Efficient Computation of Regret-ratio Minimizing Set: A Compact Maxima Representative

Abolfazl Asudeh[‡], Azade Nazi[‡], Nan Zhang[††], Gautam Das[‡]
[‡]University of Texas at Arlington; [††]George Washington University
[‡]{ab.asudeh@mavs, azade.nazi@mavs, gdas@cse}.uta.edu, [††]nzhang10@gwu.edu

ABSTRACT

Finding the maxima of a database based on a user preference, especially when the ranking function is a linear combination of the attributes, has been the subject of recent research. A critical observation is that the *convex hull* is the subset of tuples that can be used to find the maxima of any linear function. However, in real world applications the convex hull can be a significant portion of the database, and thus its performance is greatly reduced. Thus, computing a subset limited to r tuples that minimizes the *regret ratio* (a measure of the user's dissatisfaction with the result from the limited set versus the one from the entire database) is of interest.

In this paper, we make several fundamental theoretical as well as practical advances in developing such a compact set. In the case of two dimensional databases, we develop an optimal linearithmic time algorithm by leveraging the ordering of skyline tuples. In the case of higher dimensions, the problem is known to be NP-complete. As one of our main results of this paper, we develop an approximation algorithm that runs in linearithmic time and guarantees a regret ratio, within any arbitrarily small user-controllable distance from the optimal regret ratio. The comprehensive set of experiments on both synthetic and publicly available real datasets confirm the efficiency, quality of output, and scalability of our proposed algorithms.

1. INTRODUCTION

1.1 Motivation

A maxima query returns a tuple from a large database of n tuples, preferentially selected and returned according to a ranking/utility function that is used to model user preferences. Such queries are very useful in application domains where end-users are interested in multi-criteria decision making, and would like to see the most important tuples from the potentially huge answer space. Thus, much recent studies, including online, view-based and index-based techniques such as [5, 9–11, 15] have focused on this direction.

In many applications, especially in databases containing numeric attributes, the ranking function used to model user preferences is expressed in the form of a linear combination of query attributes – i.e. $\sum w_i A_i$. Finding the top houses in a real estate database based on a linear combination of some criteria such as price and floor area [29], or finding the best NBA player based on a linear combination of his performance criteria such as points and assists, are a few examples of this class of ranking function.

A critical observation is that if the tuples are viewed as points in a high-dimensional space, the convex hull is the subset of points that can be used to find the maxima for any linear ranking function [5, 8]. However, in some real world applications such a convex hull can be overwhelmingly large, and therefore its performance is greatly reduced because many tuples have to be examined during query processing [15]. The size of such a set is highly correlated with the number of attributes, i.e., the number of convex hull tuples radically increases with the number of attributes/dimensions. Even in the case of two-dimensional database, the number of convex hull tuples might also be large. As has been studied in [13], the "curvature" of the shape of a region within which the database tuples are distributed greatly affects the number of convex hull tuples. As the curvature increases the number of convex hull tuples increases. For example, when n points are uniformly distributed inside a convex polygon with k sides, the expected number of the convex hull points is $O(k \log n)$, while this value is $O(n^{\frac{1}{3}})$ when points are uniformly distributed inside a circle.

Consequently, it is of interest to develop a set limited to $r << n$ tuples (where r is an input parameter). Given such a reduced set, for a given ranking function, we can identify the maximum of the reduced set and return it as the query answer. One can observe a tradeoff between the size of the set and the accuracy of query answers (i.e., how the result might differ from the real maximum over the entire table). The task then is to design the most accurate set, i.e., the subset of r tuples for which the "user dissatisfaction" over all possible ranking functions is minimized.

Prior works on convex hull discovery in high-dimensions, such as [1, 16], focus on designing efficient approximate algorithms with small approximate ratios. Thus, their goal is to discover a set that is as similar to the real convex hull as possible, rather than resolving the problem of a large convex hull, and usually find a *super-set* of the convex hull. There has also been work, such as [4, 26], on reducing the *skyline* [2] (the maxima representative that applies to more general *monotonic* ranking functions rather than just linear functions) size. However, their objective in ranking the skyline tuples is not minimizing the user dissatisfaction on maxima queries. For example, [4] relaxes the notion of domination to "$k-$domination" in order to increase the chance domination and reduce the skyline.

The problem investigated in our paper, which we call the *Regret-ratio Minimizing Set (RRMS)* problem, has been studied in prior papers. Nanongkai et. al. [22] introduced the notion of *regret ratio* in order to measure the user dissatisfaction with the top result returned by the representative set. Given a set of r tuples and a specific ranking/utility function, they define regret ratio as the ratio of the difference between the scores of the top tuple in the set and the

SIGMOD'17, May 14-19, 2017, Chicago, Illinois, USA

 ISBN 978-1-4503-4197-4/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3035918.3035932

top tuple in the entire database, divided by score of the top tuple in the entire database. Given a set (or space) of ranking functions, the maximum regret ratio is the regret ratio with the largest value. The RRMS problem seeks to find the subset of r tuples that minimizes the maximum regret ratio. It is known that for arbitrary dimensions, the problem is NP-hard [6]. The two state of art algorithms for arbitrary dimensions are (a) a greedy heuristic with unproven theoretical guarantees, which is based on executing $O(nr)$ linear programs in total, and (b) a simple space discretization approach that produces an approximate regret ratio that is within a fixed distance from the optimal and has the time complexity of $O(nd + r)$ [22]. Further investigation on this problem has also been done by [6] for the special case of two dimensions (where the problem is not NP-hard), and a quadratic ($O(n^2r)$) algorithm to find the optimal set has been developed, that leverages the notions of geometric duality and line arrangements.

1.2 Technical Highlights

In this paper, we make several fundamental theoretical as well as practical advances for the RRMS problem, in both two-dimensional and high-dimensional databases.

In the case of two-dimensional databases, we develop an innovative dynamic programming algorithm to find the optimal set by leveraging the total order property of the tuples that occur in the skyline of the database. Our two-dimensional exact polynomial time algorithm (**2D-RRMS**) runs in $O(rs \log s \log c)$ time, where s and c are the number of skyline and convex hull tuples in the database respectively. This is a huge improvement over the $O(n^2r)$ algorithm proposed by [6], which is based on the notions of geometric duality and line arrangements.

Next, as perhaps one of the major results of this paper, we develop an approximation algorithm (**HD-RRMS**) that guarantees a regret ratio that is within a small *user-controllable* distance from the optimal regret ratio. This algorithm is based on several innovative ideas. First, we model the problem conceptually as an infinitely large matrix *min-max* problem [24], where the rows are the tuples and the (infinitely many) columns correspond to each possible ranking function. Given an user controlled discretization parameter, we discretize the ranking functions space into a bounded number of functions (based on the control parameter) in the polar system. We then take the advantage of the linear-size discretized problem space in order to find the optimal value for the discretized matrix min-max problem in $O(n \log n)$ time, which is an approximate solution for the original problem. To do so, we convert the problem into linear number of fixed-size instances of the *set-cover* problem [12]. Thus, our eventual algorithm is able to guarantee a regret ratio that is within a user-controllable distance from the optimal regret ratio. The HD-RRMS algorithm is a theoretical algorithm, because although it runs in linearithmic time (assuming that the dimensions of the space and the user controlled parameter are both constants), the proportionality constant in the running time is large (exponentially dependent on the number of dimensions), mainly due to the large size of the set cover instances. Therefore, we make an important practical adaptation to HD-RRMS, by replacing subroutine calls to an exact set-cover algorithm with calls to the well-known greedy approximate set-cover algorithm [7].

Beside the theoretical analysis, we also provide extensive experimental results over three publicly available real-world datasets, i.e., an Airline dataset, Department of Transportation (DOT), and Basketball dataset (NBA), with sizes up to several million records. We also used synthetic datasets to evaluate the performance of the proposed algorithms in the presence of different correlation models, i.e., correlated, independent, and anti-correlated. All experimental results confirm that our algorithms not only are more efficient than the existing solutions, and are scalable, but also produce representative sets with smaller regret ratios.

1.3 Summary of Contributions

In summary, we make the following main contributions:

- For the two-dimensional scenario we propose a *linearithmic* time dynamic programming algorithm **2D-RRMS** which is much faster than the existing quadratic time algorithm.
- We develop **HD-RRMS**, an algorithm for higher dimensions that can approximate the regret ratio to within a user controlled parameter. This algorithm discretizes the ranking function space and models the problem as a discrete matrix min-max problem. Although the HD-RRMS algorithm is linearithmic in theory, it may become inefficient in practice. We propose how to make the algorithm practical.
- We perform a comprehensive set of experiments on synthetic (with different correlation models) and real datasets of size up to several million records that demonstrate the efficiency, scalability, and effectiveness of our algorithms.

2. PRELIMINARIES

Database Model: Consider a database, D with m numeric attributes $\mathcal{A} = \{A_1, \ldots, A_m\}$. Let $Dom(A_i) \geq 0$ be the domain of attribute A_i. The database may also have non-numeric attributes, but since they are usually not part of any ranking function, they are not considered in our context. We assume there are n distinct tuples in the database. For a tuple $t \in D$, we use $t[A_i] \in Dom(A_i)$ to denote the non-NULL value of attribute A_i in t.

Consider a ranking function $F(\cdot) : D \rightarrow \mathbb{R}$ that assigns a score to each tuple t in the database. The ranking function sorts the tuples based on the scores assigned to them. Given such a user specified function, the database determines the tuple with maximum score that should be returned. In this paper, we assume a tuple $t \in D$ outranks a tuple $t' \in D$ based on F, if $F(t) > F(t')$. Furthermore, we follow the existing work [6, 22] and focus on the popular-in-practice [5, 29] linear ranking functions, defined by Equation 1.

$$F(t) = \sum_{i=1}^{m} w_i \cdot t[A_i] \tag{1}$$

$w_i \in [0, 1]$, in Equation 1, is the weight of the attribute A_i. Please note that since we are only restricted to positive weights, everything is confined to the first quadrant.
We also define the *contour* of F as the sets of m-attribute-value combinations that have the same score based on F.

Convex hull: The convex hull is the boundary of the smallest convex region containing all the tuples in D and is formed by the subset of tuples on the boundary [5]. In this paper, we denote the convex hull by $\mathcal{C}$. As proved in [8], $\mathcal{C}$ is the minimal subset that can be used to find the maxima for any linear ranking function[1]. Formally (considering $\mathcal{F}$ as the set of all possible linear ranking functions):

$$\begin{aligned} &(i)\, \forall F \in \mathcal{F}, \exists t \in \mathcal{C} \text{ s.t. } \forall t' \in D, F(t) \geq F(t') \text{ and} \\ &(ii)\, \forall t \in \mathcal{C}, \exists F \in \mathcal{F} \text{ s.t. } \forall t' \in \mathcal{C} \backslash \{t\}, F(t) > F(t') \end{aligned} \tag{2}$$

Depending on the number of tuples in D, and their distributions, size of the convex hull, $|\mathcal{C}| = c$, may be large. As discussed in § 1, even in a two-dimensional database a large number of tuples,

[1]The maxima representative of more general monotonic ranking functions is *skyline* [2]. Skyline is the set of non-dominated tuples in the database, where a tuple t dominates a tuple t' ($t \succ t'$), *iff* (i)$\forall A \in \mathcal{A}$, $t[A] \geq t'[A]$ and (ii) $\exists A \in \mathcal{A}$ such that $t[A] > t'[A]$.

