Noname manuscript No.
(will be inserted by the editor)

An Experimental Survey of Regret Minimization Query and Variants: Bridging the Best Worlds between Top-k Query and Skyline Query

Min Xie · **Raymond Chi-Wing Wong** · **Ashwin Lall**

Received: date / Accepted: date

Abstract When faced with a database containing millions of tuples, a user may be only interested in a (typically much) smaller representative subset. Recently, a query called the *regret minimization query* was proposed towards this purpose to create such a subset for users. Specifically, this query finds a set of tuples that minimizes the user regret (measured by how far the user's favorite tuple in the selected set is from his/her favorite tuple in the whole database). The regret minimization query was shown to be very useful in bridging the best worlds between two existing well-known queries, top-*k* queries and skyline queries: like top-*k* queries, the total number of tuples returned in this new query is controllable, and like skyline queries, this new query does not require a user to specify any preference function. Thus, it has attracted a lot of attention from researchers in the database community.

Various methods were proposed for regret minimization. However, despite the abundant research effort, there is no systematic comparison among the existing methods. This paper surveys this interesting and evolving research topic by broadly reviewing and comparing the state-of-the-art methods for regret minimization. Moreover, we study different variants of the regret minimization query that has garnered considerable attention in recent years and present some interesting problems that have not yet been addressed in the literature. We implemented 12 state-of-the-art methods published in top-tier venues such as SIGMOD and VLDB from 2010 to 2018 for obtaining regret minimization sets, and give an experimental comparison under various parameter settings on both synthetic and real datasets. Our evaluation shows that the optimal choice of methods for regret minimization depends on the application demands. This paper provides an empirical guideline for making such a decision.

Min Xie
The Hong Kong University of Science and Technology
E-mail: mxieaa@cse.ust.hk

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
E-mail: raywong@cse.ust.hk

Ashwin Lall
Denison University
E-mail: lalla@denison.edu

1 Introduction

Nowadays, a database system usually contains millions of tuples and an end user might be interested in finding his/her favorite tuples in the database. Consider the following scenario for a car database where each car is described by some attributes. Alice visits the car database and wants to find a car with high horse power (HP) and high miles per gallon (MPG) (i.e., HP and MPG are the two attributes picked by Alice, based on which she makes a decision). Note that the car database can be very large and it may consist of thousands of cars and thus, it might be impossible for Alice to go through every car tuple in the database. A possible solution is that the database system provides some operators to show a representative subset of cars to Alice. Such operators can be regarded as *multi-criteria decision-making* tools. In order to decide which cars to be shown to Alice, we implicitly assume that there is a preference function, called a *utility function*, in Alice's mind. Based on this function, we can compute a *utility* for each car in the database. A high utility indicates that this car is favored by Alice and a car with the highest utility is a *favorite* car of Alice. Depending on whether the utility function is provided to the database system, different operators were proposed towards multi-criteria decision-making. Examples are the *top-k query*, the *skyline query* and the *regret minimization query*.

In the setting of the traditional top-*k* query [14, 22, 23, 30, 38], a user is required to provide his/her exact utility function explicitly to the database system. Then, the *k* tuples with the highest utilities are returned to the user. For example, Alice's utility function can have weight 70% for HP and weight 30% for MPG. Here, a higher weight indicates that the corresponding attribute is more important to Alice. With this utility function, each car's utility is computed, and the k cars with the highest utilities are shown to Alice. Unfortunately, it is hard for most users to provide their utility functions explicitly to the database system and even the users themselves might not know their exact utility functions.

Alternatively, the *skyline* query [8, 9, 24, 26, 29] could be used if the utility function (assumed to be monotonic) is not provided to the database system. In particular, a *"domination"* concept is applied. A tuple p is said to *dominate* another tuple q if p is not worse than q on each attribute and p is better than q on at least one attribute. For example, car p with HP 300 and MPG 30 dominates car q with HP 250 and MPG 25 since no matter what utility function Alice has, the utility of car p is always higher than the utility of car q and thus, car p is more desirable to Alice. The skyline query returns all tuples that are not dominated by any other tuples to the users and those tuples are also called the *skyline tuples*. It is easy to see that the user's favorite tuple must be a skyline tuple. Unfortunately, the output size of a skyline query is un-controllable. In the worst case, the whole database can be returned by a skyline query, resulting in its difficulty in providing a small representative subset to the users.

