lemma will follow.

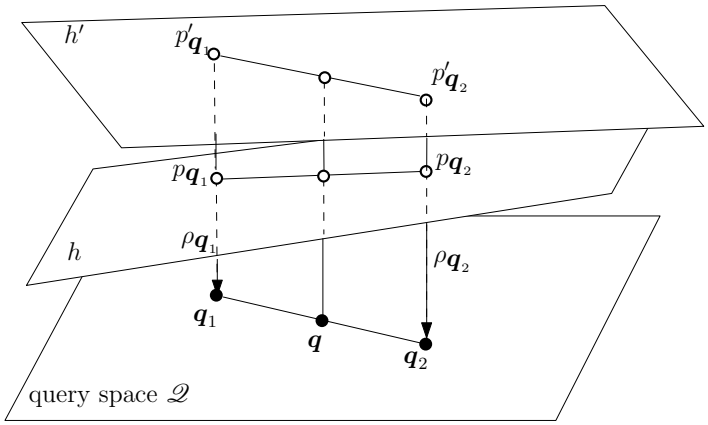

Fig. 10. Proof of Proposition 2

Let $p'_{\boldsymbol{q}_1}$ (or $p'_{\boldsymbol{q}_2}$) be the intersection point between h' and $\ell_{\boldsymbol{q}_1}$ (or $\ell_{\boldsymbol{q}_2}$, resp.); Figure 10 illustrates this for $d = 3$. Assume that h' belongs to neither $S_{\boldsymbol{q}_1}$ nor $S_{\boldsymbol{q}_2}$, which means that $p'_{\boldsymbol{q}_1}$ and $p'_{\boldsymbol{q}_2}$ must be on or above h. Hence, the entire segment $p'_{\boldsymbol{q}_1}p'_{\boldsymbol{q}_2}$ must be on or above h. Therefore, the $\boldsymbol{q}$-rank of h' cannot be lower than that of h, contradicting $h' \in S_{\boldsymbol{q}}$. □

Proposition 3. *Fix an arbitrary plane $h \in H$. Consider any $m \geq 2$ queries $\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_m$ in $\mathcal{Q}$. For any query $\boldsymbol{q}$ that is a linear combination of $\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_m$, $rank_{\boldsymbol{q}}(h) \leq (\sum_{i=1}^{m} rank_{\boldsymbol{q}_i}(h)) - (m-1)$.*

Proof. We will prove the proposition by induction on m. The base case of $m = 2$ is simply Proposition 2. Assuming correctness on $m = t \geq 2$, next we prove the claim on $m = t + 1$.

Let us write $\boldsymbol{q}$ as $\sum_{i=1}^{m} \alpha_i \boldsymbol{q}_i$ where $\alpha_1, ..., \alpha_m$ are real values in $[0, 1]$ satisfying $\sum_{i=1}^{m} \alpha_i = 1$. We consider $\alpha_m > 0$ (otherwise, the claim holds by the inductive assumption). Introduce:

$$\boldsymbol{q}' = \sum_{i=1}^{m-1} \frac{\alpha_i}{1-\alpha_m} \boldsymbol{q}_i$$

Since $\boldsymbol{q}'$ is a linear combination of $\alpha_1, ..., \alpha_{m-1}$, by the inductive assumption we know $\text{rank}_{\boldsymbol{q}'}(h) \leq (\sum_{i=1}^{m-1} \text{rank}_{\boldsymbol{q}_i}(h)) - (m-2)$. Notice that $\boldsymbol{q} = (1-\alpha_m)\boldsymbol{q}' + \alpha_m \boldsymbol{q}_m$. By Proposition 2, we have $\text{rank}_{\boldsymbol{q}}(h) \leq \text{rank}_{\boldsymbol{q}'}(h) + \text{rank}_{\boldsymbol{q}_m}(h) - 1 \leq (\sum_{i=1}^{m} \text{rank}_{\boldsymbol{q}_i}(h)) - (m-1)$. □

Lemma 16 now follows from the above proposition, using the well-known fact that any vector $\boldsymbol{q}$ inside a simplex Δ can be expressed as a linear combination of the d vertex vectors of Δ.

6.2 Algorithm

Our algorithm attacks Problem 2 and takes a set H of planes in the dual space as the input.

algorithm space-partition (H)
1. $\mathcal{S} = \emptyset$
2. *Enqueue*($\mathcal{Q}$)
3. **while** *queue* is not empty **do**
4. R = *Dequeue*( ) /* R is a rectangle in $\mathcal{Q}$ */
5. **if** R is protected by a plane in $\mathcal{S}$ **then continue**
6. **if** $\exists$ a plane $h \in H$ that protects R **then**
7. add h to $\mathcal{S}$ (if multiple such h's exist, add the one with the lowest aggregated rank on R)
else
8. split R into two rectangles of the same size on the dimension where R has the longest extent; call *Enqueue* on both two rectangles
9. return $\mathcal{S}$

Fig. 11. The space-partition algorithm

Rectangle protection. Let R be a hyper-rectangle in the query space $\mathcal{Q}$ and $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, ..., $\boldsymbol{q}_{2^{d-1}}$ be the query vectors at the corners of R. Consider an arbitrary plane $h \in H$. Define $r_1, ..., r_d$ as the d *greatest* values in

$$\{\text{rank}_{\boldsymbol{q}_1}(h), \text{rank}_{\boldsymbol{q}_2}(h), ..., \text{rank}_{\boldsymbol{q}_{2^{d-1}}}(h)\}.$$

We say that h *protects* R if

$$\sum_{i=1}^{d} r_i \quad \leq \quad k + d - 1. \tag{5}$$

The sum $\sum_{i=1}^{d} r_i$ is the *aggregated rank* of h on R.

Lemma 17. *If h protects R, h has a $\boldsymbol{q}$-rank at most k for any query $\boldsymbol{q}$ inside R.*

Proof. Any point $\boldsymbol{q}$ in R must be covered by at least one simplex that is defined by d corners of R. Let those corners be $\boldsymbol{q'}_1, \boldsymbol{q'}_2, ...,$ and $\boldsymbol{q'}_d$. By Lemma 16, $\text{rank}_{\boldsymbol{q}}(h) \leq (\sum_{i=1}^{d} \text{rank}_{\boldsymbol{q'}_i}(h)) - (d-1)$. By the definition of $r_1, ..., r_d$, we know $\sum_{i=1}^{d} \text{rank}_{\boldsymbol{q'}_i}(h)) \leq \sum_{i=1}^{d} r_i$. Therefore, $\text{rank}_{\boldsymbol{q}}(h) \leq \sum_{i=1}^{d} r_i - (d-1) \leq k + (d-1) - (d-1) = k$, where the second inequality used (5). □

Space partition. Lemma 17 inspires us to divide $\mathcal{Q}$ into non-overlapping $(d-1)$-dimensional rectangles each protected by a plane in H. The *space partition* algorithm in Figure 11 starts with an empty $\mathcal{S}$ (Line 1) and a queue storing only one rectangle, i.e., $\mathcal{Q}$ itself. In each iteration, the algorithm removes a rectangle R from the queue (Lines 3-4) and checks whether R is protected by a plane in $\mathcal{S}$ (Line 5). If not, it adds to S a plane $h \in H$ that protects R (Lines 6-7). If such an h does not exist, the algorithm splits R into two equi-size rectangles on the dimension where R has the longest extent and enqueue both. When the queue is empty, every possible query falls in a protected rectangle, making it safe to return $\mathcal{S}$ as a k-rank-regret representative (Line 9).

When $\mathcal{Q}$ is finite, the algorithm is guaranteed to terminate due to two observations. First, each split creates strictly smaller rectangles. Second, when a rectangle R contains only one query $\boldsymbol{q}$, the plane $h \in H$ with $\boldsymbol{q}$-rank 1 definitely protects R (in this case, $r_1 = r_2 = ... = r_d = 1$ and, hence, (5) holds). $\mathcal{Q}$ is finite in practice because a weight representation has a bounded precision (e.g., 64 bits) in a computer.

algorithm random-kset(P)
1. $\mathcal{K} = \emptyset$
   repeat
2. generate a weight vector $\boldsymbol{w}$ following $\mathcal{D}$
3. obtain the k-set $P_{\boldsymbol{w}}(k)$
4. **if** $P_{\boldsymbol{w}}(k) \notin \mathcal{K}$ **then** add $P_{\boldsymbol{w}}(k)$ to $\mathcal{K}$
   /* note: if $P_{\boldsymbol{w}}(k) \in \mathcal{K}$, we say that $\boldsymbol{w}$ is *captured* */
5. **until** *slen* $= O(\log n)$ queries are captured in a row
6. return $\mathcal{K}$

Fig. 12. The random-kset algorithm

7 LEVERAGING A KNOWN QUERY DISTRIBUTION

In this section, we assume a known distribution $\mathcal{D}$ for the user-specified weight vector $\boldsymbol{w}$ and present a simple algorithm to complement Theorem 15.