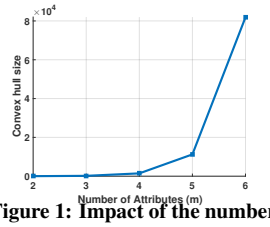

Figure 1: Impact of the number of attributes in $\mathcal{C}$

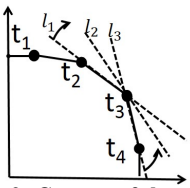

Figure 2: Contour of the ranking functions in a 2D example.

$O(n^{\frac{1}{3}})$, may place in the convex hull [13]. If the convex hull is small, it can be pre-computed, and can then be extremely useful in efficiently finding the maximum scoring tuple of any user-specified ranking function, since a linear scan that examines c tuples will suffice rather than having to examine all n database tuples. The problem with a large $\mathcal{C}$ is that the scan becomes slow. In high dimensional databases, the problem is even worse because when m increases, more and more tuples belong to $\mathcal{C}$, which results in an exponential growth in the size of the convex hull [27]. Obviously, when $\mathcal{C}$ is a large portion of D, it loses its power as a representative. Figure 1 shows how c grows with number of attributes, m, when tuples are uniformly distributed.

Performance Measure: Based on Equation 2, every tuple t in $\mathcal{C}$ is the maximum of at least one linear ranking function. Let $\mathcal{F}_t$ be the set of linear ranking functions for which t is the maximum. If a tuple t is removed from $\mathcal{C}$, the next "best" alternative (the tuple that outranks all the tuples in $\mathcal{C}\backslash t$) is returned as the maximum of a ranking function $F \in \mathcal{F}_t$.

Consider the example provided in Figure 2, where $\mathcal{C} = \{t_1, t_2, t_3, t_4\}$ is the convex hull. The lines between l_1 clockwise to l_3 represent $\mathcal{F}_{t_3}$. If we remove t_3 then t_2 will be returned as the maximum for the ranking functions from l_1 clockwise to l_2, while t_4 will be returned for the ones from l_3 anti-clockwise to l_2. One may notice from the figure that as the function gets closer to l_2 (from either sides), the difference between $\mathcal{F}_{t_3}$ and the best alternative increases, i.e. the maximum score difference happens exactly for the ranking function represented by l_2.

In this paper, we use the *maximum regret-ratio* defined by [22] in order to measure the error of removing a tuple t from a database D, formally defined by Equation 3[2] Intuitively, the maximum regret-ratio is the worst-case score (ratio) difference between the true maximum tuple and the one in the compact set.

$$\forall t \in D,\ E(t, D) = \sup_{\forall F \in \mathcal{F}_t} \frac{\min_{\forall t' \in D\backslash\{t\}}(F(t) - F(t'))}{F(t)} \quad (3)$$

The regret-ratio can be extended to measure the error of removing a set of tuples $T \subset D$. Specifically, this error is the maximum score-difference ratio of each tuple in T and its best alternative in the remaining convex hull, formally defined by Equation 4.

$$\forall T \subset D,\ E(T, D) = \max_{\forall t \in T}\left(\sup_{\forall F \in \mathcal{F}_t} \frac{\min_{\forall t' \in D\backslash\{T\}}(F(t) - F(t'))}{F(t)}\right) \quad (4)$$

In the rest of the paper, for simplicity we use E_T as $E(T, D)$.

2.1 Problem Definition

In this paper, we consider the problem of Regret-ratio Minimizing Set, i.e., given a database D and an integer $r \geq 1$, our objective is to find a set of at most r tuples such that the regret-ratio is minimum. This problem is formally defined as follows:

[2]We use the notation $E(T, D)$ to show $rr_D(D\backslash T, \mathcal{L})$ in [22].

> REGRET-RATIO MINIMIZING SET PROBLEM (RRMS): Given a database D and an integer $r \geq 1$, find a subset $\hat{\mathcal{C}} \subseteq D$ such that (1) $|\hat{\mathcal{C}}| \leq r$ and (2) $E(D\backslash\hat{\mathcal{C}}, D)$ is minimum.

In the following, we propose our efficient algorithms to solve the Regret-ratio Minimizing Set problem in both two-dimensional and high-dimensional databases.

3. 2D REGRET-RATIO MINIMIZING SET

We start by considering the two-dimensional scenario which, as discussed in the introduction, not only has significant theoretical implications but also represents popular use cases in practice.

We show in Theorem 1 that the search space can be reduced to the skyline tuples rather than in all n tuples in the database.

THEOREM 1. *Let T be the set of tuples which are removed from D. The maximum regret-ratio of the optimal solution for the regret-ratio minimizing set problem on $\mathcal{S}$ is the same as the maximum regret-ratio of the optimal solution for the regret-ratio minimizing set problem on D, i.e., $E(T, D) = E(T, \mathcal{S})$.*

PROOF. Please refer to Appendix A. □

Considering Theorem 1, we first order all skyline tuples in a two-dimensional table from top left to the bottom right, i.e. $\mathcal{S} = \{t_1, t_2, \cdots, t_s\}$. We add two dummy tuples t_0 and t_{s+1} to the left of the top left skyline tuple t_1 and to the right of the bottom right skyline tuple t_s respectively. In other words, $t_0 = (0, \max t_i[A_2])$, and $t_0 = (\max t_i[A_1], 0)$, where $\max t_i[A_j]$ is the maximum value of the A_j in all skyline tuples. Figure 3 shows a dataset with 5 skyline tuples $\{t_1, t_2, \cdots, t_6\}$ and two dummy tuples t_0 and t_7. Next, we propose a graph model for the two-dimensional Regret-ratio Minimizing Set problem and we propose a polynomial time algorithm using dynamic programming in order to find the r tuples such that the maximum regret-ratio is minimum.

3.1 Graph Modeling

3.1.1 Reduction to Path Search in Graph

We model the two-dimensional Regret-ratio Minimizing Set problem as a weighted complete graph $G = (V, E)$, where V is the set of skyline tuples, $\{t_1, t_2, \cdots, t_s\}$, and two dummy tuples, t_0 and t_{s+1}. Edge weight $w(t_i, t_j)$ represents the regret-ratio of removing all skyline tuples between t_i and t_j. Note that as we proved in the Theorem 1, the optimal solution is a subset of skyline tuples. Using this graph model, our goal is to find a path from t_0 to t_{s+1} with at most r intermediate tuples such that tuples follow an increasing order of the subscript in the path and the maximum of the edge weights are minimized. Next, we discuss how to efficiently compute the edge weight $w(t_i, t_j)$. Using the graph model we first discuss the baseline solution for the problem, and then we describe the detail of the dynamic programming approach.

3.1.2 Edge weight computation

Each convex hull tuple is the maximum for a set of ranking functions. For example in Figure 4, t_1 is the maximum for all linear ranking functions from $F \in [0, \theta_1]$, while t_3 is the representative of all linear ranking functions from $F \in [\theta_1, \theta_2]$. Similarly, t_4, and t_5 are the representatives of all linear ranking functions of $F \in [\theta_2, \theta_3]$ and $F \in [\theta_3, \pi/2]$ respectively. One may note that moving from the top-left to the bottom-right, the contours of the all ranking functions where the convex hull tuples are maxima form a sorted range of the angles from 0 to $\pi/2$. For example, in Figure 4, the sorted list $\ell = [0, \theta_1, \theta_2, \theta_3, \pi/2]$ shows all possible

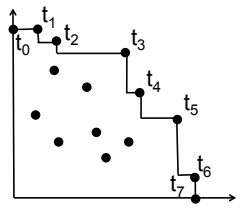

Figure 3: Dataset with skyline tuples $\{t_1, t_2, \cdots t_6\}$ and two dummy tuples t_0 and t_7.

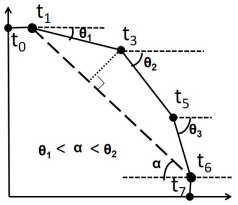

Figure 4: Error of removing $\{t_2, \cdots, t_5\}$

linear ranking functions where the convex hull tuples are the maxima. In other words the ith element of l shows all ranking functions $F \in [\theta_{i-1}, \theta_i]$ where the ith convex hull tuple is the maximum.

In Theorem 2, we prove that for two-dimensional databases, the function F that causes the maximum regret-ratio for removing the tuples between two skyline tuples t_i and t_j is specified by the line between tuples t_i and t_j. Intuitively, when we keep t_i and t_j while removing all skyline tuples in between, and start tallying the loss from removing t_{i+1}, ..., t_{j-1} in order, then the loss (max over all ranking functions) must first increase and then decrease. Having identified the max-loss function, we now take the advantage of the sorted angle list (ℓ) and apply *binary search* on ℓ to find the convex hull tuple between t_i and t_j which is the maximum for the ranking function represented by the line passing through t_i and t_j. Algorithm 1 shows the pseudocode of the function **ComputeEdgeWeight** for the two tuples t_i and t_j. Note that if it can not find a convex hull tuple between t_i and t_j the edge weight is zero (line 4-5). Clearly since we use binary search on the sorted list l, we are able to find weight of each edge in $O(\log c)$, where c is the size of the convex hull.

Algorithm 1 ComputeEdgeWeight

1: **Input:** Tuples t_i and t_j, Sorted list $l = [0, \theta_1, \cdots, \pi/2]$
2: **Output:** Edge weight $w(t_i, t_j)$
3: **if** i = 0 **then return** $t_1[A_2] - t_j[A_2]$
4: **if** j = s+1 **then return** $t_s[A_1] - t_i[A_1]$
5: compute α, where t_i and t_j are the maxima of all linear ranking functions $F \in [0, \alpha]$.
6: k = Use binary search on l to find the location of α
7: **if** $i \leq k \leq j$ **then** $w(t_i, t_j) = \frac{\min(F_\alpha(t_k) - F_\alpha(t_i), F_\alpha(t_k) - F_\alpha(t_j))}{F_\alpha(t_k)}$
8: **else** $w(t_i, t_j) = 0$
9: **return** $w(t_i, t_j)$

THEOREM 2. *In 2D, after removing the tuples between two skyline tuples t_i and t_j, the maximum regret-ratio occurs for the function, F, corresponding to the line between tuples t_i and t_j.*

PROOF. Please refer to Appendix B. □

For example, let us consider the edge between t_1 and t_5 (representing the removal of $\{t_2, \cdots, t_4\}$). As shown in Figure 4, t_1 and t_5 will be the representative of all linear ranking functions $F \in [0, \alpha]$. Using binary search on l, it turns out that $\alpha \in (\theta_1, \theta_2)$, i.e., t_3 is the tuple which is removed and has the maximum loss. Therefore, $w(t_1, t_5) = \frac{\min(F_\alpha(t_3) - F_\alpha(t_1), F_\alpha(t_3) - F_\alpha(t_5))}{F_\alpha(t_3)} = \frac{F_\alpha(t_3) - F_\alpha(t_1)}{F_\alpha(t_3)}$ (line 4 in Algorithm 1). As another example let us consider $w(t_1, t_2)$. Since there are no convex hull tuples between these two tuples, $w(t_1, t_2) = 0$. Nevertheless, t_3 is the maximum for the function represented by the line passing through t_1 and t_2, which is not removed by considering the edge t_1 to t_2.

