RRR: Rank-Regret Representative

Abolfazl Asudeh*, Azade Nazi†, Nan Zhang‡, Gautam Das§, H. V. Jagadish¶

*¶University of Michigan; †Google AI; ‡Pennsylvania State University; §University of Texas at Arlington
{asudeh,jag}@umich.edu; azade@google.com; nan@ist.psu.edu; gdas@uta.edu

ABSTRACT

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 to include 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 data set, 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 data set. In contrast, users do understand the notion of rank ordering. Therefore, we consider the position of the items in the ranked list for 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 NP-complete. We use a geometric interpretation of items to bound their ranks on ranges of functions and to utilize combinatorial geometry notions for developing effective and efficient approximation algorithms for the problem. Experiments on real datasets demonstrate that we can efficiently find small subsets with small rank-regrets.

ACM Reference Format:
Abolfazl Asudeh, Azade Nazi, Nan Zhang, Gautam Das, H. V. Jagadish. 2019. RRR: Rank-Regret Representative. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3300080

1 INTRODUCTION

Given a dataset with multiple attributes, the challenge is to combine the values of multiple attributes to arrive at a rank. In many applications, especially in databases with numeric attributes, a weight vector $\vec{w}$ is used to express user preferences in the form of a linear combination of the attributes, i.e., $\sum w_i A_i$. Finding flights based on a linear combination of some criteria such as price and duration [1], diamonds based on depth and carat [2], and houses based on price and floor area [2] are a few examples.

The difficulty is that the concept of "best" lies in the eyes of the beholder. Different 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 [3] set as representing the desirable items in the data set. Such a skyline (resp. convex hull) is the smallest subset of the data that is guaranteed to contain the top choice of a user based on any monotonic (resp. linear) ranking function. Borzsony et. al. [4] initiated the skyline research in the database community and since then 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 [5, 6], especially when there are multiple attributes. Hence, several researchers have tackled [7, 8] 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 function. There has been considerable attention given to the regret-ratio minimizing set [5, 9] problem and its variants [10–16]. Let m_o be the maximum score of the tuples in dataset based on a scoring function f. Also, let m_a be the maximum score for a subset of data. The regret-ratio of the subset for f is the

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3300080

[†] Azade Nazi's work done as a graduate student at the University of Texas, Arlington.

ratio of $(m_o - m_a)$ to m_o. 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 are pointed out in § 7.

Unfortunately, in most real situations, the actual score is a "made up" number with no direct significance. This is even more so the case when attribute values are drawn from different domains. In fact, the score itself could also be on a made-up scale. Considering the regret as a ratio helps, but is far from being a complete solution. For example, wine ratings appear to be on a 100 point scale, with the best wines in the high 90s. However, wines rated below 80 almost never make it to a store. In § 6.2, we conduct an experiment over a collection of 100 top wines. The rating of the best wine in the dataset is at 98 points. A regret of 6 points gives a very small regret ratio of .06, but actually only promises a wine with a rating of 92, which is below median! In other words, a small value of regret ratio can actually result in a large swing in rank. In the case of wines, at least the rating scales see enough use that most wine-drinkers would have a sense of what a score means. But consider choosing a hotel. If a website takes your preferences into account and scores a hotel at 17.2 for you, do you know what that means? If not, then how can you meaningfully specify a regret ratio?

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

Since items in a dataset are usually not uniformly distributed by score, solutions that minimize regret-ratio do not typically minimize rank-regret. In this paper, we seek to find the smallest subset of the given data set that has *rank-regret* of k. We call this subset the *order k rank-regret representative* of the database. (We will write this as k-RRR, or simply as RRR when k is understood from context). The order 1 rank-regret representative of a database (for linear ranking functions) is its convex hull: guaranteed to contain the top choice of any linear ranking function. The convex hull is usually very large: almost the entire data set with five or more dimensions [5, 6]. By choosing a value of k larger than 1, we can drastically reduce the size of the rank-regret representative set, while guaranteeing everyone a choice in their top k even if not the absolute top choice.

Unfortunately, finding RRR is NP-complete, even for three dimensions. However, we find a bound on the maximum rank of an item for a function and use it for designing efficient approximation algorithms. We also find the connection of the RRR problem with well-known notions in combinatorial geometry such as k-set [17], a set of k points in d dimensional space separated from the remaining points by a hyperplane. We show how the k-set notion can be used to find a set that guarantees a rank-regret of k and a logarithmic approximation ratio on the output size. We then show how a smart partitioning of the function space offers an elegant way of finding the rank-regret representative.

Summary of contributions:

- We propose the rank-regret representative as a way of choosing a small subset of the dataset guaranteed to contain at least one good choice for every user.
- We provide a key theorem that, given the rank of an item for a pair of functions, bounds the maximum rank of the item for any function "between" these functions.
- For the 2D case, we provide an approximation algorithm 2DRRR that guarantees to achieve the optimal output size and the approximation ratio of 2 on the rank-regret.
- We identify the connection of the problem with the combinatorial geometry notion of k-set. We review that k-set enumeration can be modeled by graph traversal. Using the collection of k-sets, for the general case with constant number of dimensions, we model the problem by geometric hitting set, and propose the approximation algorithm MDRRR that guarantees the rank-regret of k and a logarithmic approximation-ratio on its output size. We also propose a randomized algorithm for k-set enumeration, based on the coupon collector's problem.
- We propose a function space partitioning algorithm MDRC that, for a fixed number of dimensions, guarantees a fixed approximation ratio on the rank-regret. As confirmed in the experiments, applying a greedy strategy while partitioning the space makes this algorithm both efficient and effective in practice.
- We conduct extensive experiments on two real datasets to verify the efficiency and effectiveness of our proposal.

In the following, we first formally define the terms, provide the problem formulation, and study its complexity in § 2. We provide the geometric interpretation of items, a dual space, and some statements in § 3 that play key roles in the technical sections. In § 4, we study the 2D problem and propose an approximation algorithm for it. § 5 starts by revisiting the k-set notion and its connection to our problem. Then we provide the hitting set based approximation algorithm, as well as the function space partitioning based algorithm, for the general multi dimensional case. Experiment results and related work are provided in § 6 and 7, respectively, and the paper is concluded in § 8.

2 PROBLEM DEFINITION

Database: Consider a database $\mathcal{D}$ of n tuples, each consisting of d attributes $\mathcal{A} = \{A_1, A_2, \cdots, A_d\}$ that may be involved

id	x_1	x_2
t_1	0.8	0.28
t_2	0.54	0.45
t_3	0.67	0.6
t_4	0.32	0.42
t_5	0.46	0.72
t_6	0.23	0.52
t_7	0.91	0.43

Figure 1: A 2D dataset

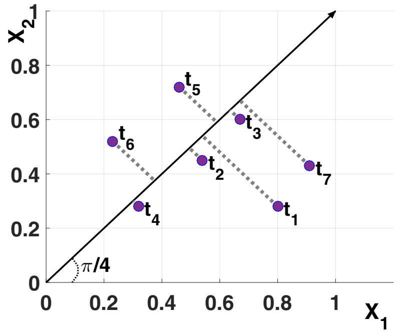

Figure 2: Items of Fig. 1 ordered by $f = x_1 + x_2$

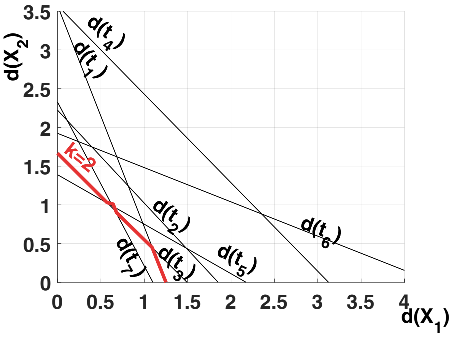

Figure 3: Dual presentation of items in Fig. 1

in a user's preference function[1]. Without loss of generality, we consider $A_i \in \mathbb{R}$ for all $i \in [1, d]$. We represent each tuple $t \in \mathcal{D}$ as a d-dimensional vector $\{t[1], t[2], \cdots t[d]\}$.

Ranking function: Consider a *ranking function* $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that maps each tuple t to a numerical score. We say t_i *outranks* t_j if and only if $f(t_i) > f(t_j)$. We apply any arbitrary tie-breaker when $f(t_i) = f(t_j)$. For each $t \in \mathcal{D}$, let $\nabla_f(t)$ be the *rank* of t in the ordered list of $\mathcal{D}$ based on f. In other words, there are exactly $\nabla_f(t) - 1$ tuples in $\mathcal{D}$ that outrank t according to f.

Ranking functions can take a wide variety of forms. A popular type of ranking functions studied in the database literature is *linear ranking functions*, i.e.,

$$f(t) = \sum_{i=1}^{d} w_i \cdot t[i], \tag{1}$$

where $\vec{w} = \{w_1, w_2, \cdots, w_d\}$ ($\forall i \in [1, d], w_i > 0$) is a weight vector used to capture the importance of each attribute to the final score of a tuple. We use $\mathcal{L}$ to refer to the set of all possible linear ranking functions.

Maxima representation: For a given database $\mathcal{D}$, if the set of ranking functions of interest is known - say $\mathcal{F}$ - then we can derive a compact *maxima representation* of $\mathcal{D}$ by only including a tuple $t \in \mathcal{D}$ if it represents the *maxima* (i.e., is the No. 1 ranked tuple) for at least one ranking function in $\mathcal{F}$. For example, if we focus on linear ranking functions in $\mathcal{L}$, then the maxima representation of $\mathcal{D}$ is what is known in the computational geometry and database literature as the *convex hull* [18] of $\mathcal{D}$. Similarly, the set of *skyline* tuples [4], a superset of the convex hull, form the maxima representation for the set of all monotonic ranking functions [19].

A problem with the maxima representation is its potentially large size. For example, depending on the "curvature" of the shape within which the tuples are distributed, even in 2D, the convex hull can be as large as $O(n^{1/3})$ [6]. The problem gets worse in higher dimensions [5, 20]. In practice, even for a database with dimensionality as small as $d = 5$, the convex hull can often be as large as $O(n)$[5].

To address this issue, we propose in this paper to relax the definition of maxima representation in order to reduce its size. Specifically, instead of requiring the representation to contain the top-1 item for every ranking function, we allow the representation to stand so long as it contains at least one of the top-k items for every ranking function. This tradeoff between the compactness of the representation and the "satisfaction" of each ranking function is captured in the following formal definitions of *rank regret*:

Definition 1. *Given a subset of tuples $X \subseteq \mathcal{D}$ and a ranking function f, the rank-regret of X for f is the minimum rank of all tuples in X according to f. Formally,*

$$RR_f(X) = \underset{\forall t \in X}{argmin}(\nabla_f(t))$$

Definition 2. *Given a subset of tuples $X \subseteq \mathcal{D}$ and a set of ranking functions $\mathcal{F}$, the rank-regret of X for $\mathcal{F}$ is the maximum rank-regret of X for all functions in $\mathcal{F}$ - i.e.,*

$$RR_{\mathcal{F}}(X) = \underset{\forall f \in \mathcal{F}}{argmax}(RR_f(X))$$

Definition 3. *Given a set of ranking functions $\mathcal{F}$ and a user input $k \geq 1$, we say $X \subseteq \mathcal{D}$ is a rank-regret representation of $\mathcal{D}$ if and only if X has the rank-regret of at most k for $\mathcal{F}$, and no other subset of $\mathcal{D}$ satisfies this condition while having a smaller size than X. Formally:*

$$\min_{\forall X \subseteq \mathcal{D}} \|X\| \qquad s.t.\ RR_{\mathcal{F}}(X) \leq k$$

Problem Formulation: Finding the rank-regret representative of the dataset $\mathcal{D}$ is our objective in this paper. Therefore, we define the problem as follows:

> **Rank-Regret Representative (RRR) Problem:** *Given a dataset $\mathcal{D}$, a set of ranking functions $\mathcal{F}$, and a user input k, find the rank-regret representative of $\mathcal{D}$ for $\mathcal{F}$ and k according to Definition 3.*

---

[1] Each tuple could also include additional attributes that are not involved in the user preferences. We do not consider these attributes for the purpose of this paper.

