On Finding Rank Regret Representatives

ABOLFAZL ASUDEH, University of Illinois Chicago
GAUTAM DAS, University of Texas at Arlington
H. V. JAGADISH, University of Michigan
SHANGQI LU, Chinese University of Hong Kong
AZADE NAZI, Google Brain
YUFEI TAO, Chinese University of Hong Kong
NAN ZHANG, American University
JIANWEN ZHAO, Chinese University of Hong Kong

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 two-dimensional space but is NP-hard on three 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, and so on. Experiments on real datasets demonstrate that we can efficiently find small subsets with small rank-regrets.

CCS Concepts: • **Theory of computation → Approximation algorithms analysis**; • **Information systems → Top-k retrieval in databases;**

Additional Key Words and Phrases: Rank regret representatives, epsilon-nets, top-k search, computational geometry, spatial databases

Authors' addresses: A. Asudeh, Department of Computer Science, University of Illinois Chicago, Chicago, Illinois; email: asudeh@uic.edu; G. Das, Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, Texas; email: gdas@uta.edu; H. V. Jagadish, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan; email: jag@umich.edu; S. Lu, Y. Tao, and J. Zhao, Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China; emails: {sqlu, taoyf, jwzhao}@cse.cuhk.edu.hk; A. Nazi, Google Brain, Mountain View, California; email: azade@google.com; N. Zhang, Department of Information Technology and Analytics, American University, Washington DC; email: nzhang@american.edu.

0362-5915/2022/08-ART10 $15.00
https://doi.org/10.1145/3531054

ACM Reference format:
Abolfazl Asudeh, Gautam Das, H. V. Jagadish, Shangqi Lu, Azade Nazi, Yufei Tao, Nan Zhang, and Jianwen Zhao. 2022. On Finding Rank Regret Representatives. *ACM Trans. Database Syst.* 47, 3, Article 10 (August 2022), 37 pages.
https://doi.org/10.1145/3531054

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 [12], diamonds based on depth and carat [35], and houses based on price and floor area [35] 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 [11] 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, respectively) ranking function. Since the introduction of skylines to the database community [13], a large body of work has been conducted in this area. A major issue with such representatives is that they can be a large portion of the dataset [8, 37], especially when there are multiple attributes. Hence, several researchers have tackled [17, 61] 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* [8, 51] problem and its variants [3, 16, 22, 41, 43, 50, 63]. 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, in particular, for ranking applications. To see a specific example, let us consider wine ratings.

Example. Each year, Wine Spectator publishes a list of top wines reviewed over the past 12 months.[1] This annual list honors successful wineries, regions, and vintages around the world. Let us consider their 2017 list. The dataset contained 100 items, defined over the attributes `rating`, `vintage year`, and `price`. 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 $6/98 \approx 0.061$. The small regret-ratio indicates the

[1] http://top100.winespectator.com/lists/.

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." A similar story holds for a ranking function that considers the combination of `vintage year` and `rating` with equal weights (after normalizing each attribute to the same scale). In this case, an item satisfying 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 better than that item according to the aforementioned ranking function.

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 article, 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 [8, 37]. 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.

Example (cont.). As explained earlier, a small difference in regret ratio can actually result in a large swing in rank. However, consider a subset that satisfies the rank-regret of 6. Such a subset should contain one of the top 6 (i.e., top 6%, in other words) wines based on rating, which serves as a good approximation for the top-1. "Cantina del Pino Barbaresco Ovello" (with rating of 97) is a good representative.

Before moving to our technical contributions, we would like to underscore the *complimentary* nature of ranks and scores: Neither should be regarded as the absolute winner. There are applications where scores are more meaningful, but there are also those where ranks matter more. One major difference between ranks and scores is that the act of "ranking" has a notion of *direct* competition at its heart, while the sense of competition is subtler with scores. Consider, as two well-known applications, credit scores vs. university rankings. Credit scores are used to evaluate the creditworthiness of an individual and are independent from how others perform. Therefore, any one who has a credit above a specific threshold can be a candidate for a loan. On the contrary, when one is interested in finding a top university, seeing a "high" score is seldom enough, because one must consider how this score stands *in comparison* with other universities. Therefore, while regret-ratio makes sense for credit scores, rank-regret is the proper measure for university rankings.

Rank-regrets have the advantage of being insensitive to *scaling*, as is an important feature, because in practice various dimensions can have drastically different domain lengths (e.g., price measured in dollar vs. ratings measured in percentile). The rank-regret of a subset remains the same no matter how each dimension is scaled, while the ratio-regret of a subset can change significantly. Preprocessing the data with normalization helps only to a rather limited extent, because normalization is only one possible way to scale and it is not clear at all why it is the best way. With

Table 1. Summary of Formal Results

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	$O(nk)$	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 5.5)	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.

another moment of thoughts, one would realize that the root cause is still the fact that the score of an item is not a reliable indication of its competition rank.

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.
(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 two-dimensinal (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 article.

A short version of this article appeared in Reference [9]. Compared to that preliminary work, the current article presents a more comprehensive treatment of the k-rank-regret problem. The new contributions include bullets (2) and (3), new 2D and 3D approximation algorithms in bullet (4), a more powerful rank-sum lemma in bullet (5), and experimentation with all the new algorithms.

The rest of the article 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 leverages a query distribution to discover a good solution, while Section 7 introduces our space partitioning algorithm. Section 8 evaluates our algorithms with extensive experiments. Section 9 reviews the previous work directly related to ours. Finally, Section 10 concludes the article with a summary of findings.

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 $\boldsymbol{p} = (\boldsymbol{p}[1], \ldots, \boldsymbol{p}[d])$ where $\boldsymbol{p}[i] = p[i]$.

A *weight vector* is a d-dimensional vector $\boldsymbol{w} = (\boldsymbol{w}[1], \ldots, \boldsymbol{w}[d])$ 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 $1, 2, \ldots, t$, respectively. Denote by $\mathcal{W}$ the set of all possible weight vectors.

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

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

and its *maximum rank regret* as

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

S is a *k-rank-regret representative* of P if $\text{MRR}(S) \leq k$. The problem studied in this article 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; this input set will serve as our running example throughout the article. Consider the weight vector $\boldsymbol{w} = (20, 1)$. The $\boldsymbol{w}$-score of o_1 is $(-1) \cdot 20 + 24 \cdot 1 = 4$. 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. Thus, the 3-set of $\boldsymbol{w}$ is $\{o_6, o_5, o_4\}$. Let $S = \{o_3, o_4\}$. Its $\boldsymbol{w}$-rank regret $RR_{\boldsymbol{w}}(S) = 3$, namely, the $\boldsymbol{w}$-rank of o_4, which is smaller than that of o_3. Later, we will see that $\text{MRR}(S) = 3$, i.e., S is a 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 [25] and establish its equivalence to Problem 1. Define

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

Compared to $\mathcal{W}$, $\mathcal{W}_{[d] \neq 0}$ 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 Equation (1)) is exactly* $\max_{\boldsymbol{w} \in \mathcal{W}_{[d] \neq 0}} RR_{\boldsymbol{w}}(S)$.

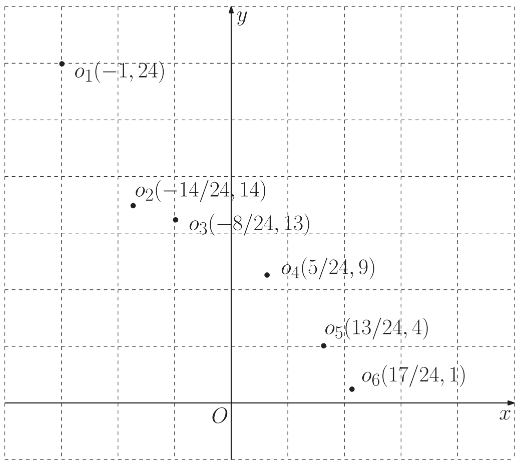

Fig. 1. The input set P for our running example.

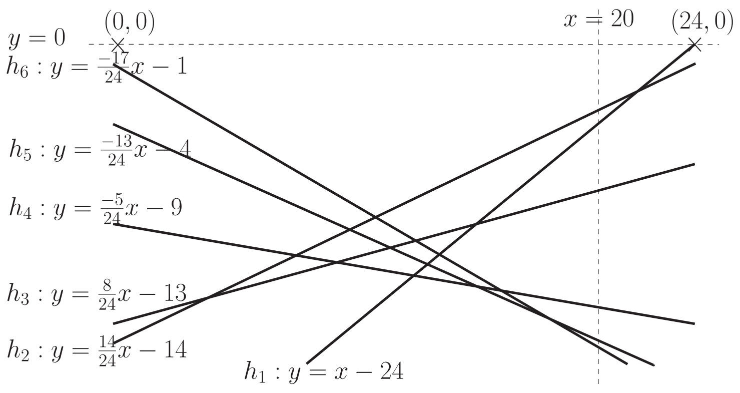
$(0, 0)$
$x = 20$
$(24, 0)$
$y = 0$
$h_6 : y = \frac{-17}{24}x - 1$
$h_5 : y = \frac{-13}{24}x - 4$
$h_4 : y = \frac{-5}{24}x - 9$
$h_3 : y = \frac{8}{24}x - 13$
$h_2 : y = \frac{14}{24}x - 14$
$h_1 : y = x - 24$

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

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 $\boldsymbol{w'} = (\frac{\boldsymbol{w}[1]}{\boldsymbol{w}[d]}, \ldots, \frac{\boldsymbol{w}[d-1]}{\boldsymbol{w}[d]}, 1)$. 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 $o = (o[1], \ldots, o[d])$ 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. Consider, again, the set P of points $o_1, o_2, \ldots, o_6$ in Figure 1. Figure 2 shows the corresponding dual planes (which are lines in 2D) $h_1, h_2, \ldots,$ and h_6, which constitute the set H. Take o_1 as an example. Recall from Figure 1 that o_1 has coordinates $(-1, 24)$, i.e., $o_1[1] = -1$

and $o_1[2] = 24$. From Equation (2), we know that its dual plane h_1 is given by the equation $x[2] = -o_1[1] \cdot x[1] - o_1[2]$, which is $x[2] = -(-1) \cdot x[1] - 24$. In Figure 2, $x[1]$ and $x[2]$ correspond to the "x" and "y" dimensions, respectively. This explains why h_1 is represented by the equation $y = x - 24$.