Next, we discuss a baseline solution for RRMS problem in 2D, based on the proposed graph modeling.

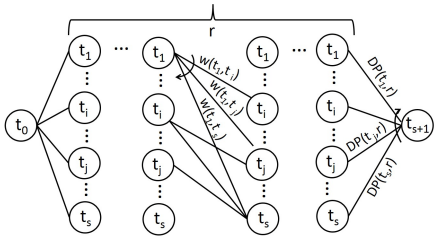

Figure 5: Dynamic programming approach of 2D-RRMS Algorithm.

3.2 Baseline Solution

Given the graph model and weight definition, a baseline solution is to compute all weights of the graph and then enumerate all paths from t_0 to t_{s+1} with at most r intermediate tuples. Among those paths the one whose maximum edge weight is the minimum is the solution. Clearly this is inefficient because it takes time quadratic in the number of skyline tuples ($O(r^2)$) to calculate all edge weights in the graph. Moreover, it has to enumerate all $\sum_{l=0}^{r} \binom{n-2}{l}$ paths from t_0 to t_{s+1} with at most r intermediate tuples, which can take exponential time. Next, we leverage the locality property of the skyline tuples in order to propose a polynomial time algorithm **2D-RRMS** using a dynamic programming approach.

3.3 Dynamic Programming Algorithm

Let $DP(t_i, r')$ be the optimal solution, i.e., a path from t_i to t_{s+1} with at most $r' \leq r$ intermediate nodes which minimizes the error. Thus, $DP(t_0, r)$ would be the solution to our problem. The recursive formula for the dynamic programming is given by Equation 5:

$$\begin{aligned} DP(t_i, 0) &= \max(w(t_0, t_i), w(t_i, t_{s+1})) \\ DP(t_i, r') &= \min_{\forall j > i} \left(max \left(w(t_i, t_j), DP(t_j, r' - 1) \right) \right) \end{aligned} \quad (5)$$

Since skyline tuples are ordered, they provide two important properties which are helpful to efficiently solve the recursive Equation 5:

1. $w(t_i, t_j) \leq w(t_i, t_{j+1})$
2. $DP(t_j, r') \leq DP(t_{j-1}, r')$.

Figure 5 shows the construction of the dynamic programming algorithm for DP. It contains t_0 and t_{s+1} at its first and last tuples. Every cell (i, j) in the middle matrix represents $DP(t_i, r - j)$. As shown in the figure, the weights increase from top to bottom, while DP increases from bottom to top.

In order to find the min value in Equation 5, Algorithm 2 divides the space between t_i and t_{s+1} into two halves and picks the one in the middle as t_m. Note that the weight computation will be done online as needed. If the edge weight $w(t_i, t_m)$ is not previously calculated it will call the function **ComputeEdgeWeight** in Algorithm 1 (lines 6-16). Then if $w(t_i, t_m) \geq DP(t_m, r' - 1)$, we can ignore the nodes between t_m and t_{s+1} because, based on property (1), $w(t_i, t_m)$ is the smallest among them and those branches will not find a better result (line 14). In this case, Algorithm 2 continues by dividing the tuples between t_{i+1} and t_{m-1}. On the other hand, if $w(t_i, t_m) < DP(t_m, r' - 1)$ (line 15), we can ignore the nodes between t_i and t_m because, based on property (2), $DP(t_m, r' - 1)$ is the smallest among them. In this case, Algorithm 2 continues to divide the tuples between t_{m+1} and t_s.

THEOREM 3. *Time complexity of the algorithm 2 (2D-RRMS) is in $O(rs \log s \log c)$.*

PROOF. Please refer to Appendix C. □

Algorithm 2 2D-RRMS

Input: Integer $r \geq 1$, Skyline tuples $\mathcal{S} = \{t_0, t_2, \cdots, t_{s+1}\}$, Sorted list $l = [0, \theta_1, \cdots, \pi/2]$
Output: The optimal regret-ratio $DP(t_0, r)$
for i from 1 to s **do** $DP(t_i, 0) = \max(w(t_0, t_i), w(t_i, t_{s+1}))$
for r' from 1 to r **do**
  for i from 1 to s **do**
    $low = i + 1$, $high = s$
    while True **do**
      if $low = high$ **then**
        $DP(t_i, r') = \max(w(t_i, t_{low}), DP(t_{low}, r' - 1))$
        break
      end if
      Let t_m be the middle tuple in $\{t_{low}, \cdots, t_{high}\}$
      if $w(t_i, t_m)$ is unknown **then**
        $w(t_i, t_m) =$ **ComputeEdgeWeight**(t_i, t_m, l)
      if $w(t_i, t_m) \geq DP(t_m, r' - 1)$ **then** $high = m$
      else $low = m + 1$
    end while
  end for
end for
return $DP(t_0, r)$

4. HD REGRET-RATIO MINIMIZING SET

It is well known that the size of convex hull grows exponentially with the data dimensionality (i.e., the number of attributes) [27] (also shown in Figure 1). The need for designing a compact representation of high dimensional databases thus becomes even more pronounced. We address the discovery of regret-ratio minimizing sets over high-dimensional databases in this section.

Specifically, we start with discussing the deficiency of the existing heuristic solution. Then, we introduce a conceptual model of the problem as an infinitely large matrix *min-max* problem [24]. We "operationalize" such a conceptual model with a matrix discretization approach that provides a user-controllable discretization parameter. After discretizing the problem space to a manageable size, we then construct a reduction to *set-cover* [12] which solves the min-max problem deterministically in (theoretically) $O(n \log n)$ time, while guaranteeing a regret ratio within any arbitrarily small user-controllable distance from the optimal regret ratio. We make this algorithm more practical by incorporating an existing approximation algorithm for set-cover. Our final algorithm outperforms existing solutions by an order of magnitude over real-world datasets, as we shall show in the experimental evaluations.

4.1 Problem with Existing Heuristic Solution

Finding r tuples that minimize the regret-ratio over a high-D database has been proven to be NP-hard [6]. The existing solutions limit to a greedy heuristic (named as GREEDY) and a simple space discretization approach that produces a regret ratio within *a fixed distance from the optimal*, both proposed by Nanongkai et. al. [22]. The basic idea of GREEDY is to start by selecting the point that has the highest value on the first attribute, and then iteratively selecting the point with the maximum error from the selected points and adding it to the set. This algorithm is not designed to provide any approximate-ratio guarantee. In addition, as we shall show below, it performs quite badly in some cases. Specifically, for any *arbitrarily large* value v, we can always find a case in which the regret-ratio of the solution provided by GREEDY is no better than v times the optimal solution.

Given $v > 0$, let $\epsilon = 1/(2 + v)$. Consider a 3-dimensional database D which contains four tuples $t_0(1, 0, 0), t_1(0, 1, 0), t_2(0, 0, 1)$, and $t_3(1-\epsilon, 1-\epsilon, 1-\epsilon)$, along with an arbitrary number of other tuples that are distributed in the region $[0, 1-\epsilon) \times [0, 1-\epsilon) \times [0, 1-\epsilon)$. One can see that, when we run GREEDY with output-size requirement $r = 3$, GREEDY will pick t_0, t_1, and t_2, while the optimal solution is t_3 together with two of t_0, t_1, and t_2.

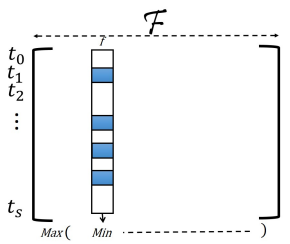

Figure 6: Illustration of Matrix M.

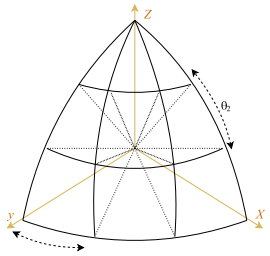

Figure 7: Illustration of space partitioning example: $m = 3$, $\gamma = 3$.

To see how bad the regret ratio for GREEDY's solution is, note that its regret ratio is equal to the distance between point t_2 and the line passing through points t_0 and t_1, which is equal to $1 - 2\epsilon$. Meanwhile, the regret ratio for the optimal solution is ϵ. Thus, the approximate-ratio is $(1 - 2\epsilon)/\epsilon \geq v$.

4.2 Conceptual Model

An intuitive illustration of our conceptual model is shown in Figure 6. Specifically, consider a matrix M that has the n tuples as its rows, while its columns consist of all possible linear ranking functions ($\mathcal{F}$). Each cell $M[t_i, f]$ of the matrix is the regret-ratio of t_i with regard to the ranking function f - i.e., the regret ratio for f if we only have t_i in the minimizing set. Thus for each tuple $t_i \in \mathcal{C}$ and the set of ranking functions $f \in \mathcal{F}_{t_i}$ (the set for which t_i is the maximum), $M[t_i, f]$ is zero, while this value is greater than zero for other tuples. If we keep r rows of the matrix, the regret-ratio for each function, is the minimum value (among the selected rows) on its corresponding column, and the regret-ratio of these r tuples is the maximum assigned regret-ratio of all columns. We can see the problem cleanly transforms to a *min-max* problem over the matrix.

This conceptual model, unfortunately, has an important issue which makes it impractical: $\mathcal{F}$ is continuous and therefore requires an infinite number of columns to capture! This issue inspires us to develop a *matrix discretization* approach which eventually leads to a linearithmic approximate solution that offers a guaranteed approximation ratio adjustable by a user-controlled parameter.

4.3 Matrix Discretization

In order to resolve the aforementioned issues with the conceptual model, we first discretize $\mathcal{F}$ by only considering a subset of all possible linear ranking functions $F \subset \mathcal{F}$ as the columns of the matrix M. We take the help of Polar coordinate system for selecting F. Note that, with help of the polar system, each point is denoted by one magnitude and $m - 1$ angles. For example, tuple $t(1, 1)$ is transformed to $t\langle\sqrt{2}, \pi/4\rangle$ in polar and $t'(1, 0, 1)$ to $t'\langle\sqrt{2}, 0, \pi/4\rangle$.

Specifically, we introduce a user-controllable parameter γ which determines the size of F, by dividing each angle into γ equal-size partitions. Thus, for a given value γ each angle partition is:

$$\alpha = \frac{\pi}{2\gamma} \tag{6}$$

Applying this discretization policy, the total number of selected functions ($|F|$) is:

$$|F| = (\gamma + 1)^{m-1} \tag{7}$$

Algorithm 3 shows the pseudo-code of the DISCRETIZE algorithm, that partitions the ranking function space based on α and se-

lects F. For example, when $m = 3$ and $\gamma = 3$, we have $\alpha = \pi/6$ according to (6). Figure 7 shows the three-dimensional function space discretization based on α.