We note that there is a dual formulation of the problem - i.e., a user specifies the output size $|X|$, and aims to find X that has the minimum rank-regret. Interestingly, a solution for the RRR problem can be easily adopted for solving this dual problem. Given the solver for RRR, for the set size x, one may apply a binary search to vary the value of k in the range $[1, n]$ and, for each value of k, call the solver to find RRR. If the output size is larger than x, then the search continues in the upper half of the search space for k, or otherwise moves to the lower half. Given an optimal solver for RRR, this algorithm is guaranteed to find the optimal solution for the dual problem at a cost of an additional $\log n$ factor in the running time.

In the rest of the paper, we focus on $\mathcal{L}$, the class of linear ranking functions.

Complexity analysis: The decision version of RRR problem asks if there exists a subset of size r of $\mathcal{D}$ that satisfies the rank-regret of k. Somewhat surprisingly, even though no solution for RRR exists in the literature, we can readily use previous results to prove the NP-completeness of RRR. Specifically, the (k, ϵ)-regret problem studied in Agrawal et al. [14] asks if there exists a set that guarantees the maximum regret-ratio of ϵ from the top k-th item of any linear ranking function. Note that the $(2, 0)$-regret problem is the equivalent of RRR problem for $k = 2$. Given that the NP-completeness proof in [14] covers the $(2, 0)$-case when $d \geq 3$, through a reduction to the NP-completeness of the convex polytope vertex cover (CPVC) problem proven by Das et al. [21], the NP-completeness of RRR follows.

We would like to reemphasize that even though the complexity of RRR was established in existing work, RRR is still a novel problem to study because all previous work in the regret ratio area focused on the case where $\epsilon > 0$. In other words, they seek approximations on the absolute score achieved by tuples in the compact representation - a strategy which, as discussed in the introduction, could lead to a significant increase on rank regret because many tuples may congregate in a small band of absolute scores. RRR, on the other hand, focus on the rank perspective (i.e., $\epsilon = 0$) and assumes no specific distribution of the absolute scores.

3 GEOMETRIC INTERPRETATION OF ITEMS

In this section, we use the geometric interpretation of items, explain a dual transformation, and propose a theorem that plays a key role in designing the RRR algorithms.

Each item $t \in \mathcal{D}$ with d scalar attributes can be viewed as a point in $\mathbb{R}^d$. As an example, Figure 1 shows a sample dataset with $n = 7$ items, defined over $d = 2$ attributes. Figure 2 shows these items as the points in $\mathbb{R}^2$. In this space, every linear preference function f with the weight vector $\vec{w} = \{w_1, w_2, \cdots, w_d\}$ can be viewed as a ray that starts at the origin and passes through the point $\{w_1, w_2, \cdots, w_d\}$. For each item $t \in \mathcal{D}$, consider the orthogonal line to the ray of f that passes through t; let the projection of t on f be the intersection of this line with the ray of f. The ordering of items based on f is the same as the ordering of the projection of them on f where the items that are farther from the origin are ranked higher. For example, Figure 2 shows the ray of the function $f = x_1 + x_2$, as well as the ordering of items based on it. As shown in the figure, the items are ranked as t_7, t_3, t_5, t_1, t_2, t_6, and t_4, based on $f = x_1 + x_2$. Every ray starting at the origin in $\mathbb{R}^d$ is represented by $d - 1$ angles. For example in $\mathbb{R}^2$, every ray is identified by a single angle. In Figure 2, the ray of function $f = x_1 + x_2$ is identified by the angle $\theta = \pi/4$.

Small changes in the weights of a function will move the corresponding ray slightly, and hence change the projection points of items. However, it may not change the ordering of items. In fact, while the function space is continuous and the number of possible weight vectors is infinite, the number of possible orderings between the items is, combinatorially, bounded by $n!$.

In order to study the ranking of items based on various functions, throughout this paper, we consider the dual space [17] that transforms a tuple t in $\mathbb{R}^d$ to the hyperplane $\mathsf{d}(t)$ as follows:

$$\mathsf{d}(t) : \sum_{i=1}^{d} t[i].x_i = 1 \tag{2}$$

In the dual space, the ray of a linear function f with the weight vector $w = \{w_1, w_2, \cdots, w_d\}$ remains the same as the original space: the origin-starting ray that passes through the point w. The ordering of items based on a function is specified by the intersection of hyperplanes $\mathsf{d}(t_i)$ with it. However, unlike the original space, the intersections that are closer to the origin are ranked higher. Using Equation 2, every tuple in two dimensional space gets transformed to the line $\mathsf{d}(t) : t[1].x_1 + t[2].x_2 = 1$. Figure 3 shows the items in the example dataset of Figure 1 in the dual space. Looking at the intersection of dual lines with the x_1 axis in Figure 3, one can see that the ordering of items based on $f = x_1$ is t_7, t_1, t_3, t_2, t_5, t_4, and t_6; hence, for any set X containing t_7 or t_1, for $f = x_1$ (i.e., $w = \{1, 0\}$), $RR_f(X) \leq 2$.

The set of dual hyperplanes defines a dissection of $\mathbb{R}^d$ into connected convex cells named as the arrangement of hyperplanes [17]. The borders of the cells in the arrangement are $d - 1$ dimensional facets. For example, in Figure 3, the arrangement of dual lines dissect the $\mathbb{R}^2$ space into connected convex polygons. The borders of the convex polygons are one dimensional line segments. For every facet in the arrangement consider a line segment starting from the origin and ending on it. Let the level of the facet be the number of hyperplanes intersecting this line segment. We define a top-k border (or simply k-border) as the set of facets having level k. For example, the red chain consisting of piecewise linear segments in Figure 3, shows the top-k border for $k = 2$. For any function f, the hyperplanes intersecting the ray of f on or below the top-k border are the top-k. Looking at the red line in Figure 3, one may confirm that:

- The top-k border is not necessarily convex.
- A dual hyperplane $\mathsf{d}(t_i)$ may contain more than one facet of the top-k border. For example, $\mathsf{d}(t_3)$ in Figure 3 contains two line segments of the top-2 border.

In the following, we propose an important theorem that is the key to designing the 2D algorithm, as well as the practical algorithm in MD.

Theorem 1. *For any item $t \in \mathcal{D}$ consider two (if any) functions f and f' where $\nabla_f(t) \leq k_1$ and $\nabla_{f'}(t) \leq k_2$. Also, consider a line segment $l_{f,f'}$ starting from a point on the ray of f and ending at a point on the ray of f'. For any function f'' that its ray intersects $l_{f,f'}$, $\nabla_{f''}(t) \leq k_1 + k_2$.*

Proof. We use the dual space and prove the theorem by contradiction. In the dual space, consider the 2D plane passing through the rays of f and f' – referred as $\mathbb{R}^2_{f,f'}$. Note that $\mathbb{R}^2_{f,f'}$ is the affine space for the origin starting rays that intersect $l_{f,f'}$. The intersection of each hyperplane $\mathsf{d}(t_i)$ and this plane is a line that we name as $\mathsf{L}(t_i)$. The arrangement of lines $\mathsf{L}(t_i)$, $\forall t_i \in \mathcal{D}$, identify the orderings of items $t \in \mathcal{D}$ based on any origin-starting ray (function) that falls in $\mathbb{R}^2_{f,f'}$. This is similar to Figure 3 in that the arrangement of lines $\mathsf{d}(t_i)$ identify the possible ordering of items in Figure 1. For any pair of items t_1 and t_2, the intersection of the lines $\mathsf{L}(t_1)$ and $\mathsf{L}(t_2)$ shows the function (the origin-starting ray that passes through the intersection) that ranks t_1 and t_2 equally well, while on one side of this point t_1 outranks t_2, but t_2 outranks t_1 on the other side. Note that since $\mathsf{L}(t_1)$ and $\mathsf{L}(t_2)$ are both (one dimensional) lines, they intersect at most once.

Now consider the item t and its corresponding line $\mathsf{L}(t)$ in the arrangement. Since $\nabla_f(t) \leq k_1$, there exist at most $k - 1$ lines below it on the ray of f. Moving from the ray of f toward the ray of f', in order for t to have a rank greater than $k_1 + k_2$, $\mathsf{L}(t)$ has to intersect with at least k_2 lines $\mathsf{L}(t_i)$ in a way that after the intersection points (toward f') those points outrank t. Since every pair of lines has at most one intersection point, $\mathsf{L}(t)$ will not intersect with those lines any further. As a result, those (at least) k_2 points keep outranking t, and thus t cannot have a rank smaller than or equal to k_2 again, which contradicts the fact that $\nabla_{f'}(t) \leq k_2$. □

Intuitively, Theorem 1 states that if f'' lies "between" f and f', then the rank of an item based on f'' is at worst the summation of its rank in f and f'. We use this result in the next section, as well as § 5, for providing approximation algorithms for RRR.

4 RRR IN 2D

In this section, we study the special case of two dimensional (2D) data in which $d = 2$. In § 2, we discussed the complexity of the problem for $d \geq 3$. However, we believe that the complexity of the problem is due to the complexity of covering the possible top-k results and therefore, provide an approximation algorithm for 2D. We consider the items in the dual space and use Theorem 1 as the key for designing the algorithm 2DRRR. Later in § 5, we extend this theorem to design a practical algorithm for general cases.

Based on our discussion about the top-k border in the previous section, each dual line may contain multiple segments of the top-k border. As a results, for each item, the set of functions for which the item is in the top-k, is a collection of disjoint intervals. Based on Theorem 1, if we take the union of these intervals (i.e., the convex closure), we get a single interval, in which the item is guaranteed to be in the top-$2k$. This, we are effectively applying Theorem 1 to get the 2-approximation factor.

At a high-level, the algorithm 2DRRR consists of two parts. It first makes an angular sweep of a ray anchored at the origin, from the x-axis (angle 0°) toward the y-axis (angle $\pi/2°$) so that for every item $t \in \mathcal{D}$, it finds the first (smallest angle) and the last function (largest angle) for which t is in top-k. Then it transforms the problem into an instance of one-dimensional range cover [22] and solves it optimally.

