A Unified Optimization Algorithm For Solving “Regret-Minimizing Representative” Problems

Suraj Shetiya, Abolfazl Asudeh, Sadia Ahmed, Gautam Das

University of Texas at Arlington; University of Illinois at Chicago

suraj.shetiya@mavs, sadia.ahmed78@mavs, gdas@cse.uta.edu, asudeh@uic.edu

ABSTRACT

Given a database with numeric attributes, it is often of interest to rank the tuples according to linear scoring functions. For a scoring function and a subset of tuples, the regret of the subset is defined as the (relative) difference in scores between the top-1 tuple of the subset and the top-1 tuple of the entire database. Finding the regret-ratio minimizing set (RRMS), i.e., the subset of a required size that minimizes the maximum regret-ratio across all possible ranking functions, has been a well-studied problem in recent years. This problem is known to be NP-complete and there are several approximation algorithms for it. Other NP-complete variants have also been investigated, e.g., finding the set of size \( k \) that minimizes the average regret ratio over all linear functions. Prior work have designed customized algorithms for different variants of the problem, and are unlikely to easily generalize to other variants.

In this paper we take a different path towards tackling these problems. In contrast to the prior, we propose a unified algorithm for solving different problem variants. Unification is done by localizing the customization to the design of variant-specific subroutines or “oracles” that are called by our algorithm. Our unified algorithm takes inspiration from the seemingly unrelated problem of clustering from data mining, and the corresponding k-MEDOID algorithm. We make several innovative contributions in designing our algorithm, including various techniques such as linear programming, edge sampling in graphs, volume estimation of multi-dimensional convex polytopes, and several others. We provide rigorous theoretical analysis, as well as substantial experimental evaluations over real and synthetic data sets to demonstrate the practical feasibility of our approach.

1. INTRODUCTION

Data-driven decision making is challenging when there are multiple criteria to be considered. Consider a database of \( n \) tuples with \( d \) numeric attributes. In certain cases, “experts” can come up with a (usually linear) function to combine the criteria into a “goodness score” that reflects their preference for the tuples. This function can then be used for ranking and evaluating the tuples [1–4]. However, devising such a function is challenging [5, 6], hence not always a reasonable option, especially for ordinary non-expert users [7]. For instance consider a user who wants to book a hotel in Miami, FL. She wants to find a hotel that is affordable, is close to a beach, and has a good rating. It is not reasonable to expect her to come up with a ranking function, even though she may roughly know what she is looking for. Therefore, she will probably start exploring different options and may end up spending several confusing and frustrating hours before she can finalize her decision. Alternatively, one could remove the set of “dominated” tuples [8], returning a Pareto-optimal [9] (a.k.a. skyline [7, 8]) set, which is the smallest set guaranteed to contain the “best” choice of the user, assuming that her preference is monotonic [8]. In the case where user preferences are further restricted to linear ranking functions, only the convex hull of the dataset needs to be returned.

The problem with the skyline or convex hull is that they can be very large themselves, sometimes being a significant portion of the data [10, 11], hence they lose their appeal as a small representative set for facilitating decision making. Consequently, as outlined in §7, there has been extensive effort to reduce the size of the set. Nanongkai et al. [10] came up with the elegant idea of finding a small set that may not contain the absolute “best” for any possible user (ranking function), but guarantees to contain a satisfactory choice for each possible function. To do so, they defined the notion of “regret-ratio” of a representative subset of the dataset for any given ranking function as follows: it is the relative score difference between the best tuple in the database and the best tuple in the representative set. Given \( k < n \), the task is to find the regret-ratio minimizing set (RRMS), i.e., a subset of size \( k \) that minimizes the average regret-ratio across all possible ranking functions. This problem is shown to be NP-complete, even for a constant (larger than two) number of criteria (attributes) [12]. Other researchers have also considered different versions of the problem formulation. For instance Chester et. al. [13] generalize the notion of regret from the comparison of the the actual top-1 of database to the top-k. More recently, in [14, 15] the goal was to compute the representative set that minimizes the average regret-ratio across all possible functions, instead of minimizing the max regret-ratio. All these variants have been shown to be NP-complete.

Given their intractable nature, there has been significant effort in designing efficient heuristics and approximation algorithms for these problems. The RRMS problem has been investigated in several papers [11,12,16], and several approximation algorithms have been designed; the algorithms in [11,12] run in polynomial time and can approximate the max-regret ratio within any user-specified accuracy threshold. The average regret-ratio problem has been investigated in [14], and a different greedy approach has been pro-
posed, which achieves a constant (not user-specified though) approximation factor with high probability.

1.1 Technical Highlights

In this paper, we make the following observations about the previous works: (a) the proposed algorithms are dependent on the specific problem formulation, and do not seem to generalize to different variants, and (b) the focus has been on designing approximation algorithms, and not on optimal algorithms. We take a different route towards solving these problems, and our work makes two important contributions:

Firstly, we develop a unified algorithm called URM that works across different formulations of the problem, including max [10] and average [14] regret minimizing sets. The unified algorithm makes calls to a subroutine (we refer to it as an “oracle”), and it is this subroutine/oracle that needs to be customized for each problem variant. Thus the customization is localized to the design of the oracle.

Secondly, we make a connection between the various regret minimizing problems and the seemingly unrelated classical data mining problem of clustering and the well-known K-MEDOID [17] algorithm. Most variants of clustering are NP-hard, yet the K-MEDOID algorithm is extremely popular in practice and is based on a hill-climbing approach to find a local optima. One of the main technical highlights of our contributions is to take inspiration from, and design our unified algorithm based on the K-MEDOID algorithm, even though the regret minimization problems seemingly appear quite different from clustering problems. One of the consequences of our approach is that the well-known advantages of the K-MEDOID algorithm transfer over to our unified algorithm. For example, the K-MEDOID algorithm has the any-time property: given more time it can be repeatedly restarted from different random starting configurations, which gives it the ability to improve the local optima that it has discovered thus far. Our unified algorithm also has this property, which can be useful in time-sensitive applications, including a query answering system.

To achieve our two contributions, several novel and challenging technical problems had to be solved. The K-MEDOID algorithm provided inspiration, but was not easily adaptable for our case. Instead, we had to carefully model our problem as that of optimally partitioning the space ranking functions into $k$ convex geometric regions, such that for each region exactly one tuple from the database is the “representative”, i.e., it has the highest (max, or average, depending on the problem variant being solved) score for any function in that region. At a high level, our algorithm first chooses $k$ tuples randomly as our initial representative set. Then it only examines these $k$ tuples, and partitions the function space into $k$ convex regions such that the tuple associated with each region outscores the remaining $k-1$ tuples for any ranking function within its region. Then, the database is examined to update the best representative for each region, and the process iterates until a local optima is reached.

The process of examining the database for updating the best representative for a region required us to develop variant-specific oracles. For the case of the max regret-ratio, we propose different innovative strategies for designing an efficient oracle. For large regions, we design a threshold-based algorithm based on function-space discretization. For narrow regions, we model the problem as an instance of edge sampling from a weighted graph where edge weights are determined by solving (constant-sized) linear programs.

For the average regret-ratio case, designing an oracle was challenging. As another of our innovative technical highlights, we show that it is reducible to the classical problem of computing the volume of polytopes defined by the intersection of $O(n)$ halfspaces. Unfortunately, this approach has a time complexity of $\Omega(n^d)$ [18]. Even if we assume that the dimension of the database is fixed, this high complexity makes this approach only of theoretical interest. Consequently, we propose an alternative approach based on Monte-Carlo sampling of the function space.

We provide proof of convergence of our unified algorithm, as well as provide detailed theoretical analyses of our oracles. We also conduct extensive empirical experiments on both real and synthesis datasets that show the efficiency and effectiveness of our proposal.

2. PRELIMINARIES

Data Model: We consider a database $D$ in the form of $n$ tuples $t_1$ to $t_n$, defined over $d$ numeric attributes $A_1$ to $A_d$. We use the notation $t_i[j]$ to show the value of $t_i$ on attribute $A_j$. Without loss of generality, we assume that attribute values are normalized and standardized as the non-negative real numbers, $\mathbb{R}^+$. The numeric attributes are used for scoring and ranking the tuples. Additionally, the dataset may also include non-ordinal attributes that may be used in filtering, but not in ranking. Finally, for each attribute $A_i$, we assume that the larger attribute values are preferred. The values $x$ of an attribute $A_i$ with smaller-preferred nature require a straightforward transformation such as $(\max(A_i) - x) / (\max(A_i) - \min(A_i))$.

**Example 1.** Consider a dataset of 10 tuples, defined over the attributes $A_1$ to $A_3$, as shown below. The values of the attributes are normalized in range [0,100] and for all attributes the higher values are preferred. In this example $t_1[2]$ refers to the value of tuple $t_1$ on attribute $d_2$ which is equal to 75.