Recently, a regret minimization query [28] was proposed, which solves multi-criteria decision-making from a novel perspective. In particular, it overcomes the deficiencies of both the top-k query (which requires the user to provide the exact utility function) and the skyline query (which does not have a controllable output size). Instead, it maintains the major advantage of the top-k query (whose output size is controllable) and the major advantage of the skyline query (which does not require the user to provide any exact utility function). Specifically, a regret minimization query finds a small set of tuples from the database such that the utility of any user's favorite among these tuples is guaranteed to be a small fraction, quantified as the *regret ratio*, less than the utility of his/her favorite in the whole database, regardless of his/her utility function. Intuitively, the regret ratio quantifies the "regret" level of a user if s/he gets the best tuple in the selected subset, but not the best tuple in the whole database. For example, a regret minimization query on the car database returns a set of cars from the database so that Alice can find some cars in the returned set that she is interested in (since her regret ratio is small) without providing her utility function. In addition to the car database application, the regret minimization query can be applied in many other scenarios. For example, on an online shopping application, each product is usually described by multiple attributes (e.g., rating and quality). Different users can have different preferences in their minds. For example, some users might

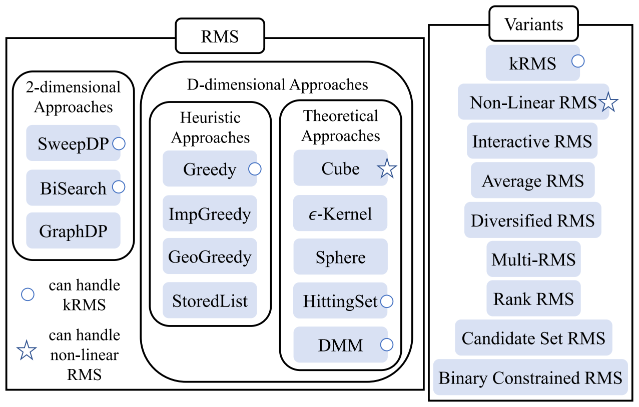


Fig. 1 Taxonomy of Regret Minimization Queries

think that a higher rating is more important while the other users might think that higher quality is more important. A regret minimization query finds a set of products minimizing the "regret" level of all users. Those products can be promoted on the home page to attract customers since no matter what preference a customer has, s/he can always find a product in the suggested set that s/he is interested in (since the regret ratio is small). Other applications of regret minimization queries includes Information Retrieval (IR) [35, 3, 37] and Recommendation Systems (RS) [19, 25, 42].

Due to the superiority of regret minimization queries, extensive efforts [28, 31, 11, 7, 4, 2, 21, 44] in the database community have been spent on finding algorithms for computing *regret minimization sets (RMS)*. However, there lacks a comprehensive comparison among them. In this paper, we give an overview of existing methods for RMS and present some interesting variants of RMS that receive considerable attention in the last decade (summarized in Figure 1). Specifically, we start with an extensive survey that covers 12 existing methods for RMS. We describe the key idea behind each method and summarize the main results known for each method. We also classify the existing methods into three categories: (1) the exact approaches for RMS when each tuple in the database is described by 2 attributes, (2) the heuristic approaches and (3) the theoretical approaches for RMS when each tuple in the database is described by d attributes ($d \geq 2$). Then, we present 9 popular variants of RMS studied in the literature. In particular, kRMS and non-linear RMS are the two major variants of RMS and we show how some existing algorithms designed for RMS can be extended to handling these variants (shown in circles and stars in Figure 1).