The first part, i.e., the angular sweep, is described in Algorithm 1 (the function FINDRANGES)[2]. For every item t the algorithm aims to find the first ($b[t]$) and the last ($e[t]$) function for which $\nabla_f(t) \leq k$. FINDRANGES initially orders the items based on their x_1-coordinates and puts them in a list L that keeps tracks of orderings while moving from x to y-axis. It uses a min-heap data structure to maintain the ordering exchanges between the adjacent items in L. Please note that each ordering exchange [23] is always between two adjacent items in L. Using Equation 2, the angle of the ordering exchange between two items L_i and L_{i+1} is as follows:

$$\theta_{L_i,L_{i+1}} = \arctan \frac{L_{i+1}[1] - L_i[1]}{L_i[2] - L_{i+1}[2]}$$

For the items that are initially in the top-k, FINDRANGES sets $b[t]$ to the angle 0°. Then, it sweeps the ray and pops the next ordering exchange from the heap. Upon visiting an ordering exchange, the algorithm updates the ordered list L. If the exchange occurs between the items at rank k and $k + 1$: (i) if this is the first time L_{k+1} enters the top-k, the algorithm sets $b[L_{k+1}]$ as the current angle, and (ii) for the item L_k that leaves the top-k, it sets $e[k]$ to the current angle.

---

[2] Pseudocode of all algorithms are provided in Appendix D.

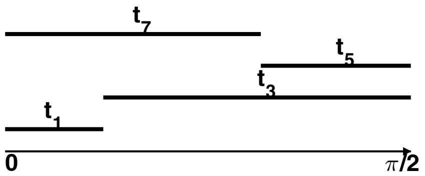

Figure 4: The ranges for Figure 3

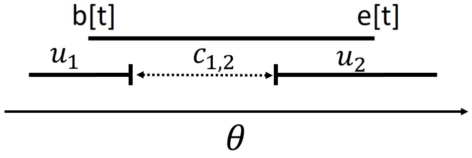

Figure 5: A contradictory example

The algorithm will update $e[k]$ later on if it becomes a top-k again. Figure 4 shows the ranges for the example dataset in Figure 1 and $k = 2$ (k-border is shown in Figure 3).

After computing the ranges for the items, the problem is transformed into a one dimensional range cover instance. The objective is to cover the function space (the range between 0° and $\pi/2°$) using the least number of ranges. The greedy approach leads to the optimal solution for this problem – that is, at every iteration, select the range with the maximum coverage of the uncovered space.

At every iteration, the uncovered space is identified by a set of intervals. Due to the greedy nature of the algorithm, the range of each remaining item intersects with at most one uncovered interval. To explain this by contradiction, consider an item t that its range intersects with two (or more) uncovered intervals (Figure 5). Let u_1 and u_2 be these intervals. Also, let us name the covered space between u_1 and u_2 as $c_{1,2}$. (i) Since the range of t intersects with both u_1 and u_2, $c_{1,2}$ is contained within the range of t, which implies the range of t is larger than $c_{1,2}$. (ii) $c_{1,2}$ should be covered by the range of at least one previously selected item t'. Also, since the ranges of items are continuous, the range of t' cannot be larger than $c_{1,2}$. As a result, the range of t' is less than the range of t, which contradicts the fact that the ranges are selected greedily.

Using this observation, after finding the ranges for each item, 2DRRR (Algorithm 2) uses a sorted list to keep track of the uncovered intervals. The elements of the list are in the form of $\langle \theta_i, \vdash / \dashv \rangle$, where $\vdash$ (resp. $\dashv$) specifies that this is the beginning (resp. the end) of an uncovered interval.

At every iteration, for each item that has still not been selected, the algorithm applies a binary search to find the element in U_k that $b[t_i]$ falls right before it, i.e., $U_k[1] \geq b[t_i]$ and $\nexists k' < k$ such that $U_{k'}[1] \geq b[t_i]$. Then depending on whether U_k specifies the beginning ($\vdash$) or the end ($\dashv$) of an uncovered interval, it computes how much of the uncovered region t_i covers. The algorithm chooses the item with the maximum coverage, adds it to the selected set, and updates the uncovered intervals accordingly. It stops when no more uncovered intervals are left.

As an example, for the dataset in Figure 1, if we execute Algorithm 2 on the ranges provided in Figure 4, it returns the set $\{t_3, t_1\}$.

THEOREM 2. *The algorithm* 2DRRR *runs in* $O(n^2 \log n)$ *time.*

PROOF. Intuitively, the summation of the cost of each iteration of the greedy algorithm is used to derive the running time. Please find the details of the proof in Appendix C. □

THEOREM 3. *The output size of* 2DRRR *is not more than the size of the optimal solution for RRR.*

PROOF. The proof follows from the fact that the ranges identified by Algorithm 1 provide a superset for each top-k result. Please refer to Appendix C for details. □

THEOREM 4. *The output of* 2DRRR *guarantees the maximum rank-regret of* $2k$.

PROOF. This result is easy to prove, by applying Theorem 1. The details are provided in Appendix C. □

5 RRR IN MD

In multi-dimensional cases (MD) where $d > 2$, the continuous function space becomes problematic, the geometric shapes become complex, and even the operations such as computing the volume of a shape and the set operations become inefficient. Therefore, in this section, we use the k-set notion [17] to take an alternative route for solving the RRR problem by transforming the continuous function space to discrete sets. This leads to the design of an approximation algorithm that guarantees the rank-regret of k, introduces a log approximation-ratio in the output size, and runs in polynomial time, for a constant number of dimensions. We will explain the details of this algorithm in § 5.2. Then, in § 5.3, we propose the function-space partitioning algorithm MDRC that uses the result of Theorem 1 in its design for solving the problem without finding the k-sets. Note that proposed algorithms in this section are also applicable for 2D.

5.1 k-Set and Its Connection to RRR

k-set is an important notion in combinatorial geometry with applications including half-space range search [24, 25]. Given a set of points in $\mathbb{R}^d$, a k-set is a collection of exactly k points in the point set that are strictly separable from the rest of points using a $d-1$ dimensional hyperplane.

Consider a finite set P of n points in the euclidean space $\mathbb{R}^d$. A hyperplane h partitions it into $P^+ = P \cap h^+$ and $P^- = P \cap h^-$, called half spaces of P, where h^+ (resp. h^-) is the open half space above[3] (resp. below) h [17]. The hyperplane h in the

---

[3] We use the word above (resp. below) to refer to the half space that does not contain (resp. contains) the origin.

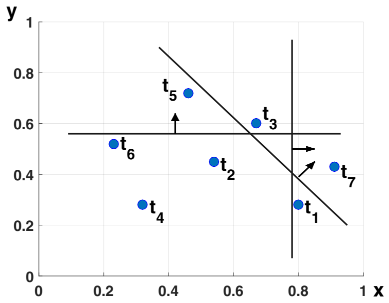

Figure 6: The k-sets of Figure 1 for $k = 2$.

Euclidean space $\mathbb{R}^d$ can be uniquely defined by a point ρ on it and a d dimensional normal vector v orthogonal to it, and has the following equation:

$$v[1](x_1 - \rho[1]) + v[2](x_2 - \rho[2]) + \cdots + v[d](x_d - \rho[d]) = 1 \quad (3)$$

A half space S of P is a k-set if $card(S) = k$. Without loss of generality, we consider the positive half spaces and $v[i] \geq 0$. That is, $S \subseteq P$ is a k-set if $\exists$ a point ρ and the positive normal vector v such that $S = h(\rho, v)^+$ and $card(h(\rho, v)^+) = k$. For example, the empty set is a 0-set and each point in the convex hull of P is a 1-set. We use $\mathcal{S}$ to refer to the collection of k-sets of P; i.e., $\mathcal{S} = \{S \subseteq P | S \text{ is a } k\text{-set}\}$. For example, Figure 6 shows the collection of k-sets for $k = 2$ for the dataset of Figure 1. As we can see, the 2-sets are $\mathcal{S} = \{\{t_1, t_7\}, \{t_7, t_3\}, \{t_3, t_5\}\}$.

If we consider items $t \in \mathcal{D}$ as points in $\mathbb{R}^d$, the notion of k-sets is interestingly related to the notion of top-k items, as the following arguments show:

- A hyperplane $h(\rho, v)$ describes the set of all points with the same score as point ρ, for the ranking function f with the weight vector v, i.e., the set of attribute-value combinations with the same scores as ρ based on the ranking function f.
- If we consider a hyperplane $h(\rho, v)$ where $card(h(\rho, v)^+) = k$, the set of points belonging to $h(\rho, v)^+$ is equivalent to the top-k items of $\mathcal{D}$ for the ranking function with weight vector v.

Lemma 5. *Let $\mathcal{S}$ be the collection of all k-sets for the points corresponding to the items $t \in \mathcal{D}$. For each possible ranking function f, there exists a k-set $S \in \mathcal{S}$ such that top-$k(f)$=S.*

Proof. We provide the proof by contradiction. Please refer to Appendix C for the details. □

Based on Lemma 5, all possible answers to top-k queries on linear ranking functions can be captured by the collection of k-sets. This will help us in solving the RRR problem in § 5.2. As we shall explain in § 7, the best known upper bound on the number of k-sets in $\mathbb{R}^2$ and $\mathbb{R}^3$ are $O(nk^{1/3})$ [26] and $O(nk^{3/2})$ [27]. For $d > 3$, the best known upper bound is $O(n^{d-\varepsilon})$ [28], where $\varepsilon > 0$ is a small constant[4]. However, as we shall show in § 6, in practice $|\mathcal{S}|$ is significantly smaller than the upper bound.

In the technical report [29], we review the k-set enumeration. For the 2D case, a ray sweeping algorithm (similar to Algorithm 1) that follows the k-border finds the collection of k-sets. For higher dimensions, the enumeration can be modeled as a graph traversal problem [30]. The algorithm considers the k-set graph $G(V, E)$ in which the vertices are the k-sets and there is an edge between two k-sets if the size of their intersection is $k - 1$. We discuss the connectivity of the graph, and explain how to traverse it and enumerate the k-sets.

Next, we use the k-set notion for developing an approximation algorithm for RRR that guarantees a rank-regret of k and a logarithmic approximation ratio on the output size.

5.2 MDRRR: Hitting-Set Based Approximation Algorithm

As we discussed in § 5.1 the collection of k-sets contains the set of all possible top-k results for the linear ranking functions. As a result, a set of tuples $X \subseteq \mathcal{D}$ that contains at least one item from each k-set is guaranteed to have at least one of the items in the top-k of any linear ranking function; which implies that X satisfies the rank regret of k. On the other hand, since every k-set $S = h(\rho, v)^+$ is at least the top-k of the linear function f with the weight vector v, a subset $X' \subseteq \mathcal{D}$ that does not contain any of the items of a k-set S does not satisfy the rank regret of k.

One can see that given the collection of k-sets, our RRR problem is similar to the *minimum hitting set problem* [31]. Given a universe of n items $\mathcal{D}$, and a collection of sets $\mathcal{S}$ where each set $S \in \mathcal{S}$ is a subset of $\mathcal{D}$, the minimum hitting set problem asks for the smallest set of items $X' \subseteq \mathcal{D}$ such that X' has a non-empty intersection with every set S of $\mathcal{S}$. The minimum hitting set problem is known to be NP-complete [31] and the existing approximation algorithm provides a factor of $O(\log n)$ from the optimal size c. A deterministic polynomial time algorithm with an improved factor of $O(\delta \log \delta c)$ had been proposed by [22] for a specific instance of this problem – the *geometric hitting set problem* – where δ is the Vapnik Chervonenkis dimension (VC-dimension). The VC-dimension is defined as the cardinality of the largest set of points $Y \subseteq \mathcal{D}$ that can be *shattered* by $\mathcal{S}$, i.e., the system introduced by $\mathcal{S}$ on Y contains all the subsets of Y [32]. In the RRR problem, since the k-sets are defined by half spaces, the VC-dimension is d (the number of attribute) [22, 33].