Recall that the main deficiency of Theorem 15 lies in enumerating all the k-sets. The *random-kset* algorithm in Figure 12 alleviates the issue by utilizing the knowledge of $\mathcal{D}$. At the beginning, the algorithm initializes an empty $\mathcal{K}$ (Line 1), which at the end will contain the k-sets found. In each iteration, it samples a weight vector $\boldsymbol{w}$ from $\mathcal{D}$ and retrieves its k-set $P_{\boldsymbol{w}}(k)$ (Lines 2-3). If $P_{\boldsymbol{w}}(k)$ is already in $\mathcal{K}$, we say that $\boldsymbol{w}$ is *captured*; otherwise, we add $P_{\boldsymbol{w}}(k)$ to $\mathcal{K}$ (Line 4). The algorithm terminates after the current $\mathcal{K}$ captures *slen* weight vectors in a row (Line 5). The $\mathcal{K}$ returned at Line 6 is then fed to the hitting set approach to compute the final representative $\mathcal{S}$.

By setting $slen = (\ln n)/\ln \frac{1}{1-\delta} \approx \frac{1}{\delta} \ln n$, the following holds with probability at least $1 - 1/n$:

$$\mathbf{Pr}_{\boldsymbol{w}\sim\mathcal{D}}[\mathcal{S} \cap P_{\boldsymbol{w}}(k) \neq \emptyset] \quad \geq \quad 1 - \delta. \tag{6}$$

To see why, suppose that (6) is not true, namely, $\mathbf{Pr}_{\boldsymbol{w}\sim\mathcal{D}}[\mathcal{S} \cap P_{\boldsymbol{w}}(k) = \emptyset] \geq \delta$. Remember that $\mathcal{S}$ hits all the k-sets in the $\mathcal{K}$ returned by *random-kset*. Thus, $\mathcal{S} \cap P_{\boldsymbol{w}}(k) = \emptyset$ implies $P_{\boldsymbol{w}}(k) \notin \mathcal{K}$. By combining all these, we know that $\mathcal{K}$ fails to capture a $\boldsymbol{w}$ drawn from $\mathcal{D}$ with probability at least δ. However, in that case, the probability for $\mathcal{K}$ to capture *slen* independent weight vectors continuously should be very slim. Indeed, the probability is at most $1/n$ under our choice of *slen*.

8 EXPERIMENTS

After providing the algorithms and rigorous theoretical analysis, in this section we present comprehensive experiments to evaluate our proposal in practical scenarios. To do so, using real datasets, we first provide a proof-of-concept experiment that highlights the motivation of finding rank-regret representatives. We will then turn our attention to evaluating the performance of different algorithms under various settings.

8.1 Experiments Setup

Datasets. We used three real datasets in the experiments. All values were normalized into the range $[0, 1]$ and discretized into granularity of 0.01.

- *BlueNile (BN) dataset*[2]*:* Blue Nile is the largest online diamond retailer in the world. We collected its catalog that contained 116,300 diamonds at the time of our collection. We

[2]www.bluenile.com/diamond-search?

considered the scalar attributes `Carat`, `Depth`, `LengthWidthRatio`, `Table`, and `Price`. For all attributes, except `Price`, higher values were preferred. The value of diamonds is sensitive to these measurement such that small changes in scores may mean a lot in terms of the quality of the jewel. For example, while the listed diamonds at Blue Nile range from 0.23 carat to 20.97 carat, minor changes in the carat affect the price. We considered two similar diamonds, where one was 0.5 carat in weight and the other was 0.53 carat. Even though all other measures were similar, the second diamond was 30% more expensive than the first one. This is also true for `Depth`, `LengthWidthRatio`, and `Table`. The phenomenon that *slight changes in the scores may dramatically affect the value* (and the rank) of the items highlights the motivation of rank-regret.

- *US Department of Transportation (DoT) flight dataset*[3]*:* This database is widely used by third-party websites to identify the on-time performance of flights, routes, airports, and airlines. After removing the records with missing values, the dataset contains 457,892 records, for all flights conducted by the 14 US carriers in the last months of 2017; we consider the scalar attributes `Dep-Delay`, `Taxi-Out`, `Actual-elapsed-time`, `Arrival-Delay`, `Air-time`, and `Distance` for our experiments.

- *Wine dataset*[4]*:* Each year, Wine Spectator publishes a list of top wines reviewed over the past 12 months. This annual list honors successful wineries, regions, and vintages around the world. We collected their list of top wines for 2017. The dataset contained 100 items, defined over the attributes `rating`, `vintage year`, and `price`. We will use this dataset in Section 8.2 for validating our proposal.

As mentioned, BN and DoT datasets are 5D and 6D in their entirety. We generated d-dimensional versions (where $d \in [2, 5]$ for BN and $d \in [2, 6]$ for DoT) of each dataset by including the first d attributes in the order mentioned earlier.

Algorithms Evaluated. We will evaluate all the algorithms proposed in this paper under different settings. Specifically, we will present two sets of experiments for the two dimensional (2D) and multi-dimensional cases (MD) where $d \geq 3$, involving the following algorithms:

- *net-extreme-skyline (2D, MD):* Proposed in Section 3, this algorithm uses sampling to construct an ϵ-net. It further shrinks the size of the ϵ-net by removing the dominated items. Following Theorem 4 and Lemma 3, using a sample size of $O(\frac{n}{k} \log n)$ the algorithm returns a k-representative set with probability at least $1 - 1/n^2$.

- *exact (2D):* The exact 2D algorithm works based on Theorem 7 and finds an optimal k-representative by constructing an envelop chain as a sequence of line segments in $\tilde{O}(nk)$ time.

- *shallow-cutting (2D):* The shallow-cutting algorithm, proposed in Section 5.2, provides a 2D regret-approximation algorithm for finding a $(2 + \delta)k$-representative of size at most OPT, where $\delta > 0$ can be an arbitrarily small constant, in $O(n \log n)$ time. The value of δ was set to 1, i.e., the regret approximation ratio was 3.

- *k-set (2D, MD):* The k-set algorithm is a size-approximation algorithm that provides a k-representative of size at most $OPT \cdot O(\log OPT)$ by first enumerating the k-sets and modeling

[3]www.transtats.bts.gov/DL_SelectFields.asp?
[4]http://top100.winespectator.com/lists/

the problem as an instance of hitting set. We will see in our experiments that this algorithm is expected to perform well when k is small.

- *rand-k-set (2D, MD):* Due to the high complexity of enumerating the k-sets, the randomized algorithm in Section 7 serves as a practical alternative for enumerating the k-sets. Algorithm rand-k-set is the same as the k-set algorithm, except that the former uses the randomized algorithm for enumerating the k-sets. The distribution $\mathcal{D}$ was set to *uniformity* for rand-k-set, which essentially says that we aimed to capture all weight vectors, instead of biasing towards particular vectors. The parameter δ for rand-k-set was set to 0.01.

- *space-partitioning (2D, MD):* The space-partitioning algorithm works based on the rank sum lemma proposed in Section 6.1. As will be shown in the experiments, this algorithm is expected to perform well as long as k is not excessively small.

We preceded each of the above methods with a preprocessing step to shrink the input P of Problem 1. As defined in Section 3, an object o *dominates* another one o' if $o[i] \geq o'[i]$ for all $i \in [1, d]$. The *k-skyband* of P includes every object $o \in P$ that is dominated by at most $k-1$ other objects in P. The k-set of any weight vector must be fully contained in the k-skyband [48]. Therefore, as opposed to P itself, we can solve Problem 1 on its k-skyband instead, which is usually much smaller. For any fixed dimensionality d, the k-skyband can be found in $\tilde{O}(n)$ time [52].

Evaluation measurements. We will evaluate the algorithms using three measures: (i) time, (ii) representative size, and (iii) rank-regret. Time evaluates the efficiency of an algorithm, while representative size and rank-regret measures evaluate how effective the algorithm is in finding good and compact representatives.