We performed a comprehensive experimental evaluation on the 12 existing methods for RMS on synthetic datasets [6] with different distribution characteristics (e.g., correlated datasets and anti-correlated datasets) and six commonly used real-world datasets with up to five million tuples [28, 31, 11, 7, 4, 2, 21, 44]. The experimental results could give an insight to researchers for RMS. According to our experiments, there is no single algorithm which dominates the other algorithms in all aspects. Specifically, some algorithms (e.g., 2d-BISEARCH [7]) solve RMS optimally but it is restricted when each tuple in the database is described by 2 attributes while some other algorithms (e.g., GREEDY [28]) are heuristic-based, but they are executable on datasets of any dimensionalities. Some algorithms (e.g., CUBE [28]) constructs a solution for RMS efficiently but the empirical maximum regret ratios of their solutions are large, which means that the users can be regretful if they see the solutions, while some other algorithms (e.g., HITTINGSET [2]) spend more time to return the solutions but they are good at constructing a small representative subset of the whole database for the users. The best choice of algorithms depends on the user demands.

The rest of the paper is organized as follows. The formal definition of regret minimization set (RMS) and some known properties/theoretical lower bounds on this problem are described in Section 2. In Section 3, we survey the existing methods for RMS, summarize the main results for each method and provide a comprehensive comparison among them. Different variants of RMS are described in Section 4 and experimental evaluations on both real and synthetic datasets are presented in Section 5. Some open problems that have not yet been explored in the literature are summarized in Section 6 while conclusions are found in Section 7.

2 Problem Definition

The input to our problem is a tuple set D with n tuples (i.e., $|D| = n$) in a d-dimensional space where each dimension corresponds to an attribute of a tuple. In this paper, we assume that the dimensionality d is a fixed constant. Note that each tuple in D could be described by more than d attributes, but the user will select precisely d of them that s/he is interested in, and based on which s/he makes decisions.

2.1 Terminologies

We use the word "tuple" and "point" interchangeably and use the word "attribute" and "dimension" interchangeably in the rest of the paper. Denote the i-th value of a d-dimensional point $p \in D$ by $p[i]$ where $i \in [1,d]$ and denote the L2-norm of p by $\|p\|$. Without loss of generality, we assume that the value in each dimension is non-negative and a larger value in each dimension is preferable to all users. If a smaller value is preferable in a dimension (e.g., price), we can modify the dimension by subtracting each value from the maximum value so that it satisfies the above assumption. Recall that in a car database, each car is associated with 2 attributes, HP and MPG. Consider the example in Table 1. The car database, i.e., $D = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, contains six 2-dimensional points, each of which represents a car in the database.

Car(p)	HP	MPG	$f_{0.4,0.6}(p)$	$f_{0.2,0.8}(p)$	$f_{0.7,0.3}(p)$
p_1	40	40	40	40	40
p_2	120	36	69.6	52.8	94.8
p_3	180	24	86.4	55.2	133.2
p_4	200	8	84.8	46.4	142.4
p_5	70	8	32.8	20.4	51.4
p_6	60	24	38.4	31.2	49.2
Regret Ratio of $\{p_1, p_4\}$			$1 - \frac{86.4}{84.8} = 1.85\%$	$1 - \frac{46.4}{55.2} = 15.9\%$	$1 - \frac{142.4}{142.4} = 0\%$

Table 1 Car Database and Car Utilities

Similar to [28,31,27,23,10], the user's happiness can be modeled by an unknown *utility function*, denoted by f, which is a mapping $f\colon \mathbb{R}^d_+ \to \mathbb{R}_+$. Denote the *utility* of a point p in D w.r.t. f by $f(p)$. A high utility indicates that p is favored by the user and a point with the highest utility is a *favorite* point of the user. For each user, we define a *regret ratio* based on his/her utility function f.

Definition 1 ([28]) Given a set $S \subseteq D$ and a utility function f, the *regret ratio* of S over D w.r.t. f, denoted by $\mathsf{rr}_D(S, f)$, is defined to be $\frac{\max_{p\in D} f(p) - \max_{p\in S} f(p)}{\max_{p\in D} f(p)} = 1 - \frac{\max_{p\in S} f(p)}{\max_{p\in D} f(p)}$.