---

[4] Note that this is polynomial for a constant d.

Next we formally show the mapping of the RRR problem into the geometric hitting set problem, and provide the detail of approximation algorithm.

Mapping to Geometric hitting set problem: Given a set space $R = (D, \mathcal{S})$, where $\mathcal{S}$ is the collection of k-sets and $D = \underset{\forall s_i \in \mathcal{S}}{\cup} S_i$ is the set of points, find the smallest set $X \subseteq D$ such that $\forall S \in \mathcal{S}, \exists t \in X$ s.t. $t \in S$.

In MDRRR (Algorithm 3), we use the approximation algorithm for the geometric hitting set problem that is proposed in [22] using the concept of ϵ-nets [34]. More formally, an ϵ-net of D for $\mathcal{S}$ is a set of points $X \subseteq D$ such that X contains a point for every $S \in \mathcal{S}$ with size of at least $\epsilon|D|$. Algorithm 3 shows the psudocode of MDRRR, the approximation algorithm that uses the mapping to geometric hitting set problem. The algorithm initializes the weight of each point to one. It then iteratively, in polynomial time, selects (using weighted sampling) a small-sized set of tuples $X \subseteq D$ that intersects all highly weighted sets in $\mathcal{S}$. More formally if a set $X \subseteq D$ intersects each k-set S of $\mathcal{S}$ with weight larger than $\epsilon W(D)$, where $W(D)$ is the total weights of of points in D, then X is an ϵ-net. If X is not a hitting set (lines 4-9), then the algorithm doubles the weight of the points in the particular sets S of $\mathcal{S}$ missed by X.

Discussion: In summary, considering the one-to-one mapping between the RRR problem and the geometric hitting set problem over the collection of k-sets, we can see that:

- MDRRR guarantees rank-regret of k. That is because MDRRR is guaranteed to return at least one item from each k-set in $\mathcal{S}$, the set of all top-k results.
- MDRRR guarantees the approximation ratio of $O(d \log dc)$, where c is the optimal output size and d is the number of attributes.
- MDRRR runs in polynomial time. This is because it has been shown in [22] that the number of iterations the algorithm must perform is at most $O(c \log \frac{n'}{c})$, where n' is the number of points in D, and c is the size of the optimal hitting set. Moreover, recall that MDRRR needs the collection of k-sets, which can be enumerated by traversing the k-set graph [29] and runs in polynomial time.

Nevertheless, although it runs in polynomial time, the MDRRR algorithm is quite impractical as described above. It needs the collection of k-sets ($\mathcal{S}$), as input. Therefore, its efficiency depends on the k-set enumeration and the size of $|\mathcal{S}|$. Although, as we shall show in § 6, in practice the size of $|\mathcal{S}|$ is reasonable and, as explained in the technical report [29], the k-set graph traversal algorithm is linear in $|\mathcal{S}|$, the algorithm does not scale beyond a few hundred items in practice. The reason is that while exploring each k-set, it needs to solve much as n linear programs, each of size n constraints over d variables. This makes the enumeration extremely inefficient. Therefore, we need to explore practical alternatives to the k-set enumeration algorithm.

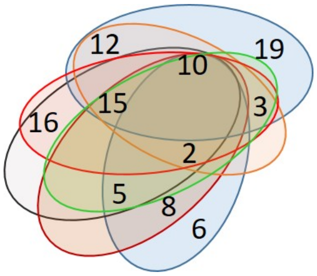

Figure 7: Illustration of overlap between the k-sets of a sample of 20 items from the DOT dataset (c.f. § 6) while $d = 2$

Hence, we propose K-SET$_r$, a sampling-based alternative for the k-set enumeration. K-SET$_r$ is a randomized approach based on the *coupon collector's problem* [35] that considers the one mapping between the linear ranking functions and the k-sets and generates random ranking functions for k-set enumeration. Further details about K-SET$_r$ is provided in Appendix A.

Let $\mathcal{S}_r$ be the collection of k-sets discovered by K-SET$_r$. After finding $\mathcal{S}_r$, we pass it, instead of $\mathcal{S}$ to MDRRR. Since K-SET$_r$ does not guarantee the discovery of all k-sets, the output of the hitting set algorithm does not guarantee the rank-regret of k for the missing k-sets. However, the missing k-sets (if any) are expected to be in the very small regions that has never been hit by a randomly generated function. Also, the fact that the adjacent k-sets in the k-set graph vary in only one item, further reduces the chance that a missing k-set is not covered. Therefore, this is very unlikely that the top-k of a randomly generated function is not within the output.

On the other hand, since K-SET$_r$ finds a subset of k-sets, the output size for running the hitting set on top of the subset (i.e., $\mathcal{S}_r$) is not more than the output size of running the hitting set on $\mathcal{S}$. As a result, the output size remains within the logarithmic approximation factor.

5.3 MDRC: Function Space Partitioning

Given the collection of k-sets, the hitting set based approximation algorithm MDRRR guarantees the rank-regret of k while introducing a logarithmic approximation in its output size. Despite these nice guarantees, MDRRR still suffers from k-set enumeration, as it can only be executed after the k-sets have been discovered. Therefore, as we shall show in § 6, in practice it does not scale well for large problem instances. One observation from the k-set graph is the high overlap between the k-sets, as the adjacent k-sets differ in only one item. As a result many of them may share at least one item. For example, we selected 20 random items from the DOT (Department of Transportation) dataset (c.f. § 6) while setting $d = 2$. By performing an angular sweep of a ray from the x-axis to the y-axis while following the k-border (see Figure 3), we enumerated the k-sets. In Figure 8, we illustrate the overlap between these k-sets. The figure confirms the high overlap between the k-sets where the item with id 2 appears in all except one of the sets. This motivates the idea of finding these items without enumerating the k-sets. In addition, the top-k of two similar functions (where the angle between their corresponding rays is small) are more likely to intersect.

We use these observations in this subsection and propose the function-space partitioning algorithm MDRC which (similar to the 2D algorithm 2DRRR) leverages Theorem 1 in its design. The algorithm is based on the extension of Theorem 1 that bounds the rank of an item that appears in the top-k of the functions corresponding to the corners of a convex polytope in the function space.

MDRC considers the function space in which every function (i.e., a ray starting from the origin) in $\mathbb{R}^d$ is identified as a set of $d-1$ angles. Rather than discovering the k-sets and transforming the problem to a hitting set instance, here our objective is to cover the *continuous function space* (instead of the discrete k-set space). Intuitively, we propose a recursive algorithm which, at every recursive step, considers a hyper-rectangle in the function space, and either assigns a tuple to the functions in the space, or uses a round robin strategy on the $d-1$ angles to break down the space in two halves, and to continue the algorithm in each half. This partitioning strategy is similar to the Quadtree data structure [36]. The reason for choosing this strategy is to maximize the similarity of the functions in the corners of the hyper-rectangles to increase the probability that their top-k sets intersect. MDRC also follows a *greedy* strategy in covering the function space, by partitioning a hyper-rectangle only if it cannot assign a tuple to it.

Consider the space of possible ranking functions in $\mathbb{R}^d$. This is identified by a set of $d-1$ angles $\Theta = \{\theta_1, \theta_2, \cdots, \theta_{d-1}\}$, where $\theta_i \in [0, \pi/2]$. To explain the algorithm, consider the binary tree where each node is associated with a hyper-rectangle in the angle space, specified by a range vector of size $d-1$. The root of the tree is the complete angle space, that is the hyper-rectangle defined between the ranges $[0, \pi/2]$ on each dimension. Let the level of the nodes increase from top to bottom, with the level of the root being 0. Every node at level l uses the angle $\theta_{l\%(d-1)+1}$ to partition the space in two halves, the negative half (the left child) and the positive half (the right child). Figure 8 illustrates an example of such tree for 3D. The root uses the angle θ_1 to partition the space. The left child of the root is associated with the rectangle specified by the ranges $\{[0, \pi/4], [0, \pi/2]\}$ and the right child shows the one by $\{[\pi/4, \pi/2], [0, \pi/2]\}$. The nodes at level 1 use the angle $\theta_{1\%2+1} = \theta_2$ for partitioning the space.

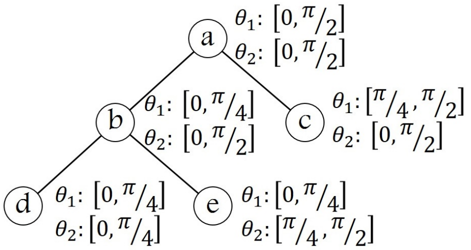

Figure 8: Illustration of space partitioning and the recursion tree of Algorithm 5

At every node, the algorithm checks the top-k items in the corners of the node's hyper-rectangle and if there exists an item that is common to all of them, returns it. Otherwise, it generates the children of the node and iterates the algorithm on the children. The algorithm combines the outputs of each of the halves as its output. We implement MDRC as a recursive algorithm (Algorithm 5). The algorithm is started by calling MDRC $(\mathcal{D}, n, d, k, 0, \{[0, \pi/2]\ |\forall 0 < i < d\})$.

As a running example for the algorithm, let us consider Figure 8. The algorithm starts at the root, partitions the space in two halves, as the intersection of the top-k of its hyper-rectangle's corners are empty, and does the recursion at nodes b and c. The node c finds the item t_c which appears in the top-k of all of its corners and returns it to a. Node b, however, cannot find such an item and does the recursion by partitioning its hyper-rectangle along the angle θ_2. Nodes d and e find the items t_d and t_e and return them to b which returns $\{t_d, t_e\}$ to the root. The root returns $\{t_c, t_d, t_e\}$ as the representative.

THEOREM 6. *The algorithm* MDRC *guarantees the maximum rank-regret of dk.*

PROOF. This proof uses Theorem 1 to extend the maximum rank bound from one dimensional ranges to $(d-1)$ dimensions. Please find the details in Appendix C. □

Theorem 6 uses the result of Theorem 1 to provide an upper bound on the maximum rank of the items assigned to each hyper-rectangle, for the functions inside it. However, as we shall show in § 6, the rank-regret of its output in practice is much less. For all the experiments we ran, the output of MDRC satisfied the maximum rank of k for all settings. Also, following the greedy nature in partitioning the function space, as we shall show in § 6, the output of MDRC in all cases was less than 40. In addition, in § 6, we show that this algorithm is very efficient and scalable in practice.

6 EXPERIMENTAL EVALUATION

6.1 Datasets

US Department of Transportation flight database (DOT)[5]*:* This database is widely used by third-party websites to identify

---

[5] www.transtats.bts.gov/DL_SelectFields.asp?

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, over the scalar attributes `Dep-Delay`, `Taxi-Out`, `Actual-elapsed-time`, `Arrival-Delay`, `Air-time`, `Distance`, `Taxi-in`, and `CRS-elapsed-time`. For `Air-time` and `Distance` higher values are preferred while for the rest of attributes lower values are better.