Default values. In every experiment, we vary one parameter while fixing the other parameters to the following default values: $k = 8$, $d = 4$, and $n = 116,300$ for BN and $457,892$ for DoT.

8.2 Proof of Concept

Users of a dataset with multiple attributes may have different preferences for finding the "best" fit for their need. A Pareto-optimal set contains the best of any preference function but can be very large. To reduce the output size, the *regret* approach returns a small subset that bounds the loss of quality for all users. While regret-ratio defines regret on the score, rank-regret does it on ranks. Recall that a subset achieves a regret-ratio of δ for a scoring function f if $\frac{m_{sub}-m_{all}}{m_{sub}} \leq \delta$, where m_{all} is the maximum score of the objects in dataset under f and m_{sub} is the maximum score of the objects in the aforementioned subset. The subset's rank-regret under f, on the other hand, is k if it contains one of the top-k objects based on f in the dataset.

As mentioned in Section 1, an issue with defining regret by score is that the actual scores assigned by functions are usually artificially "made up" and do not carry a significant meaning. Furthermore, there might even be non-linearity in the raw attribute values themselves. That is, uniform changes in attribute values may not uniformly change the "quality" of an item with that value. These motivate the rank difference as an alternative for defining regret. In this experiment, we use our wine dataset to showcase the phenomenon in real world.

Consider a user who prefers the wine with maximum rating. Wine ratings are in the scale of 0 to 100. In our Wine dataset, the wine with the highest rating is "Clos des Papes Châteauneuf-du-Pape" whose rating is 98. A regret of 6 points on the rating gives a small regret-ratio of 0.06. The small

regret-ratio indicates the containment of a “good” representative for the user’s choice: note that any subset that contains a wine with rating of 92 satisfies this regret-ratio. However, a wine with this rating is even below the median of the dataset based on ranking! An example of such a wine is "Volver Alicante Tarima Hill Old Vines". On the other hand, consider a subset that satisfies the rank-regret of top-6 (top-6%, in other words). Such a subset should contain one of the top 6 wines based on rating, a good approximation for the top-1. "Cantina del Pino Barbaresco Ovello" (with rating of 97) is such a good representative.

A similar story holds for a ranking function that considers the combination of vintage year and rating with equal weights on the normalized values. In this case, an item that satisfies the small regret-ratio of 0.05 falls in the middle of the ranked list, i.e., half of the wines in the dataset approximate the top choice based on this function better than that item.

8.3 Performance Evaluation

Having provided the proof of concept, we proceed to evaluate the performance of our algorithms under different settings. As explained in Section 8.1, in every experiment, we study the impact of varying one parameter, while fixing the other parameters to their default values. We start the experiments by evaluating the 2D algorithms and will then move to MD where $d \geq 3$.

2D, varying k. The value of k greatly impacts the ability to reduce the size of the rank-regret representative. For example, when $k = 1$, all the items on the boundary of the convex hull appear in the representative. As the value of k increases, it gives us the freedom to have more choices for every ranking function, hence more opportunity to find items that cover large portions of the ranking functions. Besides the size of the representative, the running time of the algorithms may also depend on the choice of k. Therefore, as the first 2D experiment, we vary the value of k while fixing other parameters to their default values. The results are provided in Figures 13 to 18.

Figures 13 and 14 show the time taken by each algorithm to find a representative. First, one can observe that the k-set algorithm was significantly slower than all other algorithms and its running time rapidly increased as the value of k increased. The reason for the algorithm’s bad running time is that it requires enumerating all the k-sets before solving a hitting set problem. Therefore, the running time of the algorithm significantly depends on the number of k-sets. The number of k-sets, on the other hand, depends on the value of k. As observed in the experiments on both the BN and DoT datasets, the increase in the value of k resulted in a significant increase in the number of k-sets, causing the poor running time of the algorithm. Even though the rest of the algorithms did not have running times as bad as k-set, still rand-k-set had a worse running time and it got worse as k increased. The main reason why rand-k-set outperformed k-set in the running time is that, compared to the graph enumeration approach for finding the k-sets, rand-k-set used a more efficient randomized algorithm for the same purpose. We also note that, at least theoretically, rand-k-set may miss the k-sets of certain weight vectors, resulting in a potentially smaller number of k-sets in some cases. Among other algorithms, the exact 2D algorithm, even though initially fast, took noticeably more time than the others as k increased. Net-extreme-skyline (labeled `nes` in the legend) had a stable running time (but not the fastest) for different values of k. The shallow-cutting and space-partitioning algorithms (labeled `sc` and `sp` in the legend) had similar running time and both were significantly faster than all other algorithms across all values of k. We note that shallow-cutting is an algorithm specifically designed for 2D, while space-partitioning works for arbitrary dimensionalities.

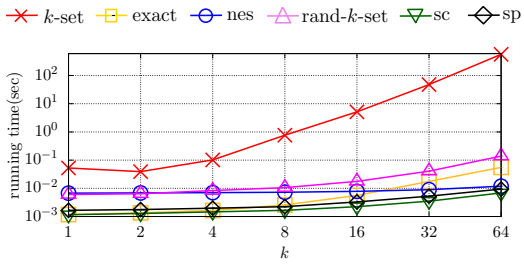

Fig. 13. BN, 2D: impact of varying k on time

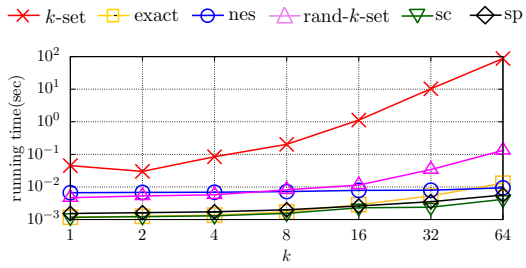

Fig. 14. DoT, 2D: impact of varying k on time

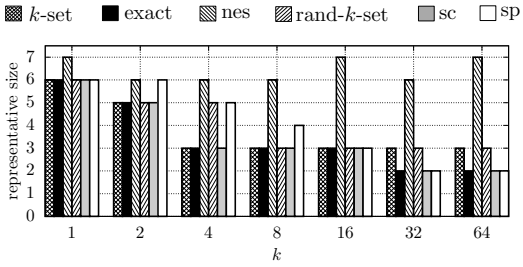

Fig. 15. BN, 2D: impact of varying k on output size

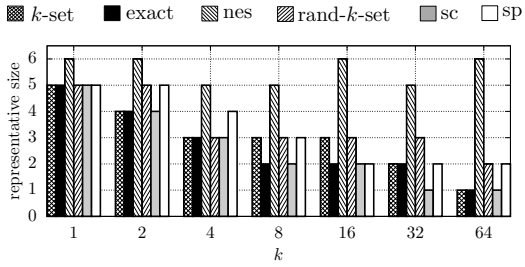

Fig. 16. DoT, 2D: impact of varying k on output size

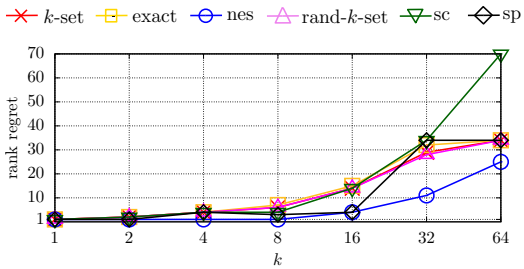

Fig. 17. BN, 2D: impact of varying k on output rank regret

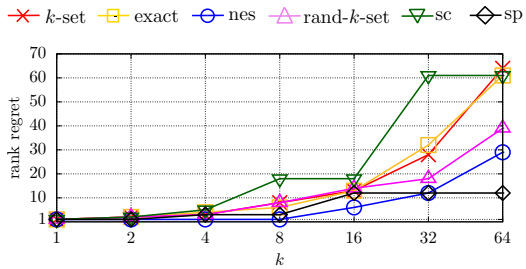

Fig. 18. DoT, 2D: impact of varying k on output rank regret

Figures 15 and 16 show the size of the output (representative set) found by each algorithm, while Figures 17 and 18 show the rank-regret of the output. Please note that the exact algorithm guarantees to the optimal set (i.e., minimum size), while the output of the other algorithms is approximate. Among the approximation algorithms, net-extreme-skyline consistently returned the largest sets but its output always satisfied the rank-regret of k. The outputs of all other algorithms were very close to optimality, a strong indication that they are effective in finding compact sets. The k-set and space partitioning algorithms always guarantee the rank-regret of k, rand-k-set and net-extreme-skyline ensure the guarantee with very high probability, and shallow cutting was parameterized for a 3-approximate assurance on rank-regret. By comparing the exact algorithm with all other algorithms in Figures 17 and 18, one can notice that, interestingly, except shallow-cutting, the output of all algorithms satisfied the rank-regret of k. In fact, the same is nearly true for shallow-cutting whose rank regret was always bounded by k, except in a single case (BN, $k = 64$),