Define the *query space* $\mathscr{Q}$ as the set of all possible $(d-1)$-dimensional vectors $\boldsymbol{q} = (\boldsymbol{q}[1], \ldots, \boldsymbol{q}[d-1])$ 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 $(\boldsymbol{q}[1], \ldots, \boldsymbol{q}[d-1], -\infty)$. 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 $1, 2, \ldots, t$, respectively.

Example (cont.). The dimensionality d of the dual space is 2. Hence, the query space $\mathscr{Q}$ in Figure 2 is one dimensional ($= d - 1$), because of which we will simplify the vector representation $\boldsymbol{q}$ into a real value q. Consider the query $q = 20$. Line ℓ_q is the vertical line $x = 20$, namely, the line parallel to dimension $d = 2$ (i.e., y-axis) and passing the point $(\boldsymbol{q}[1], -\infty) = (20, -\infty)$. The lines in H intersect ℓ_q in the bottom-up order of $h_6, h_5, h_4, h_3, h_1, h_2$ (see Figure 2). The q-ranks of h_6, h_5, h_4, h_3, h_1, and h_2 are, therefore, $1, 2, \ldots, 6$, respectively.

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

Proposition 1. *Fix any canonical weight vector $\boldsymbol{w} \in \mathscr{W}_{[d]=1}$. Set $\boldsymbol{q} = (\boldsymbol{w}[1], \ldots, \boldsymbol{w}[d-1])$. 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 $\boldsymbol{w} = (20, 1)$; recall that $\boldsymbol{w}$ is canonical if and only if $\boldsymbol{w}[2] = 1$. As mentioned in an earlier example, 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, namely, $h_6, h_5, h_4, h_3, h_1, h_2$, respectively (see the previous example for how the q-ranks are computed.

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

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

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 $\mathcal{S} = \{$dual plane of $o \mid o \in S\}$.*

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

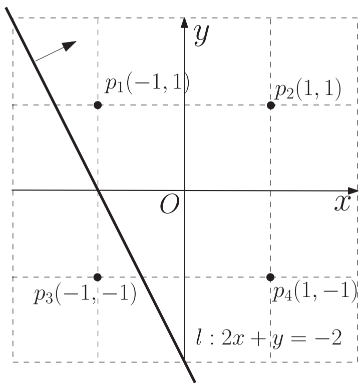

Fig. 3. Illustration of ϵ-net.

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 $c_1, \ldots, c_{d+1}$ are real-valued coefficients. The halfspace is *non-negative* if $c_1, c_2, \ldots, c_d$ 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.

Example. Consider the set $P = \{p_1, p_2, p_3, p_4\}$ of 2D points shown in Figure 3. $S = \{p_1, p_4\}$ is a 1/2-net. Let us examine the (non-negative) halfspace $2x + y \geq -2$ (with boundary line l). Because (i) the halfspace covers 3 points (p_1, p_2, p_4) in P and (ii) 3 is greater than $4 \cdot (1/2) = 2$, the 1/2-net definition demands that the halfspace should contain at least one point in S, which is indeed the case (actually, both points in S fall in the halfspace).

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 $\boldsymbol{w} = (\boldsymbol{w}[1], \ldots, \boldsymbol{w}[d])$. Let $o \in P$ be an object with $\boldsymbol{w}$-rank k; specifically, if no such objects exist (due to ties in scores), then 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 $\boldsymbol{w} = (c[1], \ldots, c[d])$. 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. □

Obtaining an ϵ-net is simple. As proved in Reference [38], 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 the following:

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 $1 - 1/n^2$.*

```
algorithm net-extreme-skyline ($P$)
1. sample a set $S_{sam}$ of $O(\frac{n}{k}\log n)$ points from $P$ with replacement
2. $EXT(S_{sam}) \leftarrow$ the extreme set of $S_{sam}$
3. $S \leftarrow$ the skyline of $EXT(S_{sam})$
4. return $S$
```

Fig. 4. The net-extreme-skyline algorithm.

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 Reference [44]). 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* [13]. 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*, as shown in Figure 4.

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 $o[1], \ldots, o[d]$, 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. [22] 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

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

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 Equation (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$(S) = 0$, then $\text{gain}_{\boldsymbol{w}}(P, k) \leq \text{gain}_{\boldsymbol{w}}(S, 1)$ 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. □

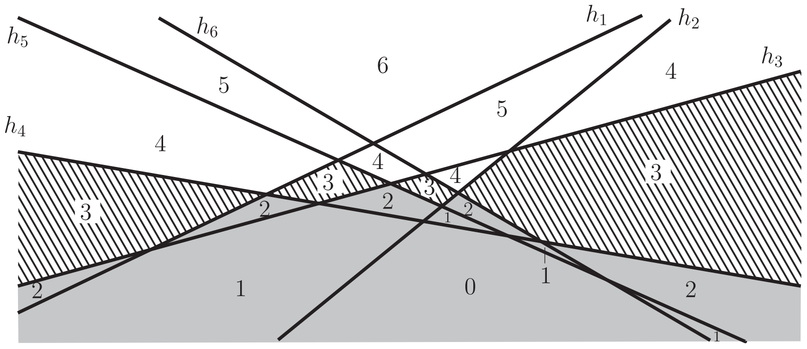

Fig. 5. Illustration of levels. Each number indicates the level of the corresponding cell. The gray area is the (≤ 2)-level, while the striped area is the 3-level.

In 2D space, the k-regret minimizing set problem can be settled in $\tilde{O}(n^2)$ time [16] (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. [3] 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 $|S|$, then 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 toward 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.

Example. To illustrate the above concepts, Figure 5 shows a set H of 2D planes (lines) that are taken directly from Figure 2. Every number indicates the level at the point where the number is placed. The gray area represents the (≤ 2)-level, the striped area represents 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 5) whose edges we refer to as *boundary edges*. There are $O(nk)$ boundary edges [5, 23] and they can be computed in $\tilde{O}(nk)$ time [32]. See Figure 6 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

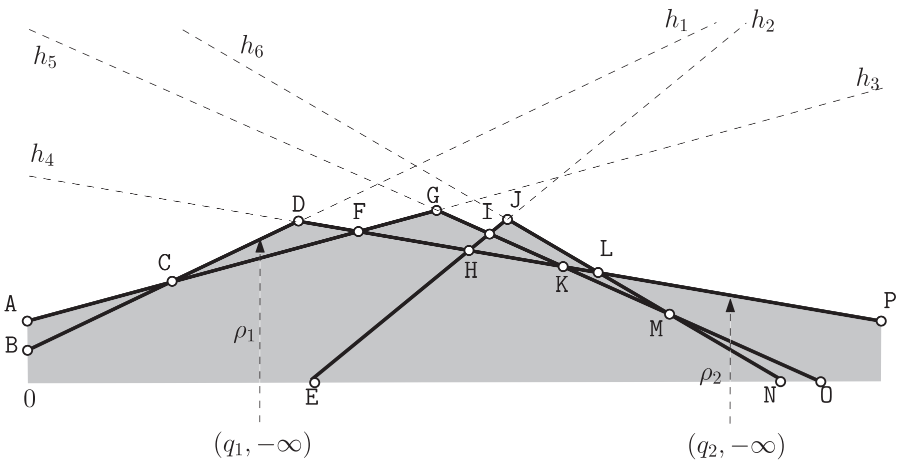

Fig. 6. The boundary edges in the (≤ 2)-level (taken from Figure 5) are shown in bold segments.

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 6 corresponds to the (≤ 2)-level defined by the set of lines in Figure 5. 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 3-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. Define an *envelop chain* as a sequence C of line segments $\sigma_1, \sigma_2, \ldots, \sigma_{|C|}$ such that:

- every segment of C is in the ($\le k-1$)-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 6. 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 $C = \sigma_1, \sigma_2, \ldots, \sigma_{|C|}$ 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 $\sigma_1, \sigma_2, \ldots, \sigma_{|C|}$.