Blue Nile (BN)[6]*:* Blue Nile is the largest diamonds online retailer in the world. We collected its catalog that contained 116,300 diamonds at the time of our collection. We consider the scalar attribute `Carat`, `Depth`, `LengthWidthRatio`, `Table`, and `Price`. For all attributes, except `Price`, higher values are preferred. The value of the diamonds highly depend on these measurement, small changes in these scores may mean a lot in terms of the quality of the jewel: For example, while the listed diamonds range from 0.23 carat to 20.97, minor changes in the carat affects the price. We considered two similar diamonds, where one is 0.5 carat and the other is 0.53 carat. Even though all other measures are similar for both diamonds, the second is 30% more expensive than the first one. This is also correct for `Depth`, `LengthWidthRatio`, and `Table`. *Such settings where slight changes in the scores may dramatically affect the value* (and the rank) of the items, highlight the motivation of rank-regret.

Wine dataset[7]*:* Each year, Wine Spectator publishes a list of top wines reviewed over past 12 months. This annual list honors successful wineries, regions and vintages around the world. We collect their list of top wines for 2017. The dataset contains 100 items, defined over the attributes `rating`, `vintage year`, and `price`. We use this dataset in § 6.2 for validating our proposal.

We normalize the values of datasets in a way that a value v of a higher-preferred attribute A as $v/\max(A)$ and for each lower-preferred attribute A, we do it as $(\max(A)-v)/\max(A)$.

Next, we start the experiments by validating our proposal in § 6.2, using the wine dataset. Then, we evaluate the performance of the proposed algorithms in § 6.3.

6.2 Validation

Users of a dataset with multiple attributes may have different preferences for finding the "best" fit for their need. Pareto-optimal is the set that contains the best of any preference function, however, may be large. A neat way of finding a smaller subset is by defining a notion of regret and bounding it for any specific user. While regret-ratio defines regret on the score, rank-regret does it based on the ranking of items. Let m_o be the maximum score of an item in the dataset based on f, while m_a is the one for a subset of data. The subset satisfies a regret-ratio of δ for f, if $\frac{m_o-m_a}{m_o} \leq \delta$. The rank-regret of the subset, on the other hand, is k, if it contains one of the top-k of the dataset based on f.

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

Consider a user who prefers the wine with maximum rating. Wine ratings are in the scale of 0 to 100. In our dataset, the wine with the maximum rating is "Clos des Papes Châteauneuf-du-Pape" with the rating of 98. A regret of 6 points on the rating gives a small regret-ratio of 0.06. The small regret-ratio indicates the containment of a good "representative" for the user's choice. Note that any subset that contains a wine with rating of 92 satisfies this regret-ratio. However, a wine with this ranking is even below the median of the dataset based on rating! An example of such a wine is "Volver Alicante Tarima Hill Old Vines". On the other hand, consider a subset that satisfies the rank-regret of top-6 (top-6%, in other words). Such a subset should contain one of the top 6 wines based on rating, a good approximation for the top-1. "Cantina del Pino Barbaresco Ovello" (with rating of 97) is such a good representative.

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

6.3 Performance Evaluation

After validating our proposal, here we study the performance of our algorithms. In addition to the efficiency, we evaluate the effectiveness of the proposed algorithms. That is, we study if the algorithms can find a small subset with bounded rank-regret based on k. We consider the running time as the efficiency measure and the rank-regret of output set, as well as its size, for effectiveness. Computing the exact rank-regret of a set needs the construction of the arrangement of items in the dual space which is not scalable to the large settings. Therefore, in the experiments for estimating the rank-regret of a set in MD, we draw 10,000 functions uniformly at random (based on Lines 4 to 6 of Algorithm 4) and consider them for estimating the rank-regret.

---

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

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

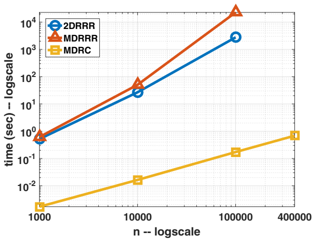

Figure 9: DOT dataset, 2D, Efficiency: Impact of dataset size (n)

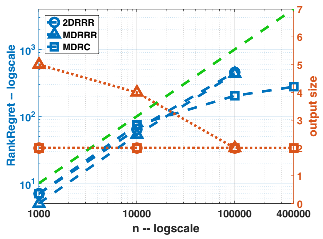

Figure 10: DOT dataset, 2D, Effectiveness: Impact of dataset size

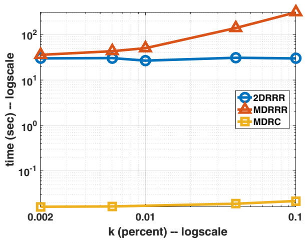

Figure 11: DOT dataset, 2D, Efficiency: Impact of k

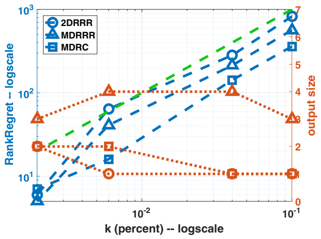

Figure 12: DOT dataset, 2D, Effectiveness: Impact of k

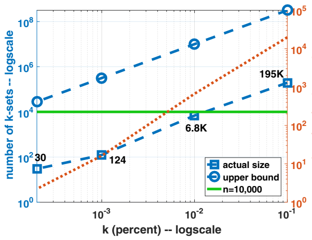

Figure 13: DOT dataset, MD: Impact of k on $|\mathcal{S}|$

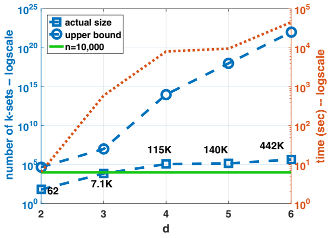

Figure 14: DOT dataset, MD: Impact of d on $|\mathcal{S}|$

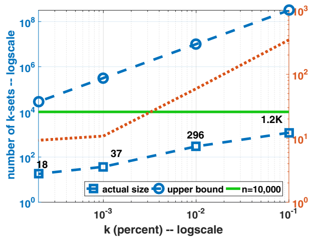

Figure 15: BN dataset, MD: Impact of k on $|\mathcal{S}|$

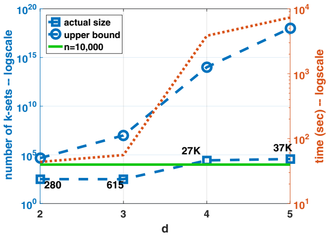

Figure 16: BN dataset, MD: Impact of d on $|\mathcal{S}|$

6.3.1 *Setup.* All experiments were performed on a Linux machine with a 3.8 GHz Intel Xeon processor and 64 GB memory. The algorithms are implemented using Python 2.7.

Algorithms evaluated. In addition to the theoretical analyses, we evaluate the algorithms proposed in this paper. In § 4, we proposed 2DRRR, the algorithm that uses Theorem 1 to transform the problem into one dimensional range covering. This quadratic algorithm guarantees the approximation ratio of 2 on the maximum rank regret of its output. In this section, we shall show that in all the cases it generated an output with maximum rank of k. For 2D, we implemented the ray-sweeping algorithm (similar to Algorithm 1) that enumerates the k-sets by following the changes in the k-border (Figure 3). We also implemented the k-set graph based enumeration[29] for MD. We did not include the results here, but we observed that it does not scale beyond a few hundred items (that is because it need to solve much as $O(nk)$ linear programs for a single k-set). Instead, we apply the randomized algorithm K-SET$_r$ for finding the k-sets (while setting the termination condition c to 100). The MD algorithms proposed in § 5 are the hitting-set based algorithm MDRRR and the space function covering algorithm MDRC. As we explained in § 1 and 7, all of the existing algorithms proposed for different varieties of regret-ratio consider the score difference, as the measure of regret and apply the optimization based on it. Still to verify this, we consider comparing with them as the baseline. As we shall further explain in § 7, [5, 14] propose similar approximation algorithms for the regret-ratio minimizing problem with controllable additive approximation factors. Both of these works are based on discretizing the function space and transforming the problem into hitting set instances [14, 16]. We adopt the HD-RRMS algorithm [5] as baseline.

Default values. For each experiment, we study the impact of varying one variable while fixing others to their default values. The default values are as following: (i) dataset size (n): 10,000, (ii) number of attributes (d): 3, and (iii) k: top-1%.

6.3.2 *2D Results.* We use a ray sweeping algorithm, similar to Algorithm 1, to enumerates the k-sets by following the changes in the k-border. We also use the ray sweeping to find out the (exact) rank regret of a set in 2D. Due to the space limitations, for 2D, we only provide the plots for the DOT dataset. Figures 9 and 10 show the performance of the algorithms for varying the dataset size (n) from 1000 to 400,000. The running times of 2DRRR and MDRRR are dominated by the time required by the sweeping line algorithms for finding the ranges (Algorithm 1) and the k-sets. Since these two algorithms have similar structure, their running times are similar. Still, because the sweeping ray algorithm is quadratic, these algorithms did not scale beyond 100K items. On the other hand MDRC does not depend on finding the k-set or sweeping a line. Rather, it partitions the space until top-k of two corners of each range intersect. Due to the binary search nature of the algorithm that breaks the space by half at every iteration, soon the functions in the two ends of each range become similar enough to share an item in their top-k. Therefore, the algorithm performs very well in practice, and scales well for large settings. For example, it took less than a second to run MDRC for 100K items, while 2DRRR and MDRRR required several thousand seconds. See Figure 9. In Figure 10,

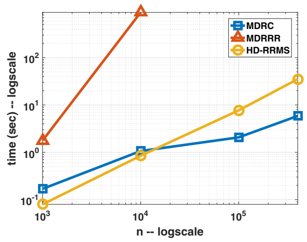

Figure 17: DOT, MD, Efficiency: Impact of dataset size (n)

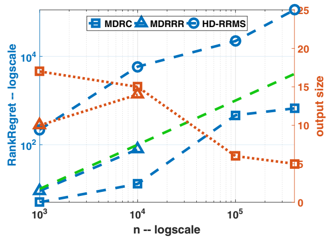

Figure 18: DOT, MD, Effectiveness: Impact of dataset size

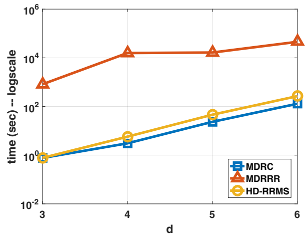

Figure 19: DOT, MD, Efficiency: Impact of # of attributes (d)

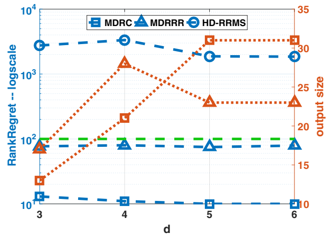

Figure 20: DOT, MD, Effectiveness: Impact of # of attributes

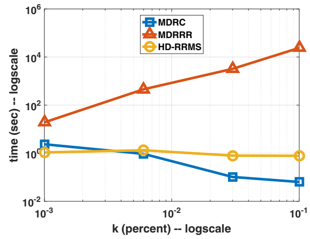

Figure 21: DOT, MD, Efficiency: Impact of k

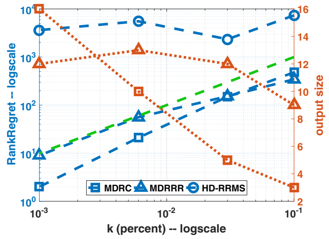

Figure 22: DOT, MD, Effectiveness: Impact of k

and all other plots with, two y-axes, the left axis show the rank-regret and the right one is the output size. The dashed green line show the border for the rank-regret of 1%.