Algorithm 3 DISCRETIZE

Input: Control Parameter γ, Number of attributes m
Output: Discretized functions F
$F = \{\}$, $\alpha = \frac{\pi}{2\gamma}$
for i from 1 to $m-1$ **do** $\theta[i] = 0$
for i from 1 to γ^{m-1} **do**
  $r = 1$
  {*transforming the function from polar system to scalar*}
  for j from m downto 2 **do**
    $v[j] = r\cos(\theta[j-1]\alpha)$
    $r = r\sin(\theta[j-1]\alpha)$
  end for
  $v[1] = r$
  $F = F \cup \{v\}$
  {*finding the next function*}
  for j from 1 until $\theta[j] < \gamma$ **do** $\theta[j] = 0$
  $\theta[j] = \theta[j] + 1$
end for
return F

THEOREM 4. *If a set T of tuples guarantee a regret-ratio threshold of ϵ for all the ranking functions in $f \in F \subseteq \mathcal{F}$, constructed based on the angle partitioning (α) in Equation 6, the maximum regret-ratio of those points for any ranking function $f' \in \mathcal{F}$ is:*

$$\epsilon' \leq c\epsilon + (1-c) \tag{8}$$

where $c = \frac{\cos(\alpha'/2)\cos(\pi/4)}{\cos(\pi/4-\alpha'/2)}$ *and* $\alpha' = 2\arcsin(\sqrt{\frac{1-cos^{m-1}\alpha}{2}})$.

PROOF. Please refer to Appendix D. □

Considering the guarantee provided in Theorem 4, we discuss our approximate algorithm over the discretized function space in the next section.

4.4 HD-RRMS Algorithm

In this section, we first model the problem as the discretized min-max problem. We then take the advantage of the linear-size discretized problem space in order to find the optimal value for the discretized matrix min-max problem.

4.4.1 DMM: Discretized min-max Problem

Given the (above described) discretized matrix M, and the value r, find a set of r rows that minimizes the maximum of the minimum values of all columns among the selected r rows. Formally, find:

$$\min_{\forall R \subseteq \mathcal{S} \text{ s.t. } |R|=r} \max_{\forall f \in F} \min_{\forall i \in R} (M[i,f]) \tag{9}$$

If we assume that the number attributes, m, is bounded by a constant, and the user controlled parameter γ is a constant, then the total number of discrete functions $|F|$ becomes bounded by a (albeit large) constant. Recall that M is a matrix with n rows and $|F|$ columns, where each cell shows the regret-ratio. Given any set of r rows, *the solution to the discretized min-max problem is one of the cell values*. The total number of such values is, at most, $n.|F|$, which since $|F|$ is bounded by a constant, is in $O(n)$. We consider each distinct value in cells of the matrix M as a possible ϵ threshold of the following problem.

MRST: Minimum Rows Satisfying the given Threshold problem: Given the discretized matrix M and the threshold value ϵ, find the minimum number of rows in matrix M such that for each column of matrix M, the minimum value of each column among the selected rows is less than or equal to ϵ. Formally:

$$\text{minimize } |R|, \forall R \subseteq \mathcal{S} \text{ where } \max_{\forall f \in F} \min_{\forall i \in R} (M[i,f]) \leq \epsilon \tag{10}$$

Consider an oracle that solves the MRST problem. HD-RRMS sorts all the distinct values in matrix M, and applying the binary search strategy, passes the cell-values to the oracle to find the set of rows that satisfy the threshold. Based on the fact that the size of returned set is either less than r or not, it continues the search in lower/upper half. Algorithm 4 shows the pseudo-code of the HD-RRMS algorithm in the presence of the MRST oracle.

Algorithm 4 HD-RRMS

Input: The discretized matrix M and the value r
Output: selected tuples (T_A)
v = sorted list of distinct values in M
$T_A = \{\}$, $\epsilon_{min} = \infty$
$low = 0$, $high = |v|$
while $low < high$ **do**
  $mid = \frac{low+high}{2}$
  R = **MRST**(M, $v[mid]$)
  if $|R| \leq r$ **then**
    $T_A = R$, $\epsilon_{min} = v[mid]$
    $high = mid - 1$
  else
    $low = mid + 1$
  end if
end while
return T_A

Suppose T_O is the optimal set with the minimum regret-ratio. Note that since the output of the HD-RRMS algorithm is the optimal solution for a subset of ranking functions, ϵ_{min} is less than or equal to the regret-ratio of T_O:

$$\epsilon_{min} \leq E_{D \setminus T_O} \tag{11}$$

That is because, if there is no subset of tuples with size of at most r that satisfy the regret-ratio of less than ϵ_{min} for F (which is a subset of $\mathcal{F}$), no set will have the regret-ratio of less than ϵ_{min} for its super-set ($\mathcal{F}$). Moreover, based on Theorem 4 we know $E_{D \setminus T_A} \leq c\epsilon_{min} + (1-c)$. From Equation 11 $c\epsilon_{min} + (1-c) \leq cE_{D \setminus T_O} + (1-c)$. Thus:

$$E_{D \setminus T_A} \leq cE_{D \setminus T_O} + (1-c) \tag{12}$$

Since $c \leq 1$, based on Equation 12, the regret-ratio of the set discovered by HD-RRMS is within $(1-c)$ distance from the regret-ratio of the optimal solution.

MRST Oracle: The only missing part of the algorithm is the MRST oracle. We model the MRST problem with the set-cover problem [12] by constructing the matrix M' as following:

$$\forall i, f: \; M'[i,f] = \begin{cases} 0, & \text{if } M[i,f] > \epsilon \\ 1, & otherwise \end{cases} \tag{13}$$

M' contains constant number of columns and s rows. Since the number of columns is bounded by a constant, there exists at most a constant number (more precisely, the power-set of $|F|$) of distinct combinations for the row values of M'. In the next step, the algorithm (giving a distinct id to each value-combination of columns) parses M' and removes the duplicate rows. As a result, M' becomes a matrix whose number of rows and columns are bound by constants! Next we transform the MRST problem to the *set-cover* problem as follows:

- Each column f in M' is an item in the set-cover problem.

- Each row i in M' is a set $\mathbb{S}_i$ in the set-cover problem such that $\forall$ column f, $f \in \mathbb{S}_i$ *iff* $M'[i, f] = 1$.

Now the problem is converted to a set-cover instance with a constant number of items and constant number of sets. Thus, even though the set-cover problem is NP-complete in general, an optimal solution for this instance is, *theoretically*, possible in constant-time order. MRST uses the set-cover solver to find the minimum number of sets that cover all the columns and returns its corresponding rows as the optimal solution for the MRST problem. Algorithm 5 shows the pseudo-code of MRST oracle.

4.4.2 Time Complexity of the HD-RRMS Algorithm

Since the size of the discretized matrix M is $n|F|$, the binary search over the possible values of ϵ takes $\log(n|F|)$ steps. At each step, a set-cover with $|F|$ items and $\min(2^{|F|}, n)$ sets is constructed. Converting M to the set-cover instance takes $n|F|$ time, while solving it takes, at most, $2^{\min(2^{|F|},n)}|F|$ time.
Since, $|F| = \gamma^m$, the total time to solve HD-RRMS is:

$$\log(n\gamma^m).(n\gamma^m + (2^{\min(2^{\gamma^m},n)}.\gamma^m)) \tag{14}$$

Assuming that γ and m are constants, the running-time of HD-RRMS is, *theoretically*, in $O(n\log(n))$.

Algorithm 5 MRST Oracle

Input: The discretized matrix M and the threshold ϵ
Output: The set of tuples satisfying the threshold on M
{*constructing the matrix* M'}
for i from 1 to s **do**
  for f in columns of M **do**
    $M'[i, f] = 1$ **if** $(M[i, f] \leq \epsilon)$ **else** 0
  end for
end for
{*removing the duplicate rows*}
seen = {}
for i in rows of M' **do**
  if id($M'[i]$) $\in$ seen) **then**
    M'.remove(i)
  else
    seen = seen $\cup\{i\}$
  end if
end for
{*constructing the set-cover problem*}
items={}, sets={}
for f in columns of M' **do** items = items $\cup\{f\}$
for i in rows of M' **do**
  set={}
  for f in columns of M' **do**
    if $M'[i, f] == 1$ **then** set = set $\cup\{f\}$
  end for
  sets = sets $\cup\{$ set $\}$
end for
return set-cover(items, sets)

4.4.3 Practical HD-RRMS Algorithm

Although HD-RRMS is linearithmic in theory, it can be inefficient in practice. Even with the existence of an efficient set-cover solver (such as [23]), depending on the values of γ and m, solving this problem may become infeasible.

To make the HD-RRMS Algorithm practical, we solve the set-cover instances approximately using the well-known greedy approximate algorithm for set-cover [7]. Since this algorithm guarantees a log factor in the approximate-ratio, applying it adds another level of approximation and increases the set size to up to $r\log(|F|) = rm\log(\gamma)$ (rather than r) tuples, while maintaining the distance from the optimal regret ratio in the bound provided in Theorem 4. Alternatively, one can find a new value (r') for set-cover size such that $r' = \frac{r}{m\log(\gamma)}$. In this way, we can make sure that the set size will not be more than r. However, the distance from the optimal solution may become larger than what provided in Theorem 4. Applying the greedy approximate set-cover reduces the running time to:

$$\begin{aligned}&\log(n\gamma^m).(n\gamma^m + (\min(2^{\gamma^m}, n).\gamma^m))\\ \leq\ &2n\gamma^m\log(n\gamma^m)\end{aligned} \tag{15}$$

Given the importance of the choice of γ in Equation 15, we evaluate the impact of the discretization control parameter (γ) in § 6.3 (Figures 24, 25, and 26). On examining the experiment results, it turns out even though increasing the value γ increases the running time significantly, the improvement on the quality of results drops quickly. In our experiments, choosing γ between 4 and 6 seemed to be appropriate.

5. DISCUSSION

5.1 Top-k Extension

For enabling efficient computation of Top-k queries (instead of only Top-1 queries) requires us to extend the compact set for higher values of $k > 1$, while minimizing the dissatisfaction of the k^{th} top tuple of the compact set versus the k^{th} tuple of the actual database, An easy way of adopting the existing algorithms is an iterative approach with k iterations. At each iteration we discover the maxima set over the remaining tuples. Then we remove the tuples in the maxima set, as well as the tuples falling outside of the convex shape formed by the set, and start the next iteration over the remaining tuples in the database. One can see that this method will construct k compact layers around the data and can serve for discovering the Top-k. However, studying the theoretical guarantees of such adaptation requires further investigation in a future work.

5.2 Alternative Matrix Discretization

As an alternative for matrix discretization proposed in § 4.3, we can change the algorithm to let the user specify the size of function space as the control parameter (rather than the value γ) which makes $|F|$ independent from the value of m. Now the discretization problem is reduced to the problem of evenly distributing $|F|$ points on the surface of a quarter hyper-sphere (the vector from the origin to a point on surface represents a function $f \in F$).