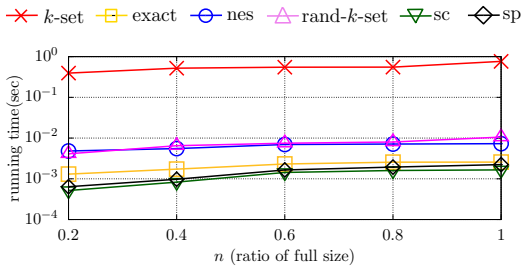

Fig. 19. BN, 2D: impact of varying n on time

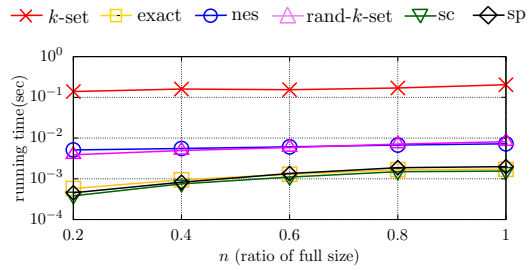

Fig. 20. DoT, 2D: impact of varying n on time

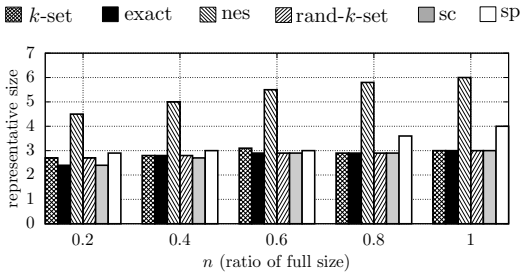

Fig. 21. BN, 2D: impact of varying n on output size

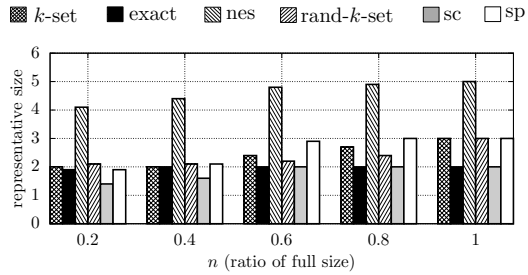

Fig. 22. DoT, 2D: impact of varying n on output size

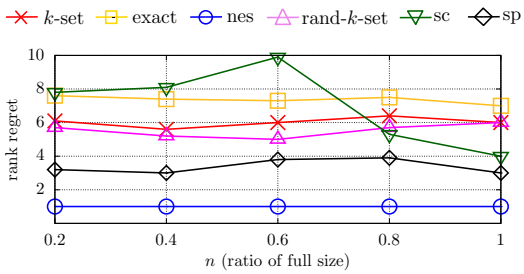

Fig. 23. BN, 2D: impact of varying n on output rank regret

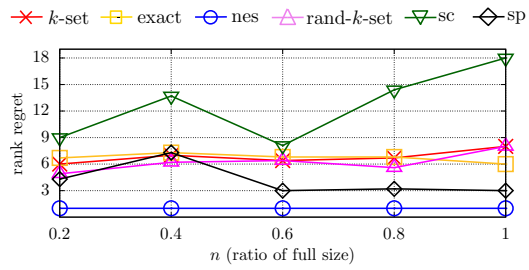

Fig. 24. DoT, 2D: impact of varying n on output rank regret

2D, varying the dataset size (n). Rank regret representatives are compact representatives that are intended to be significantly smaller than the dataset size. The connection to ϵ-nets (Section 3) provides an upper-bound on the size of the representative set which, however, needs to be $1/k$ of the original dataset. Therefore, our earlier results in Figures 15 and 16 suggest that traditional sampling approaches for finding an ϵ-net are not necessarily effective in practical scenarios. Our objective is to find the minimal set that satisfies the rank regret constraint. Recall that Section 3 also gave a lower-bound n/k on the size of any ϵ-net in the worst case. Despite this negative result, Figures 15 and 16 indicate that this lower bound can be excessively pessimistic on real data. To further demonstrate these phenomena, in the next experiment, we varied the dataset size, while observing the algorithms performance, rank-regret, and the output size for each algorithm and setting.

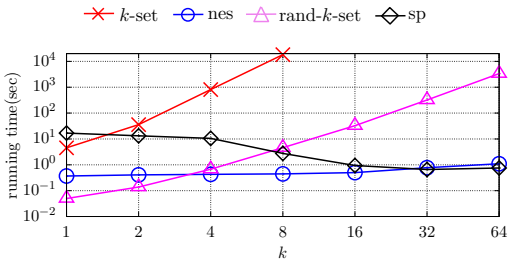

Fig. 25. BN, MD: impact of varying k on time

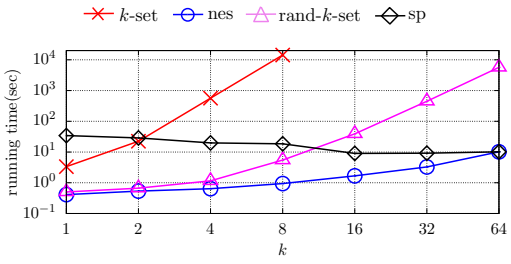

Fig. 26. DoT, MD: impact of varying k on time

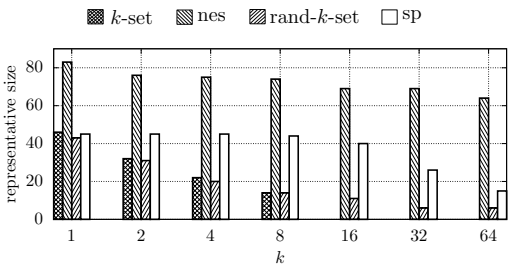

Fig. 27. BN, MD: impact of varying k on output size

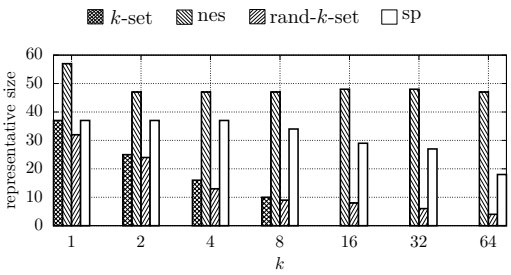

Fig. 28. DoT, MD: impact of varying k on output size

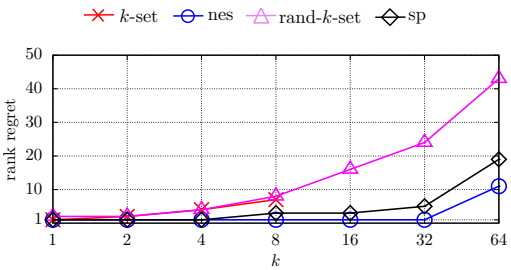

Fig. 29. BN, MD: impact of varying k on rank regret

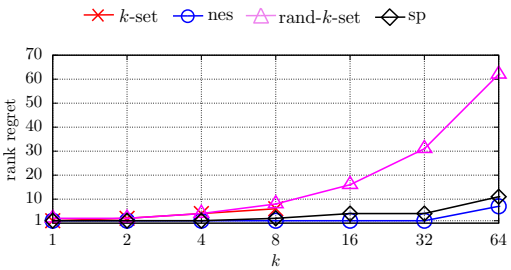

Fig. 30. DoT, MD: impact of varying k on rank regret

The results are provided in Figures 19 to 24. For every dataset, we controlled n by randomly selecting 20%, 40%, 60%, 80%, and 100% of the data. First, as in Figures 19 and 20, the running time of the algorithms was stable as the value of n increased. Among different algorithms, k-set had the longest running time and shallow-cutting had the least. Figures 21 and 22 show the output size for the BN and DoT datasets, while Figures 23 and 24 show the rank-regret of the generated results obtained by different algorithms. Similar to the previous experiments, the output of net-extreme-skyline had the maximum size, while the others were close to the optimum (the output size of the exact algorithm). Furthermore, all algorithms returned representatives achieving a rank-regret of k, except shallow-cutting, which guaranteed 3-approximation. These observations imply that all algorithms except shallow-cutting found near-optimal solutions. A perhaps more important observation is that even though the theoretical lower bound from the ϵ-net interpretation suggests that the output size should be only a constant factor smaller than the dataset (recall $k = 8$, the default value, here), in practice this number may be only a handful and hardly increase with n.