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

Proof. 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 $H[C] = \mathcal{S}$ such that the x-projection of C covers $\mathcal{Q}$. This implies the correctness of the lemma.

```
algorithm 2D-exact ($H$)
1.  compute the ($\leq k-1$)-level induced by $H$
2.  $E \leftarrow$ the set of (directed) boundary edges in the ($\leq k-1$)-level
3.  sort, in ascending order, the edges in $E$ using their right endpoints' $x$-coordinates
4.  for each $e \in E$ (in the sorted order) do
5.      if the x-projection of $e$ covers coordinate 0 then  $minlen(e) \leftarrow 1$
6.      else
7.          $p \leftarrow$ the left endpoint of $e$
8.          $\mathrm{IN}(p) \leftarrow$ the incoming edges of $p$ in $E$
9.          for each $e' \in \mathrm{IN}(p)$
10.             if $e'$ and $e$ are on the same line in $H$ then $w(e') \leftarrow minlen(e')$
11.             else if $e'$ has a greater slope than $e$ then $w(e') \leftarrow minlen(e') + 1$
12.             else $w(e') \leftarrow \infty$
13.         $minlen(e) \leftarrow \min_{e' \in \mathrm{IN}(p)} w(e')$
14. $e^* \leftarrow$ a terminal edge $e \in E$ with the smallest $minlen(e)$
15. $C^* \leftarrow$ an optimal envelop chain whose last segment contains $e^*$
    /* the x-projection of $C^*$ covers $[0, \infty)$ and $|C^*| = minlen(e)$*/
16. return $C^*$
```

Fig. 7. The 2D exact algorithm.

The upper boundary of the ($\leq$ 0)-level of $\mathcal{S}$ must be an envelop chain C. Furthermore, every $h \in \mathcal{S}$ must contribute an edge to the ($\leq$ 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. □

Example (cont.). In Figure 6, no envelop chains of length 1 have an x-projection covering $\mathcal{Q}$. However, the x-projection of C = AF, FP covers $\mathcal{Q}$. Hence, $H[C] = \{h_3, h_4\}$ must be an optimal solution (this corresponds to $\{p_3, p_4\}$ in Figure 1).

We are now ready to clarify our algorithm for computing C^*. The algorithm, summarized in Figure 7, combines a dynamic programming strategy of Reference [22] with ideas specific to our context. Let e be a boundary edge in the ($\leq k-1$)-level of H (Lines 1–3), 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 (Lines 14 and 15).

Example (cont.). Set $k = 3$. For the boundary edge CD in Figure 6, we have $minlen(\mathrm{CD}) = 1$, 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 $[0, \mathrm{D}[1]]$. Similarly, $minlen(\mathrm{LP}) = 2$, evidenced by the monotone chain C = AF, FP; this C is an optimal solution to Problem 2.

To describe the computation of $minlen(e)$ (Lines 4–13), 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 (Line 5). Otherwise, let p be the left endpoint of e and $\mathrm{IN}(p)$ be the set of incoming edges of p in the ($\leq k-1$)-level (Lines 7 and 8). For each $e' \in \mathrm{IN}(p)$, define its *contribution* as

- *minlen*(e') if e' and e are on the same line in H (Line 10);
- $1 + $ *minlen*(e') if e' has a greater slope than e (Line 11) ;
- ∞ otherwise (Line 12).

Then, *minlen*(e) equals the minimum contribution of all $e' \in \mathrm{IN}(p)$ (Line 13).

If m is the total number of boundary edges in the $(\leq k-1)$-level, then 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 above dynamic programming process using 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

This section will present algorithms for solving Problem 1 with strong approximation guarantees. Section 5.1 first illustrates our high-level objectives in terms of approximation quality and running time. Then, Section 5.2 will introduce the *shallow cutting technique*, which we apply to design 2D and 3D algorithms in Sections 5.3 and 5.4, respectively. Sections 5.5 and 5.6 are dedicated to dimensionalities 4 and higher.

5.1 Overview of Our Approximation Schemes

Denote by *OPT* the size of an optimal k-rank-regret representative of the input P. In terms of *result quality*, our goal is to compute a c_1k-rank-regret representative of size $c_2 \cdot OPT$ where $1 \leq c_1 = \tilde{O}(1)$ and $1 \leq c_2 = \tilde{O}(1)$. If an algorithm can always return such representatives, then we call it a *bi-criteria approximation algorithm*. Even better, if an algorithm guarantees $c_1 = 1$ and $c_2 \geq 1$, then we refer to it as a *size-approximation algorithm*; similarly, if an algorithm guarantees $c_1 \geq 1$ and $c_2 = 1$, then we refer to it as a *regret-approximation algorithm*.

The k-rank-regret representative problem is NP-hard when $d \geq 3$ (Section 4.1). To tackle this computation barrier, we want to design bi-criteria algorithms finishing in $f(k) \cdot \tilde{O}(n)$ time, where $f(k)$ is a monotonic function depending only on k and satisfying $f(k) = \tilde{O}(1)$ for $k = O(\operatorname{polylog} n)$. Such algorithms have practical significance, because users prefer small values of k in real-world applications. In particular, when $k = O(\operatorname{polylog} n)$ (we believe this already fulfills the needs of most applications), the running time of those algorithms is bounded by $\tilde{O}(n)$.

Motivated by this, we say that a bi-criteria approximation algorithm $\mathscr{A}$ is *fixed-parameter near-linear* if its running time is bounded by $f(k) \cdot \tilde{O}(n)$. This name suggests that if the parameter k is "fixed" (i.e., $\tilde{O}(1)$), then the computation time is "near-linear" (i.e., $\tilde{O}(n)$). We will strive to design such algorithms whenever possible. As it will turn out, they exist for dimensionalities $d = 2$ and 3 but do not exist for $d \geq 4$ (unless major breakthroughs could be made in computational geometry). For $d \geq 4$, therefore, we will drop the fixed-parameter near-linear requirement and, instead, aim to design bi-criteria algorithms that terminate in polynomial time (for arbitrary k).

5.2 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, and so on. 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 dth coordinate is $-\infty$. Figure 8 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 $-\infty$.

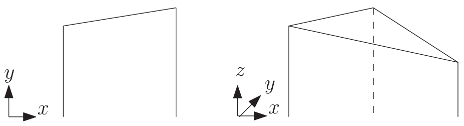

Fig. 8. The 2D and 3D prisms.

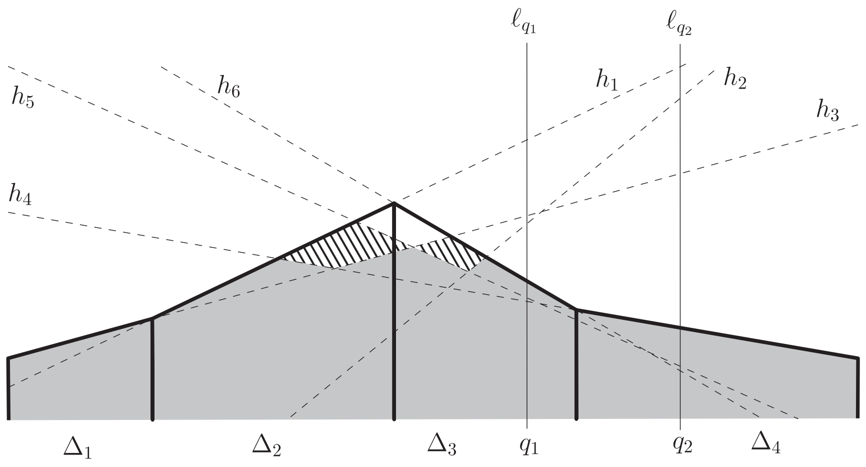

Fig. 9. A $(1, 1/3)$-shallow cutting in 2D space. All the lines come from Figure 6. The gray area is the (≤ 2)-level, the striped area is a part of the 3-level, and the closed white region is a part of the 4-level.

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. To further illustrate the above concepts, Figure 9 shows a $(\lambda, k/n)$-shallow cutting Ξ where $n = 6$, $k = 2$, and $\lambda = 1$. Ξ consists of the four prisms $\Delta_1, \ldots, \Delta_4$ as shown. The union of all those prisms covers the (≤ 2)-level, but is also contained in the (≤ 4)-level. The conflict set H_{Δ_4}, for example, includes h_4, h_5, and h_6.

Lemma 8 ([1, 20]). *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.*

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

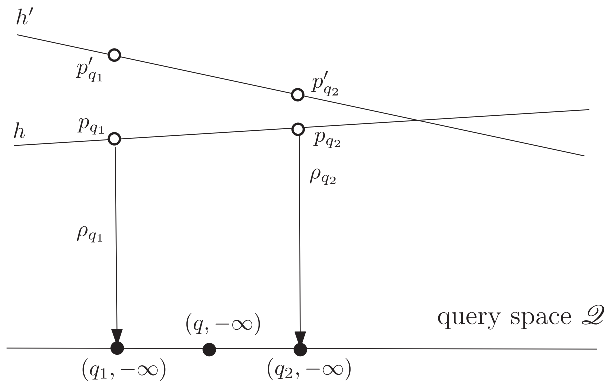

Fig. 10. Proof of Lemma 9.

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 9. 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}, respectively), as shown in Figure 10. 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. Figure 11 shows the pseudocode of our 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 (Line 1).

For each line $h \in H$, we generate an interval $\mathcal{I}_h$ as follows (Line 4). First, identify the *leftmost* (or *rightmost*) prism Δ_1 (or Δ_2, respectively) intersecting h. Let σ_1 be the part of h that appears in Δ_1; note that σ_1 is a 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, respectively). The interval $\mathcal{I}_h$ is the minimum bounding interval of I_1 and I_2. See Figure 12 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, $rank_q(h) < (2+\delta)k$ 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

```
algorithm 2D-shallow-cutting ($H$)
1. obtain a $(\delta/2, (k-1)/n)$-shallow cutting $\Xi$ on $H$ and, thus, the conflict sets $\{H_\Delta \subseteq H \mid \Delta \in \Xi\}$
2. $\mathcal{I} \leftarrow \emptyset$
3. for $h \in H$ do
4.     generate the interval $\mathcal{I}_h$ for $h$ (from the leftmost and rightmost prisms intersecting $h$)
5.     $\mathcal{I} \leftarrow \mathcal{I} \cup \{\mathcal{I}_h\}$
6. find a minimum subset $\mathcal{J} \subseteq \mathcal{I}$ such that the union of the intervals in $\mathcal{J}$ covers $\mathcal{Q}$
7. $\mathcal{S} \leftarrow \{h \in H \mid \mathcal{I}_h \in \mathcal{J}\}$
8. return $\mathcal{S}$
```

Fig. 11. The 2D shallow cutting algorithm.