The algorithm 2DRRR guarantees the optimal output size. For all settings its output also had the rank-regret of less than k, confirming that it returned the optimal solution. On the other hand, MDRRR guarantees the rank-regret of k and provides the logarithmic approximation ratio on its output size. This is also confirmed in the figure, where the rank regret of the output of MDRRR is always below the green line. However, the size of its output is more than the optimal for two (out of three) settings. the space partitioning algorithm MDRRR provides the output which in all cases satisfied the rank-regret of k and also its output size was the minimum, confirming that it also discovered the optimal output. In Figures 11 and 12, we fix the dataset size and other variables to the default and study the effect of changing k on the efficiency of the algorithm and the quality of their outputs. Similar to Figure 9, 2DRRR and MDRRR have similar running times (due to applying the ray sweeping algorithm) and MDRC runs within a few milliseconds for all settings. On the other hand, in Figure 12, the output size of MDRC is in all cases, except one, equal to the optimal output size (of 2DRRR) while, due to its logarithmic approximation ratio, the hitting set based MDRRR generates larger outputs.MDRRR guarantees the rank-regret of k, which is confirmed in the figure. MDRC also provided the maximum rank-regret of k for all settings and 2DRRR did so for all, except $k = 0.004\%$ for which its maximum rank regret was slightly above the threshold.

k-set size. Next, we compare the actual size of k-sets with the theoretical upper-bounds, using the K-SET$_r$ algorithm. To do so, we select the DOT and BN datasets, set number of items to 10K and study the impact of varying k and d. The results are provided in Figures 13, 14, 15, and 16. The left-y-axis in the figures show the size and the right-y-axis show the running time of the K-SET$_r$ algorithm. The horizontal green line in the figures highlight the number of items $n = 10$K. Figures 13 and 15 show the results for varying k for DOT and BN, respectively. First, as observed in the figures, the actual sizes of the k-sets are significantly smaller than the best known theoretical upper-bound for 3D ($O(nk^{3/2})$ [27]). In fact, the number of k-sets is closer to n than the upper-bounds. Second, the number of k-sets for $k = 10\%$ is significantly larger than the number of k-sets for smaller values of k. Recall that the k-sets are densely overlapping, as the neighboring k-sets in the k-set graph only differ in one item. As k increases (up until $k = 50\%$), for each node of the k-set graph the number of candidate transitions to the neighboring k-sets increases which affect $|\mathcal{S}|$ as well. Although significantly smaller than the upper bound, still the sizes are large enough to make the k-set discovery impractical for large settings. For example, running the K-SET$_r$ algorithm for the DOT dataset and $k = 10\%$ took more than ten thousand seconds. The observations for varying d (Figures 14 and 15) are also similar. Also, the gap between the theoretical upper-bound for $d \geq 4$ and the actual k-sets sizes show how loose the bounds are.

6.3.3 MD Results. Here, we study the algorithms proposed for the general cases where $d \geq 3$. MDRRR is the hitting set based algorithm that, given the collection of k-sets, guarantees the rank-regret of k and a logarithmic increase in the output size. So far, the 2D experiments confirmed these bounds. The other algorithm is the space partitioning algorithm MDRC which is designed based on Theorem 1. Given the possibly large number of k-sets and the cost of finding them (even using the randomized algorithm K-SET$_r$), this algorithm is designed to prevent the k-set enumeration. MDRC uses the fact that the k-sets are highly overlapping and recursively partitions the space (see Figure 8) into several hypercubes and stops the recursion for each hypercube as soon as the intersection of the top-k items in its corners is not empty. This algorithm performs very well in practice, as after a few iterations, the functions in the corners become similar enough to share at least one item in their top-k. Also, the maximum rank-regret of the item that appear in the top-k of the corners of the hyper-rectangle for the functions inside the hypercube is much smaller than the bound provided in Theorem 6. We so far observed it in the 2D experiments where in all cases the rank-regret of the output of MDRC is less than k, while the output size also was always close to the optimal output size.

In addition to these algorithms, we compare the efficiency and effectiveness of our algorithms against, HD-RRMS [5], the recent approximation algorithm proposed for regret-ratio minimizing problem. Since HD-RRMS takes the index size as the input, we first run the MDRC algorithm and pass its output size to HD-RRMS. Having a different optimization objective (on the regret-ratio), as we shall show, the output of HD-RRMS fails to provide a bound on the rank-regret. In the first experiment, fixing the other variables to their default values, we vary the dataset size n from 1000 to 400,000 for DOT and from 1000 to 100,000 for BN[8]. Figures 17, 18, 23, and 24 show the results. Figures 17 and 18, 23 show the running time of the algorithms for DOT and BN, respectively. Looking at these figures, first MDRRR did not scale for 100K items. The reason is that MDRRR needs the collection of k-sets in order to apply the hitting set. For a very large number of items even the K-SET$_r$ algorithm does not scale. HD-RRMS has a reasonable running time in all cases. MDRC has the least running time for large values of n and in all cases it finished in less than a few seconds. The reason is that after a few recursions, the functions in the corners of the hypercubes become similar and share an item in their top-k. Figures 18 and 24 show the effectiveness of the algorithms for these settings. The left-y-axes show the maximum rank-regret of an output set while the right-y-axes show the output size. The green lines show the rank-regret of k border. First, the output size for all settings is less than 20 items, which confirm the effectiveness of algorithms for finding a rank-regret representative. As explained in § 5.2, MDRRR guarantees the rank-regret of k, which is observed here as well. As expected, HD-RRMS fails to provide a rank-regret representative in all cases. Both for DOT and BN, the maximum rank-regret of the output of HD-RRMS are close to n, the maximum possible rank-regret. For example, for DOT and n =400K, the rank-regret of HD-RRMS was 112K, i.e., there exists a function for which the top-1 based on the output of HD-RRMS has the rank 112,000. Based on Theorem 6, for these settings, the rank-regret of the output of MDRC is guaranteed to be less than $4k$ for all cases. However, in practice we expect the rank-regret to be smaller than this. This is confirmed in both experiments for DOT (Figure 18) and BN (Figure 24) where the output of MDRC provided the rank-regret of k.

Next, we evaluate the impact of varying the number of dimensions. Setting n to 10,000 and k to 1% of n (i.e. 100), we the number of attributes, d, from 3 to 6 for DOT and from 3 to 5 for BN. Figures 19, 20, 25, and 20 show the results. The running times of the algorithms for DOT and BN are provided in Figures 19 and 25. Similar to the previous experiments, since the hitting set based algorithm MDRRR requires the collection of k-sets, it was not efficient. Both HD-RRMS and MDRC performed well in both experiments. On the other hand, looking at Figures 20 and 20 HD-RRMS fails to provide a rank-regret representative, as in all settings there the rank-regret of its output was several thousands, while the maximum possible rank-regret is $n = 10,000$. The outputs of proposed algorithms in § 5, as expected, satisfied the requested rank-regret. Interestingly, the output of MDRC had a lower rank-regret, especially for DOT where its rank-regret was around 10 for all settings. The output of both MDRRR and MDRC was less than 40, for all settings and both datasets, which confirm the effectiveness of them as the representative.

In the last experiment, we evaluate the impact of varying k. For both datasets, while setting n to 10,000 and d to 3, we varied k from 0.1% of items (i.e., 10) to 10% (i.e., 1000). Figures 21, 22, 27, and 28 show the results. Looking at Figures 21 and 27 which show the running time of the algorithms for DOT and BN, respectively, MDRRR had the worst performance, and it got worse as k increased. The bottleneck in MDRRR is the k-set enumeration, and (looking at Figures 13 and 15) it increased by k, as the number of k-sets increased. Both HD-RRMS and MDRC were efficient for all settings. One interesting fact in these plots is that the running time of MDRC decreases as k increases. This is despite the fact that, as showed in Figures 13 and 15, the number of k-sets increased. The reason for the decrease, however, is simple. The probability that the top-k of corners of a hypercube share an item increases when looking at larger values of k where each set contains more items. Although HD-RRMS was efficient in all settings, similar to the previous experiments it fails to provide a rank-regret representative as the rank-regret of its output is not bounded. The outputs of MDRRR and MDRC, on the other hand, had smaller rank-regret than the requested k in all settings for both datasets. Again, the output sizes in all settings were less than 20, which confirm the effectiveness of them as the rank-regret representative.

6.3.4 *Summary of results.* To summarize, the experiments verified the effectiveness and efficiency of our proposal. While the adaptation of the regret-ratio based algorithm HD-RRMS fails to provide a rank-regret representative, 2DRRR, MDRRR, and MDRC found small sets with small rank-regrets. Although the rank-regret of the outputs of 2DRRR and MDRC can be larger than k, in our experiments and our measurements those were always below k. MDRRR provided small outputs that as expected, always guarantees the rank-regret of k.

---

[8] Due to the space limitations, the MD experiment results of BN are provided in Appendix B.

Interestingly, the output size of MDRC was around the size of the one by MDRRR, which verifies the effect of the greedy behavior of MDRC. The output sizes in all the experiments were less than 40, confirming the effectiveness of the representatives. The quadratic 2DRRR and the hitting-set based algorithm MDRRR scaled up to a limit, whereas MDRC had low running time at all scales.

7 RELATED WORK

The problem of finding preferred items of a dataset has been extensively investigated in recent years, and research has spanned multiple directions, most notably in top-k query processing [37] and skyline discovery [4]. In top-k query processing, the approach is to model the user preferences by a ranking/utility function which is then used to preferentially select tuples. Fundamental results include access-based algorithms [38–41] and view-based algorithms [42, 43]. In skyline research, the approach is to compute subsets of the data (such as skylines and convex hulls) that serve as the data representatives in the absence of explicit preference functions [3, 4, 44]. Skylines and convex hulls can also serve as effective indexes for top-k query processing [19, 45, 46].

Efficiency and effectiveness have always been the challenges in the above studies. While top-k algorithms depend on the existence of a preference function and may require a complete pass over all of the data before answering a single query, representatives such as skylines may become overwhelmingly large and ineffective in practice [5, 6]. Studies such as [7, 8] are focused towards reducing the skyline size. In an elegant effort towards finding a small representative subset of the data, Nanongkai et al. [9] 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 [5, 10–12, 14–16, 47] studied different challenges and variations of the problem. Chester et al. [13] generalize the regret-ratio notion to k-regret, in which the regret is considered to be the difference between the top result of the subset and the actual top-k result (instead of the top-1 result). They also prove that the problem is NP-complete for variable number of dimensions. [14, 15] prove that the k-regret problem is NP-complete even when $d = 3$, using the polytope vertex cover problem [21] for the reduction. As explained in § 2, this also proves that our problem is NP-complete for $d \geq 3$. For the case of two dimensional databases, [13] proposes a quadratic algorithm and [5] improves the running time to $O(n \log n)$. The cube algorithm and a greedy heuristic [9] are the first algorithms proposed for regret-ratio in MD. Recently, [5, 14] 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 [5] considers the original regret-ratio problem while [14] considers the k-regret variation. 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.

The geometric notions used in this paper, such as arrangement, dual space, and k-set, are explained in detail in [17]. Finding bounds on the number of k-sets of a point set do not lead to promising results on the upper bound of the size of $\mathcal{S}$. Lovasz and Erdos [48, 49] initiated the study of k-set notion and provided an upper bound on the maximum number of k-sets in $\mathbb{R}^2$. The problem in $\mathbb{R}^2$ has also been studied in [50–53]. The best known upper bound on the number of k-sets in $\mathbb{R}^2$ and $\mathbb{R}^3$ are $O(nk^{1/3})$ [26] and $O(nk^{3/2})$ [27], respectively. For higher dimensions, finding an upper bound on the number of k-sets has been extensively studied [17, 28, 51, 54]; the best known upper bound is $O(n^{d-\varepsilon})$ [28], where $\varepsilon > 0$ is a small constant. The problem of enumerating all k-sets has been studied in [55, 56] for 2D and [30] for MD.