One way for doing so, is to adopt the force-directed drawing algorithms [18], such as the Barycentric algorithm [25], for evenly distributing the points. Suppose each point (bound to exist on the surface of the hyper-sphere) is a particle with a fixed positive charge. The idea is that if we throw $|F|$ particles on the hyper-sphere, they start moving (based on the average of forces between them) until they form an even distributed in which the superposition of forces is zero. The algorithm thus is as following: until the superposition of forces on all the points has not converged to zero, compute the force on each point and move it on the hyper-sphere based on the direction and the magnitude of it. Another alternative is relaxing the notion of an even distribution of the points to the randomly at uniform distribution. Thus, we can uniformly (at random) select each functions, in the polar space, by specifying the value of each angle uniformly at random between 0 and $\Pi/2$. Now, we can compute the expected angle distance between the two neighboring functions which specifies a expected bound based on Theorem 4.

6. EXPERIMENTS

6.1 Experimental Setup

Hardware and Platform: All our experiments were performed on a Core-I7 machine running Ubuntu 14.04 with 8 GB of RAM. The algorithms were implemented in Python.

Real-world Datasets: We used three publicly available real-world datasets, i.e., Airline dataset, Department of Transportation (DOT), and Basketball dataset (NBA).

- *Airline dataset*[3]*:* The 2008 Airline dataset has $5,810,463$ records with 13 attributes, out of which two of them (namely *Actual Elapsed Time* and *Distance*) are ordinal. We used this dataset to test the performance of the two-dimensional algorithms over a large dataset.
- *DOT dataset*[4]*:* The flight on-time dataset is published by the US Department of Transportation. It records, for all flights conducted by the 14 US carriers in January 2015, attributes such as scheduled and actual departure time, taxiing time and other detailed delay metrics. The dataset consists of 457,013 tuples over 28 attributes with 7 ordinal attributes *Dep-Delay, Taxi-out, Taxi-in, Actual-elapsed-time, Air-time, Distance, ArrivalDelay*.
- *NBA dataset*[5]*:* This basketball dataset contains the points for the combination of player/team/season up to 2009. It contains $21,961$ tuples and 17 ordinal attributes: *gp, minutes, pts, oreb, dreb, reb, asts, stl, blk, turnover, pf, fga, fgm, fta, ftm, tpa, tpm*.

Synthetic Data: We also used synthetic data to evaluate the performance of our algorithms in the presence of different kind of correlations between attributes. Note that such correlations affect the performance of baseline solutions, e.g., the size of the skyline, as the more correlated all attributes are, the smaller the skyline becomes. Moreover, the correlations could affect the regret ratio of our outputs too. Specifically, we used the method proposed in [2] to generate three datasets with correlated, independent, and anti-correlated attributes. Each dataset has $10M$ tuples with 10 attributes.

Algorithms Evaluated: For the two-dimensional case, we evaluated the performance of our two-dimensional algorithm, 2D-RRMS, and compared its performance with the two-dimensional Sweeping-Line algorithm proposed in [6]. Sweeping-Line is a quadratic algorithm that considers the points in the dual space and covers the function space using a sweeping line while updating the regret-ratio of the points, as their corresponding lines intersect. We used the Nested Block Loop algorithm [2] in order to compute the skylines for our 2D-RRMS algorithm.

We also evaluated the performance of the high-dimensional algorithm discussed in § 4, namely GREEDY [22] and HD-RRMS (Algorithm 4). In the practical implementation of HD-RRMS algorithm, we applied the greedy approximate solution to solve the set-cover problem [12]. Nonetheless, to be fair in comparing the algorithms' performance, we only accept the set-cover result if its size is at most r. Note that our HD-RRMS algorithm features two main ideas: One is the conceptual model (of matrix min-max problem) along with its practical discretization, and the other is the reduction to set-cover and the corresponding approximation algorithm. To test the effectiveness of these two ideas separately, we devised another greedy algorithm called HD-GREEDY, which simply applies an iterative greedy approach over the discretized matrix generated by the first idea (i.e., as explained in § 4.3). Specifically, HD-GREEDY iteratively picks a tuple that minimizes the max of the min value of the columns for the selected set of tuples. The complexity of HD-GREEDY is $O(rn)$, because each iteration requires passing through the matrix once, while computing the reduction in the matrix max column value only takes $O(1)$. As one can see, HD-GREEDY uses only the (discretized version of the) conceptual model, but not our second idea of reduction to set-cover. As we shall show latter in the section, the performance of HD-GREEDY almost always falls in between GREEDY and HD-RRMS, thus demonstrating the effectiveness of both of our ideas.

As mentioned in § 1, the previous studies on approximating high-dimensional convex hulls [1] and relaxing skyline definitions [4] differ in objective from our paper (which aims to minimize the user dissatisfaction on maxima queries). Nonetheless, in order to provide a broader context for the efficacy of our algorithms, we still implemented both [1] and [4], and studied the possibility of applying them for regret minimization.

Performance Measures: For the two-dimensional case, since our 2D-RRMS algorithm always guarantees optimality in terms of regret ratio, we focus our evaluations on the *execution time*. Specifically, to be fair to the sweeping-line algorithm, we considered the execution time of our 2D-RRMS algorithm to be the SUM of the execution time of both the skyline computation process (we used the skyline computation algorithm in [2]) and the actual execution of 2D-RRMS. For the high-dimensional case, we used both the regret-ratio and the overall execution time of an algorithm measure its performance - naturally, the shorter the execution time and the smaller the regret ratio, the better.

6.2 Two-dimensional Experimental Result

Figures 8 and 9 show the performance of our 2D-RRMS algorithm (circle marker) and the Sweeping-Line algorithm proposed in [6] (triangle marker). We tested both algorithms over the correlated, independent, and anti-correlated synthetic datasets. In these figures, green dotted line, blue dashed line, and red solid line are used for the correlated, independent, and anti-correlated synthetic datasets respectively.

Impact of the dataset size (n): In these set of experiments, we varied the dataset size from 5K to 10M for each of correlated, independent, and anti-correlated synthetic datasets. Figure 8 shows the execution time of each algorithm. From the figure, one can see that 2D-RRMS algorithm outperforms the Sweeping-Line algorithm by orders of magnitude. As expected, the performance of the Sweeping-Line algorithm does not depend on the correlation of the attributes because it considers all the joins between points in the dual environment. The 2D-RRMS algorithm performs better for the correlated and independent datasets because the anti-correlated case generates a larger skyline which affects the performance of the Nested Block Loop algorithm [2] used for skyline discovery. Note that, while utilizing more efficient skyline algorithms will reduce the measured execution algorithm of 2D-RRMS, we did not further pursue this direction as it is orthogonal to our research. In addition, 2D-RRMS already outperforms the sweeping-line algorithm by an order of magnitude even in the anti-correlated case.

Impact of the output size (r): Next, we fixed the dataset size to 40K, and varied the output size (r) from 3 to 10 (Figure 9). Again in all experiments, 2D-RRMS algorithm significantly outperforms the Sweeping-Line algorithm. Since the time complexity of the Sweeping-Line algorithm, $O(rn^2)$, is quadratic in dataset size and linear in output size, varying r does not affect the performance of the algorithm. The execution time of our 2D-RRMS algorithm also does not change by varying r. The reason is actually *not* because the execution time of our algorithm has no dependency on r, but because the pre-processing step (skyline discovery) actually domi-

[3] http://kt.ijs.si/elena_ikonomovska/datasets/airline/2008_14col.data.bz2

[4] http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

[5] http://www.databasebasketball.com/

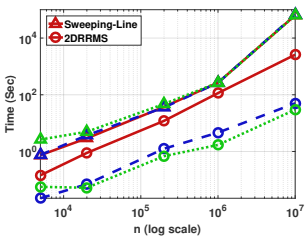

Figure 8: 2D, Impact of dataset size (n) on correlated, independent, and anti-correlated datasets

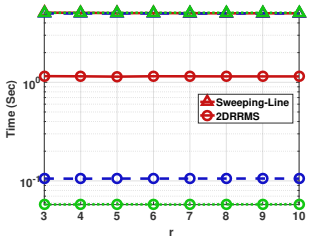

Figure 9: 2D, Impact of output size (r) on correlated, independent, and anti-correlated datasets

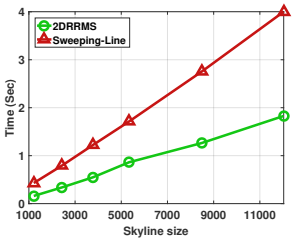

Figure 10: 2D, Impact of skyline size (n) on skyline-only datasets

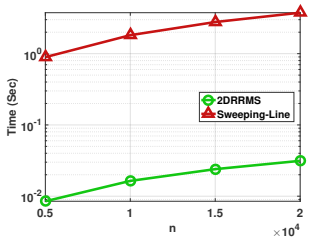

Figure 11: 2D, NBA dataset, varied the dataset size (n)

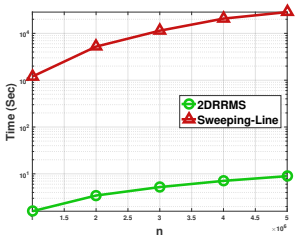

Figure 12: 2D, Airline dataset, varied the dataset size (n)

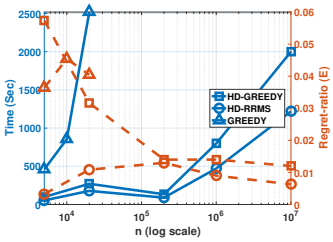

Figure 13: HD, Impact of dataset size (n) on correlated dataset

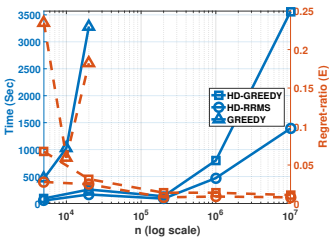

Figure 14: HD, Impact of dataset size (n) on independent dataset

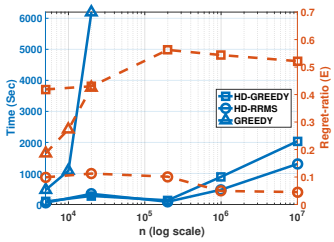

Figure 15: HD, Impact of dataset size (n) on Anti-correlated dataset

nates the overall running time, and this pre-processing step is independent to r.

Experiment on the skyline-only datasets: To test the skyline-size effect on the performance of our algorithm, we generated synthetic, "skyline-only" datasets (i.e., in which every tuple is on the skyline), with varying sizes. We did so by drawing uniformly at random from all points inside the 2D unit circle, and then iteratively removing a point if it is dominated by others. We generated 6 such skyline-only datasets of sizes 1212, 2431, 3782, 5335, 8488, and 12032 (skyline) tuples, respectively. Figure 10 shows the performance of the algorithms. We can see that in all cases 2D-RRMS outperformed the sweeping line algorithm significantly. Indeed, the improvement is even more pronounced when the skyline size becomes larger.