For example, given a utility function $f_{0.4,0.6}$ where $f_{a,b}(p) = a \times p[1] + b \times p[2]$ and a point p_4 in Table 1, the utility of p_4 w.r.t. $f_{0.4,0.6}$ is $f_{0.4,0.6}(p_4) = 0.4 \times 200 + 0.6 \times 8 = 84.8$. The utilities of remaining points in D w.r.t. $f_{0.4,0.6}$ are computed similarly in Table 1. Consider a set $S = \{p_1, p_4\}$ (shown shaded in Table 1). The point with the highest utility in S w.r.t. $f_{0.4,0.6}$ is p_4 and its utility is equal to 84.8 while the point with the highest utility in D w.r.t. $f_{0.4,0.6}$ is p_3 and its utility is equal to 86.4. Then, we can compute $\mathsf{rr}_D(S, f_{0.4,0.6})$ to be $1 - \frac{\max_{p\in S} f_{0.4,0.6}(p)}{\max_{p\in D} f_{0.4,0.6}(p)} = 1 - \frac{84.8}{86.4} = 1.85\%$.

Given a set $S \subseteq D$, we have $\max_{p\in S} f(p) \le \max_{p\in D} f(p)$ (since S is a subset of D) and thus, the regret ratio in Definition 1 ranges from 0 to 1. A user is *happy* (some papers use the term *not regretful*) with a given set S if his/her regret ratio is close to 0 since the highest utility in S is close to the highest utility in D (i.e., the best tuple in the selected set S is close to his/her favorite tuple in the whole D). Table 2 summarizes the frequently used notations in the paper.

Unfortunately, in real cases, it is difficult to obtain the user's exact utility function. Thus, we assume that the user's utility function in a function class, denoted by FC. Examples of function classes include the *linear* [28] and *multiplicative* function class [32]. Then, the *maximum regret ratio* of a set S is defined over a function class FC, which can be regarded as the worst-case regret ratio w.r.t. a utility function in FC.

Definition 2 ([28]) Given a set $S \subseteq D$ and a function class FC, the *maximum regret ratio* of S over D w.r.t. FC, denoted by $\mathsf{mrr}_D(S, \mathsf{FC})$, is defined to be $\sup_{f\in \mathsf{FC}} \mathsf{rr}_D(S, f)$ [1].

[1] We define the maximum regret ratio using the supremum instead of the maximum since the function class FC can consist of an infinite number of utility functions and a maximum may not exist.

Notation	Meaning
D	The set of d-dimensional points ($\|D\| = n$)
$f(p)$	The utility of p w.r.t. a function f
FC	A utility function class
L	The linear utility function class
$\mathsf{rr}_D(S, f)$	The regret ratio of S over D w.r.t. f
$\mathsf{mrr}_D(S, \mathsf{FC})$	The maximum regret ratio of S w.r.t. FC
r	The maximum output size, i.e., $\|S\| \le r$
ε	The required maximum regret ratio, i.e., $\mathsf{mrr}_D(S, \mathsf{FC}) \le \varepsilon$
r_ε	The smallest size of any set with maximum regret ratio at most ε
ε_r	The smallest maximum regret ratio of any set with at most r points
$k\text{-max}_{p\in D} f(p)$	The k-th highest utility among points in D
$k\text{-}\mathsf{rr}_D(S, f)$ ($k\text{-}\mathsf{mrr}_D(S, \mathsf{FC})$)	The (maximum) k-regret ratio of S

Table 2 Frequently Used Notations

To illustrate, assume that FC consists of three utility functions $f_{0.4,0.6}$, $f_{0.2,0.8}$ and $f_{0.7,0.3}$ in Table 1. By following a similar procedure before, we can compute $\mathsf{rr}_D(S, f_{0.4,0.6}) = 1.85\%$, $\mathsf{rr}_D(S, f_{0.2,0.8}) = 15.9\%$ and $\mathsf{rr}_D(S, f_{0.7,0.3}) = 0\%$. Then, the maximum regret ratio $\mathsf{mrr}_D(S, \mathsf{FC})$ is computed to be $\sup_{f\in\mathsf{FC}} \mathsf{rr}_D(S, f) = \max\{1.85\%, 15.9\%, 0\%\} = 15.9\%$.