(or q_2, respectively). 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 5.2). 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 $\mathcal{I} = \{\mathcal{I}_h \mid h \in H\}$ (Line 5). 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 $\mathcal{Q}$ (Line 6). 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 (Lines 7 and 8).

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 $h \in \mathcal{S}$.

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., Reference [9]). □

5.4 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.

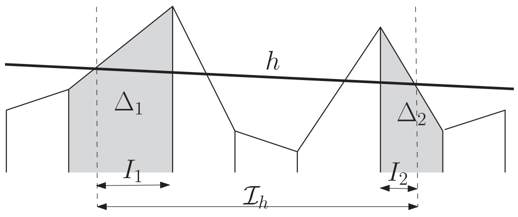

Fig. 12. Illustration of $\mathcal{I}_h$.

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.*

Remark. Theorem 13 is mainly of theoretical interest. Its primary purpose is to prove the existence of fixed-parameter near-linear algorithms in 3D space. The algorithm in Theorem 13, unfortunately, is a bit sophisticated and may not be suitable for practical implementation (for this reason, we will omit it in the experiments). In Sections 6 and 7, we will develop alternative algorithms for 3D space.

5.5 Hardness of Dimensions $d \geq 4$

In this subsection, we will prove that no fixed-parameter near-linear algorithms exist for $d \geq 4$ subject to a conjecture on a well-known problem 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 [48] that has running time $O(n^{2-2/(\lceil d/2 \rceil+1)+\epsilon})$. In his 1996 paper [18], 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 provethe following.

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 $f(k) = O(\text{polylog}\, n)$ for $k = O(\text{polylog}\, n)$. 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 $w[1], \dots, w[4]$. 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 jth instance. X must be the union of $S^*_1, \dots, S^*_{16}$.

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 jth 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 $S = S_1 \cup S_2 \cup \cdots \cup S_{16}$. 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, then 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.

5.6 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 $\{K_1, \ldots, K_s\}$ 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 $O(nk^{1/3})$ [26] for $d = 2$, $O(nk^{3/2})$ for $d = 3$ [57], and in general $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil - c_d})$ for fixed $d \geq 4$ [4], 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 [32] in 2D, $\tilde{O}(nk^2 + sk)$ expected time [2] in 3D, and $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil} + sk)$ expected time [49] 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 that 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 $d = 3$, while that of Theorem 13 is $\tilde{O}(nk^2)$. If $k = \text{polylog}\, n$, then the two time complexities are within a factor of $\tilde{O}(1)$. For larger k, e.g., when k is a polynomial of n, Theorem 15 actually yields a better time bound. However, Theorem 15 produces size-approximation algorithms (i.e., the approximation ratio on size is 1; see Section 5.1), whereas Theorem 13 can only guarantee $\tilde{O}(1)$ approximation ratios on size and rank regret.

```
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. 13. The k-set enumeration algorithm.

Remark. 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 Reference [36] 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. We will not delve into further details in this article.

k-set enumeration. Theorem 15 requires enumerating all possible k-sets, for which purpose the algorithms in References [4, 7, 26, 57] are rather complicated. To alleviate the issue, we describe a practical method that does not improve the complexities in Theorem 15, but is much easier to implement. Our method is adapted from an algorithm in Reference [7], which, however, requires the sophisticated concept of *k-set polytope* (see Reference [7] for details). Instead, we will adopt an intuitive graph perspective.

We call a k-set *clean* if it has size k.[2] As far as the hitting set approach is concerned, it suffices to consider the set $\mathcal{K}_{clean}$ of clean k-sets [4, 7, 26, 57]. 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 13 shows an algorithm for generating $\mathcal{K}_{clean}$ incrementally by performing a BFS (breadth 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 $\mathcal{K}_{clean}$, then the algorithm checks whether K' is a valid k-set. If so, then K' is a neighbor vertex of K in G and, hence, is added to $\mathcal{K}_{clean}$ and to the queue (Lines 8 and 9). After the BFS finishes, the final $\mathcal{K}_{clean}$ is returned (Line 10).

Deciding whether 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 $\boldsymbol{o} \in P \setminus K'$.

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

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

```
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. 14. The random-kset algorithm.

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

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 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 14 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 and 3). If $P_{\boldsymbol{w}}(k)$ is already in $\mathcal{K}$, then 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 $1 - 1/n$:

$$\mathbf{Pr}_{\boldsymbol{w}\sim\mathcal{D}}[\mathcal{S} \cap P_{\boldsymbol{w}}(k) \neq \emptyset] \quad \geq \quad 1 - \delta. \tag{4}$$

To see why, suppose that (4) 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*.

7 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 7.1 and then explain the algorithm in Section 7.2.

7.1 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 $\text{rank}_{\boldsymbol{q}}(h) \leq k$.

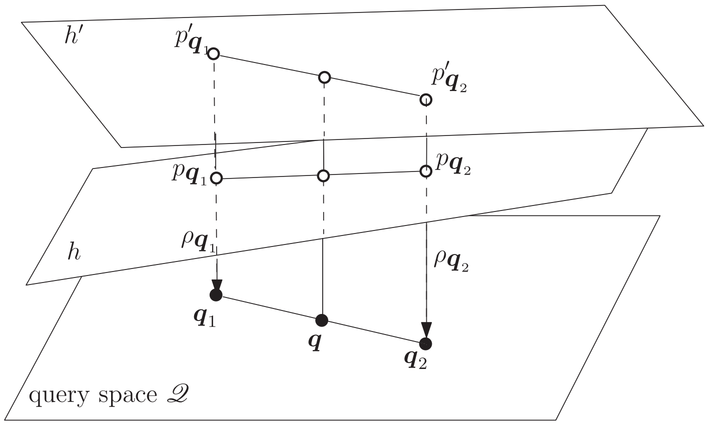

Fig. 15. Proof of Proposition 2.

The rest of this subsection serves as a proof of the following.

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

$$rank_{\boldsymbol{q}}(h) \quad \le \quad \left(\sum_{i=1}^{d} rank_{\boldsymbol{q}_i}(h)\right) - (d-1). \tag{5}$$

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, $rank_{\boldsymbol{q}}(h) \le rank_{\boldsymbol{q}_1}(h) + rank_{\boldsymbol{q}_2}(h) - 1$.*

Proof. The proof uses the same ideas as in the proof of Lemma 9. Each query $\boldsymbol{q}$ corresponds to the point $(\boldsymbol{q}[1], \ldots, \boldsymbol{q}[d-1], -\infty)$ at the bottom of the dual space $\mathbb{R}^d$. Denote by $\ell_{\boldsymbol{q}}$ the line parallel to dimension d and passing $(\boldsymbol{q}[1], \ldots, \boldsymbol{q}[d-1], -\infty)$. 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 $|S_{\boldsymbol{q}}| \le |S_{\boldsymbol{q}_1}| + |S_{\boldsymbol{q}_2}|$, from which the lemma will follow.

Let $p'_{\boldsymbol{q}_1}$ (or $p'_{\boldsymbol{q}_2}$) be the intersection point between h' and $\ell_{\boldsymbol{q}_1}$ (or $\ell_{\boldsymbol{q}_2}$, respectively); Figure 15 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 $m \ge 2$ queries $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_m$ in $\mathcal{Q}$. For any query $\boldsymbol{q}$ that is a convex combination of $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_m$, $rank_{\boldsymbol{q}}(h) \le (\sum_{i=1}^{m} rank_{\boldsymbol{q}_i}(h)) - (m-1)$.*

Before proving the proposition, we would like to remind the reader that $\boldsymbol{q}$ is a convex combination of $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_m$ if and only if $\boldsymbol{q} = \sum_{i=1}^{m} \alpha_i \boldsymbol{q}_i$ for m real values $\alpha_1, \ldots, \alpha_m$ in $[0, 1]$ satisfying $\sum_{i=1}^{m} \alpha_i = 1$.