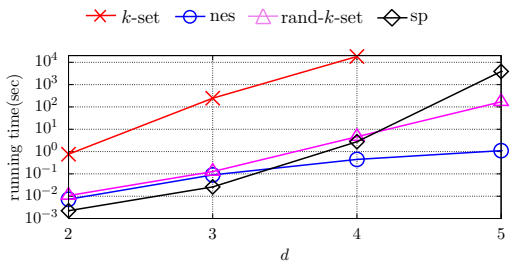

Fig. 31. BN, MD: impact of varying d on time

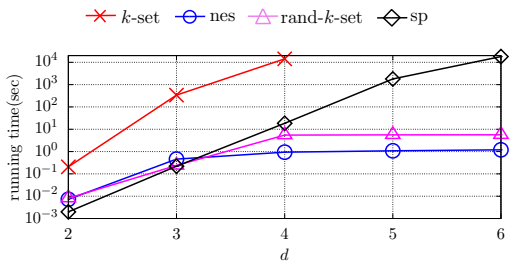

Fig. 32. DoT, MD: impact of varying d on time

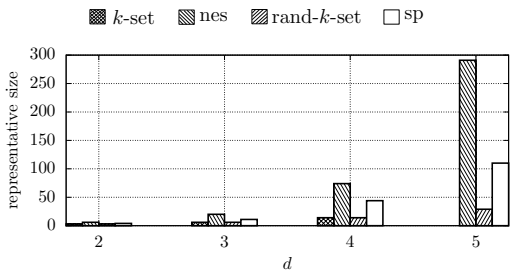

Fig. 33. BN, MD: impact of varying d on output size

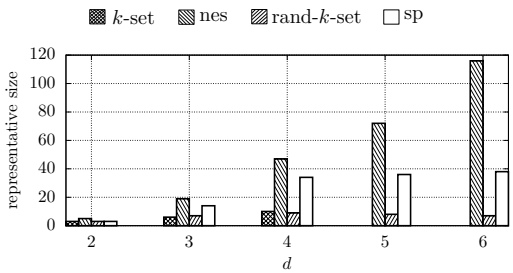

Fig. 34. DoT, MD: impact of varying d on output size

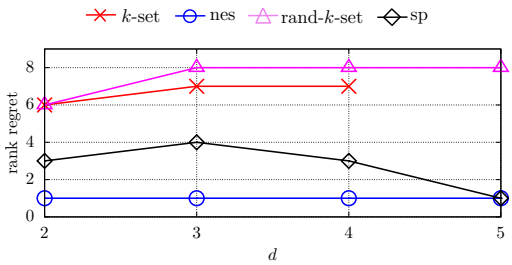

Fig. 35. BN, MD: impact of varying d on rank regret

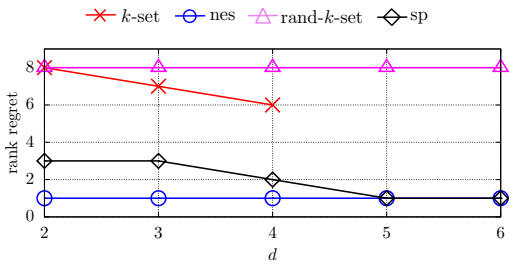

Fig. 36. DoT, MD: impact of varying d on rank regret

MD, varying k. After evaluating the 2D solutions, we now turn our attention to MD where $d \geq 3$. In the upcoming experiment, we study the impact of varying the value of k on the performance of different MD algorithms. Figures 25 to 30 show the results across different settings for the BN and DoT datasets.

First, looking at the running time of the algorithms in Figures 25 and 26, net-extreme-skyline was the fastest across different cases, while k-set did not scale well with k. An interesting observation, however, is that while the running time of k-set and rand-k-set monotonically *increased* with k, that of space-partitioning actually *decreased* as k went up. The reason for the increase in the running time of k-set and rand-k-set is that (assuming $k < n/2$) the number of k-sets escalates as k increases. This forces both algorithms to spend more time enumerating the k-sets and solving the hitting set problem. For larger k, however, the space-partitioning algorithm finds more opportunity to prune the search space, simply because it essentially looks for common elements in larger sets, which are the top-k results (which are supersets of the results of smaller k). These observations

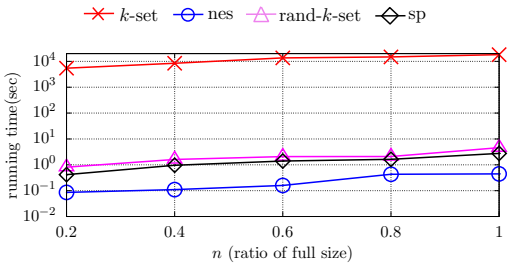

Fig. 37. BN, MD: impact of varying n on time

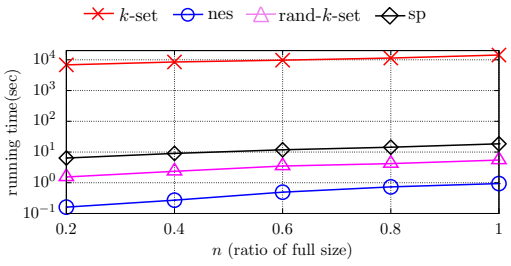

Fig. 38. DoT, MD: impact of varying n on time

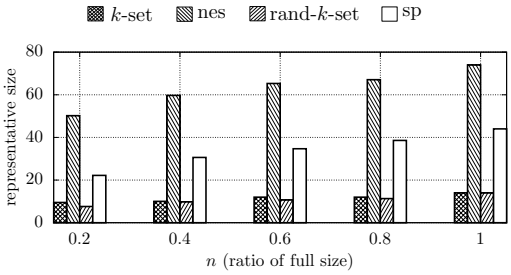

Fig. 39. BN, MD: impact of varying n on output size

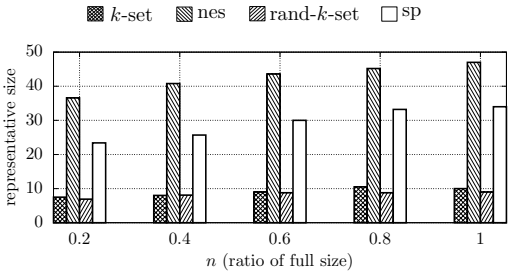

Fig. 40. DoT, MD: impact of varying n on output size

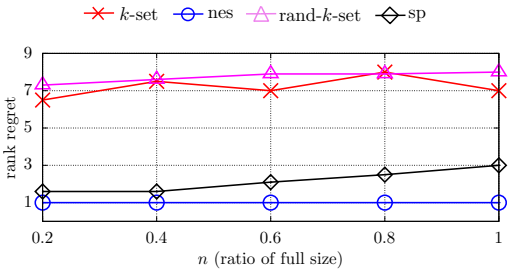

Fig. 41. BN, MD: impact of varying n on rank regret

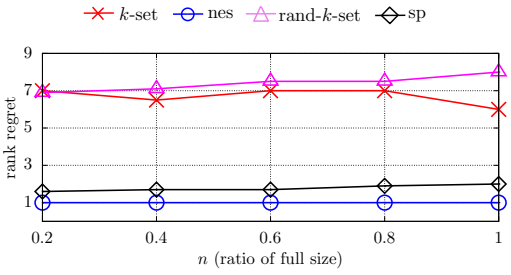

Fig. 42. DoT, MD: impact of varying n on rank regret

together indicate that (rand-)k-set and space partitioning are *complimentary algorithms* for finding rank-regret representatives in different settings.

Next, we studied the output size (Figures 27 and 28) and rank-regret (Figures 29 and 30) for the BN and DoT datasets. Recall that, in theory, the space partitioning and k-set algorithms guarantee the rank-regret of k, while rand-k-set and net-extreme-skyline guarantee the same with very high probability. However, in all settings across the two datasets, every algorithm managed to find k-rank representatives. The net-extreme-skyline algorithm, in spite of being fast, failed to find compact representatives, especially as k increases. The rand-k-set and k-set algorithms generated the smallest outputs, and the size of their representative decreased as k increased. In particular, for all settings with $k > 10$ in both datasets the output size was always less than 10, fully echoing the motivation of rank-regret representatives.