Real datasets: Figures 11 and 12 show the total execution time of the two-dimensional algorithms over two real datasets, namely Airline dataset and NBA dataset (by only considering Air-time and ArrivalDelay attributes), respectively. For the NBA dataset (Figure 11) we varied the dataset size (n) from 5K to 20K. We used the Airline dataset to evaluate the performance of the algorithms over a large real dataset. Figure 12 shows their performance when we varied the dataset size (n) from 1M to 5M. In both experiments, 2D-RRMS algorithm outperforms the Sweeping-Line algorithm by the orders of magnitude. For example, 2D-RRMS algorithm executed in less than 10 seconds in the Airline dataset for n =5M, while Sweeping-Line algorithm took tens of thousand seconds!

6.3 High-dimensional Experimental Result

The performance of the HD algorithms (§ 4) is studied under three correlation models, i.e., correlated, independent, and anti-correlated, on synthetic datasets created based on [2]. The default values for the dataset size, number of attributes, output size, and the control parameter to $n = 10$K, $m = 4$, $r = 5$, and $\gamma = 4$ respectively. We studied the impact of each parameter individually as well. Note that in all HD Figures 13 to 28 the left (blue) y-axis shows the execution time of the algorithms while the right (orange) y-axis shows the regret-ratio (E). The triangle, rectangle, and circle line markers represent GREEDY, HD-GREEDY, and HD-RRMS algorithms respectively, while blue solid lines shows the execution time and dashed orange lines represent the regret-ratio.

Impact of the dataset size (n): In these set of experiments, we varied the size of the dataset (n) from 5K to 10M and evaluated the performance of the three aforementioned algorithms on the synthetic datasets. Figures 13, 14, and 15 show the results for correlated, independent, and anti-correlated datasets respectively. We have compared both the execution time and the regret-ratio of the algorithms to evaluate their performance.

As explained in § 4, the GREEDY algorithm suffers from running n LP optimizations before picking a single tuple, which lead to high execution time. As shown in these figures (13, 14, and 15) the GREEDY algorithm did not scale in any of the experiments, e.g., it required several thousands of seconds for $n = 20$K. On the other hand, the other two algorithms, namely HD-GREEDY and HD-RRMS scaled perfectly in all the experiments as their performances were less dependent on the dataset size. Yet among the two algorithms, HD-RRMS algorithm performs better than HD-GREEDY algorithm, both in time and regret-ratio.

Now let us discuss more about the regret-ratio of the outputs of the algorithms. The fluctuation in the regret-ratio of the output of the GREEDY algorithm in Figure 14 confirms the fact that it cannot guarantee the output quality. Since HD-GREEDY algorithm deals with the discretized subset of function $F \in \mathcal{F}$ and runs the greedy approach on top of it (while GREEDY applies the greedy manner on the whole set of function, $\mathcal{F}$) one expect that the output quality of the GREEDY algorithm should be better than the HD-GREEDY algorithm. However, in Figure 14 and part of the Figure 13, the regret-ratio of the output of the HD-GREEDY algorithm is better than that of the GREEDY algorithm. This is due to the starting point selected by the GREEDY algorithm, where it does not select the first point greedily and simply picks the maximum on the first dimension. Since the latter points are selected based on the initial point, a bad selection of it may highly propagate to the final quality of the output. On the other hand, this problem cannot easily get fixed in the GREEDY algorithm; because in order to construct the LPs, the GREEDY algorithm needs a *non-empty* set of so far selected points – i.e. it cannot start the greedy optimization without pre-selecting at least one point! One way to resolve this, is to run the algorithm with all possible choices of initial point, which multiplies the algorithm complexity with n. Applying the set-cover idea, the output of the HD-RRMS algorithm had less regret-ratio than the

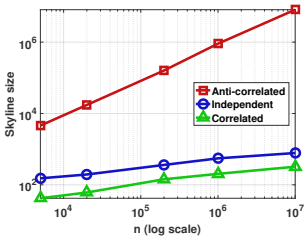

Figure 16: HD, skyline size when varying the dataset size (n)

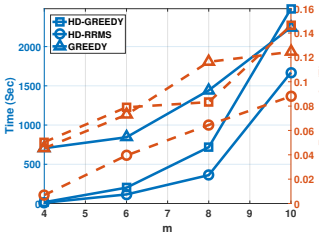

Figure 17: HD, Impact of number of attributes (m) on correlated dataset

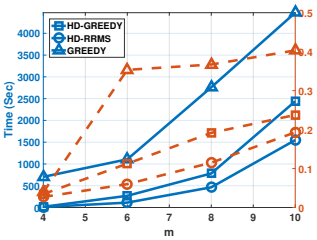

Figure 18: HD, Independent, Impact of number of attributes (m) on independent dataset

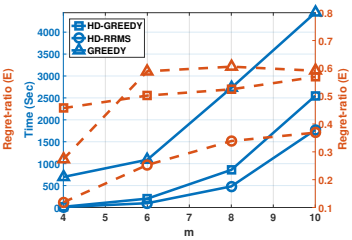

Figure 19: HD, Impact of number of attributes (m) on Anti-correlated dataset

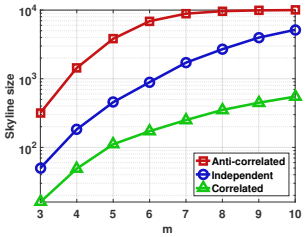

Figure 20: HD, skyline size when varying the number of attributes (m)

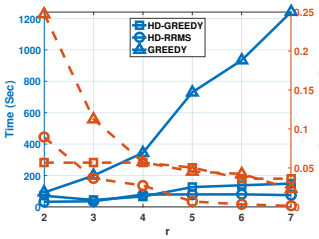

Figure 21: HD, Correlated, Impact of output size (r) on correlated dataset

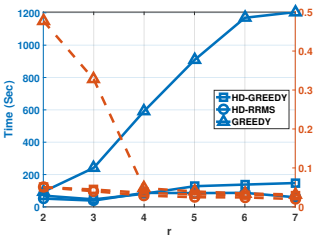

Figure 22: HD, Independent, Impact of output size (r) on independent dataset

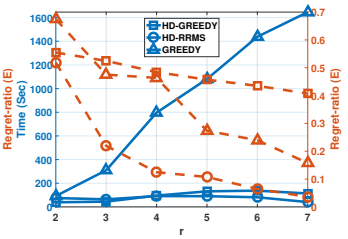

Figure 23: HD, Impact of output size (r) on Anti-correlated dataset

outputs of the both GREEDY and HD-GREEDY algorithms.

Additionally, the skyline size for each of these experiments is reported in Figure 16. As expected, in anti-correlated dataset, most of the tuples are skyline, while the size is less in the two other datasets. Despite of such differences, however, our HD-RRMS algorithm significantly outperforms the competitors in all these cases.

Impact of the number of attributes (m): In these set of experiments, we varied the number of the attributes (m) from 4 to 10 for all the three synthetic datasets, setting the dataset size, and output size to $n = 10$K and $r = 5$ respectively. Figures 17, 18, and 19 show the results for the correlated, independent, and anti-correlated datasets respectively. As expected for all three datasets the HD-RRMS algorithm outperforms the GREEDY and HD-GREEDY algorithms both in execution time and regret-ratio. While the execution time of the HD-GREEDY algorithm is almost similar to the HD-RRMS algorithm, it outperforms the Greedy algorithm. The regret-ratio of the output of the HD-GREEDY algorithm is better in Figure 18 and part of Figure 19, which is, as explained, due to the bad initial point selected by the GREEDY algorithm.

The skyline sizes are reported in Figure 20. After 6 dimensions, almost all the tuples in anti-correlated database and most of the tuples in the independent dataset are skyline. Once again, HD-RRMS algorithm significantly outperforms the competitors even when almost all tuples are on the skyline.

Impact of the output size (r): Figures 21, 22, and 23 show the results for correlated, independent, and anti-correlated datasets, where the output size (r) is varied from 2 to 7, and dataset size and number of attributes are set to $n = 10$K and $m = 4$ respectively. As expected, in these set of experiments, also, HD-RRMS algorithm outperforms the HD-GREEDY and GREEDY algorithms in both time and regret-ratio, while the execution time of the HD-GREEDY algorithm is almost similar. The regret-ratio of the output of the HD-GREEDY algorithm is better than the GREEDY algorithm in Figures 21, 22, and part of 23, due to the bad initial point selection in GREEDY algorithm. One interesting fact in these figures is the bad performance of the GREEDY algorithm for the small output sizes (especially 2 and 3). This is due to the fact that the first point was not selected greedily and the other point(s) was selected with maximum (regret-ratio) distance to it, which (together with the initial point) may not be a good selection. However, as the algorithm picks more points, the effect of the initial point reduces.

Impact of the discretization control parameter (γ): In these set of experiments, we studies the effect of the value of γ in the performance of the HD-RRMS and HD-GREEDY algorithms. Figures 24, 25, and 26 show the performance of the algorithms for correlated, independent, and anti-correlated datasets respectively. While increasing the value of γ does not highly affect the performance of the HD-GREEDY algorithm, it highly affects on the performance of the HD-RRMS algorithm. On the other hand, increasing the value of γ increases the number of columns of the discretized matrix by γ^m and directly affects on the running time of the algorithms. Looking at the figures, at least in these experiments, selecting γ between 4 and 6 seems appropriate, as it seems to reach a point of saturation where increasing γ benefits little in terms of execution time.

Real datasets: We also evaluated the performances of the high-dimensional algorithms over the DOT and NBA real datasets. We set the default values of the number of attributes, number of outputs and discretization control parameter to $m = 4$, $r = 5$, and $\gamma = 6$ respectively. We set the default dataset size of NBA and DOT to $n = 10$K and $n = 100$K respectively.

In the first set of the experiments we varied the dataset size (n) from 100K to 400K for DOT and from 5K to 20K for NBA dataset. Figures 27 and 29 show the performance of the algorithms for the DOT and NBA respectively. Similar to the synthetic experiments, in both experiments the HD-RRMS and HD-GREEDY algorithms run faster in the order of magnitude, while the the GREEDY algorithm does not perform well as the input size (n) increased, e.g., it requires more than $20,000$ seconds for $n = 400$K in DOT. From the regret-ratio point of view the HD-RRMS algorithm outperforms the other two algorithms. The HD-GREEDY algorithm has a better output quality than the GREEDY algorithm for some of the cases.