<table>
<thead>
<tr>
<th>tuple</th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>60</td>
<td>80</td>
<td>79</td>
</tr>
<tr>
<td>$t_2$</td>
<td>55</td>
<td>75</td>
<td>63</td>
</tr>
<tr>
<td>$t_3$</td>
<td>75</td>
<td>60</td>
<td>59</td>
</tr>
<tr>
<td>$t_4$</td>
<td>68</td>
<td>70</td>
<td>75</td>
</tr>
<tr>
<td>$t_5$</td>
<td>80</td>
<td>75</td>
<td>73</td>
</tr>
</tbody>
</table>

Ranking Model: The database tuples are ranked using the scores assigned by a ranking function. For every tuple $t \in D$, a ranking function $f: \mathbb{R}^d \rightarrow \mathbb{R}^+$ assigns a non-negative score to $t$. A tuple $t_i$ outranks $t_j$ based on $f$ if its score is larger than the one of $t_j$. Following the literature in regret-minimizing context [10–13], we consider the class of linear ranking functions in this paper.

The score of a tuple $t$ based on a linear function $f$ with a weight vector $\vec{w} = \{w_1, w_2, \ldots, w_d\}$ is computed as:

$$f_{\vec{w}}(t) = \sum_{i=1}^{d} w_i \cdot t[i]$$

In the rest of paper, we simplify $f_{\vec{w}}(t)$ to $f(\vec{w}, t)$ when $\vec{w}$ is clear in the context. As an example, let us consider Example 1, while choosing $\vec{w} = \{25, 5, 25\}$ for the ranking function. Using this function, $f(t_1) = 25 \times 60 + 5 \times 80 + 25 \times 77 = 74.25$. Computing the score of other tuples, the ordering of the tuples in $D$ based this function is $\{t_5, t_1, t_7, t_9, t_10, t_4, t_6, t_2, t_8, t_3\}$.

In a $d$-dimensional space every tuple is presented as a point, and every linear function can be modeled as a origin-starting ray that passes through the point specified by its weight vector. For example, the function $f$ with the weight vector $\vec{w} = \{25, 5, 25\}$ is modeled as the ray that starts from the origin and passes through the point $(25, 5, 25)$. The ordering of tuples based on $f$ is specified by the ordering their projections on the ray of $f$ (please refer to [5] for further details). As a result, the universe of origin-starting rays in the first quadrant of the $d$ dimensional space forms the universe of linear functions. We call this the function space.

\[\text{Note that a large class of non-linear functions can fit this model after a straightforward linearization [19].}\]
A tuple $t_i$ of a database is said to dominate tuple $t_j$ if each of the attribute values for $t_i$ is not smaller than that of $t_j$'s while there exists an attribute $A_k$ where $t_i[k] > t_j[k]$. For instance, in Example 1, $t_1 : (60, 80, 77)$ dominates $t_2 : (55, 75, 63)$. The set of tuples from the database which are not dominated tuples in the database is known as the skyline (or Pareto-optimal) [7-9].

A maxima representative, or simply a representative, is a subset $S \subseteq D$ that is used for finding the maximum of $D$ for any arbitrary ranking function $f$. Skysky is the minimal representative of $D$ that guarantees the containment of the maximum of any function in the class of monotonic ranking functions. That is the reason it is popular for multi-criteria decision making (in the absence of a ranking function). Similarly, convex-hull is the minimal representative that contains the maximum for the class of linear ranking functions.

However, the skyline or convex-hull may contain a large portion of the database, which diminishes their applicability as a small set for decision making [10,11]. For instance, the skyline of Example 1 is \{t_1, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}, which includes 80% of the tuples.

### 2.1 Regret optimization measures

Regret ratio is a measure of dissatisfaction of a user when she sees results returned from the subset instead of the entire database. Let $f$ be a ranking function, then regret ratio is defined as the ratio of the difference in ranking function scores between the top database tuple and the top tuple from the representative set to the score of the top database tuple:

**Definition 1 (Regret Ratio).** Given a function $f$, let tuple $t$ be argmax$_{t \in D} f(t)$. The regret ratio of set $S \subseteq D$ for a ranking function $f$ can be computed as

$$ rr(f, S) = \min_{t' \in S} \frac{f(t) - f(t')} {f(t)} \quad (2) $$

For instance, in Example 1, consider the set $S = \{t_2, t_3, t_4\}$ and the function $f$ with the weight vector $\overrightarrow{w} = (0.25, 0.25, 0.25)$. $t_3$ is the max for this function ($f(t_3) = 75.75$) while $t_4$ is the tuple with the max score ($f(t_4) = 70.75$) in $S$. Therefore, $rr(f, S)$ in this example is $75.75 - 70.75)/75.75 ≈ 0.066$.

For a universe of ranking functions, an aggregate over the regret ratio of each ranking function is considered as the regret ratio of the representative set for the universe.

In this paper, we provide a unified model that can handle a wide variety of aggregates (any $\ell^p$ norm):

**Definition 2. $\ell^0$ norm regret measure:** Given a database $D$, a representative $S \subseteq D$, a real number $p \geq 1$, and a set of ranking functions $F$, the regret-ratio for the $\ell^0$ norm is defined as:

$$ RR_F(S) = \text{Agg}_{f \in F} rr(f, S) $$

where Agg$^p$ is the $\ell^p$ norm aggregate measure.

While different aggregates can be used here, so far the literature has considered maximum and average regret ratio minimizing sets. Without limiting our proposal to a specific value of norm, we especially show the adaptation of our framework for the existing aggregates, i.e., (i) $\ell_{\infty}$: maximum regret ratio and (ii) $\ell_1$: average regret ratio. We will lay out the formal definitions of these measures in the remainder of this section.

**Definition 3. Maximum ($\ell^\infty$ norm) Regret Ratio:** Given a database $D$ and a set of functions $F$, the maximum regret ratio [10] of a set $S \subseteq D$ is the maximum value of regret ratio for the set $S$ over the set of all possible ranking functions $F$. That is,

$$ RR_F(S) = \sup_{f \in F} rr(f, S) \quad (3) $$

While the maximum regret ratio looks at the worst case regret ratio for a set of functions, a different measure for user dissatisfaction is the average regret ratio.

**Definition 4. Average ($\ell^1$ norm) Regret Ratio:**

Given a database $D$, a set of ranking functions $F$, and a probability distribution $\eta(.)$ where $\eta(f)$ is the probability of each function $f \in F$, the average regret ratio [14] of a set $S \subseteq D$ is defined as

$$ ARR_F(S) = \int_{f \in F} \eta(f) \times rr(f, S) df \quad (4) $$

Even though the regret ratio notions are defined for general classes of functions, the majority of the existing work consider $F$ as the class of linear ranking functions (Eq. 1) [10–14, 20]. Also, for average regret ratio, the uniform distribution is considered as the probability distribution of ranking functions $\eta(.)$ [14]. We follow the literature on these. In the rest of the paper, we simplify the notations $RR_F(S)$ and $ARR_F(S)$ to $RR(S)$ and $ARR(S)$ for the class of linear ranking functions ($\mathcal{L}$).

### 2.2 Problem Definition

In this paper, we consider the problem of finding a compact representative of size $k$ from a database such that the regret optimization measure is minimized. Formally:

**$\ell^p$ Regret Ratio Representative Problem:**

Given a database $D$, a set of ranking functions $F$, and a value $k$, find the representative $S \subseteq D$ for $F$ such that $|S| = k$ and $\ell^p$ norm regret measure (Definition 2) of $S$ is minimized.

In particular, the problem for the max. (resp. avg.) regret ratio is to find a set $S \subseteq D$ of size $k$ that minimizes the max. (resp. avg.) regret ratio. For any constant number of dimensions larger than 2, the maximum regret ratio problem is NP-complete [12]. Similarly, the problem for average regret ratio is proven to be NP-complete [14].

The two problems we consider in this paper are the max and average regret ratio optimizing set problems. In section 3, we describe the Regret Minimizing Framework algorithm, a general framework to solve the regret ratio class of problems, followed by sections 4 and 5 which discuss the details of the oracles for the Maximum and Average regret ratio problems respectively. We show empirical results for both these problems in section 6.

### 3. UNIFIED REGRET MINIMIZATION

#### 3.1 Overview

In this section, we propose our Unified Regret Minimizer (URM) algorithm which is inspired by the $k$-MEDOID algorithm. Before providing the details of the algorithm, it is necessary to explain two central ideas in this paper: (i) representative and (ii) region of a tuple $t$, defined in definition 5 and definition 6 respectively.