MD, varying the number d of dimensions. In this experiment, we evaluate different MD algorithms for different values of d. The results are provided in Figures 31 to 36.

Let us first look into the running time of the algorithms across different settings (Figures 31 and 32). The k-set algorithm failed to scale beyond four dimensions because the exact (graph-traversal) algorithm in Section 5.5 for enumerating the k-sets did not finish within the time budget (20,000 seconds). In contrary, the rand-k-set algorithm (being efficient in finding the k-sets) scaled much better with respect to d. The time performance of the space-partitioning algorithm worsened as d became larger, due to the curse of dimensionality, i.e., significant enlargement in the search space. The net-extreme-skyline had the best time performance but, as discussed next, it failed to find compact representatives. Figures 33 and 34 show the output size, and Figures 35 and 36 show the rank-regret of the generated output for the BN and DoT datasets. Similar to the previous experiments, all algorithms were able to ensure the rank-regret of $k = 8$ across different settings. Evidently, the representative sets of net-extreme-skyline were fairly large, especially as d increased, while (rand-)k-set managed to secure small representatives in all cases.

MD, varying the dataset size n. Finally, we conclude our experiments by studying the impact of varying the dataset size on the performance of different algorithms. To do so, similar to the corresponding 2D experiments, we selected 20% to 100% of the BN and DoT datasets for the default values of $k = 8$ and $d = 4$. The results are provided in Figures 37 to 42.

Looking at Figures 37 and 38, one can see that, even as the value of n increased, all algorithms had a stable running time for all values of n. Also, looking at Figures 41 and 42, one can see that the output of all algorithms satisfied the rank-regret requirement ($k = 8$) in all settings, as is consistent with the previous experiments. As shown in Figures 39 and 40, the output of net-extreme-skyline was the largest, while (rand-)k-set could find representatives with size around 10 in all the scenarios.

9 RELATED WORK

The problem of finding preferred items of a data set has been extensively investigated in recent years, and research has spanned multiple directions, most notably in top-k query processing [36] and skyline discovery [11]. In top-k query processing, the approach is to model the user preferences by a ranking/utility function which is then used to preferentially select tuples. Fundamental results include access-based algorithms [12, 29, 30, 41] and view-based algorithms [20, 35]. In skyline research, the approach is to compute subsets of the data (such as skylines and convex hull points) that serve as the data representatives in the absence of explicit preference functions [9, 11, 50]. Skylines and convex hull points can also serve as effective indexes for top-k query processing [8, 17, 55].

Efficiency and effectiveness have always been the challenges in the above studies. While top-k algorithms depend on the existence of a preference function and may require a complete pass over all of the data before answering a single query, representatives such as skylines may become overwhelmingly large and ineffective in practice [7, 33]. Studies such as [14, 54] are focused towards reducing the skyline size. In an elegant effort towards finding a small representative subset of the data, Nanongkai et al. [46] introduced the regret-ratio minimizing representative. The intuition is that a "close-to-top" result may satisfy the users' need. Therefore, for a subset of data and a preference function, they consider the score difference between the top result of the subset versus the actual top result as the measure of regret, and seek the subset that minimizes its maximum regret over all possible linear functions. Since then, works such as [4, 7, 13, 37, 38, 45, 49, 56] studied different challenges and variations of the problem. As discussed in Section 4.1, Chester et

al. [18] generalize the regret-ratio notion to k-regret ratio, and Agarwal et al. [4] prove that the k-regret minimizing set problem is NP-complete even when $d = 3$. For the case of two dimensional databases, [18] proposes a quadratic algorithm. The cube algorithm and a greedy heuristic [46] are the first algorithms proposed for regret-ratio in dimensionality $d \geq 3$. Recently, [4, 7] independently propose similar approximation algorithms for the problem, both discretizing the function space and applying the hitting set, thus, providing similar controllable additive approximation factors. The major difference is that [7] considers the original regret-ratio problem while [4] considers the k-regret variation. It is important to note that the above prior works consider the score difference as the regret measure, making their problem setting different from ours, since we use the rank difference as the regret measure.

Lovasz and Erdos [27, 40] initiated the study of k-set notion and provided an upper bound on the maximum number of k-sets in $\mathbb{R}^2$. Finding an upper bound on the number of k-sets has also been studied in [22, 25, 26, 47, 53] for $d = 2$ and in [5, 23, 24, 51, 53] for $d \geq 3$. The problem of enumerating all k-sets has been studied in [28] for 2D and [5, 6, 22, 51] for higher dimensions.

The geometric notions used in this paper, such as arrangement, dual space, ϵ-nets, shallow cuttings, and extreme sets are explained in detail in [2, 15, 16, 21, 24, 34, 39, 42, 43].

10 CONCLUSIONS

In this paper, we proposed a rank-regret measure that is easier for users to understand, and often more appropriate, than regret computed from score values. We defined *rank-regret representative* as the minimal subset of the data containing at least one of the top-k of any possible ranking function. Our systematic study contains an optimal polynomial time algorithm in 2D space, an NP-hardness proof in 3 or more dimensions, approximation algorithms of various dimensionalities under different approximation schemes, a space-partition algorithm leveraging an interesting rank-sum lemma, and a randomized algorithm utilizing the knowledge of query distribution. In addition to theoretical analyses, we conducted empirical experiments on real data that verified the effectiveness and efficiency of our techniques. The proposed algorithms nicely complement each other and together constitute an adequate set of solutions covering a great variety of practical scenarios.

APPENDIX

PROOF OF LEMMA 1

Clearly, $\text{MRR}(S) \geq \max_{\boldsymbol{w} \in \mathscr{W}_{[d]\neq 0}} RR_{\boldsymbol{w}}(S)$ because $\mathscr{W}_{[d]\neq 0} \subset \mathscr{W}$. The subsequent discussion will show $\max_{\boldsymbol{w} \in \mathscr{W}_{[d]\neq 0}} RR_{\boldsymbol{w}}(S) \geq \text{MRR}(S)$, which will establish the lemma.

Set $r = \text{MRR}(S)$; and suppose that $\max_{\boldsymbol{w} \in \mathscr{W}_{[d]\neq 0}} RR_{\boldsymbol{w}}(S) < r$. There must exist $\boldsymbol{w^*} \in \mathscr{W}$ with $\boldsymbol{w^*}[d] = 0$ such that $RR_{\boldsymbol{w^*}}(S) = r$.[5] Let $o^* \in S$ be an object that has the highest $\boldsymbol{w^*}$-score in S; clearly, $\text{rank}_{\boldsymbol{w^*}}(o^*) = r$. P must have $r - 1$ objects $o_1, o_2, ..., o_{r-1}$ outside S such that the $\boldsymbol{w^*}$-score of o_j ($j \in [1, r-1]$) is strictly larger than that of o^*.

Construct $\boldsymbol{w'} \in \mathscr{W}_{[d]\neq 0}$ where $\boldsymbol{w'}[i] = \boldsymbol{w}[i]$ for each $i \in [1, d-1]$ and $\boldsymbol{w'}[d] = \delta$ where $\delta > 0$ is infinitesimally small. As $\boldsymbol{o_j} \cdot \boldsymbol{w^*} > \boldsymbol{o^*} \cdot \boldsymbol{w^*}$ for every $j \in [1, r-1]$, a sufficiently low δ ensures $\boldsymbol{o_j} \cdot \boldsymbol{w'} > \boldsymbol{o^*} \cdot \boldsymbol{w'}$. In other words, the $\boldsymbol{w'}$-rank of o^* is at least r.

[5] Otherwise, every weight vector $\boldsymbol{w}$ achieving $RR_{\boldsymbol{w}}(S) = \text{MRR}(S)$ must fall in $\mathscr{W}_{[d]\neq 0}$, implying $\max_{\boldsymbol{w} \in \mathscr{W}_{[d]\neq 0}} RR_{\boldsymbol{w}}(S) \geq \text{MRR}(S)$).