In the second set of experiments, we varied the number of attributes (m) from 3 to 6. In both experiments, the HD-RRMS algorithm has the best running time, while the HD-GREEDY algorithm performs similarly. While the HD-RRMS algorithm has the

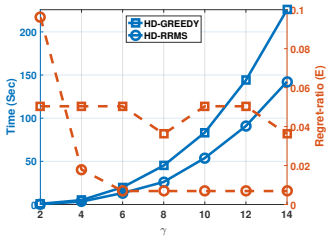

Figure 24: HD, Impact of number of partitions (γ) on correlated dataset

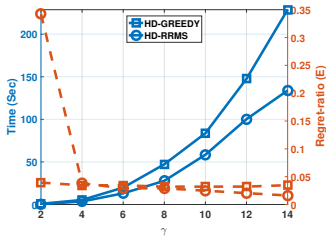

Figure 25: HD, Impact of number of partitions (γ) on independent dataset

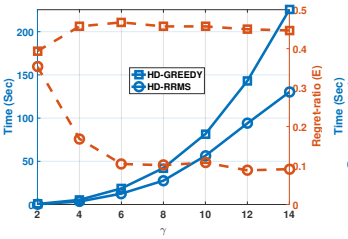

Figure 26: HD, Impact of number of partitions (γ) on Anti-correlated dataset

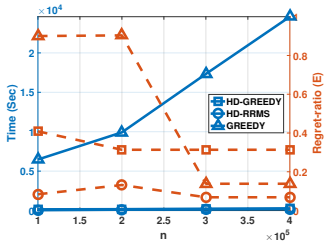

Figure 27: HD, DOT dataset, varied the dataset size (n)

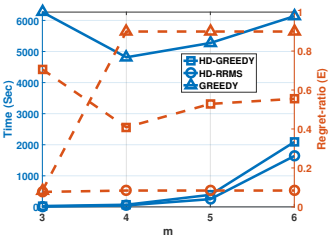

Figure 28: HD, DOT dataset, varied the number of attributes (m)

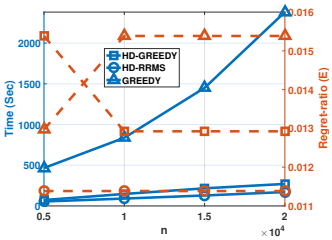

Figure 29: HD, NBA dataset, varied the dataset size (n)

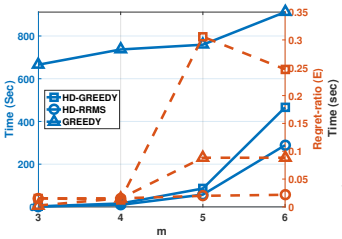

Figure 30: HD, NBA dataset, varied the number of attributes (m)

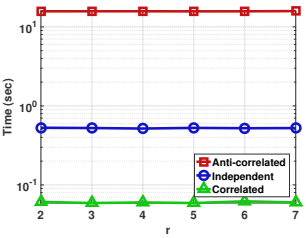

Figure 31: Adopting the k-dominant skyline

best output quality in both experiments, the GREEDY algorithm provides better output quality for NBA and the output of the HD-GREEDY algorithm is better for the DOT dataset.

Adopting the state-of-the-art: In order to test whether one could achieve a reasonable regret-ratio by selecting a subset according to existing techniques that were not designed to optimize the regret-ratio, we implemented a number of existing algorithms for related problems. First, we tested the partitioning-based approximate convex hull method proposed in [1]. As expected, since the goal of this algorithm is efficiently discovering a set which is as similar to the real convex hull as possible, the method is not adoptable here. For example, in these experiments, the size of the discovered set was always larger than the original convex hull, defeating the purpose of generating a more compact representation of the original data.

In another set of experiments, we considered the existing technique of finding the k-dominant skyline [4]. Once again, minimizing the dissatisfaction on maxima query is not its objective - testing it here is solely to demonstrate that it should not be used for the purpose of minimizing regret-ratio. k-dominant skyline is a subset of the skyline when we relax the definition of dominance to k-dominance (i.e., a tuple t k-dominates a tuple t' if there are $k \leq m$ attributes in which t is better than or equal to t' and is better in at least one of these k attributes). k-dominant skyline is the set of tuples are not k-dominated by any other points in the database.

In order to adopt the k-dominant skyline for finding a set of at most r tuples, we do a binary search over the value of k and at each iteration, if the size of the discovered set is larger than r, we increase the value of k and reduce it otherwise. Somewhat surprisingly, we found a major problem with this adaptation - the chance of returning the empty set is high! To understand it, let us consider the case where $m = 2$ and the tuples are distributes inside a circle. The expected number of convex hull points (a subset of skyline) is $O(n^{\frac{1}{3}})$ [13], meaning that having $k = 2$ would often lead to too large an output. However, if we reduce k from 2 to 1, the output size will most likely become zero because, for every skyline point, there is very likely a point that is better on either x or y! Indeed, we ran an experiment on $n = 10000$ $m = 4$ over the three synthetic datasets - in all experiments, the returned set was empty. Figure 31 shows the running time of the algorithm over each of the datasets.

7. RELATED WORK

Top-k discovery algorithms: These algorithms can be divided into on-demand query processing and index construction. On-demand Top-k algorithms focus on the data access methods. For example, *NRA* [11] considers the existence of one sorted list of tuples for each attribute, and finds the Top-k only by exploring the lists, while *TA* [10] applies both random and sorted access. *CA* [10], *Upper/Pick* [3], and [20] are the more advanced algorithms in this category. Focusing on client-server databases, [21] views the existing query engine as the only data access method and designs efficient Top-k discovery algorithms for them. Besides, methods like *PREFER* [15] and *LPTA* [9], employ the materialized views to increase the efficiency of Top-k discovery process. While the first class of Top-k algorithms focuses on efficiently answering the queries on demand, the other set of works aims toward indexing the data beforehand, such that they can answer future queries fast. For example, *ONION* [5] constructs k layers of convex hull that can serve as the representative for linear ranking functions. [29] adds the notion of robustness as the set that performs the best in the worst case scenario. The recent work in this category includes [14,19,28], etc.

Approximate convex hull and skyline reduction algorithms: Given the complexity of convex hull discovery algorithms, especially in high-dimensions, designing effective approximate algorithms, with tight approximate-ratios, for finding the convex hull has attracted many researchers. For example, J. L. Bently et. al. [1] propose a FPTAS ϵ-approximate convex hull algorithm for the two-dimensional algorithms and extend it to high-dimensions. Similarly, [16] partitions the space of skyline tuples into several sub-regions and finds the convex hull in each sub-region, which will result in finding a super-set of convex hull. The objective here is to approximately find a set which is as similar to the original convex hull, not reducing the size of it. On the other hand, a set of work, such as [4,26], aim toward reducing the skyline size. For example, Chan et. al. [4] relax the notion of domination to "$k-$domination" in order to increase the chance domination and reduce the skyline size. However, their objective in ranking the skyline tuples is not minimizing the user dissatisfaction on maxima queries.

Regret-ratio minimizing problem: The authors in [6,22] focused on regret minimization of a database to support multi-criteria de-

cision making. Regret-ratio and Regret-ratio minimizing problem were first introduced in [22] in order to minimize the maximum user dissatisfaction of a Top-k query. They proposed the so called CUBE algorithm to provide an upper-bound guarantee for the regret-ratio of the optimal solution and more importantly they showed it is independent of the input size (n). They also provided the GREEDY heuristic for the problem that runs $O(nr)$ linear programs and picks the points greedily. Chester et.al. [6] extended the regret-ratio notion to k-regret ratio which measures how far from a k^{th} "best" tuple is the "best" tuple in a subset. They propose k-regret minimizing sets problem and proved that it is NP-hard in a high-dimensional database. They also proposed the quadratic Sweeping-Line algorithm for the two-dimensional scenario. Kessler et. al. [17] extended the k-regret minimizing notion to nonlinear functions. The focus of this paper is designing efficient algorithms for the regret-ratio minimizing problem proposed in [22]. In two-dimensional case, compared to quadratic existing Sweeping-Line, we propose the linearithmic dynamic programming algorithm 2D-RRMS, while for the high-dimensional case we provide the linearithmic HD-RRMS that guarantees a user controllable distance from the optimal solution.

8. CONCLUSION

In this paper, we made several fundamental theoretical as well as practical advances in developing a compact maxima representative. We have studied both two-dimensional as well as high-dimensional databases to find a set limited to only r tuples that minimizes the maximum regret-ratio. In the case of two-dimensional databases, we have developed an innovative dynamic programming algorithm to build the optimal index that runs in linearithmic time. We have developed an innovative linearithmic algorithm that guarantees a regret ratio that is within a user-controllable distance from the optimal regret ratio. Our comprehensive set of experiments on synthetic (with different correlation models) and real datasets of size up to several million records confirm the efficiency, scalability, and effectiveness of our algorithms.

9. ACKNOWLEDGMENTS

The work of Abolfazl Asudeh, Azade Nazi, and Gautam Das was supported in part by the National Science Foundation under grant 1343976, the Army Research Office under grant W911NF-15-1-0020, and a grant from Microsoft Research. Nan Zhang was supported in part by the National Science Foundation, including under grants 1343976, 1443858, 1624074, and by the Army Research Office under grant W911NF-15-1-0020. This contribution was made possible in part by NPRP grant #07-794-1-145 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

10. REFERENCES

[1] J. L. Bentley, F. P. Preparata, and M. G. Faust. Approximation algorithms for convex hulls. *Communications of the ACM*, 25(1):64–68, 1982.

[2] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[3] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *TODS*, 2002.

[4] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.

[5] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: indexing for linear optimization queries. In *SIGMOD*, volume 29, 2000.

[6] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. *VLDB*, 7(5), 2014.

[7] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[8] G. B. Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[9] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *VLDB*, 2006.

[10] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *Journal on Discrete Mathematics*, 2003.

[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 2003.

[12] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 1998.

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

[14] J.-S. Heo, J. Cho, and K.-Y. Whang. Subspace top-k query processing using the hybrid-layer index with a tight bound. *Data & Knowledge Engineering*, 2013.

[15] V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *VLDB*, 13(1), 2004.

[16] S.-Y. Ihm, K.-E. Lee, A. Nasridinov, J.-S. Heo, and Y.-H. Park. Approximate convex skyline: a partitioned layer-based index for efficient processing top-k queries. *KBS*, 2014.

[17] T. Kessler Faulkner, W. Brackenbury, and A. Lall. k-regret queries with nonlinear utilities. *VLDB*, 8(13), 2015.

[18] S. G. Kobourov. Force-directed drawing algorithms. 2004.

[19] J. Lee, H. Cho, S. Lee, and S.-w. Hwang. Toward scalable indexing for top-k queries. *TKDE*, 2014.

[20] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2), 2004.

[21] A. Asudeh, N. Zhang, and G. Das. Query reranking as a service. *VLDB*, 9(11), 2016.

[22] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. *VLDB*, 3, 2010.

[23] P. Nobili and A. Sassano. A separation routine for the set covering polytope. In *IPCO*, 1992.

[24] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[25] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc*, 13(3):743–768, 1963.

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

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

[28] A. Asudeh, S. Thirumuruganathan, N. Zhang, and G. Das. Discovering the skyline of web databases. *VLDB*, 2016.

[29] D. Xin, C. Chen, and J. Han. Towards robust indexing for ranked queries. In *VLDB*, 2006.

APPENDIX

A. PROOF OF THEOREM 1

Let T be the set of tuples which are removed from D. The maximum regret-ratio of the optimal solution for the regret-ratio minimizing set problem on $\mathcal{S}$ is the same as the maximum regret-ratio of the optimal solution for the regret-ratio minimizing set problem on D, i.e., $E(T, D) = E(T, \mathcal{S})$.