**Definition 5.** Given a function $f$ and a set of tuples $S \subseteq D$, a tuple $t \in S$ is a representative for $f$ if it has the maximum score, based on $f$ among elements of $S$. Formally:

$$ \rho(f, S) = \text{argmax}_{t \in S} f(t) $$

**Definition 6 (Region of the Tuple $t$).** Given a set $S$ and a tuple $t \in S$, the region of $t$ is the set of functions for which $t$ is the representative. That is,

$$ R_t(S) = \{ f \in \mathcal{L} | \rho(f, S) = t \} $$


For example, Fig. 1 shows the regions of the tuples \( \{t_1, t_5, t_7, t_9\} \) in Example 1.

Having the necessary definitions in place, we now provide a brief overview of the K-MEDOID and then show the transformation of our problem to it. Recall that given a database \( \mathcal{D} \), the output size \( k \) and a regret measure, the URM algorithm finds a set of \( k \) representative tuples from \( \mathcal{D} \), such that the regret measure is minimized.

K-MEDOID algorithm: An iterative algorithm that partitions a set of \( n \) objects into \( k \) clusters, such that the distance between objects belonging to a cluster and their cluster center are minimized. While many clustering algorithms (such as KNN [21]) have cluster centers that may not belong to the set of \( n \) objects, the K-MEDOID algorithm sets itself apart by choosing the cluster centers from the set of \( n \) objects. More details about the algorithm is provided in [17].

Although the clustering problem and the K-MEDOID algorithm seem different from our regret minimization problems, the notion of region of a tuple, provided in Definition 6 is the key in the problem transformation. As we shall show in the following, we see the problem as a partitioning of the function space into \( k \) convex regions while the tuples are the centroids.

Before providing further technical details in § 3.2, we would like to highlight a key difference between the nature of the problems: While the K-MEDOID algorithm deals with a countable set of objects \( n \), the URM algorithm deals with an infinite set of objects (functions). The K-MEDOID algorithm clusters \( n \) objects \( (n \) being a discrete and finite number) into \( k \) clusters, each of which contains finite number of objects. In contrast, as we shall elaborate next, the URM algorithm clusters the continuous space of ranking functions, which contains an infinite number of functions into \( k \) convex regions of ranking functions.

The URM algorithm forms the basis for a unified framework to find compact representatives for a variety of regret measures. It does this by abstracting the various notions of regret measures into an oracle. At a high level, URM operates as follows. It is initialized with a set of \( k \) tuples, which form the compact representatives for the first iteration of the algorithm. These \( k \) tuples are used to partition the entire function space into \( k \) convex regions, such that for each region the corresponding tuple is the representative. In each iteration, we replace the \( k \) tuples with a potentially new set of \( k \) tuples that improves the regret measure. This is done by choosing, for each convex region of functions, the tuple from the database \( \mathcal{D} \) that is the best representative for that region (this is accomplished by making calls to the oracle). The function space is repartitioned into \( k \) new convex regions, and the iterations continue until we converge to a local optima (i.e., the \( k \) tuples do not change).

### 3.2 Algorithm development

The key in the design of URM is that a set of \( k \) representative tuples partition the function space into \( k \) convex regions, such that each of the tuples \( t_i \) is the “representative” for all functions of its region \( R_i \). This enables adopting the K-MEDOID technique for clustering the function space.

**Theorem 1.** Consider the set \( S = \{t_1, \ldots, t_k\} \) as the compact representative of database \( \mathcal{D} \) and the region of functions that are represented by \( t_i \) be \( R_i \). Then the followings hold:

1. The regions \( R_1, \ldots, R_k \) partition the function space.
2. For each tuple \( t_i \), \( R_i \) is convex.
3. Each region \( R_i \) is the intersection of \( (k-1) \) half spaces.

**Proof.** Consider an arbitrary ranking function \( f \). From Eq. 2, the regret ratio \( r = r(f, S) \) is the regret ratio of the set \( S \) for the function \( f \). The tuple from the set \( S \) which achieves this regret ratio is \( t = \operatorname{argmin}_{i \in S} (f(t) - f(t_i))/f(t) \).

This tuple \( t \in S \) is the representative for the function \( f \). The region \( R_t \) consists of all the functions that have \( t_i \) as the representative. As the above property is applicable to the set of all ranking functions, regions \( R_t, \ldots, R_k \) partition the function space into \( k \) regions. This proves property 1.

The region of functions for which the tuple \( t_i \) is ranked higher than tuple \( t_j \) is a half-space denoted by \( H_{ij}^+ \), which is defined by the inequality:

\[
\sum_{i=1}^{k} w_j \cdot (f(t_i) - t_j) \geq 0
\]

The hyper-plane represented by the left-hand side of Eq. 5 is denoted as ordering exchange hyper-plane as it divides the space into two regions, one in which tuple \( t_i \) is better than \( t_j \) and vice versa in the other. As Eq. 5 represents a half-space and half-spaces are convex, \( H_{ij}^+ \) is a convex region. Similarly, let \( H_{ij}^- \) be the region of functions where \( t_i \) is better than \( t_j \). As \( R_i \) is the region of functions where \( t_i \) is the representative, we need to consider the function space where \( t_i \) is better than all the other tuples, which is given by:

\[
R_i = \bigcap_{j=1}^{k} H_{ij}^+
\]

This proves property 3.

An interesting observation from Theorem 1 is that each region is described as a list of \( (k-1) \) half-spaces (equations), which is independent of the number of dimensions.

As an example let us consider Fig. 1, which shows the regions for the tuples \( \{t_1, t_5, t_7, t_9\} \) in Example 1. Table 1 shows the three half-spaces that define each of the regions. For instance, the region of \( t_1 \) is described by the half-spaces between \( t_1 \) and \( t_5 \), \( t_7 \), and \( t_9 \). Each of these regions are drawn in Fig. 1. One can verify that these regions are convex and partition the function space.

**Theorem 1** shows that the regions of a set of tuples \( S = \{t_1, \ldots, t_k\} \) partition the function space into \( k \) non-overlapping convex clusters. Now we show how this enables adopting the K-MEDOID algorithm. Consider a database \( \mathcal{D} \) and an initial set \( S = \{t_1, \ldots, t_k\} \) of \( k \) representative tuples in the first iteration of the algorithm. Let the region of functions \( R_i \) for tuple \( t_i \) be represented as the intersection of \( (k-1) \) half-spaces. Based on Definition 6, \( t_i \) is preferred over all tuples \( t_j \in S\setminus\{t_i\} \) for any function \( f \in R_i \). But it does not necessarily mean that \( t_i \) is preferred over all tuples \( t_j \in \mathcal{D}\setminus\{t_i\} \) for each \( t_i \) there may be another tuple in the dataset that can introduce a smaller regret compared to \( t_i \) for \( R_i \). This is a key observation in designing the subsequent iterations of the algorithm. To do so, we rely on the existence of the “regret optimization oracle”, that finds
Algorithm 1 URM Algorithm

Input: D, Regret Oracle Orc, Initial set of k tuples initial S
Output: Representative S
1: S ← initial
2: repeat
3: S′ ← new List
4: for tuple t in S do
5: \( R_t \leftarrow \text{RegionOf}(t, S \setminus \{t\})^2 \)
6: \( t' \leftarrow \text{Orc}(D, R_t) \) // the best tuple for \( R_t \)
7: Add \( t' \) to S′
8: end for
9: S ← S′
10: until Convergence
11: return S

the best tuple in the database for \( R_t \). This oracle is dependent on the variant of the regret measure we are seeking to optimize.

Regret optimization oracle: The regret optimization oracle is the variant-specific part of our overall approach. It is used to find a representative tuple from the database which best optimizes a specific regret measure for a convex region of ranking functions \( R_t \) (e.g., max regret, average regret, more general \( \ell_p \)-norm regret, etc). Formally, given a database \( D \), a particular variant of regret measure of interest, and a convex region of functions \( R_t \) defined by the intersection of half-spaces, the regret optimization oracle finds the tuple (and the corresponding regret measure) from the database which minimizes the regret measure over all functions in \( R_t \).

For now, we assume that such a variant-specific oracle exists, and we proceed with describing how our unified algorithm URM can leverage such an oracle (details about the design of the oracle for different problem variants are deferred to §4 and §5). At every iteration of the algorithm, for every region of functions \( R_t \), the regret optimization oracle is used to find the new representative for the region \( R_t \). The set of \( k \) tuples obtained from \( k \) calls to the regret optimization oracle form the new set of representatives for the next iteration of the algorithm. The iterations continue until the representative set of tuples ceases to change.

The pseudo code of the URM algorithm is given in Algorithm 1. Also, Algorithm 2 shows the pseudo code for finding the region \( R_t \) for a tuple \( t_i \) in the form of the intersection of \((k-1)\) half-spaces.