By the assumption $\max_{\boldsymbol{w}\in\mathscr{W}_{[d]\neq 0}} RR_{\boldsymbol{w}}(S) < r$, we know $RR_{\boldsymbol{w}'}(S) \leq r-1$. Hence, S must have an object o' whose $\boldsymbol{w}'$-rank is at most $r-1$. This object must have a $\boldsymbol{w}'$-score higher than that of o^*, namely:

$$\left(\sum_{i=1}^{d-1} o'[i]\cdot \boldsymbol{w}^*[i]\right) + o'[d]\cdot\delta \quad > \quad \left(\sum_{i=1}^{d-1} o^*[i]\cdot \boldsymbol{w}^*[i]\right) + o^*[d]\cdot\delta.$$

Using the definition of o^* and δ being infinitesimally small, we can assert $\sum_{i=1}^{d-1} o'[i]\cdot \boldsymbol{w}^*[i] = \sum_{i=1}^{d-1} o^*[i]\cdot \boldsymbol{w}^*[i]$, namely, o' and o^* have the same $\boldsymbol{w}^*$-score.

Given $\boldsymbol{o_j}\cdot\boldsymbol{w}^* > \boldsymbol{o}^*\cdot\boldsymbol{w}^* = \boldsymbol{o'}\cdot\boldsymbol{w}^*$ for every $j\in[1,r-1]$, we conclude that $\boldsymbol{o_j}\cdot\boldsymbol{w'} > \boldsymbol{o'}\cdot\boldsymbol{w'}$ for a sufficiently small $\delta > 0$. This means that P has at least $r-1$ objects whose $\boldsymbol{w}'$-scores are strictly higher than that of o', which contradicts $\text{rank}_{\boldsymbol{w}'}(o') \leq r-1$.

PROOF OF LEMMA 6

It is obvious that $H[C^*]$ hits all possible queries and, hence, is a legal solution to Problem 2. Next, we will prove that *every* optimal solution $\mathcal{S}$ to Problem 2 defines an envelop chain C with $H[C] = S$ such that the x-projection of C covers $\mathcal{Q}$. This implies the correctness of the lemma.

The upper boundary of the (≤ 0)-level of $\mathcal{S}$ must be an envelop chain C. Furthermore, every $h\in\mathcal{S}$ must contribute an edge to the (≤ 0)-level; otherwise, h is completely above C, because of which $\mathcal{S}\setminus\{h\}$ must still hit all the queries, giving a contradiction to the optimality of $\mathcal{S}$. C is thus the envelop chain promised.

PROOF OF THEOREM 13

We use Lemma 8 to obtain a $(\delta,(k-1)/n)$-shallow cutting Ξ. For each $\Delta\in\Xi$, define XY_Δ as the xy-projection of Δ. Each plane $h\in H_\Delta$ (where H_Δ is the conflict set of Δ; see Section 5.1) intersects Δ into a polygon, whose xy-projection is represented as $XY_h(\Delta)$.

The polygons $XY_h(\Delta)$ of all $h\in H_\Delta$ induce an *arrangement*, which is a set $\mathcal{A}_\Delta$ of $O(k^2)$ polygons in the xy-plane satisfying

- the union of all the polygons in $\mathcal{A}_\Delta$ is XY_Δ, and
- for every $h\in H_\Delta$ and every polygon $A\in\mathcal{A}_\Delta$, $XY_h(\Delta)$ either entirely covers A or is non-overlapping with A.

Define

$$\mathcal{A} \quad = \quad \bigcup_{\Delta\in\Xi} \mathcal{A}_\Delta.$$

The polygons in $\mathcal{A}$ are non-overlapping; and their union is precisely $\mathbb{R}^2$.

Given a plane $h\in H$, define $\Xi(h)$ as the set of prisms in Ξ intersecting with h, i.e., $\Xi(h) = \{\Delta\in\Xi \mid h\in H_\Delta\}$. Define:

$$Z_h = \left\{A\in\mathcal{A} \;\middle|\; \exists\Delta\in\Xi(h) \text{ such that } A\in\mathcal{A}_\Delta\right\}.$$

Lemma 18. *For any plane $h\in H$ and any query $\boldsymbol{q}$ covered by Z_h, $rank_q(h)\leq(1+\delta)k$.*

Proof. As before, denote by $\ell_{\boldsymbol{q}}$ the vertical line in $\mathbb{R}^d$ that is parallel to dimension d and passes $(\boldsymbol{q}[1],...,\boldsymbol{q}[d-1],-\infty)$. Let p be the intersection between h and $\ell_{\boldsymbol{q}}$. Since $\boldsymbol{q}$ is covered by Z_h, we know that p must be covered by a prism in Ξ. Since the union of all the prisms of Ξ is covered by

the ($\leq (1+\delta)(k-1)$)-level of H, the level of p must be at most $(1+\delta)(k-1)$. This means that $\text{rank}_{\boldsymbol{q}}(h) \leq (1+\delta)k$. □

Consider any optimal solution $\mathcal{S}^*$ to Problem 2. We have:

Lemma 19. *$\bigcup_{h\in\mathcal{S}^*} Z_h$ covers $\mathcal{Q}$.*

Proof. Assume, on the contrary, that the union fails to include a query $\boldsymbol{q} \in \mathcal{Q}$. By definition of $\mathcal{S}^*$, there exists a plane $h \in \mathcal{S}^*$ whose $\boldsymbol{q}$-rank is at most k. Let p be the intersection point between h and $\ell_{\boldsymbol{q}}$. By $\text{rank}_{\boldsymbol{q}}(h) \leq k$, the level of p is at most $k-1$. Now, consider the prism $\Delta \in \Xi$ whose xy-projection covers $\boldsymbol{q}$. We assert that p must fall inside Δ; otherwise, p falls outside the union of all the prisms of Ξ, contradicting the fact that the union must contain the ($\leq k-1$)-level of H. However, Δ covering p implies that $XY_h(\Delta)$ covers $\boldsymbol{q}$, which in turn indicates the existence of an $A \in Z_h$ covering $\boldsymbol{q}$, giving a contradiction. □

We now find a small $\mathcal{S} \subseteq H$ such that $\bigcup_{h\in\mathcal{S}} Z_h$ covers $\mathcal{Q}$; the existence of $\mathcal{S}$ is guaranteed by Lemma 19. It is rudimentary to apply a greedy set cover algorithm over $\{Z_h \mid h \in H\}$ to find an $\mathcal{S}$ with size at most $|\mathcal{S}^*|O(\log n) = OPT \cdot O(\log n)$. As every query must be covered by the Z_h of at least one $h \in \mathcal{S}$, Lemma 18 ensures that $\text{MRR}'(\mathcal{S}) \leq (1+\delta)k$.

To analyze the running time, we observe:

$$\begin{aligned}
& \sum_{h\in H} |Z_h| \\
= & \sum_{A\in\mathcal{A}} \text{number of planes } h \in H_\Delta \text{ s.t. } XY_h(\Delta) \text{ covers } A, \text{ where } \Delta \text{ is the prism with } A \in \mathcal{A}_\Delta \\
\leq & \sum_{A\in\mathcal{A}} |H_\Delta| \text{ where } \Delta \text{ is the prism with } A \in \mathcal{A}_\Delta \\
\leq & \sum_{A\in\mathcal{A}} O(k) \qquad \text{(applying the definition of shallow cutting)} \\
= & \; O(|\mathcal{A}| \cdot k) = O(nk^2)
\end{aligned}$$

where the last equality used $|\mathcal{A}| = O(k^2) \cdot |\Xi|$ and $|\Xi| = O(n/k)$ (Lemma 8). After explicitly generating the Z_h of every $h \in H$, the greedy set-cover algorithm runs in $O(nk^2)$ time. This concludes the proof of Theorem 13.

REFERENCES

[1] [n.d.]. Reference anonymized due to double blind. ([n. d.]).

[2] Peyman Afshani and Timothy M. Chan. 2009. On Approximate Range Counting and Depth. *Discrete & Computational Geometry* 42, 1 (2009), 3–21.

[3] Pankaj K. Agarwal, Mark de Berg, Jiri Matousek, and Otfried Schwarzkopf. 1998. Constructing Levels in Arrangements and Higher Order Voronoi Diagrams. *SIAM Journal of Computing* 27, 3 (1998), 654–667.

[4] Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. 2017. Efficient Algorithms for k-Regret Minimizing Sets. In *International Symposium on Experimental Algorithms*. 7:1–7:23.

[5] Noga Alon, Imre Barany, Zoltan Furedi, and Daniel J. Kleitman. 1992. Point Selections and Weak e-Nets for Convex Hulls. *Combinatorics, Probability & Computing* 1 (1992), 189–200.