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

```
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. 16. The space-partition algorithm.

Let us write $\boldsymbol{q}$ as $\sum_{i=1}^{m} \alpha_i \boldsymbol{q}_i$ where $\alpha_1, \ldots, \alpha_m$ are real values in $[0, 1]$ such that $\sum_{i=1}^{m} \alpha_i = 1$. We consider $0 < \alpha_m < 1$ (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 convex combination of $q_1, \ldots, q_{m-1}$, by the inductive assumption we know $\text{rank}_{\boldsymbol{q}'}(h) \le (\sum_{i=1}^{m-1} \text{rank}_{\boldsymbol{q}_i}(h)) - (m-2)$. Notice that $\boldsymbol{q} = (1-\alpha_m)\boldsymbol{q}' + \alpha_m \boldsymbol{q}_m$. By Proposition 2, we have $\text{rank}_{\boldsymbol{q}}(h) \le \text{rank}_{\boldsymbol{q}'}(h) + \text{rank}_{\boldsymbol{q}_m}(h) - 1 \le (\sum_{i=1}^{m} \text{rank}_{\boldsymbol{q}_i}(h)) - (m-1)$. □

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 convex combination of the d vertex vectors of Δ.

7.2 Algorithm

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

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

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

We say that h *protects* R if

$$\sum_{i=1}^{d} r_i \le k + d - 1. \tag{6}$$

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

Lemma 17. *If h protects R, then 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 $\boldsymbol{q}'_1, \boldsymbol{q}'_2, \ldots,$ and $\boldsymbol{q}'_d$. By Lemma 16, $\text{rank}_{\boldsymbol{q}}(h) \le (\sum_{i=1}^{d} \text{rank}_{\boldsymbol{q}'_i}(h)) - (d-1)$. By the definition of $r_1, \ldots, r_d$, we know $\sum_{i=1}^{d} \text{rank}_{\boldsymbol{q}'_i}(h) \le \sum_{i=1}^{d} r_i$. Therefore, $\text{rank}_{\boldsymbol{q}}(h) \le \sum_{i=1}^{d} r_i - (d-1) \le k + (d-1) - (d-1) = k$, where the second inequality used (6). □

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 16 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, then it adds to $\mathcal{S}$ a plane $h \in H$ that protects R (Lines 6 and 7). If such an h does not exist, then 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, $r_1 = r_2 = \cdots = r_d = 1$ and, hence, (6) holds). $\mathcal{Q}$ is finite in practice, because a weight representation has a bounded precision (e.g., 64 bits) in a computer.

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

- *BlueNile dataset*[3]*:* **Blue Nile (BN)** 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 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 flight dataset*[4]*:* The **US Department of Transportation (DoT)** 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.

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.

[3] www.bluenile.com/diamond-search?.
[4] www.transtats.bts.gov/DL_SelectFields.asp?.

Algorithms Evaluated. We will evaluate all the algorithms proposed in this article under different settings. Specifically, we will present two sets of experiments for the 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 5.3, 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 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 6 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 toward 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 7.1. 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 [53]. 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 [58].

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 Performance Evaluation

Having provided the proof of concept, we proceed to evaluate the performance of our algorithms under different settings.

Number of extreme points. As mentioned in Section 1, the 1-rank-regret representative of a dataset comprises the points on the boundary of the convex hull, i.e., the extreme points, which

Table 2. The Sizes of Extreme Points at Various Dimensions

dimension	BN dataset	DoT dataset
$d = 2$	19	13
$d = 3$	69	56
$d = 4$	206	168
$d = 5$	553	445
$d = 6$	–	1012

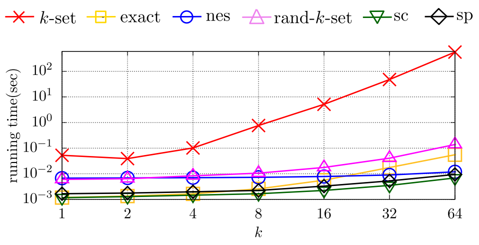

Fig. 17. BN, 2D: Impact of varying k on time.

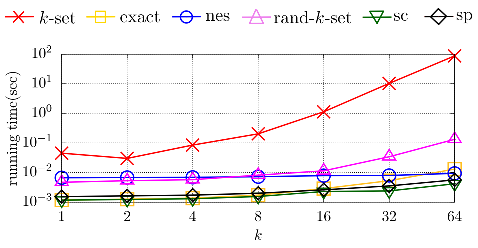

Fig. 18. DoT, 2D: Impact of varying k on time.

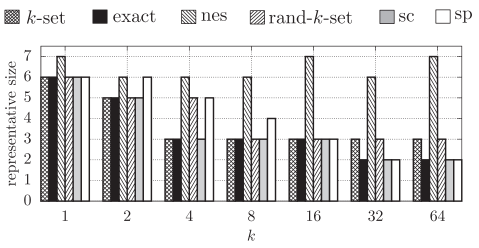

Fig. 19. BN, 2D: Impact of varying k on output size.

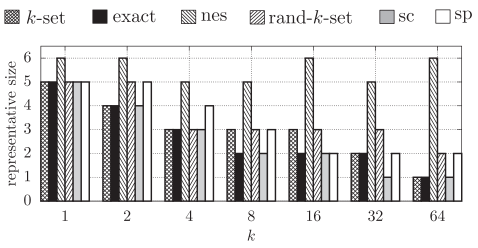

Fig. 20. DoT, 2D: Impact of varying k on output size.

guarantee to contain the top choice of any linear ranking function. However, the number of extreme points can be very large. Table 2 shows the number of extreme points in the BN and DoT datasets at various dimensionalities. As will become evident in the upcoming experiments, the number of extreme points is usually several times the size of the rank-regret representative we find. This further strengthens our motivation and supports the necessity of developing efficient algorithms for discovering (small) rank-regret representatives.

2D, varying k. Henceforth, we will follow the paradigm explained in Section 8.1, namely, in every experiment, we will study the impact of one parameter, while fixing the other parameters to their default values.

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 17 to 22.

Figures 17 and 18 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

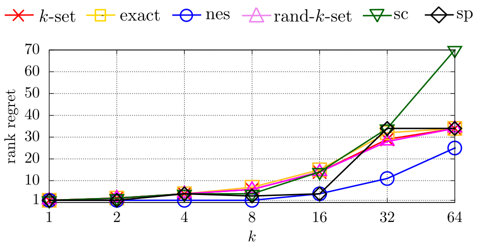

Fig. 21. BN, 2D: Impact of varying k on output rank regret.

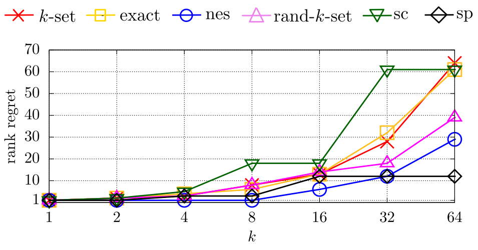

Fig. 22. DoT, 2D: Impact of varying k on output rank regret.

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, however, 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.

Figures 19 and 20 show the size of the output (representative set) found by each algorithm, while Figures 21 and 22 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 21 and 22, 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, $k = 64$).

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 that, however, needs to be $1/k$ of the original dataset. Therefore, our earlier results in Figures 19 and 20 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

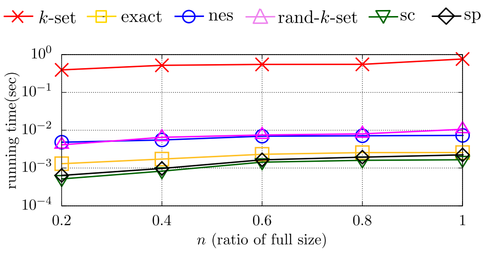

Fig. 23. BN, 2D: Impact of varying n on time.

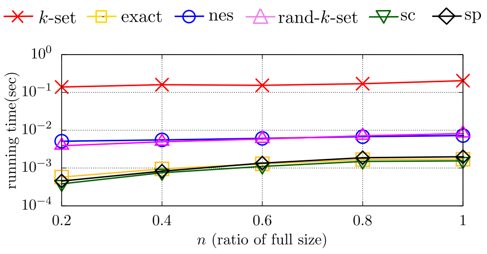

Fig. 24. DoT, 2D: Impact of varying n on time.

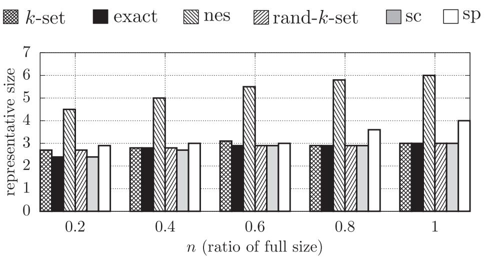

Fig. 25. BN, 2D: Impact of varying n on output size.

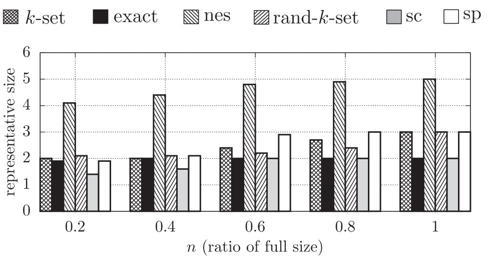

Fig. 26. DoT, 2D: Impact of varying n on output size.

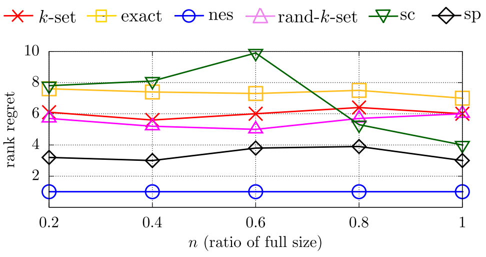

Fig. 27. BN, 2D: Impact of varying n on output rank regret.

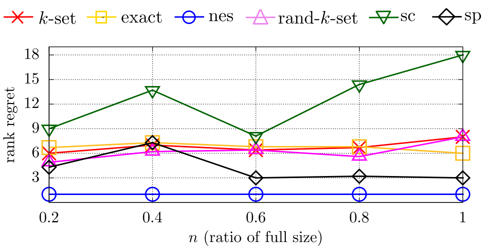

Fig. 28. DoT, 2D: Impact of varying n on output rank regret.

also gave a lower-bound n/k on the size of any ϵ-net in the worst case. Despite this negative result, Figures 19 and 20 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.

The results are provided in Figures 23 to 28. For every dataset, we controlled n by randomly selecting 20%, 40%, 60%, 80%, and 100% of the data. First, as in Figures 23 and 24, 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 25 and 26 show the output size for the BN and DoT datasets, while Figures 27 and 28 show the rank-regret of the generated results obtained by different algorithms. Similarly 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

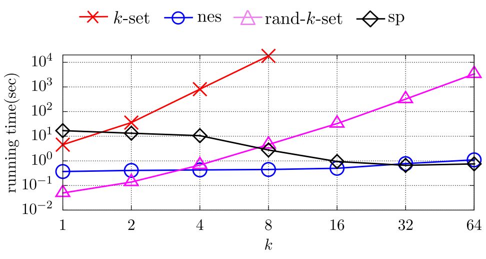

Fig. 29. BN, MD: Impact of varying k on time.

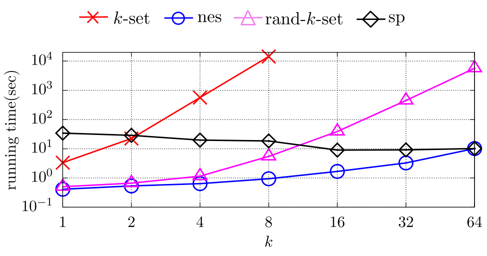

Fig. 30. DoT, MD: Impact of varying k on time.

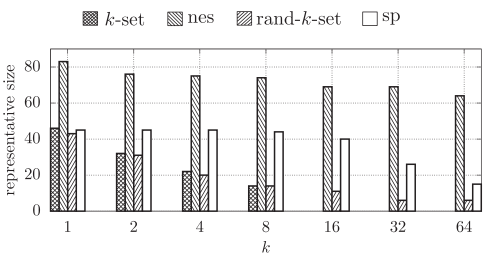

Fig. 31. BN, MD: Impact of varying k on output size.