Algorithm 2 RegionOf

Input: A set of \( k \) tuples \( S \) and a tuple \( t \in S \)
Output: \( R_t \) in the form of intersection of \((k-1)\) half-spaces
1: \( R_t \leftarrow \) new set
2: for tuple \( t_j \) in \( S \setminus \{t\} \) do
   // \( H_j \) is defined as as \( \sum_{i=1}^{d} H_j[i] \times w_i \geq 0 \)
3: \( H_j \leftarrow \) new list of size \( d \)
4: for \( i \leftarrow 1 \) to \( d \) do \( H_j[i] = t[i] - t_j[i] \)
5: Add \( H_j \) to \( R_t \)
6: end for
7: return \( R_t \)

\footnote{Note that the region of a tuple can be empty unless it belongs to the convex hull. In case of a region of a tuple being empty we add a random tuple in its place.}

Compared to the existing literature for regret ratio minimizing problem, the URM algorithm has some unique and important features. URM provides a unified framework for the \( \ell^n \) and \( \ell^\infty \) classes of regret ratio measures. Our iterative algorithm has the any-time property where the user may stop at any time/iteration and still find a set of compact representatives. The any-time property can be very useful in case of real time query answering systems and other time-sensitive applications.

3.3 Proof of convergence for \( \ell^n \) and \( \ell^\infty \) norms

A critical requirement for the iterative algorithm is the convergence guarantee. Here, we prove that our algorithm converges to a local optima, for the \( \ell^n \) and \( \ell^\infty \) norm class of regret ratio measures.

Theorem 2. In each successive iteration of the URM algorithm, the regret measure for the set \( S : \{t_1, \cdots, t_k\} \) improves for the \( \ell^n \) norm regret measure.

Proof. Let us consider a set \( S : \{t_1, \cdots, t_k\} \) as the compact representatives before the iteration and let set \( S' : \{t'_1, \cdots, t'_k\} \) be the compact representatives after the iteration. The \( \ell^n \) norm for the set \( S \) can be described by

\[
RR = \sqrt{\sum_{i=1}^{k} \int_{f \in R_t} (rr(f, S))^n} = \sqrt{\sum_{i=1}^{k} \int_{f \in R_t} (rr(f, t'_i))^n}
\]

During the iteration, for every region \( R_t \) the regret minimization oracle finds a tuple from the database \( D \) that minimizes the \( \ell^n \) norm regret measure. Hence, we know that,

\[
\forall i \in \{1, \cdots, k\}, \int_{f \in R_t} (rr(f, S))^n \geq \int_{f \in R'_t} (rr(f, t'_i))^n
\]

As the equation for \( RR' \) can be written as,

\[
RR' = \sqrt{\sum_{i=1}^{k} \int_{f \in R_t} (rr(f, t'_i))^n}
\]

Using Eq. 7 in Eq. 8, we get

\[
\sqrt{\sum_{i=1}^{k} \int_{f \in R_t} (rr(f, t'_i))^n} \geq \sqrt{\sum_{i=1}^{k} \int_{f \in R_t} (rr(f, t'_i))^n}
\]

As the region of \( t_i \) differs from that of \( t'_i \), we can conclude

\[
rr(f, t'_i) \geq (rr(f, S')) \Rightarrow RR \geq RR'
\]

\( \square \)

A similar proof for \( \ell^\infty \) norm can be proved.

Theorem 3. In each successive iteration of the URM algorithm, the regret measure for the set \( S : \{t_1, \cdots, t_k\} \) improves for the \( \ell^\infty \) norm regret measure(maximum regret ratio).

Proof. Let us consider a set \( S : \{t_1, \cdots, t_k\} \) as the compact representatives before the iteration and let set \( S' : \{t'_1, \cdots, t'_k\} \) be

Table 1: Table containing the inequalities that define each region of the representative \( \{t_1, t_5, t_7, t_9\} \).

<table>
<thead>
<tr>
<th>Region for ( t_1 )</th>
<th>( t_1 - t_5: -20A_1 + 5A_2 + 4A_3 \geq 0 )</th>
<th>( t_1 - t_7: -20A_1 + 2A_2 + 3A_3 \geq 0 )</th>
<th>( t_1 - t_9: -26A_1 + 12A_2 + 3A_3 \geq 0 )</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region for ( t_2 )</td>
<td>( t_2 - t_5: -20A_1 + 5A_2 - 4A_3 \geq 0 )</td>
<td>( t_2 - t_9: -6A_1 + 7A_2 - 3A_3 \geq 0 )</td>
<td>( t_2 - t_9: 25A_1 - 10A_2 - 6A_3 \geq 0 )</td>
<td>Green</td>
</tr>
<tr>
<td>Region for ( t_3 )</td>
<td>( t_3 - t_5: A_1 - 2A_2 + 3A_3 \geq 0 )</td>
<td>( t_3 - t_9: -19A_1 + 3A_2 + 7A_3 \geq 0 )</td>
<td>( t_3 - t_9: 25A_1 - 10A_2 - 6A_3 \geq 0 )</td>
<td>Blue</td>
</tr>
<tr>
<td>Region for ( t_4 )</td>
<td>( t_4 - t_5: 26A_1 - 12A_2 - 3A_3 \geq 0 )</td>
<td>( t_4 - t_9: 6A_1 - 7A_2 + 3A_3 \geq 0 )</td>
<td>( t_4 - t_9: 25A_1 - 10A_2 - 6A_3 \geq 0 )</td>
<td>Orange</td>
</tr>
</tbody>
</table>

\( 1 \leq i \leq \alpha \)

\[ \sum_{i=1}^{\alpha} (rr(f, S))^n = \sum_{i=1}^{\alpha} (rr(f, t'_i))^n \]

\[ RR = \sqrt{\sum_{i=1}^{\alpha} (rr(f, S))^n} = \sqrt{\sum_{i=1}^{\alpha} (rr(f, t'_i))^n} \]

\[ RR' = \sqrt{\sum_{i=1}^{\alpha} (rr(f, t'_i))^n} \]

\[ rr(f, t'_i) \geq (rr(f, S')) \Rightarrow RR \geq RR' \]

\( \square \)

A similar proof for \( \ell^\infty \) norm can be proved.
the compact representatives after the iteration. The \( \ell^\infty \) norm regret measure for the set \( S \) can be described by

\[
RR = \max_{i=1}^{k} \sup_{f \in R_i} rr(f, t_i)
\]

During the iteration, for every region \( R_i \) the regret minimization oracle finds a tuple from the database \( D \) that minimizes the \( \ell^\infty \) norm regret measure. Hence, we know that

\[
\forall i \in \{1, \ldots, k\} \sup_{f \in R_i} rr(f, t_i) \geq \sup_{f \in R_i} rr(f, t'_i)
\]

Using equations 9 and 10 we get,

\[
\max_{i=1}^{k} \sup_{f \in R_i} rr(f, t_i) \geq \max_{i=1}^{k} \sup_{f \in R_i} rr(f, t'_i) \geq \sup_{f \in F} rr(f, S')
\]

\[
RR \geq RR'
\]

\( \square \)

3.4 Running Example

To provide a better understanding of the algorithm, in this section we provide a run of the algorithm over Example 1, while considering max as the target regret ratio measure. Let the set \( S = \{t_1, t_2, t_3, t_4\} \) be the initial representative for the algorithm. The region of each tuple is highlighted in Table 1. Based on Definition 6, each tuple \( t_i \) is in charge of its own region \( R_i \), highlighted in Fig. 1, i.e., \( t_5 \) is the best tuple in \( S \) for all the ranking functions lying inside \( R_i \). In the next iteration the URM algorithm goes through each of these regions to find better representatives. To do this, it calls the oracle \( MaxO \). We shall provide the details of this oracle in § 4. For each region \( R_i \), the oracle finds the best tuple in the entire dataset that has the minimum value of max regret ratio the functions in \( R_i \). After the first iteration, the representative changes to \( \{t_1, t_2, t_3, t_5\} \). This new representative creates a different partitioning of the function space, shown in Fig. 2. We report the max regret ratio scores of this iteration in Table 2. As we can see, the new representative has a lower score of max regret ratio, which is due to the convergence property of the algorithm, discussed in § 3.3. While in this example, only one of the tuples changed during the iteration, in practice more than one tuple may change during an iteration. The algorithm then calls the oracle \( MaxO \) to find the best tuples for the new regions and continues until convergence.

4. MAX REGRET RATIO ORACLE

So far in this paper, we discussed the regret-minimizing problem in general, assuming the existence of an oracle for computing the regret. In this section we focus on the original (and dominant) measure of regret-ratio: max regret-ratio [10]. Given a database \( D \) and a region of ranking functions \( R \), the oracle \( MaxRROR \) finds the tuple that has the least maximum regret ratio score for all functions in \( R \). The region of functions \( R \) is formulated as the intersection of half spaces, \( R = \{H_1, \ldots, H_{k-1}\} \). The objective is to design an efficient oracle for this case.

We first model the problem into a weighted directed complete graph, and use it for calculating the max regret ratio. Then, we propose three optimization techniques to make the oracle efficient and scalable.