8 FINAL REMARKS

In this paper, we proposed a rank-regret measure that is easier for users to understand, and often more appropriate, than regret computed from score values. We defined *rank-regret representative* as the minimal subset of the data containing at least one of the top-k of any possible ranking function. Using a geometric interpretation of items, we bound the maximum rank of items on ranges of functions and utilized combinatorial geometry notions for designing effective and efficient approximation algorithms for the problem. In addition to theoretical analyses, we conducted empirical experiments on real datasets that verified the validity of our proposal. Among the proposed algorithms, MDRC seems to be scalable in practice: in all experiments, within a few seconds, it could find a small subset with small rank-regret.

9 ACKNOWLEDGEMENTS

The work of Abolfazl Asudeh and H. V. Jagadish was supported in part by NSF grants No. 1741022 and 1250880. Nan Zhang was supported in part by NSF grants No. 1343976, 1801539, and 1850605. The work of Gautam Das was supported in part by grant W911NF-15-1-0020 from the Army Research Office, NSF grant No. 1745925, and a grant from AT&T.

REFERENCES

[1] Abolfazl Asudeh, Nan Zhang, and Gautam Das. Query reranking as a service. *VLDB*, 9(11), 2016.

[2] Yeshwanth D. Gunasekaran, Abolfazl Asudeh, Sona Hasani, Nan Zhang, Ali Jaoua, and Gautam Das. Qr2: A third-party query reranking service over web databases. In *ICDE Demo*, 2018.

[3] Abolfazl Asudeh, Gensheng Zhang, Naeemul Hassan, Chengkai Li, and Gergely V. Zaruba. Crowdsourcing pareto-optimal object finding by pairwise comparisons. In *CIKM*, 2015.

[4] S Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, 2001.

[5] Abolfazl Asudeh, Azade Nazi, Nan Zhang, and Gautam Das. Efficient computation of regret-ratio minimizing set: A compact maxima representative. In *SIGMOD*. ACM, 2017.

[6] Sariel Har-Peled. On the expected complexity of random convex hulls. *arXiv preprint arXiv:1111.5340*, 2011.

[7] Chee-Yong Chan, HV Jagadish, Kian-Lee Tan, Anthony KH Tung, and Zhenjie Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.

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

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

[10] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino. Interactive regret minimization. In *SIGMOD*. ACM, 2012.

[11] Sepanta Zeighami and Raymond Chi-Wing Wong. Minimizing average regret ratio in database. In *SIGMOD*. ACM, 2016.

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

[13] Sean Chester, Alex Thomo, S Venkatesh, and Sue Whitesides. Computing k-regret minimizing sets. *VLDB*, 7(5), 2014.

[14] Pankaj K Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Efficient algorithms for k-regret minimizing sets. *LIPIcs*, 2017.

[15] Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan. k-regret minimizing set: Efficient algorithms and hardness. In *LIPIcs*, 2017.

[16] Nirman Kumar and Stavros Sintos. Faster approximation algorithm for the k-regret minimizing set and related problems. In *ALENEX*. SIAM, 2018.

[17] Herbert Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10. Springer Science & Business Media, 1987.

[18] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[19] Abolfazl Asudeh, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. Discovering the skyline of web databases. *VLDB*, 2016.

[20] W Weil and J Wieacker. Stochastic geometry, handbook of convex geometry, vol. a, b, 1391–1438, 1993.

[21] Gautam Das and Michael T Goodrich. On the complexity of optimization problems for 3-dimensional convex polyhedra and decision trees. *Computational Geometry*, 8(3), 1997.

[22] Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite vc-dimension. *DCG*, 14(4):463–479, 1995.

[23] Abolfazl Asudeh, H.V. Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In *SIGMOD*. ACM, 2019.

[24] Bernard Chazelle and Franco P Preparata. Halfspace range search: an algorithmic application of k-sets. In *SOCG*. ACM, 1985.

[25] Kenneth L Clarkson. Applications of random sampling in computational geometry, ii. In *SOCG*. ACM, 1988.

[26] Tamal K Dey. Improved bounds for planar k-sets and related problems. *DCG*, 19(3):373–382, 1998.

[27] Micha Sharir, Shakhar Smorodinsky, and Gábor Tardos. An improved bound for k-sets in three dimensions. In *SOCG*. ACM, 2000.

[28] Noga Alon, Imre Bárány, Zoltán Füredi, and Daniel J Kleitman. Point selections and weak ε-nets for convex hulls. *Combinatorics, Probability and Computing*, 1(03):189–200, 1992.

[29] Abolfazl sudeh, Azade Nazi, Nan Zhang, Gautam Das, and H. V. Jagadish. Rrr: Rank-regret representative. *CoRR*, abs/1802.10303, 2018.

[30] Artur Andrzejak and Komei Fukuda. Optimization over k-set polytopes and efficient k-set enumeration. In *WADS*, 1999.

[31] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[32] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[33] P Assouad. Densité et dimension. *Ann. Institut Fourier (Grenoble)*, 1983.

[34] David Haussler and Emo Welzl. ÉŻ-nets and simplex range queries. *DCG*, 2(2):127–151, 1987.

[35] P Erdős. On a classical problem of probability theory. *Magy. Tud. Akad. Mat. Kut. Int. Kőz.*, 6(1-2), 1961.

[36] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 1974.

[37] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *CSUR*, 40(4):11, 2008.

[38] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *JCSS*, 2003.

[39] Ronald Fagin, Ravi Kumar, and D Sivakumar. Comparing top k lists. *Journal on Discrete Mathematics*, 2003.

[40] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *TODS*, 2002.

[41] Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2), 2004.

[42] Vagelis Hristidis and Yannis Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB*, 13(1), 2004.

[43] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. Answering top-k queries using views. In *VLDB*, 2006.

[44] Md Farhadur Rahman, Abolfazl Asudeh, Nick Koudas, and Gautam Das. Efficient computation of subspace skyline over categorical domains. In *CIKM*. ACM, 2017.

[45] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R Smith. The onion technique: indexing for linear optimization queries. In *SIGMOD*, 2000.

[46] Dong Xin, Chen Chen, and Jiawei Han. Towards robust indexing for ranked queries. In *VLDB*, 2006.

[47] Peng Peng and Raymond Chi-Wing Wong. Geometry approach for k-regret query. In *ICDE*. IEEE, 2014.

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

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

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

[51] Géza Tóth. Point sets with many k-sets. *DCG*, 26(2), 2001.

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

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

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

[55] Herbert Edelsbrunner and Emo Welzl. Constructing belts in two-dimensional arrangements with applications. *SICOMP*, 15(1):271–284, 1986.
[56] Timothy M Chan. Remarks on k-level algorithms in the plane. *Manuscript, Department of Computer Science, University of Waterloo, Waterloo, Canada*, 1999.
[57] George Marsaglia et al. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43(2), 1972.

APPENDIX

A K-SET$_R$: SAMPLING FOR k-SET ENUMERATION

Here we propose a sampling-based alternative for the k-set enumeration, based on the many to one mapping between the linear ranking functions and the k-sets. That is, while a k-set is the top-k of infinite number of linear ranking functions, every ranking function is mapped to only one k-set, the set of top-k tuples for that function. Instead of the exact enumeration of the k-sets, which requires solving expensive linear programming problems for the discovery of the k-sets, we propose a randomized approach based on the *coupon collector's problem* [35]. The coupon collector's problem describes the "collect the coupons and win" contest. Given a set of coupons, consider a sampler that every time picks a coupon uniformly at random, with replacement. The requirement is to keep sampling until all coupons are collected. Given a set of v coupons, it has been shown that the expected number of samples to draw is in $\Theta(v \log v)$. We use this idea for collecting the k-sets by generating random ranking functions and taking their top-k results as the k-sets. This is similar to the coupon collector's problem setting, except that the probabilities of retrieving the k-sets are not equal. For each k-set, this probability depends on the portion of the function space for which it is the top-k. Therefore, rather than applying a k-set enumeration algorithm, K-SET$_r$ (Algorithm 4), repeatedly generates random functions and computes their corresponding k-sets, stopping when it does not find a new k-set after a certain number of iterations. The algorithm returns the collection of k-sets it has discovered, as $\mathcal{S}_r$. Recall that the function space in MD, is modeled by the universe of origin-starting rays. The set of points on the surface of the (first quadrant of the) unit hypersphere represent the universe of origin-starting rays. Therefore, uniformly selecting points from the surface of the hypersphere in $\mathbb{R}^d$, is equivalent to uniformly sampling the linear functions. Algorithm 4 adopts the method proposed by Marsaglia [57] for uniformly sampling points on the surface of the unit hypersphere, in order to generate random functions. It generates the weight vector of the sampled function as the absolute values of d random normal variables. We note that since the k-sets are not collected uniformly by K-SET$_r$, its running time is not the same as coupon collector's problem, but as we shall show in § 6, it runs well in practice.

B BLUENILE MD EXPERIMENT RESULTS

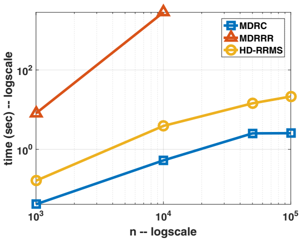

Figure 23: BN, MD, Efficiency: Impact of dataset size

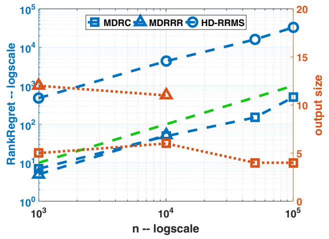

Figure 24: BN, MD, Effectiveness: Impact of dataset size

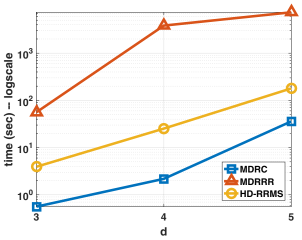

Figure 25: BN, MD, Efficiency: Impact of number of attributes (d)

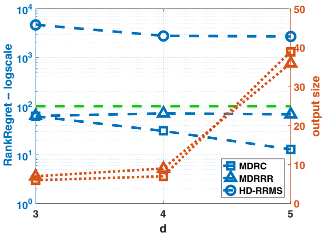

Figure 26: BN, MD, Effectiveness: Impact of number of attributes (d)

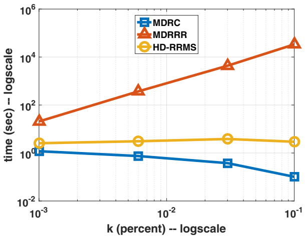

Figure 27: BN, MD, Efficiency: Impact of k

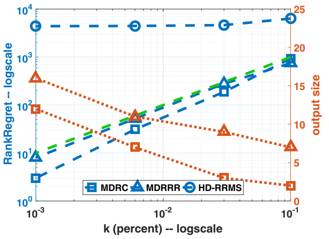

Figure 28: BN, MD, Effectiveness: Impact of k

C PROOFS

THEOREM 2. *The algorithm* 2DRRR *is in* $O(n^2 \log n)$.