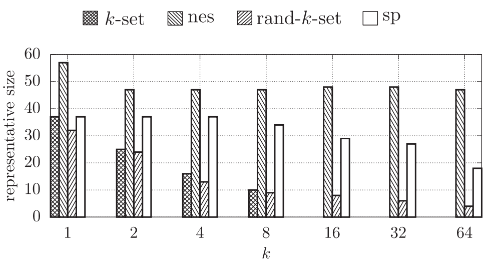

Fig. 32. DoT, MD: Impact of varying k on output size.

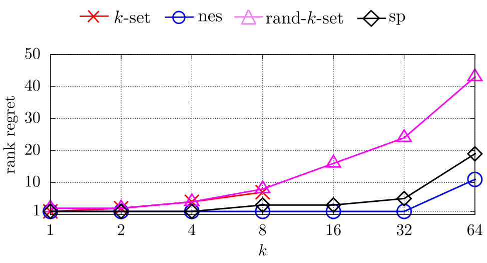

Fig. 33. BN, MD: Impact of varying k on rank regret.

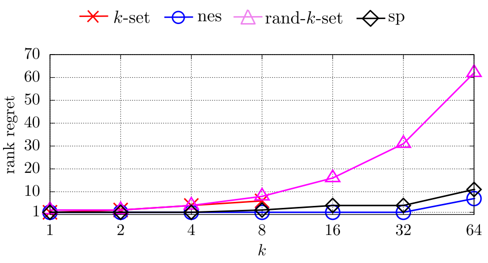

Fig. 34. DoT, MD: Impact of varying k on rank regret.

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.

MD, varying k. After evaluating the 2D solutions, we now turn our attention to MD where $d \geq 3$. In the upcoming experiments, we study the impact of varying the value of k on the performance of different MD algorithms. Figures 29 to 34 show the results across different settings for the BN and DoT datasets.

First, looking at the running time of the algorithms in Figures 29 and 30, 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

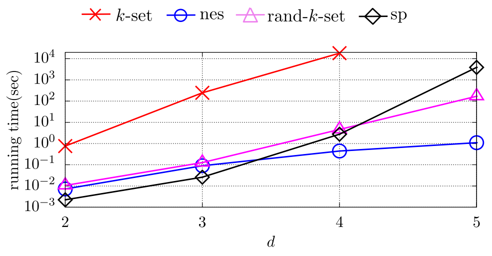

Fig. 35. BN, MD: Impact of varying d on time.

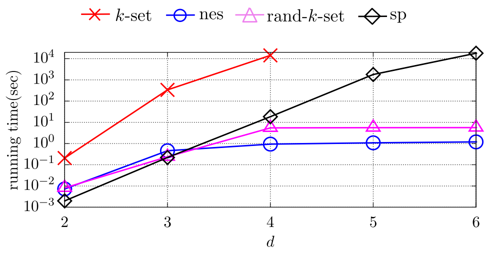

Fig. 36. DoT, MD: Impact of varying d on time.

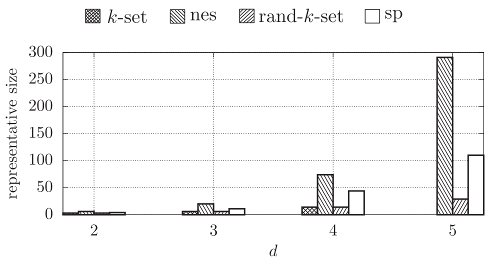

Fig. 37. BN, MD: Impact of varying d on output size.

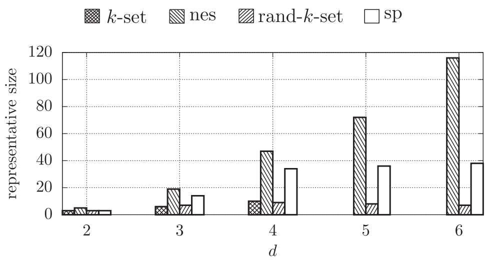

Fig. 38. DoT, MD: Impact of varying d on output size.

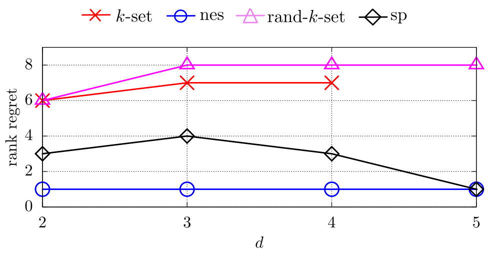

Fig. 39. BN, MD: Impact of varying d on rank regret.

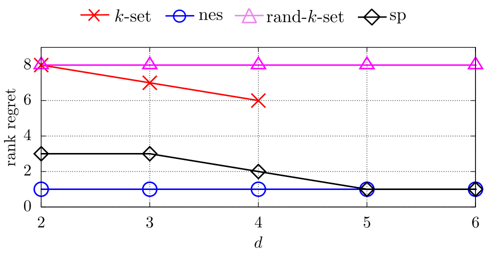

Fig. 40. DoT, MD: Impact of varying d on rank regret.

are the top-k results (which are supersets of the results of smaller k). These observations 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 31 and 32) and rank-regret (Figures 33 and 34) 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 their representative sizes 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 35 to 40.

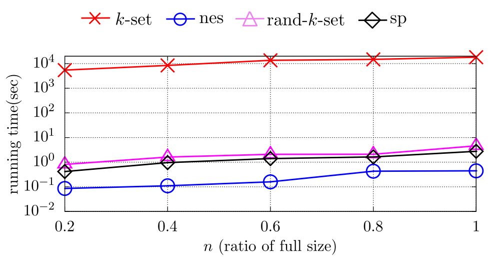

Fig. 41. BN, MD: Impact of varying n on time.

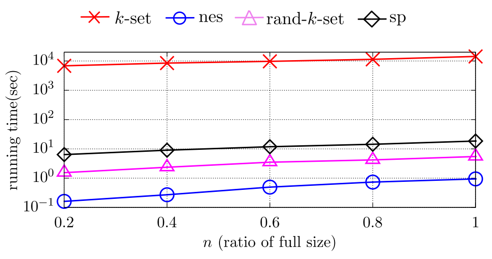

Fig. 42. DoT, MD: Impact of varying n on time.

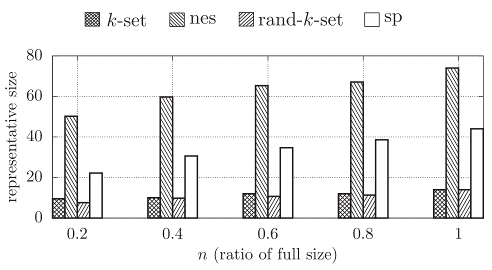

Fig. 43. BN, MD: Impact of varying n on output size.

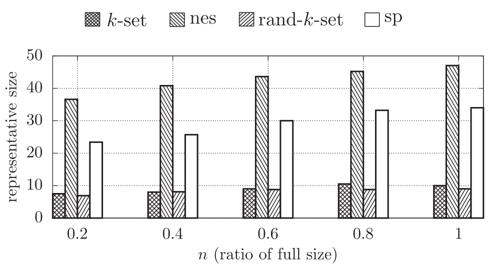

Fig. 44. DoT, MD: Impact of varying n on output size.

Let us first look into the running time of the algorithms across different settings (Figures 35 and 36). The k-set algorithm failed to scale beyond four dimensions, because the exact (graph-traversal) algorithm in Section 5.6 for enumerating the k-sets did not finish within the time budget (20,000 seconds). In contrast, 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 37 and 38 show the output size, and Figures 39 and 40 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 41 to 46.

Looking at Figures 41 and 42, 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 45 and 46, 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 43 and 44, 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 [40] and skyline discovery [13]. In top-k query processing, the approach is to model the user preferences

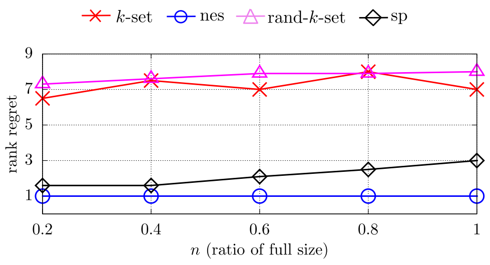

Fig. 45. BN, MD: Impact of varying n on rank regret.

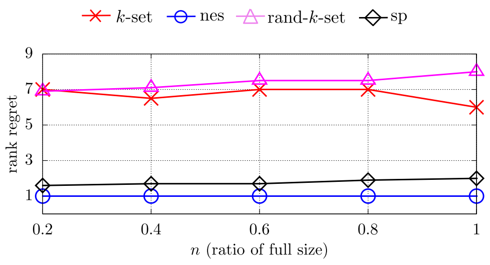

Fig. 46. DoT, MD: Impact of varying n on rank regret.

by a ranking/utility function that is then used to preferentially select tuples. Fundamental results include access-based algorithms [15, 33, 34, 46] and view-based algorithms [24, 39]. 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 [11, 13, 55]. Skylines and convex hull points can also serve as effective indexes for top-k query processing [10, 21, 62].

Efficiency and effectiveness have always been the ch6allenges 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 [8, 37]. Studies such as [17, 61] are focused toward reducing the skyline size. In an elegant effort toward finding a small representative subset of the data, Nanongkai et al. [51] 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 References [3, 8, 16, 41, 43, 50, 54, 63] studied different challenges and variations of the problem. As discussed in Section 4.1, Chester et al. [22] generalize the regret-ratio notion to k-regret ratio, and Agarwal et al. [3] prove that the k-regret minimizing set problem is NP-complete even when $d = 3$. For the case of two-dimensional databases, Reference [22] proposes a quadratic algorithm. The cube algorithm and a greedy heuristic [51] are the first algorithms proposed for regret-ratio in dimensionality $d \geq 3$. Recently, References [3, 8] 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 [8] considers the original regret-ratio problem while Reference [3] 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.

We now review results relevant to the geometric notions that were used in this article to develop new algorithms. Such notions include ϵ-net (for the net-extreme-skyline algorithm, Figure 4), ($\leq k$)-level in 2D space (for the 2D exact algorithm, Figure 7), shallow cutting (for the 2D and 3D shallow cutting algorithms, Figure 11 and Theorem 13), and k-set (for the k-set enumeration algorithm, Figure 13).

Haussler and Welzl [38] proved that a random sample (with replacement) of size $O(\frac{d}{\epsilon} \log \frac{1}{\delta\epsilon})$ from a d-dimensional points set P is an ϵ-net (for halfspaces) with probability at least $1 - \delta$. We utilized this result in designing the net-extreme-skyline algorithm. For certain dimensionalities (in particular, 2 and 3), it is possible to produce even smaller ϵ-nets; we refer the reader to References [6, 14, 19, 42, 44, 60] for details.

The notion of $(\leq k)$-level, defined on a set H of lines, is fundamental in computational geometry. Recall from Section 4.2.1 that the $(\leq k)$-level is partitioned into non-overlapping polygons using the lines in H. In the special case of $k = n$ (where $n = |H|$), the set of polygons that constitute the $(\leq n)$-level is called the *arrangement* of H. The arrangement can be computed in $\tilde{O}(n^2)$ time (the algorithm is easy to implement; see Reference [25]). Another special case worth mentioning is $k = 0$. The (≤ 0)-level (namely, the 0-level) consists of a single polygon, whose boundary is called the *lower envelope* and can be computed in $O(n \log n)$ time (again, the algorithm is easy to implement; see Reference [25]). For general values of k, Alon and Gyori were the first to prove that the $(\leq k)$-level has $O(nk)$ boundary edges (recall that a boundary edge is an edge of a polygon in the $(\leq k)$-level); the bound is tight in the worst case, meaning that the number of boundary edges can reach $\Omega(nk)$. Clarkson and Shor [23] provided an alternative (somewhat simpler) argument to prove the same bound. To establish Theorem 7, we leveraged an $O(n \log n + nk)$-time algorithm for computing the $(\leq k)$-level; the algorithm was due to Everett et al. [32].

The algorithm of Reference [32] is a bit complicated and difficult to implement. There exist heuristic methods for finding the $(\leq k)$-level that, although not attractive in worst-case time complexity, are much simpler to implement. One such method is *Onion* [21]. Next, we illustrate the Onion approach for $k = 1$, because the extension to higher k values is straightforward. Let H be a set of lines whose (≤ 1)-level is to be computed. First, find the set L_0 of lines that define the lower envelope of H (which, as mentioned before, takes $O(|L_0| \log |L_0|)$ time). Then, we remove L_0 from H (as if peeling off the out-most layer of an onion) and obtain $H_1 = H \setminus L_0$. In the same fashion, we find the set L_1 of lines that define the lower envelope of H_1. All the lines relevant to the (≤ 1)-level of H must be in $L = L_0 \cup L_1$. We can now compute the arrangement of L (which, as mentioned before, takes $\tilde{O}(|L|^2)$ time) and then derive the (≤ 1)-level of H from the line arrangement. For small k, the size of L is considerably smaller than n, thus allowing the method to terminate fast.