4.1 Graph Transformation

In order to find the tuple with the least max regret ratio in a region \( R \), we transform the problem into a graph exploration instance. Consider a weighted directed complete graph \( G \), as illustrated in Fig. 3, with \( n \) nodes and \( n(n-1) \) edges such that:

- Every tuple \( t \in D \) translates to the node \( t \) in \( G \).
- The weight \( w_{t \rightarrow t'} \) of an edge \( t \rightarrow t' \) (from node \( t \) to node \( t' \)) is equal to the max regret ratio of replacing \( t' \) with \( t \) in the region \( R \).

In order to compute the weight of an edge \( t \rightarrow t' \) we use the (fixed size – independent of \( n \)) linear programming (LP) shown in Eq. 11.

Having the graph \( G \) constructed, the max regret ratio of assigning a tuple \( t \) as the representative (removing all other tuples \( t' \in D \setminus \{t\} \)) is the maximum weight of its outgoing edges. This is proved in Theorem 4.

**Theorem 4.** Given the graph \( G \) of a database \( D \), a space of ranking functions \( R \) the max regret ratio of a tuple \( t \in D \) in \( R \) is:

\[
RR(t, D, R) = \max_{t' \in D \setminus \{t\}} w_{t \rightarrow t'}
\]

**Proof.** Let \( f' \in R \) be the ranking function for which the tuple \( t \) has the maximum regret ratio score. Let \( r_{f'} \) be the maximum regret ratio score and tuple \( t_{f'} \) be the tuple which has the maximum score from the database \( D \) for the function \( f' \). By definition, the equation for maximum regret ratio \( r_{f'} \) is

\[
r_{f'} = \frac{f'(t_{f'}) - f'(t)}{f'(t_{f'})}
\]

An important observation that we will use in this proof is that the equation for \( r_{f'} \) is dependent only on the tuple \( t_{f'} \) from the database. The computation of edge weight \( w_{t \rightarrow t'} \) can be formulated as:

\[
w_{t \rightarrow t'} = \sup_{f \in R} \frac{f(t') - f(t)}{f(t_{f'})}
\]

As we compute the edge weights between tuple \( t \) and all the other tuples from the database \( D \), the maximum value is computed when the comparison between tuples \( t \) and \( t_{f'} \). Hence, \( RR(t, D, R) \) is equal to \( r_{f'} \). \( \square \)

The representative tuple for a region of functions \( R \) is the node that has the minimum max-weight over its outgoing edges. Therefore, after constructing the graph \( G \), the oracle can make a pass over the graph and find the representative node to assign to the region.

Given that the LPs have a constant size, the construction of the graph \( G \) and finding the representative tuple for a region \( R \) based on it has the time complexity of \( O(n^2) \).

Even though the construction of graph \( G \) enables a polynomial algorithm for the max regret ratio oracle, it is still a quadratic algorithm which is not efficient and scalable in practice. Therefore, in the rest of this section we propose different approaches for making the oracle more efficient.

The idea is to find the representative tuple of the region without the complete construction of \( G \). For instance, following the convergence property of the URM algorithm, we know that, at every iteration, the max regret ratio of the representative of a region \( R \) is not more than the one for the representative from the previous iteration. We can exploit this property by using the regret ratio value of the representative for the region \( R \) as a threshold during the regret ratio value computation for the region \( R_k \). That is, for any node in graph \( G \), we ignore computing the weights of its outgoing edges, as soon as we find a edge that is not smaller than the threshold for this region. Following the idea of not computing the weights of all edges in \( G \), next, we propose a threshold-based algorithm for the max regret ratio oracle.
Figure 2: Tuples in \( \{t_1, t_5, t_6, t_9\} \) partition the function space into convex regions. This is the updated representative after one iteration of URM, with \( \{t_1, t_5, t_7, t_9\} \) as the initial set. The new representative has the max regret ratio 0.0444.

\[
\text{max } r \\
\text{s.t. } r \geq 0 \\
\sum_{j=1}^{d} w_j \times t'[j] = 1 \\
\sum_{j=1}^{d} w_j \times (t'[j] - t[j]) \geq r \\
\forall 0 \leq \ell \leq k : \sum_{j=1}^{d} H[i,j] \times w_j \geq 0
\]

4.2 Threshold-based Algorithm

Threshold-based algorithms are proven to be effective in practice and are the de-facto solution for many important problems such as top-k query processing [2, 22]. When the objective value is minimization, the idea is to sort the tuples based on a lower bound on their objective values, ascending. Then starting from the top of the list, while maintaining a threshold, we continue processing the tuples in the sorted order until the lower bound value of the remaining tuples in the list is larger than the current threshold. The algorithm can stop then, as the objective value of the remaining tuples cannot be less that the best known threshold.

We apply a similar strategy here. But, first, we need to find the lower bounds on the max regret ratio of the tuples in region \( R \). We apply two strategies for finding the lower bound: (i) function sampling and (ii) edge sampling, explained in § 4.2.1 and § 4.2.2, respectively. The objective is to construct a sorted vector \( V \) based on the lower bound values on the max regret ratio of the tuples.

4.2.1 Lower bound based on function sampling

The first strategy is to use function sampling for finding the lower bounds. We use the existing work [5] for sampling unbiased functions from the region \( R \). Using a set of \( N \) IID function samples drawn from \( R \), we construct a \( n \) by \( N \) table \( T \) that every row in it is a tuple and every column is one of the sampled functions. Every cell \( T[i,j] \) is the regret ratio of the tuple \( t_i \) on the sampled function \( f_j \) (Fig. 4). In order to identify the cells in \( T \), we first make a pass over the matrix and fill every cell \( T[i,j] \) with the value of \( f_j(t_i) \). Also, for every column \( j \), we keep track of its maximum value \( \max_j \). After finishing the first pass over the matrix, we do a second pass replacing each cell value \( T[i,j] \) by \((\max_j - T[i,j]) / \max_j\).

After the table \( T \) is constructed, the lower bound on the max regret ratio of each tuple \( t_i \) is the max value on row \( T[i] \). That is,

\[
\text{lower}_{RR}(t_i, R) = \max_{j=1}^{N} T[i,j] 
\]

Algorithm 3 uses the above idea and returns a sorted vector of tuples based on the lower bound estimation of their max regret ratio.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuple</td>
<td>RR</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>0.0</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>0.0223</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>0.1208</td>
</tr>
<tr>
<td>( t_9 )</td>
<td>0.0938</td>
</tr>
<tr>
<td></td>
<td>0.1208</td>
</tr>
</tbody>
</table>

Figure 3: Graph transformation for max regret ratio oracle

Table 2: An iteration of the URM algorithm corresponding to the max regret ratio problem for example 1. Regret ratio for each of the tuples in representative is shown. The last row gives the max regret ratio score for the representatives.

Algorithm 3 SortedLB_{FS}

**Input:** Database \( D \), Set of Function Samples \( F \)

**Output:** Sorted vector of tuples based on the lower bound of their max regret ratio

**Program:**

1: for \( j \leftarrow 1 \) to \( |F| \) do
2: \( \max_j \leftarrow 0 \)
3: for \( i \leftarrow 1 \) to \( n \) do
4: \( T[i,j] \leftarrow F[j](t_i) \)
5: if \( T[i,j] > \max_j \) then \( \max_j \leftarrow T[i,j] \)
6: end for
7: for \( i \leftarrow 1 \) to \( n \) do \( \text{Sorted} \leftarrow \frac{\max_j - T[i,j]}{\max_j} \)
8: end for
9: for \( i \leftarrow 1 \) to \( n \) do \( \text{Sorted} \leftarrow \frac{\max_j - T[i,j]}{\max_j} \)
10: return Sorted(\( \nu \)) on second column

Making two passes over \( T \) and then sorting the vector \( V \). Algorithm 3 is in \( O(n(N + \log n)) \). Note that, considering a fixed sampling budget, the algorithm is linearithmic.

4.2.2 Lower bound based on edge sampling

Function sampling works well for fat regions i.e. regions which have a large volume where sampling from these regions is easy. But if the space of function space is a thin region then function sampling from it would end up being costly. We propose weighted sampling of edges of \( G \) for these cases.