2.2 Problem Definition

Without knowing which function a user exactly uses in FC and the distribution of functions in FC, our goal is to find a regret minimization set $S \subseteq D$, optimizing over the worst case (maximum regret ratio), so that the worst-case regret is minimized and the happiness of *each* user is guaranteed. Formally, we define the regret minimization query (RMS).

Problem 1 (The Regret Minimization Query (RMS) [28]) Given a set D and a function class FC, we want to find a regret minimization set $S \subseteq D$ of at most r points so that the maximum regret ratio $\mathsf{mrr}_D(S, \mathsf{FC})$ is at most ε.

There are two parameters that come into play in RMS, namely (1) the maximum output size r and (2) the required maximum regret ratio ε. We assume that $r \ge d$. Otherwise, the maximum regret ratio might not be bounded [28]. In traditional RMS, we aim at minimizing (or bounding) the maximum regret ratio while fixing the output size [28,31]. Recently, however, some existing studies focus on a dual version of RMS which aims at minimizing (or bounding) the output size while fixing the maximum regret ratio [7,2]. Moreover, some recent methods relax both the maximum regret ratio and the output size simultaneously [21,4]. For the ease of illustration, we do not distinguish these variants explicitly but describe them in a unified manner in Problem 1.

In general, any function class FC can be applied in RMS and the utility functions in FC can have an arbitrary distribution. For the ease of illustration, we first focus on the class

	Results	Related Materials
RMS	Scale-invariance	Theorem 1
	Stability	Theorem 2
	Lower bound ($\Omega(r^{-\frac{2}{d-1}})$)	Theorem 3
	NP-hardness	Theorem 4

Table 3 Known Results about RMS

of *linear utility functions*, denoted by L, which is very popular in modeling user preferences [28,31,11,27,23,10]. We relax this assumption in Section 4 by considering other variants of RMS. Specifically, a utility function f is *linear* if $f(p) = u \cdot p$ where u is a *utility vector*. The utility vector u is a d-dimensional non-negative vector where $u[i]$ measures the importance of the i-th dimensional value in the user preference. In the rest of this paper, we refer to a utility function f by its utility vector u when $\mathsf{FC} = \mathsf{L}$ is clear in the context.

2.3 Properties

In this section, we introduce the *scale-invariance* and the *stability* of RMS, which are two important properties of RMS.

Scale-Invariance. Intuitively, RMS is said to be *scale-invariant* if the maximum regret ratio of a given solution set is the same even when the attribute value of each point in D is scaled by a certain factor. Specifically, we consider a scaled dataset $D' = \{p'_1, \ldots, p'_n\}$ of D where $p'_i[j] = \lambda_j p_i[j]$, $\lambda_j \ge 0$ for each $j \in [1, d]$. For example, we can create a scaled dataset D' for the car database in Table 1 by converting HP to watts and MPG to kilometers per liter by setting $\lambda_1 = 750$ and $\lambda_2 = 0.425$ since 1 HP = 750 watts and 1 MPG = 0.425 kilometers per liter. The following theorem shows that the definition, maximum regret ratio, is independent of the scale of each attribute and thus, RMS is scale-invariant.

Theorem 1 (Scale-Invariance [28]) *Let $S = \{p_{i_1}, \ldots, p_{i_k}\}$ be any subset of D and $S' = \{p'_{i_1}, \ldots, p'_{i_k}\}$ be the corresponding subset of D' where D' is a scaled dataset of D (i.e., for each p_i in D and each p'_i in D', $p'_i[j] = \lambda_j p_i[j]$ where $\lambda_j \ge 0$ and $j \in [1, d]$). We have $\mathsf{mrr}_D(S, \mathsf{L}) = \mathsf{mrr}_{D'}(S', \mathsf{L})$.*

Stability. RMS is said to be *stable* if the maximum regret ratio of any set S is independent of the *junk* points being inserted into or deleted from the database. Specifically, a point in D is said to be a *junk* point if it does not have the highest utility w.r.t. any utility function in L. Intuitively, a junk point is the point not favored by *any* user. According to the definitions above, stability is a desirable property since a database system is not allowed to manipulate the solution by strategically inserting/deleting a number of junk points not favored by any user. The stability of RMS is summarized below.