PROOF. Suppose, $\exists t \in \hat{\mathcal{C}}$ where $t \notin \mathcal{S}$.

$$\Rightarrow \exists t' \in D, \text{s.t. } t' \succ t$$

Since $\forall A \in \mathcal{A}$, $t[A] \geq t'[A]$ and $\exists A \in \mathcal{A}$, $t[A] > t'[A]$, for any ranking function F:

$$F(t) = \sum_{\forall A_i \in \mathcal{A}} w_i \cdot t[A_i] < F(t') = \sum_{\forall A_i \in \mathcal{A}} w_i \cdot t'[A_i]$$

It means by replacing t with t' the size of the set does not change, and the maximum regret-ratio after the replacement is less than or equal to the maximum regret-ratio before the replacement. Therefore, the maximum regret-ratio of the optimal solution for the skyline tuples, $\mathcal{S}$, is the same as the maximum regret-ratio of the optimal solution for all tuples in D. □

B. PROOF OF THEOREM 2

In 2D, after removing the tuples between two skyline tuples t_i and t_j, the maximum regret-ratio happens for the function, F, corresponding with the line between tuples t_i and t_j.

PROOF. Let F be the ranking function specified by the line between tuples t_i and t_j. For a tuple t between t_i and t_j and a ranking function $F' \in \bigcup_{\forall k \in [i,j]} \mathcal{F}_k$ either t_i or t_j is the maxima after removing the tuples $\{t_k | i < k < j\}$. More specifically, if (the angle of) $F' < F$, t_i (and if $F' > F$, t_j) will be the maxima. Let us name the best alternative for F' (either t_i or t_j) as t'. For example in Figure 2, l_2 shows the function F and for any ranking function between l_1 and l_2, t_2 is the best alternative, while the best alternative for the functions between l_2 and l_3 is t_4.

Suppose the lines F in Figure 32 is parallel with the line passing though the points t' and t'' (representing the function for which both t' and t'' have equal scores) and let F' be a function in $\mathcal{F}_t$ for which t' is the best alternative (the same analysis is valid for a F'' for which t'' is the best alternative). Since the line tA is perpendicular to the line $t't''$, $F(t) - F(t')$ is equal to the distance between t and A (shown as $|tA|$). On the other hand, since the lines tB and F' are perpendicular, $F'(t) - F'(t')$ is equal to the distance between t and B (shown as $|tB|$). Looking at the figure, $|tC| < |tA|$. Moreover:

$$|tC| = \frac{tB}{\cos\theta}$$

since $\cos\theta < 1$

$$\Rightarrow F'(t) - F'(t') = |tB| < |tC| < |tA| = F(t) - F(t')$$

Now given that $F(t) - F(t') > F'(t) - F'(t')$, our goal is to prove that:

$$\frac{F(t) - F(t')}{F(t)} > \frac{F'(t) - F'(t')}{F'(t)} \tag{16}$$

If $F(t) \leq F'(t)$, since $F(t) - F(t') > F'(t) - F'(t')$, then Equation 16 holds.
If $F(t) > F'(t)$:

$$F(t) - F(t') > F'(t) - F'(t') \Rightarrow F(t) - F'(t) > F(t') - F'(t')$$

Let $\sigma = F(t') - F'(t')$ and $\delta = F(t) - F'(t) - \sigma$.

$$\Rightarrow F(t) = F'(t) + \sigma + \delta \text{ and } F(t') = \sigma + F'(t')$$
$$\Rightarrow \frac{F(t) - F(t')}{F(t)} = \frac{F'(t) + \sigma + \delta - \sigma - F'(t')}{F'(t) + \sigma + \delta}$$
$$= \frac{F'(t) - F'(t') + \delta}{F'(t) + \sigma + \delta}$$

Since $\sigma \geq 0$ and $\delta \geq 0$,

$$\Rightarrow \frac{F'(t) - F'(t') + \delta}{F'(t) + \sigma + \delta} > \frac{F'(t) - F'(t')}{F'(t)}$$

□

C. PROOF OF THEOREM 3

Time complexity of the algorithm 2 (2D-RRMS) is in $O(rs \log s \log c)$.

PROOF. As shown in Figure 5, DP can get constructed considering a back-track matrix completion approach from level (column) r to level 0, while the values at level i can get computed from the level $i + 1$. The matrix has s rows and r columns. For each cell of the matrix, a binary search with order of $O(\log s)$ is applied for finding the $\min$ value in Equation 5, and at each step of the binary search, if $w(t_i, t_j)$ is unknown, it will call the Algorithm 1 to compute the edge weight which takes $O(c)$. Thus, the overall complexity of 2DEP is $O(rs \log s \log c)$. □

D. PROOF OF THEOREM 4

If a set T of tuples guarantee a regret-ratio threshold of ϵ for all the ranking functions in $f \in F \subseteq \mathcal{F}$, constructed based on the angle partitioning (α) in Equation 6, the maximum regret-ratio of those points for any ranking function $f' \in \mathcal{F}$ is:

$$\epsilon' \leq c\epsilon + (1 - c) \tag{17}$$

where $c = \frac{\cos(\alpha'/2)\cos(\pi/4)}{\cos(\pi/4 - \alpha'/2)}$ and $\alpha' = 2\arcsin(\sqrt{\frac{1 - cos^{m-1}\alpha}{2}})$.

PROOF. We want to see if the regret-ratio of each representative tuple for all ranking function $f \in F$ that are picked is within the ϵ regret-ratio bound, how much is the maximum regret-ratio for a missing ranking function. For the simplicity of the proof, let us start with the two-dimensional case where there are two attributes x and y (Figure 33). Since each missing ranking function is bounded between two selected functions with angle α, its maximum angle with its closest selected ranking function is at most $\frac{\alpha}{2}$ (the worst case happens for the ranking function which is exactly in the middle of two consequent selected functions). Consider a representative tuple for a ranking function $f \in F$ which is in the ϵ range of the top representative of f. Again the worst case happens when the regret-ratio of such tuple is exactly ϵ for the selected ranking function. Suppose the top green line and the red line in Figure 33 shows f and the missing ranking function with angle distance $\frac{\alpha}{2}$ from it respectively. Any tuple above the blue perpendicular dashed line with f is within the ϵ threshold bound for it. As specified in the figure, the intersection of the dashed blue line and the y-axis (tuple t' in the figure) maximizes the distance of the representative of f for f'. Moreover, in order to maximize the distance we put the Top-1 for f to be exactly on it, i.e. tuple t. Thus, the maximum

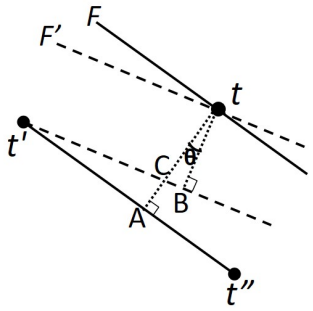

Figure 32: Illustration of the distance of function scores

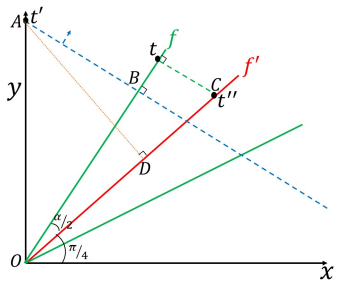

Figure 33: Illustration of maximum growth in 2D.

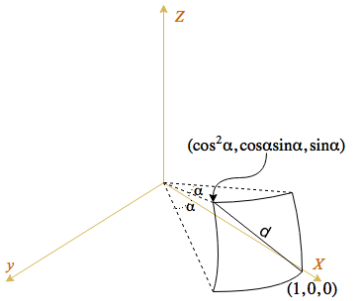

Figure 34: Illustration of the cell diameter in 3D.

regret-ratio of t' for f' is:

$$\epsilon' = \frac{f'(t'') - f'(t')}{f'(t'')} = \frac{OC - OD}{OC}$$

$$OC = \frac{f(t)}{\cos(\alpha/2)}$$

$$OD = OA\cos(\pi/4), OA = \frac{f(t')}{\cos(\pi/4 - \alpha/2)}$$

$$\Rightarrow OD = \frac{f(t')}{\cos(\pi/4 - \alpha/2)}\cos(\pi/4)$$

$$\epsilon' = \frac{\frac{f(t)}{\cos(\alpha/2)} - \frac{f(t')}{\cos(\pi/4-\alpha/2)}\cos(\pi/4)}{\frac{f(t)}{\cos(\alpha/2)}}$$

$$= \frac{f(t) - f(t')\frac{\cos(\alpha/2)\cos(\pi/4)}{\cos(\pi/4-\alpha/2)}}{f(t)}$$

Let $c = \frac{\cos(\alpha/2)\cos(\pi/4)}{\cos(\pi/4 - \alpha/2)}$

$$\Rightarrow \epsilon' = \frac{f(t) - cf(t')}{f(t)} = c\frac{f(t) - f(t')}{f(t)} + (1 - c)$$

Since $0 < c \leq q$ and $\frac{f(t)-f(t')}{f(t)} \leq \epsilon, \Rightarrow \epsilon' \leq c\epsilon + (1 - c) \leq \epsilon + (1 - c)$.

Now let us extend the computation to the high-dimensional case. As shown in Figure 7, the space partitioning can be seen as a set of (hyper-)cones originated at point $(0, \ldots, 0)$. Looking from the surface of the cone, each missing function is covered by a set of selected ranking functions in F which together form a (hyper-)trapezius around it. Consider the hyper-plane constructed between the origin the polar to scalar system).

$$d = \sqrt{(1 - \cos^{m-1}\alpha)^2 + (\cos^{m-2}\alpha\sin\alpha)^2 + \cdots + \sin^2\alpha}$$

$$= \sqrt{(1 - \cos^{m-1}\alpha)^2 + \sin^2\alpha\sum_{i=1}^{m-2}\cos^{2i}\alpha} \quad (18)$$

and the two points in the diameter of hyper-trapezius, in which the distance between the two selected points is maximum. One can see that the maximum growth in the regret-ratio of a point and a missing function happens for the function in the middle of this hyper-plane (that has the maximum angle distance with the selected ranking functions). In the following, looking at Figure 34, we compute the distance d between the two diagonal points, and use it to compute the maximum angle (α') between two adjacent ranking functions. In order to do so, we consider the corners (1,0,...,0), i.e. the point on X-axis and the point $(\cos^{m-1}\alpha,\ \cos^{m-2}\alpha\sin\alpha,\ \cos^{m-3}\alpha\sin\alpha,\ \ldots,\ \sin\alpha)$ (computed by transforming the coordinates from Following the geometric series (while replacing $\sin^2\alpha$ with $1 - \cos^2\alpha$),

$$d = \sqrt{(1 - \cos^{m-1}\alpha)^2 + (1 - \cos^{2(m-1)}\alpha)}$$

$$= \sqrt{2(1 - cos^{m-1}\alpha)} \quad (19)$$

Considering the value of $\frac{d}{2}$, while knowing that the radius of the hyper-sphere is 1,

$$\alpha' = 2\arcsin(\sqrt{\frac{1 - cos^{m-1}\alpha}{2}}) \quad (20)$$

Now using the same analysis we did for the 2D, $\epsilon' \leq c\epsilon + (1 - c) \leq \epsilon + (1 - c)$, where $c = \frac{\cos(\alpha'/2)\cos(\pi/4)}{\cos(\pi/4-\alpha'/2)}$. □