Recall that the max regret ratio of a tuple \( t_i \) in a region \( R \) is the max of the weights of its outgoing edges in \( G \) (c.f. Theorem 4). A loose lower bound on the max regret ratio value can be obtained by uniformly sampling a few edges from the graph and computing their weights. To make it more effective, we use a weighted sampling of the edges. In order to obtain the weights for the sampling process, we start with a set of ranking functions, chosen within the region \( R \). To obtain the functions, a linear program is used to find the Chebyshev center [23], which is the center of largest inscribed hyper sphere inside of \( R \). Using a normal distribution, we sample a few functions and transform these functions to lie on the surface of the hyper sphere described by the Chebyshev center. We use the sum of the scores of the tuples as a guidance for the weighted
Algorithm 4 SortedLB

Input: Database \( \mathcal{D} \), Number of samples \( N \), Number of edge samples \( N_{\text{edge}} \), Convex region of functions \( R \)
Output: Sorted vector of tuples based on the lower bound of their max regret ratio
1: \( \text{center, radius} \leftarrow \text{Compute Chebyshev Center for region } R \)
2: \( F \leftarrow \text{Sample}(N) // \text{draw } N \text{ samples} \)
3: for \( j \leftarrow 1 \) to \( |F| \) do
4: for \( i \leftarrow 1 \) to \( n \) do
5: \( \mathcal{P}[i] \leftarrow \mathcal{P}[i] + F[j](t_i) \)
6: end for
7: for \( i \leftarrow 1 \) to \( n \) do
8: \( \text{Normalize } \mathcal{P} \)
9: \( V \leftarrow \text{SortedLB}(\mathcal{D}, F) \)
10: end for
11: return \( V \)

Algorithm 5 MaxO

Input: Database \( \mathcal{D} \), Convex region of functions \( R \), Sampling budget \( N \), representative \( t \), threshold \( \tau \)
Output: The representative tuple for \( R \)
1: \( F \leftarrow \text{Sample}(R, N) // \text{draw } N \text{ samples from } R \)
2: if \( F \) is not empty then \( V \leftarrow \text{SortedLB}_{FS}(\mathcal{D}, F) \)
3: else \( V \leftarrow \text{SortedLB}_{FS}(\mathcal{D}, R, N) \)
4: for \( i \leftarrow 1 \) to \( n \) do
5: if \( V[i, 2] \geq \tau \) then break
6: \( rr \leftarrow 0 \)
7: for \( j \leftarrow 1 \) to \( n \) where \( j \neq V[i, 1] \) do
8: \( w \leftarrow \text{compute } w_{V[i, 1]} \rightarrow j \) based on Eq. 11
9: if \( w > rr \) then \( rr \leftarrow w \)
10: if \( rr \geq \tau \) then continue
11: end for
12: if \( rr < \tau \) then \( t \leftarrow V[i, 1] \)
13: end for
14: return \( t \)

ARR computation oracle requires to partition the region \( R \) into the “maxima sub-regions” such that a specific tuple \( t_i \) is the maxima for each and every of the functions in each sub-region. Let \( T_{TM} \subseteq \mathcal{D} \) be the set of tuples that have the maximum score for at least one function in \( R \). Also let \( n' \leq n \) be the size of \( T_{TM} \). For each tuple \( t \in T_{TM} \), the set of functions for which \( f \) is the maxima is a convex region, defined by the intersection of \( n' - 1 = O(n) \) half-spaces. For every function \( f \) in \( R_r \) (the maxima sub-region of a tuple \( t' \in T_{TM} \)), the regret ratio of a tuple \( t \in D \) is \( \frac{f(t') - f(t)}{f(t')} \). This, in the end, provides an exact solution for computing the ARR of a tuple in a region \( R \). However, it involves a volume computation under the regret ratio curves across the sub-regions. Unfortunately, even though the total number of sub-regions is in \( O(n) \) and each sub-region is defined as the intersection of a linear number of half-spaces, the computation of ARR within each of the sub-regions is not computationally feasible in higher dimensions. That is because, as proven by Dyer et al. [18], the exact computation of the volume of a convex shape described as the intersection of a linear number of half-spaces is #P-hard.

Fortunately, although exact volume computation is usually costly, Monte-carlo methods [24] combined with tail inequalities [25] provide strong estimation methods for the problem. Following this, Zeighami et al. [15] have shown that the ARR value can be approximated using \( N \) samples of ranking functions with an error bound of \( \epsilon \) and a confidence of \( 1 - \alpha \), where the relation between \( N, \epsilon \), and \( \alpha \) is shown in the following equation.

\[
\epsilon = \sqrt{\frac{3}{N}} \ln \frac{T}{\alpha}
\] (13)

Essentially, [15] shows that discretizing the continuous function space to a set of \( N \) uniform samples, and using the samples for finding the maxima representatives guarantees an error \( \epsilon \), with the confidence interval of \( 1 - \alpha \), as specified in Eq. 13. We follow this in the design of the ARR oracle. That is, to use the discrete set of \( N \) uniform function samples for finding the representative tuples.

Consider a database \( \mathcal{D} \) and a set of representative tuples \( S \) and their regions. For each of the regions, we want to find the tuple \( t \in \mathcal{D} \) for which the ARR score is the lowest. For any region \( R_t \), we select the functions which lie inside \( R_t \). We use this set of ranking functions to estimate the ARR score of each of the tuples in the database. Next, for each region, the tuple with the lowest ARR score replaces the previous representative of that region. The algorithm is given in 6.

6. EXPERIMENTS

6.1 Experimental setup

Datasets: For evaluating our algorithms, we have used the following datasets. We generated two synthetic datasets - Surface and Scaled along the lines of Sphere and SkyPoints in [12].

- (Real dataset) Colors [26]: This is one of the commonly used datasets for the evaluation of regret ratio and skyline problems [10, 12, 27]. In our experiments, we have used the Color Histogram dataset. It contains 68,040 tuples, each being a color image. For every image, it contains 32 attributes, where each attribute is the density of a color in the entire image.
- (Synthetic) Surface: We generated the Surface dataset by uniformly sampling points on the surface of a unit hypersphere. Therefore, by construction, all the points in the Surface dataset belong to the skyline. The dataset contains 20,000 tuples, over 12 attributes, in range \([0, 1]\).
- (Synthetic) Scaled: For the Scaled dataset, we uniformly generated points inside a unit hypersphere. Since, in a high dimensional space, a large portion of the total volume of a hypersphere sampling. The idea is that the tuples with the higher scores are more likely to be representative of the region. Hence, we use the normalized vector of the score aggregates for the tuples, as the probability distribution for edge sampling.

Weighted sampling is performed using these weights to obtain a few edges of the graph. Computing the weights of these edges gives us a tighter lower bound value for max regret ratio. These lower bound values are then used in a similar manner to the threshold based algorithm \( \S 4.2.1 \). Algorithm 4 shows the pseudo code for the weighted edge sampling algorithm.

Having the sorted list of tuples \( V \), we can now design our threshold based algorithm (Algorithm 5). Starting from the first tuple in the list, the algorithm computes the weights for the outgoing edges the current tuple in graph \( G \) (Eq. 11). The max regret ratio of the current tuple is the max of its outgoing edges. While making a pass over \( V \) and computing the max regret ratio of the tuples, the algorithm keeps track of the best known solution (the least value of max regret ratio) as the threshold, and stops as soon as the lower bound values of the remaining tuples are higher than the threshold.

5. AVERAGE REGRET RATIO ORACLE

In this section, we shift our focus to a different measure of regret ratio, namely, average regret-ratio (ARR). An exact solution for the
Algorithm 6 AvgO

Input: D, Convex region R, Function samples F in R
Output: The representative tuple for R
1: for j ← 1 to |F| do max[j] ← 0
2: for i ← 1 to n do
3: for j ← 1 to |F| do
4: scores[i,j] ← F[j](t_i)
5: if max[j] < scores[i,j] then max[j] ← scores[i,j]
6: end for
7: end for
8: min ← ∞
9: for i ← 1 to n do
10: arr[i] ← ∑_{j∈F} max[j]−scores[i,j]
11: if min > arr[i] then min ← arr[i], t ← i
12: end for
13: return t

lies near the surface area, most of the points in the Scaled dataset is also present in the skyline. The scaled dataset contains a set of 20,000 tuples, each defined over 12 attributes in range [0, 1].

- (Real dataset) DOT [28]: The flight on-time dataset is published by the US Department of Transportation (DOT). It records, for all flights conducted by the 14 US carriers in January 2015, attributes such as scheduled and actual departure time, taxing time and other detailed delay metrics. The dataset consists of 457,013 tuples and 7 ordinal attributes.

- (Real dataset) NBA [29]: NBA 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, fgm, tpa, tpm.

Hardware and Platform: All our experiments were performed on a Core-i7 machine running Ubuntu 16.04 with 64 GB of RAM. The algorithms were implemented in Python.

Evaluation: In order to assess the performance of our algorithm, we focus on two main criteria namely, efficiency and efficacy. Concretely, we evaluate based on the following metrics - (i) the quality of the results produced, i.e. the regret ratio measure of the result (ii) the amount of time taken by the algorithm.