PROOF. The complexity of the algorithm 2DRRR depends is determined by Algorithms 1 and 2. Algorithms 1 first orders the items based on x in $O(n \log n)$. Then in applies a ray sweeping from the x-axis toward y and at every intersection applies constant number of operations. The upper bound on the number of intersections in $O(n^2)$ and therefore, it is the running time of Algorithms 1. Calling Algorithm 1, generates at most n ranges, each for an item. Every iteration of Algorithm 2 is in $O(n \log n)$ as it applies a binary search on the set of uncovered intervals for each unselected item, and the number of uncovered intervals is bounded by $O(n)$. Given that the output size is bounded by n, Algorithm 2 is in $O(n^2 \log n)$. □

THEOREM 3. *The output size of* 2DRRR *is not more than the size of the optimal solution for RRR.*

Proof. Following the k-border, while sweeping a ray from x-axis to y, the top-k results change only when a line above the border intersects with it. For example, in Figure 3, moving from x-axis to y, in the intersection between $\mathrm{d}(t_3)$ and $\mathrm{d}(t_1)$, the top-2 changes from $\{t_7, t_1\}$ to $\{t_7, t_3\}$. Consider the collection of the top-k results and the range of angles of rays (named as top-k regions) that provide them. Now consider the ranges that are generated by Algorithm 1 for each item. Let us name them here as the ranges of items. These ranges mark the first and last angle for which an item is in top-k. For each top-k region R, let the set items that their ranges cover it be S_R. Each top-k region is covered by each and every item in its top-k. In addition the ranges of some other items cover each top-k region. Therefore, S_R is a superset for the top-k of R. An optimal solution with the minimum number of items from the collection of supersets that contains at least one item from each set, is not larger than the minimum number of such items from the collection of subsets. As a result, the output size of 2DRRR is not greater that the size of the optimal solution for the RRR problem. □

Theorem 4. *The output of* 2DRRR *guarantees the maximum rank-regret of* $2k$.

Proof. The proof is straightforward, following the Theorem 1. For each item t, Algorithm 1 finds a range that in its beginning and its end, t is in the top-k. Therefore, based on Theorem 1, the rank of t for each of the functions inside its range is no more than $2k$. Algorithm 2 covers the function space with the ranges generated by Algorithm 1. Hence, for each function, there exists an item t in the output where $\nabla_f(t) \leq 2k$. □

Lemma 5. *Let $\mathcal{S}$ be the collection of all k-sets for the points corresponding to the items $t \in \mathcal{D}$. For each ranking function f, there exists a k-set $S \in \mathcal{S}$ such that top-$k(f)$=S.*

Proof. The proof is straight-forward using contradiction. Consider a ranking function f with the weight vector w where the top-k is S_f and S_f does not belong to $\mathcal{S}$. Let t be the item for which $\nabla_f(t) = k$. Consider the hyperplane $h(t, w)$. For all the items in $t' \in S_f$ $f(t') \leq f(t)$ and for all items in $\mathcal{D} \backslash S_f$, $f(t') > f(t)$. Hence, all the items in S_f fall in the positive half space of h – i.e., $h(t, w)^+ = S_f$. Since $|S_f|$ is k, $card(h(t, w)^+) = k$. Therefore $h(t, w)^+ = S_f$ is a k-set and should belong to $\mathcal{S}$, which contradicts with the assumption that is does not belong to the collection of k-sets. □

Theorem 6. MDRC *guarantees the maximum rank-regret of* dk.

Proof. The proof of this theorem is based on Theorem 1. We also consider the arrangement lattice [17] for this proof. Every convex region in the $(d-1)$-dimensional space is constructed from the $d-2$ dimensional space convex facets as its borders. Each of the facets are constructed by $d-3$ dimensional facets, and this continues all the way down until the (0 dimensional) points. For example, the borders of a convex polyhedron in 3D, are two dimensional convex polygones; the borders of the polygones are (one dimensional) line segments, each specified by two points. The arrangement lattice is the data structure that describe the convex polyhedron by its i dimensional facets – $\forall\ 0 \leq i \leq d$. The nodes at level i of the lattice show the i dimensional facets, each connected to its $i-1$ dimensional borders, as well as the $i+1$ dimensional facets those are a border for.

Now, let us consider the hyper-rectangle of each of the leaf nodes in the recursion tree of MDRC (c.f. Figure 8) and let t be the tuple that appeared at the top-k of all corners of the hyper-rectangle. Consider the arrangement lattice for the hyper-rectangle of the leaf node and let us move up from the bottom of the lattice, identifying the maximum rank of t at each level of it. Since t is in the top-k of both corners of each line segment in level 1, based on Theorem 1, its rank for each point on the line is at most $2k$. Level 2 of the lattice shows the two dimensional rectangles, each built by the line segments at level 1. For every point inside each rectangle at level 2, consider a line segment on the rectangle's affine space starting from one of its corners, passing through the point and ending on the edge of the rectangle. Since the rank of the point on the corner is less than k and for any point on the edge less than $2k$, based on Theorem 1, the rank of t for the points inside the rectangles at level 2 of lattice is at most $k + 2k = 3k$. Similarly, consider each hyper-rectangle at level i of the lattice. The hyper-rectangle is built by the $(i-1)$ dimensional hyper-rectangle at level $i-1$. For every point inside the i dimensional hyper-rectangle, consider the line segment starting from a corner of the hyper-rectangle, passing through the point and hitting the edge of it. By induction, the rank of t on the $(i-1)$ dimensional edges of hyper-rectangle is at most ik. Therefore, since the rank of t on the corner is at most k, based on Theorem 1, its rank for the point inside the i dimensional hyper-rectangle is at most $k + ik = (i+1)k$. Therefore, the rank of t for every point inside the $(d-1)$ dimensional hyper-rectangle (the top of the lattice) is at most $k + (d-1)k = dk$. MDRC partitions the function space into hyper-rectangles that, for each, there exists an item t in the top-k in all of hyper-rectangle's corners (included in the output). The rank of t for every point inside the hyper-rectangle is at most dk. Since every function in the space belongs to a hyper-rectangle, there exists an item in the output that guarantees the rank of dk for it. □

D PSEUDOCODE OF ALGORITHMS

Algorithm 1 FINDRANGES

Input: 2D dataset $\mathcal{D}$, n, k

```
heap = new min-heap(); visited = new set()
$L$ = sort $\mathcal{D}$ based on $x$
for $i = 1$ to $n - 1$ do
    if $L_i[2] < L_{i+1}[2]$ /* skip if $L_i$ dominates $L_{i+1}$ */ then
        heap.push( (arctan $\frac{L_{i+1}[1]-L_i[1]}{L_i[2]-L_{i+1}[2]}$, $L_i$) )
    end if
end for
for $i = 1$ to $k$ do $b[L_i] = 0$
while heap is not empty do
    $(\theta, t)$ = heap.pop() // let $i$ be the index of $t$ in $L$
    if $i == k$ then
        if $b[L_{i+1}] == null$ then $b[L_{i+1}] = \theta$
        $e[L_i] = \theta$
    end if
    swap $L_i$ and $L_{i+1}$
    if ($L_{i-1}[1] < L_i[1]$ or $L_{i-1}[2] < L_i[2]$) and $(L_{i-1}, L_i) \notin$ visited)
    then
        heap.push( (arctan $\frac{L_i[1]-L_{i-1}[1]}{L_{i-1}[2]-L_i[2]}$, $L_{i-1}$) )
        visited.$add((L_{i-1}, L_i))$
    end if
    if ($L_{i+1}[1] < L_{i+2}[1]$ or $L_{i+1}[2] < L_{i+2}[2]$ and $(L_{i+1}, L_{i+2}) \notin$
    visited) then
        heap.push( (arctan $\frac{L_{i+2}[1]-L_{i+1}[1]}{L_{i+1}[2]-L_{i+2}[2]}$, $L_{i+1}$) )
        visited.$add((L_{i+1}, L_{i+2}))$
    end if
end while
for $i = 1$ to $k$ do $e[L_i] = \pi/2$
return $b$, $e$
```

Algorithm 2 2DRRR

Input: 2D dataset $\mathcal{D}$, n, k

```
$b$,$e$ = FindRanges($\mathcal{D}$,$n$,$k$)
$\Psi$ = new set()
$U = [\langle 0, \vdash \rangle, \langle \pi/2, \dashv \rangle]$
while $|U| > 0$ do
    $cov_m = 0$;
    for $t_i$ in $\mathcal{D} \backslash \Psi$ do
        if $b[t_i] == null$ then continue
        $k$ = the index of the element in $U$ that $b[t_i]$ fall before it (found
        by applying binary search)
        if $U_k[2] == \dashv$ then cov = min($U_k[1]$, $e[t_i]$) $-$ $b[t_i]$
        else cov = max(0, $e[t_i] - U_k[1]$)
        if cov > $cov_m$ then t = $t_i$; $cov_m$ = cov; $k_m = k$
    end for
    $\Psi$.add($t$)
    if $U_{k_m}[2] == \vdash$ then
        if $U_{k_m+1}[1] \leq e[t]$ then remove $U_{k_m}$ and $U_{k_m+1}$
        else $U_{k_m}[1] = e[t]$
    else
        if $U_{k_m}[1] > e[t]$ then
            $U$.insert($k_m$, $\langle b[t], \dashv \rangle$); $U$.insert($k_m + 1$, $\langle e[t], \vdash \rangle$)
        else
            $U_{k_m}[1] = b[t]$
        end if
    end if
end while
return $\Psi$
```

Algorithm 3 MDRRR

Input: collection of k-sets $\mathcal{S}$

```
$D = \bigcup_{\forall S_i \in \mathcal{S}} S_i$
Set weight of each point to one
while True do
    $X$ = Select the $\epsilon$-net
    if $X$ is not hitting set then
        for $S$ in $\mathcal{S}$ do
            if points in a set k-set $S$ missed by $X$ then
                Double the weights of the points in $S$
            end if
        end for
    else
        return $X$
    end if
end while
```

Algorithm 4 K-SET$_r$

Input: dataset $\mathcal{D}$, termination condition c

```
$\mathcal{S}_r = \{\}$ ,counter=0
while counter$\leq c$ do
    // generate a sample function
    for $i = 1$ to $d$ do
        $w_i = |N(0, 1)|$ // N(0,1) draws a sample from the standard
        normal distribution
    end for
    // find the corresponding $k$-set
    $S$ = top-$k(\mathcal{D}, f_w)$
    if $S \in \mathcal{S}_r$ then
        counter = counter+1
    else
        add $S$ to $\mathcal{S}_r$
        counter = 0
    end if
end while
return ($\mathcal{S}_r$)
```

Algorithm 5 MDRC

Input: The dataset $\mathcal{D}$, n, d, k, level of the node: l, ranges: R

```
$C$ = corners of the hypercube specified by $R$
$I = \bigcap_{\forall c_i \in C}$ top-$k(\mathcal{D}, c_i)$
if $|I| > 0$ then return $I[1]$
$i = l\%(d - 1) + 1$
mid = $\frac{R[i][1]+R[i][2]}{2}$
$lR = rR = R$
$lR[i][2]$ = mid; $rR[i][1]$ = mid;
return MDRC ($\mathcal{D}$, $n$, $d$, $k$, $l + 1$, $lR$)$\cup$ MDRC ($\mathcal{D}$, $n$, $d$, $k$, $l + 1$, $rR$)
```