[6] Artur Andrzejak and Komei Fukuda. 1999. Optimization over k-set Polytopes and Efficient k-set Enumeration. In *Algorithms and Data Structures Workshop (WADS)*. 1–12.

[7] Abolfazl Asudeh, Azade Nazi, Nan Zhang, and Gautam Das. 2017. Efficient Computation of Regret-ratio Minimizing Set: A Compact Maxima Representative. In *Proceedings of ACM Management of Data (SIGMOD)*. 821–834.

[8] Abolfazl Asudeh, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2016. Discovering the Skyline of Web Databases. *Proceedings of the VLDB Endowment (PVLDB)* 9, 7 (2016), 600–611.

[9] Abolfazl Asudeh, Gensheng Zhang, Naeemul Hassan, Chengkai Li, and Gergely V. Zaruba. 2015. Crowdsourcing Pareto-Optimal Object Finding By Pairwise Comparisons. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*.

[10] Abolfazl Asudeh, Nan Zhang, and Gautam Das. 2016. Query Reranking As a Service. *Proceedings of Very Large Data Bases (VLDB)* 9, 11 (2016).

[11] Stephan Borzsonyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *Proceedings of International Conference on Data Engineering (ICDE)*. 421–430.

[12] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2002. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems (TODS)* (2002).

[13] Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan. 2017. k-Regret Minimizing Set: Efficient Algorithms and Hardness. In *Proceedings of International Conference on Database Theory (ICDT)*. 11:1–11:19.

[14] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. 2006. Finding k-dominant skylines in high dimensional space. In *Proceedings of ACM Management of Data (SIGMOD)*. 503–514.

[15] Timothy M. Chan. 1996. Output-Sensitive Results on Convex Hulls, Extreme Points, and Related Problems. *Discrete & Computational Geometry* 16, 4 (1996), 369–387.

[16] Timothy M. Chan and Konstantinos Tsakalidis. 2016. Optimal Deterministic Algorithms for 2-d and 3-d Shallow Cuttings. *Discrete & Computational Geometry* 56, 4 (2016), 866–881.

[17] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R Smith. 2000. The onion technique: indexing for linear optimization queries. In *Proceedings of ACM Management of Data (SIGMOD)*.

[18] Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. 2014. Computing k-Regret Minimizing Sets. *Proceedings of the VLDB Endowment (PVLDB)* 7, 5 (2014), 389–400.

[19] Kenneth L. Clarkson and Peter W. Shor. 1989. Application of Random Sampling in Computational Geometry, II. *Discrete & Computational Geometry* 4 (1989), 387–421.

[20] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. 2006. Answering top-k queries using views. In *Proceedings of Very Large Data Bases (VLDB)*.

[21] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag.

[22] Tamal K. Dey. 1998. Improved Bounds for Planar k -Sets and Related Problems. *Discrete & Computational Geometry* 19, 3 (1998), 373–382.

[23] Tamal K Dey and Herbert Edelsbrunner. 1993. Counting triangle crossings and halving planes. In *SOCG*. ACM, 270–273.

[24] Herbert Edelsbrunner. 1987. *Algorithms in combinatorial geometry*. Vol. 10. Springer Science & Business Media.

[25] Herbert Edelsbrunner, Nany Hasan, Raimund Seidel, and Xiao Jun Shen. 1989. Circles through two points that always enclose many points. *Geometriae Dedicata* 32, 1 (1989), 1–12.

[26] Herbert Edelsbrunner and Emo Welzl. 1985. On the number of line separations of a finite set in the plane. *Journal of Combinatorial Theory, Series A* 38 (1985).

[27] P Erdős, László Lovász, A Simmons, and Ernst G Straus. 1973. Dissection graphs of planar point sets. *A survey of combinatorial theory* (1973), 139–149.

[28] Hazel Everett, Jean-Marc Robert, and Marc J. van Kreveld. 1993. An Optimal Algorithm for the (<= k)-Levels, with Applications to Separation and Transversal Problems. In *Proceedings of Symposium on Computational Geometry (SoCG)*. 38–46.

[29] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Comparing Top k Lists. *SIAM J. Discret. Math.* 17, 1 (2003), 134–160.

[30] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*.

[31] Yeshwanth Durairaj Gunasekaran, Abolfazl Asudeh, Sona Hasani, Nan Zhang, Ali Jaoua, and Gautam Das. 2018. QR2: A Third-Party Query Reranking Service over Web Databases. In *Proceedings of International Conference on Data Engineering (ICDE)*. 1653–1656.

[32] Sariel Har-Peled. 2011. *Geometric approximation algorithms*. Number 173. American Mathematical Soc.

[33] Sariel Har-Peled. 2011. On the Expected Complexity of Random Convex Hulls. *CoRR* abs/1111.5340 (2011).

[34] David Haussler and Emo Welzl. 1987. Epsilon-Nets and Simplex Range Queries. *Discrete & Computational Geometry* 2 (1987), 127–151.

[35] Vagelis Hristidis and Yannis Papakonstantinou. 2004. Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal* 13, 1 (2004), 49–70.

[36] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surveys* 40, 4 (2008).

[37] Taylor Kessler Faulkner, Will Brackenbury, and Ashwin Lall. 2015. k-regret queries with nonlinear utilities. *The VLDB Journal* 8, 13 (2015).

[38] Nirman Kumar and Stavros Sintos. 2018. Faster Approximation Algorithm for the k-Regret Minimizing Set and Related Problems. In *Proceedings of Workshop on Algorithm Engineering and Experiments (ALENEX)*.

[39] Andrey Kupavskii, Nabil H. Mustafa, and Janos Pach. 2016. New Lower Bounds for epsilon-Nets. In *Proceedings of Symposium on Computational Geometry (SoCG)*. 54:1–54:16.

[40] László Lovász. 1971. On the number of halving lines. *Ann. Univ. Sci. Budapest, Eötvös, Sec. Math* 14 (1971), 107–108.

[41] Amélie Marian, Nicolas Bruno, and Luis Gravano. 2004. Evaluating Top-k Queries over Web-accessible Databases. *ACM Trans. Database Syst.* 29, 2 (2004).

[42] Jiri Matousek. 1992. Reporting Points in Halfspaces. *Comput. Geom.* 2 (1992), 169–186.

[43] Jiri Matousek. 1993. Linear Optimization Queries. *J. Algorithms* 14, 3 (1993), 432–448.

[44] Ketan Mulmuley. 1991. On Levels in Arrangement and Voronoi Diagrams. *Discrete & Computational Geometry* 6 (1991), 307–338.

[45] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino. 2012. Interactive regret minimization. In *Proceedings of ACM Management of Data (SIGMOD)*. ACM.

[46] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J Lipton, and Jun Xu. 2010. Regret-minimizing representative databases. *The VLDB Journal* (2010).

[47] János Pach, William Steiger, and Endre Szemerédi. 1992. An upper bound on the number of planar K-sets. *DCG* 7, 1 (1992), 109–123.

[48] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)* 30, 1 (2005), 41–82.

[49] Peng Peng and Raymond Chi-Wing Wong. 2014. Geometry approach for k-regret query. In *Proceedings of International Conference on Data Engineering (ICDE)*. IEEE.

[50] Md Farhadur Rahman, Abolfazl Asudeh, Nick Koudas, and Gautam Das. 2017. Efficient Computation of Subspace Skyline over Categorical Domains. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*. ACM.

[51] Micha Sharir, Shakhar Smorodinsky, and Gabor Tardos. 2001. An Improved Bound for *k*-Sets in Three Dimensions. *Discrete & Computational Geometry* 26, 2 (2001), 195–204.

[52] Cheng Sheng and Yufei Tao. 2012. Worst-Case I/O-Efficient Skyline Algorithms. *ACM Transactions on Database Systems (TODS)* 37, 4 (2012), 26.

[53] Géza Tóth. 2001. Point sets with many k-sets. *Discrete & Computational Geometry* 26, 2 (2001).

[54] Akrivi Vlachou and Michalis Vazirgiannis. 2010. Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships. *Data Knowledge Engineering (DKE)* 69, 9 (2010).

[55] Dong Xin, Chen Chen, and Jiawei Han. 2006. Towards robust indexing for ranked queries. In *Proceedings of Very Large Data Bases (VLDB)*.

[56] Sepanta Zeighami and Raymond Chi-Wing Wong. 2016. Minimizing average regret ratio in database. In *Proceedings of ACM Management of Data (SIGMOD)*. ACM.