Algorithms Evaluated: For the max regret ratio representative problem, we have used HD-RRMS as the baseline [11], one of the recent and advanced algorithms for finding the max regret ratio representative that guarantees a tunable additive approximation. To do so, it discretizes the ranking function space and models the problem as a discrete matrix min-max problem. A combination of binary search technique and transforming the problem into fixed-size set covers are applied for solving the problem. To make the problem practical, the greedy approximation algorithm is used for the set cover instances. As we shall later show in this section, the extra approximation induced by the greedy set cover in the HD-RRMS algorithm shows up in our results and can be seen in the HD-RRMS curve when the regret ratio does not reduce with time in some cases. For the average regret ratio case, we have implemented the GREEDY-SHRINK algorithm [15]. This algorithm starts with the entire database as its representative and iteratively removes the tuple for which the increase in the average regret ratio is the least. This process is continued until k tuples are left. In addition to GREEDY-SHRINK, we also compute the global minima, using N samples from the function space, drawn based on Eq. 13 with ε = 0.01 and a confidence as 0.999. We filtered the dataset to only the skyline points. Using the N samples, a table containing the regret-ratio values for the filtered tuples is generated, similar to table 4. For every combination of k tuples as a representative, we compute the average regret-ratio value. It is important to note here that existing algorithms are all approximation algorithms and do not exhibit the anytime property. As a result, URMs is not directly comparable to the existing work. In our experiments we focus on specific properties of our algorithm.

We also compare our algorithm with several skyline reducing algorithms (which are discussed in more detail in § 7). We have implemented the kRSPGreedy algorithm from [30], EIQUIE algorithm from [31], NAIVEGreedy algorithm from [32], e-ADR greedy algorithm from [33] and SKYCOVER algorithm from [34].

6.2 Summary of experimental results

At a high level, the experiments verified the quality and efficiency of our proposal. We consider regret ratio score as the measure of quality. For the max regret ratio representative problem, URMs improves the quality of the representatives provided by HD-RRMS when used as a starting point. When provided with a time budget URMs qualitatively outperforms HD-RRMS algorithm. In case of the average regret ratio representative problem, URMs qualitatively outperforms GREEDY-SHRINK when provided with the same time budget. Our experiments show that URMs converges to the local minima very fast which increases the chance of discovering the global optima. In addition, the experiments demonstrate some of the useful properties of our approach, namely, (a) anytime property - even if the execution of the algorithm is terminated at any point of time, the algorithm will still have a representative (b) provision for getting better results - by restarting with a different set of initial points, URMs can achieve better representatives.

6.3 Results for Max Regret-Ratio

Results when initial representative is given: Similar to other clustering-inspired algorithms, the quality of the results produced by the URMs algorithm depends on the initial set of tuples. Often feeding the output of an existing approximation algorithm as the input to these iterative algorithms yield good results. In this set of experiments, we have used the compact representative returned by HD-RRMS as the initial set of tuples to the URMs algorithm. To be fair in the assessment, while creating the plots for URMs, we have taken the time to generate the initial points into consideration. Concretely, we have added the running time of HD-RRMS to the running time of URMs when creating the plots for the URMs algorithm. We run the experiments on the Surface, Colors, NBA, DOT, and the Scaled datasets. For Surface, Colors, DOT and Scaled datasets we use URMs for 4 and 5 dimensions with 20k points each. We use the same configuration used in [11] for NBA and DOT datasets. While we use 20k tuples with 4, 5 and 6 dimensions for NBA dataset, we use 400k tuples with 4 and 5 dimensions for DOT dataset. Fig. 5 and 6 plot the obtained max regret ratio to the time taken by each algorithm for Surface, Colors, Scaled and the DOT datasets for dimensions 4 and 5, respectively. Fig. 7 compares the URMs and HD-RRMS algorithms for the NBA datasets.

To get different results from HD-RRMS, we used different values of γ for discretization. Every red point shows the experimental result of one individual run of the HD-RRMS algorithm. While HD-RRMS is expected to provide better results as γ increased, in several settings, one can see an increase in the max regret ratio of the generated output. The reason is that HD-RRMS uses the greedy approach for solving the (theoretically fixed-size) set cover instances. This adds one more level of approximation to the algorithm which, in the end, results in the non-decreasing behavior of it in some cases.

In the figures, each red point is connected to a blue point by a dotted line, which represents feeding the HD-RRMS algorithm’s output to the URMs algorithm. The string of blue points connected by the solid blue line show the performance of the URMs algorithm.
Figure 5: Maximum regret ratio when *URM* uses the compact representative produced by *HD-RRMS* as the initial set [4D, K=5].

Figure 6: Maximum regret ratio when *URM* uses the compact representative produced by *HD-RRMS* as the initial set [5D, K=5].

Figure 7: Maximum regret ratio when *URM* uses the compact representative produced by *HD-RRMS* as the initial set for NBA dataset.

Figure 8: 2D Dataset Example

Figure 9: Maximum regret ratio when both *URM* and *HD-RRMS* uses a fixed time budget [4D, K=5].

Figure 10: Maximum regret ratio when both *URM* and *HD-RRMS* uses a fixed time budget [5D, K=5].

Figure 11: Maximum regret ratio when both *URM* and *HD-RRMS* uses a fixed time budget for NBA dataset.

Figure 12: Comparing skyline reducing alg. with *URM* for DOT dataset with 4 dimensions
over the subsequent iterations. First, our experiments show that URM quickly reaches the local minima of regret ratio in this setting. Also, in most of the experiments, URM improves the results of the HD-RRMS algorithm. The ones where URM could not improve the result of HD-RRMS were the ones that HD-RRMS had, by chance, discovered a local optima. Apart from the performance of the algorithm, it is interesting to see the existence of multiple local optima and the impact of the starting point on the optima discovered. For instance in Fig. 5b, each setting discovered a different local optima. The fast convergence of URM enables multiple runs of the algorithm with multiple starting points, which increases the chance of discovering the global optima.

Results on a fixed time budget: While URM is an anytime algorithm, HD-RRMS is not. That is, HD-RRMS returns a representative only after it is finished. Therefore, in order to make a fair assessment, we run the HD-RRMS algorithm with different values of \( \gamma \) and allocate the exact same amount of time taken by the HD-RRMS algorithm as the time budget for our URM algorithm. We run our experiments with Surface, Colors, NBA, DOT, and the Scaled datasets. This set of experiments demonstrate two important properties of the URM algorithm. First, as we can see from Fig. 9, 10 and 11 the regret ratio scores reduce monotonically. This is due to the convergence property of the algorithm, proved in § 3.3. In addition, the URM algorithm produces reasonably good regret ratio scores even when the allocated time budget is small. This means that a user can still get a reasonably good set of representatives even if she terminates the algorithm before it finishes. Another important result is that URM actually allows the user to find a better representative. With a fixed time budget, we restart the algorithm with a random set of starting points and repeat this process until our time budget expires. Essentially, with subsequent repetition of the URM algorithm, we find a different set of compact representatives with a better score of max regret-ratio. Another important property that our algorithm exhibits is the any time property. As URM is an iterative hill climbing algorithm, the best representative among the ones it has already visited can be consumed by the user even before the end of the time budget. This can be seen in Fig. 9 and Fig. 10 where if we were to stop the algorithm at any point of time we would get a representative marked by the blue circles. First, looking at the figures, one can see the monotonically decreasing behaviour of URM, compared to HD-RRMS. Also, HD-RRMS needs to finish at least once or will not provide any output. For instance, in Fig. 10(c), HD-RRMS did not provide any result for a time budget less than 8000 seconds. In contrast, having the anytime property, URM has an output to offer at any point of time.

Proof of concept by an example: Using the NBA dataset, we highlight a concrete example. The dataset contains performance records of different Basketball players across different seasons. Consider the case where the decision criteria are Points, Rebounds, Assists and Steals. More than 60 tuples belonged to the skyline. Offering such a large set to the user is overwhelming. Instead, using the URM algorithm while setting the output size to 6 we find the set \( \{ \text{Wilt Chamberlain} - 1967, \text{Don Buse} - 1975, \text{Nate Archibald} - 1972, \text{Michael Jordan} - 1987, \text{Wilt Chamberlain} - 1961, \text{John Stockton} - 1988 \} \) with maximum regret-ratio of 0.058. That is, the user can make selection between these 6 tuples, yet be sure that the score (quality) of its selection is not more than 5.8% percent worse than the optimal choice. For example, for the ranking function Assists+Steals, the optimal tuple is John Stockton 1990 with the score of 1398. The tuple John Stockton - 1988 in the representative set has the score of 1381 which shows how close to optimal it is.