Shallow cuttings were introduced by Matousek [47] as a tool for halfspace range reporting. He [47] gave a polynomial-time algorithm to compute a shallow cutting in any constant-dimensional space. Later, Ramos [56] presented randomized algorithms for 2D and 3D space, both of which run in $O(n \log n)$ expected time (where n is the number of planes in the input). Also focusing on 2D and 3D, Chan and Tsakalidis [20] discovered deterministic algorithms that finish in $O(n \log n)$ time. The specific form of shallow cuttings we used in Section 5.2 is a refined version of Matousek's and was proposed by Afshani and Chan [1]. They also showed [1] that, in 2D and 3D space, all the aforementioned algorithms designed to compute Matousek's version can be used to compute the refined version with the same time complexity.

There is a simpler method to compute a $(\lambda, k/n)$-shallow cutting whose size may not be bounded by $O(n/(1+k))$ but is often sufficiently small for practical use. Imagine sweeping a vertical line ℓ from left to right and, in doing so, trace out a prism Δ continuously. Specifically, Δ is two-sided: its left and top edges have been decided, but its right edge is aligned with ℓ and is still moving toward right along with ℓ. The prism's right edge is finalized at the current ℓ when, if ℓ continues to move, Δ will violate the conditions of $(\lambda, k/n)$-shallow cutting. After that, we create another two-sided prism Δ' right away. Specifically, the left edge of Δ' is aligned with ℓ, while its top edge is the $(1 + \lambda/2)k$th lowest line (in the input set) at the current position of ℓ. The process then repeats until the end.

Lovasz and Erdos [31, 45] appeared to be the first to formally investigate how many k-sets can be induced by a set of 2D points. Their work motivated a fruitful line of research on bounding the number of k-sets. See References [26, 29, 30, 52, 59] for results on dimensionalitiy $d = 2$ and References [4, 27, 28, 57, 59] for results on $d \geq 3$. The problem of enumerating all k-sets has been studied in Reference [32] for 2D and in References [4, 7, 26, 57] for higher dimensionalities. A practical algorithm for enumerating k-sets has been described in Section 5.6.

10 CONCLUSIONS

In this article, 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 three or more dimensions, approximation algorithms of various dimensionalities under different approximation schemes, a randomized algorithm utilizing the knowledge of query distribution, and a space-partition algorithm leveraging an interesting rank-sum lemma. 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.

Our work initializes several directions for future research. Recall that in 2D space we developed a bi-criteria approximation algorithm with running time $O(n \log n)$. Currently, it remains elusive to design a bi-criteria approximation algorithm with the same time complexity in 3D space. Like most of the research in the skyline literature, this article focused on low and medium dimensionalities. When the dimensionality is very large such that it can no longer be considered a constant, our algorithms would not work well. How to overcome this issue is another exciting topic for investigation. The last direction we want to mention concerns updates. In this work, we have assumed the input set of points to be static. It would be nice to have algorithms that can maintain a rank-regret representative efficiently along with the insertions/deletions on the input.

APPENDIX

PROOF OF LEMMA 1

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

Set $r =$ 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 $o_1, o_2, \ldots, o_{r-1}$ 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.

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.

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

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 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 5.2) 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

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

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:

$$Z_h = \{A \in \mathcal{A} \mid \exists \Delta \in \Xi(h) \text{ such that } A \in \mathcal{A}_\Delta\}.$$

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

Proof. As before, denote by $\ell_{\boldsymbol{q}}$ the vertical line in $\mathbb{R}^d$ that is parallel to dimension d and passes $(\boldsymbol{q}[1], \ldots, \boldsymbol{q}[d-1], -\infty)$. 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 the following.

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 the following:

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

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.

ACKNOWLEDGMENTS

Abolfazl Asudeh's research was supported in part by project 2107290 from the US National Science Foundation (NSF). Jagadish's research was supported in part by projects 1741022 and 2106176 from NSF. Gautam Das's research was supported in part by projects 2107296 and 2008602 from NSF. Yufei Tao's research was supported in part by GRF projects 142034/21 and 142078/20 from Hong Kong Research Grants Council. Nan Zhang's research was supported in part by projects 1850605, 1851637, and 2040807 from NSF.

REFERENCES

[1] Peyman Afshani and Timothy M. Chan. 2009. On approximate range counting and depth. *Discr. Comput. Geom.* 42, 1 (2009), 3–21.

[2] Pankaj K. Agarwal, Mark de Berg, Jiri Matousek, and Otfried Schwarzkopf. 1998. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM J. Comput.* 27, 3 (1998), 654–667.

[3] Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. 2017. Efficient algorithms for k-regret minimizing sets. In *Proceedings of the International Symposium on Experimental Algorithms.* 7:1–7:23.

[4] Noga Alon, Imre Barany, Zoltan Furedi, and Daniel J. Kleitman. 1992. Point selections and weak e-nets for convex hulls. *Combin. Probab. Comput.* 1, 4 (1992), 189–200. https://www.cambridge.org/core/journals/combinatorics-probability-and-computing/article/point-selections-and-weak-nets-for-convex-hulls/7C4D22465C309AA68A967AFE14FE9BEE.

[5] Noga Alon and Ervin Gyori. 1986. The number of small semispaces of a finite set of points in the plane. *J. Combin. Theory Ser. A* 41, 1 (1986), 154–157.

[6] Noga Alon and Asaf Shapira. 2008. A characterization of the (Natural) graph properties testable with one-sided error. *SIAM J. Comput.* 37, 6 (2008), 1703–1727.

[7] Artur Andrzejak and Komei Fukuda. 1999. Optimization over k-set polytopes and efficient k-set enumeration. In *Proceedings of the Algorithms and Data Structures Workshop (WADS'99).* 1–12.

[8] 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 Conference (SIGMOD'17).* 821–834.

[9] Abolfazl Asudeh, Azade Nazi, Nan Zhang, Gautam Das, and H. V. Jagadish. 2019. RRR: Rank-regret representative. In *Proceedings of ACM Management of Data Conference (SIGMOD'19).* 263–280.

[10] Abolfazl Asudeh, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2016. Discovering the skyline of web databases. *Proc. VLDB Endow.* 9, 7 (2016), 600–611.

[11] Abolfazl Asudeh, Gensheng Zhang, Naeemul Hassan, Chengkai Li, and Gergely V. Zaruba. 2015. Crowdsourcing pareto-optimal object finding by pairwise comparisons. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'15).* 753–762.

[12] Abolfazl Asudeh, Nan Zhang, and Gautam Das. 2016. Query reranking as a service. *Proc. VLDB* 9, 11 (2016), 888–899.

[13] Stephan Borzsonyi, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings of the International Conference on Data Engineering (ICDE'01).* 421–430.

[14] Herve Bronnimann, Bernard Chazelle, and Jiri Matousek. 1999. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.* 28, 5 (1999), 1552–1575.
[15] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2002. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Datab. Syst.* 27, 2 (2002), 153–187.
[16] 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 the International Conference on Database Theory (ICDT'17).* 11:1–11:19.
[17] 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 the ACM Management of Data Conference (SIGMOD'06).* 503–514.
[18] Timothy M. Chan. 1996. Output-sensitive results on convex hulls, extreme points, and related problems. *Discr. Comput. Geom.* 16, 4 (1996), 369–387.
[19] Timothy M Chan, Elyot Grant, Jochen Konemann, and Malcolm Sharpe. 2012. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12).* 1576–1585.
[20] Timothy M. Chan and Konstantinos Tsakalidis. 2016. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discr. Comput. Geom.* 56, 4 (2016), 866–881.
[21] 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 the ACM Management of Data Conference (SIGMOD'00).* 391–402.
[22] Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. 2014. Computing k-regret minimizing sets. *Proc. VLDB Endow.* 7, 5 (2014), 389–400.
[23] Kenneth L. Clarkson and Peter W. Shor. 1989. Application of random sampling in computational geometry, II. *Discr. Comput. Geom.* 4 (1989), 387–421.
[24] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. 2006. Answering top-k queries using views. In *Proceedings of Very Large Data Bases (VLDB).* 451–462.
[25] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag.
[26] Tamal K. Dey. 1998. Improved bounds for planar k -sets and related problems. *Discr. Comput. Geom.* 19, 3 (1998), 373–382.
[27] Tamal K. Dey and Herbert Edelsbrunner. 1993. Counting triangle crossings and halving planes. In *Proceedings of the Symposium on Computational Geometry (SoCG'93).* 270–273.
[28] Herbert Edelsbrunner. 1987. *Algorithms in Combinatorial Geometry.* Vol. 10. Springer Science & Business Media.
[29] Herbert Edelsbrunner, Nany Hasan, Raimund Seidel, and Xiao Jun Shen. 1989. Circles through two points that always enclose many points. *Geom. Dedicata* 32, 1 (1989), 1–12.
[30] Herbert Edelsbrunner and Emo Welzl. 1985. On the number of line separations of a finite set in the plane. *J. Combin. Theory Ser. A* 38, 1 (1985), 15–29.
[31] P. Erdős, László Lovász, A. Simmons, and Ernst G. Straus. 1973. Dissection graphs of planar point sets. In *A Survey of Combinatorial Theory*, 139–149.
[32] 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 the Symposium on Computational Geometry (SoCG'93).* 38–46.
[33] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Comparing top k lists. *SIAM J. Discret. Math.* 17, 1 (2003), 134–160.
[34] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS'01).* 102–113.
[35] 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 the International Conference on Data Engineering (ICDE'18).* 1653–1656.
[36] Sariel Har-Peled. 2011. *Geometric Approximation Algorithms.* Number 173. American Mathematical Soc.
[37] Sariel Har-Peled. 2011. On the expected complexity of random convex hulls. *CoRR* abs/1111.5340 (2011).
[38] David Haussler and Emo Welzl. 1987. Epsilon-nets and simplex range queries. *Discr. Comput. Geom.* 2 (1987), 127–151.
[39] Vagelis Hristidis and Yannis Papakonstantinou. 2004. Algorithms and applications for answering ranked queries using ranked views. *VLDB J.* 13, 1 (2004), 49–70.
[40] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surv.* 40, 4 (2008), 1–58.
[41] Taylor Kessler Faulkner, Will Brackenbury, and Ashwin Lall. 2015. k-regret queries with nonlinear utilities. *VLDB J.* 8, 13 (2015), 2098–2109.

[42] Janos Komlos, Janos Pach, and Gerhard Woeginger. 1992. Almost tight bounds for epsilon-nets. *Discr. Comput. Geom.* 7, 2 (1992), 163–173.
[43] Nirman Kumar and Stavros Sintos. 2018. Faster approximation algorithm for the k-regret minimizing set and related problems. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX'18)*. 62–74.
[44] Andrey Kupavskii, Nabil H. Mustafa, and Janos Pach. 2016. New lower bounds for epsilon-nets. In *Proceedings of the Symposium on Computational Geometry (SoCG'16)*. 54:1–54:16.
[45] László Lovász. 1971. On the number of halving lines. *Ann. Univ. Sci. Budapest, Eötvös, Sec. Math* 14 (1971), 107–108.
[46] Amélie Marian, Nicolas Bruno, and Luis Gravano. 2004. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.* 29, 2 (2004).
[47] Jiri Matousek. 1992. Reporting points in halfspaces. *Comput. Geom.* 2 (1992), 169–186.
[48] Jiri Matousek. 1993. Linear optimization queries. *J. Algor.* 14, 3 (1993), 432–448.
[49] Ketan Mulmuley. 1991. On levels in arrangement and voronoi diagrams. *Discr. Comput. Geom.* 6 (1991), 307–338.
[50] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino. 2012. Interactive regret minimization. In *Proceedings of the ACM Management of Data (SIGMOD'12)*. 109–120.
[51] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J. Lipton, and Jun Xu. 2010. Regret-minimizing representative databases. *Proc. VLDB Endow.* 3, 1 (2010), 1114–1124.
[52] Janos Pach, William Steiger, and Endre Szemeredi. 1992. An upper bound on the number of planar K-sets. *Discr. Comput. Geom.* 7, 1 (1992), 109–123.
[53] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Trans. Database Syst.* 30, 1 (2005), 41–82.
[54] Peng Peng and Raymond Chi-Wing Wong. 2014. Geometry approach for k-regret query. In *Proceedings of the International Conference on Data Engineering (ICDE'14)*.
[55] Md Farhadur Rahman, Abolfazl Asudeh, Nick Koudas, and Gautam Das. 2017. Efficient computation of subspace skyline over categorical domains. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'17)*. 407–416.
[56] Edgar A. Ramos. 1999. On range reporting, ray shooting and k-level construction. In *Proceedings of the Symposium on Computational Geometry (SoCG'99)*. 390–399.
[57] Micha Sharir, Shakhar Smorodinsky, and Gabor Tardos. 2001. An improved bound for k-sets in three dimensions. *Discr. Comput. Geom.* 26, 2 (2001), 195–204.
[58] Cheng Sheng and Yufei Tao. 2012. Worst-case I/O-efficient skyline algorithms. *ACM Trans. Database Syst.* 37, 4 (2012), 26.
[59] Géza Tóth. 2001. Point sets with many k-sets. *Discr. Comput. Geom.* 26, 2 (2001).
[60] Kasturi Varadarajan. 2010. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'10)*. 641–648.
[61] Akrivi Vlachou and Michalis Vazirgiannis. 2010. Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships. *Data Knowl. Eng.* 69, 9 (2010), 943–964.
[62] Dong Xin, Chen Chen, and Jiawei Han. 2006. Towards robust indexing for ranked queries. In *Proceedings of Very Large Data Bases (VLDB)*. 235–246.
[63] Sepanta Zeighami and Raymond Chi-Wing Wong. 2016. Minimizing average regret ratio in database. In *Proceedings of the ACM Management of Data (SIGMOD'16)*. 2265–2266.

Received November 2020; revised September 2021; accepted April 2022