6.4 Results for Avg Regret-Ratio
Comparison against the global minima: URM is an iterative algorithm where the regret ratio score monotonically decreases over time. That is, it produces better results as time passes. In contrast, the GREEDY-SHRINK algorithm generates results only at program termination. For a fair comparison between the two algorithms, we run GREEDY-SHRINK with different parameter values of \( \epsilon \) and \( \sigma \). We also demonstrate the ability of our algorithm to find the global minima given sufficient time. Therefore, in addition to the GREEDY-SHRINK algorithm we compare against the global minima, the computation of which is described in § 6.1. Concretely, we let the URM algorithm finish multiple iterations and recorded the regret ratio at the end of each iteration. Finally, we have compared the progression of these scores against the global minima along with the results from GREEDY-SHRINK. For this set of experiments, we have used 20,000 points from the Colors dataset, in a 9 dimensional space. As discussed in § 5, instead of computing the exact volumes, we approximate the average regret ratio score using \( N \) samples with an \( \epsilon \) of 0.01 and a confidence of 0.999 as described in 13. We present the results of our experiments in Fig. 13. The black dotted line indicates the global minima while the red triangles show the output from the GREEDY-SHRINK algorithm. The URM algorithm outperforms the GREEDY-SHRINK algorithm for different output sizes of \( k \). In all cases URM finds a representative with small value of average regret ratio in a small amount of time. In fact, our experiments show that it is possible to reach the global minima of average regret ratio in a short time.

6.5 Results for skyline reducing algorithms
To compare our algorithm with skyline reducing algorithms, we have implemented the algorithms kRSPGREEDY [30], \( \epsilon \)-ADR [33], EIQUE [31], NAIVEGREEDY [32], and SKYCOVER [34]. NAIVEGREEDY takes the skyline and a starting tuple as input to output a representative. To be fair in comparison, we passed every point from the skyline as a starting point and chose the set with least regret ratio. \( \epsilon \)-ADR and SKYCOVER take an error parameter \( \epsilon \) as the input and output the smallest set size satisfying \( \epsilon \)-ADR optimization criteria. To find a set of size \( k \) or smaller, we find a large \( k \) which satisfies the set size criteria and apply a binary search to get the smallest \( \epsilon \) satisfying the size requirement. The results for the DOT dataset with dimensions 4 and 5 are presented in Fig. 12 and Fig. 14. We denote kRSPGREEDY as RSP, NAIVEGREEDY as NG, EIQUE as EQ, \( \epsilon \)-ADR greedy as \( \epsilon \)-ADR and SKYCOVER as SC in the plots. The blue bar shows the regret ratio of the output of
the skyline reducing algorithms, with the name of the algorithm on
the X axis. The orange bars show the regret ratio of the output of
our (URM) algorithm when initialized with the corresponding al-
gorithm’s output. As expected URM outperforms other algorithms.

7. RELATED WORK

Over the past few decades, a major amount of research has fo-
cused on the generation of a representative of the dataset to assist
users with multi-criteria decision making. Most of these published
works can be grouped into skyline discovery, skyline reduction and
regret based compact representative computations. In this section,
in addition to discussing the related works in these three categories
we also provide a brief summary of the clustering techniques that
have inspired the URM algorithm.

Regret minimizing representatives: As an effective solution to
the problem of finding compact representatives, Nanongkai et al.
[10] introduced regret-ratio minimizing representative. The notion
of regret introduced in the paper deals with the amount of dissatis-
faction the user would express when she is shown the top item
from the representative set instead of the top item from the dataset
when provided with the user preference. Many different variations
of the original regret-minimizing representative problem have been
studied since this paper [16, 35]. One specific variant, finding a
representative set that minimizes the maximum regret ratio, has
been extensively studied by several researchers. Agarwal et al. [12]
proved that this problem is NP-complete for dimensions larger than
2. Asudeh et al. [11] introduce function space discretization and
the transformation of the problem to set-cover instances. [11, 12]
propose approximation algorithms with similar tunable approxima-
tion guarantees for the problem. Another interesting variant of this
problem is finding the average regret ratio problem. It was intro-
duced by Zeigami et al. [14, 15]. However, it is important to note
that while others have studied and proposed solutions for single
variants of the regret measures, we have proposed an unifying al-
gorithm that can work with a class of regret measures.

Skyline and convex hull: In the pursuit of efficiency, finding a
small set of representatives of the entire dataset has been of key
interest in recent years. In the absence of user preferences, sky-
line [8, 36, 37] and convex hull based algorithms obtain a part of
the dataset which behave like representatives. While convex hull is
more compact than the skyline, the computation of a convex hull is
significantly more costly in large dimensions. Bentley et al have
studied approximation algorithms with tight approximation ratios
for convex hulls [38]. However, as we move to higher dimensions,
most points in the dataset appear on the skyline or convex hull. As
a result, even though these skyline/convex hull based algorithms are
effective in lower dimensions, the large size of skyline and convex
hull render them ineffective as representatives.

Skyline reducing algorithms: Existing works have studied the
problem of reducing the size of the skyline in various scenarios [30–
34, 39–41]. To the best of our knowledge, none, except the regret-
minimizing literature, incorporate user customizable functions for
skyline reduction nor can be directly translated to the regret ratio
problem. [30] and [39] reduce the skyline size by choosing a subset
of size k of it which maximizes the number of dominated tuples by
this subset. [32] proposes distance-based skyline, a subset of size
k that minimizes the sum of the distances between the points and
their closest representative. [41] and [31] find a subset of size k with
maximum diversity. [42] propose the concept of skyline ordering,
which is skyline-based partitioning of a given data set such that an
order exists among the partitions. This ordering is then exploited
to reduce the set size to k. We note that all of the aforementioned have
a different objective function than regret-ratio and cannot provide
any guarantee on how good it is for an arbitrary function.

Similarly, [33] and [34] propose a new measure, $\epsilon$-ADR query,
which chooses a subset of the skyline such that the tuples of the
skyline when scaled by $(1+\epsilon)$, dominate the rest of the skyline
tuples. We illustrate the difference between the optimization mea-
sure of $\epsilon$-ADR query and regret ratio with an example. Consider
a dataset with n points in (2D), out of which (n-2) are equi-angularly
placed on the surface of a circle with radius 10 and the two other
points are $(20,1)$ and $(1,20)$. Note that all the points belong to
the skyline. Figure 8 shows this example for $n=7$. The opti-
mal set for size 2 is the convex hull \{(20,1), (1,20)\} with max
regret ratio 0. On the other hand, $\epsilon$-ADR query chooses a subset
of the skyline such that the tuples of the skyline when scaled by
$(1+\epsilon)$, dominate the rest of the skyline tuples. As both $(20,1)$
and $(1,20)$ have one of their attributes set to 1, the scaling required
for these tuples to dominate the other tuples is large making them
less attractive to the $\epsilon$-ADR query. For the example of Figure 8,
$\epsilon$-ADR [33] returns \{(6.66, 5.0), (5.0, 8.66)\}. Note that, instead
of \{(20,1) and (1,20), we could add points $(X,1)$ and $(1,Y)$ with
X and Y such that there exists a tuple $(a, b)$ from database which
has the property $\frac{a}{X} < b$ and $\frac{X}{a} < a$. For example, we can increase
X and Y to 49 as this dataset contains the point (7.07, 7.07) for
which $\frac{49}{7.07} = 6.93 < 7.07$.

Clustering: Clustering have long been studied to partition data into
similar groups. One of the major variants in this field is the K-
MEDOID algorithm [17]. Like many other clustering algorithms,
K-MEDOID is also an iterative algorithm that partitions the points
into clusters by minimizing the distance between the points from
the cluster center. However, the difference between the K-MEDOID
algorithm and other clustering techniques is that K-MEDOID al-
ways chooses the cluster centroids from the existing data points. In
the literature, there have been several variations of K-MEDOID
that have explored several avenues of choosing the best centroid.
Few of the most prominent are PAM, CLARA [43] and CLARANS [44].

8. FINAL REMARKS

In this paper, we proposed a unified algorithm for solving a va-
riety of “regret-minimizing representative” problems for different
aggregate norms. Unlike the existing work that design approxi-
mation algorithms for a specific variant of problem, we design an
iterative optimization algorithm to search for optima for the gen-
eral problem. To do so, we make connection with the problem of
clustering from data mining, and the corresponding K-MEDOID al-
gorithm. We propose several innovative technical contributions in
designing our algorithm and provide theoretical analysis as well as
empirical experiments that demonstrate the effectiveness of our
proposal. The experiments also highlight useful properties of our
approach that makes it a suitable practical algorithm.

Our work gives rise to several interesting optimization opportu-
nities that deserve further study. For example, currently in URM,
we update all regions concurrently. An interesting heuristic to up-
date the regions one after the other with a specific ordering which
might improve our algorithm. In particular, at every iteration, we
could find the representative tuple with the smallest region by vol-
ume as a good candidate for being replaced. This, however, is chal-
enging to develop and implement, as each update requires k ex-
clusive volume computations. We consider this, along with other
possible optimizations and open problems for future work.

9. ACKNOWLEDGEMENTS

The work of Gautam Das was supported in part by grants 1745925
and 1937143 from the National Science Foundation, and a grant
from AT&T.
10. REFERENCES