Theorem 2 (Stability [28]) *Given a set $S \subseteq D$ and a junk point p, $\mathsf{mrr}_D(S, \mathsf{L}) = \mathsf{mrr}_{D/\{p\}}(S, \mathsf{L}) = \mathsf{mrr}_{D\cup\{p\}}(S, \mathsf{L})$.*

2.4 Lower Bound and NP-hardness

In this section, we summarize the best-known lower bounds on RMS [44,27]. Informally, we show that by returning a set of at most r points from the database, it is not possible to guarantee a maximum regret ratio better than $\Omega(r^{-\frac{2}{d-1}})$.

Theorem 3 (Lower Bound [44]) *For any dimensionality d, there is a d-dimensional database such that the maximum regret ratio of any set of at most r points is at least $\frac{1}{8}(2r)^{-\frac{2}{d-1}}$.*

Corollary 1 ([27]) *For any dimensionality d and $\varepsilon \in (0,1]$, there is a d-dimensional database such that any RMS algorithm needs to return at least $\frac{1}{2}(\frac{1}{8\varepsilon})^{\frac{d-1}{2}}$ points from the database to guarantee a maximum regret ratio at most ε.*

Finding an optimal solution for RMS (i.e., finding a minimum size set guaranteeing a certain regret ratio ε or finding the minimum regret set with at most r points) was first proven to be an NP-hard problem in general by Chester et al. [11]. Formally, we formulate the decision version of RMS below, whose NP-hardness is shown in Theorem 4.

Problem 2 (Decision-RMS) Given a set D, a function class FC, an integer r and a real value ε, we want to determine whether there exists a solution set $S \subseteq D$ of at most r points so that the maximum regret ratio $\mathsf{mrr}_D(S,\mathsf{FC})$ is at most ε.

Theorem 4 (NP-Hardness [11]) *Decision-RMS is NP-hard.*

Unfortunately, the NP-hardness proof in [11] required both the size and the dimensionality of the dataset to be arbitrarily large. In particular, it was left open whether this problem is NP-hard for small dimensionalities. Cao et al. [7] and Agarwal et al. [2] resolved this issue independently by showing that RMS is NP-hard for all $d \geq 3$. Table 3 summarizes all the aforementioned known results about RMS.

2.5 Computing Maximum Regret Ratio

Given a set $S \subseteq D$, it is difficult to compute the maximum regret ratio $\mathsf{mrr}_D(S,\mathsf{L})$ directly according to Definition 2 since there are an infinite number of linear utility functions in L. In practice, we can approximate $\mathsf{mrr}_D(S,\mathsf{L})$ by sampling a finite number of utility functions (e.g., 100,000 utility functions [2]) in L, based on which we compute their regret ratios. Then, $\mathsf{mrr}_D(S,\mathsf{L})$ can be estimated to be the largest regret ratio among them. Alternatively, we can also compute the exact $\mathsf{mrr}_D(S,\mathsf{L})$. This is done by dividing the computation of $\mathsf{mrr}_D(S,\mathsf{L})$ into a finite number of smaller problems. Formally, we have the following lemma from [28].

Lemma 1 ([28]) $\mathsf{mrr}_D(S,\mathsf{L}) = \max_{p\in D} \mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$.

According to Lemma 1, we can obtain $\mathsf{mrr}_D(S,\mathsf{L})$ by computing n alternative maximum regret ratios $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ for each p in D (which are easier to be computed). Specifically, given a point p in D, we compute its $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ by formulating it as a linear programming (LP) problem [28]:

$$\begin{aligned} \max \quad & x & & \\ s.t. \quad & (p-q)\cdot u \geq x & & \forall q \in S \\ & p \cdot u = 1 & & \\ & u[j] \geq 0 & & \forall 1 \leq j \leq d \end{aligned} \tag{1}$$

where the optimal objective x^* is the desired maximum regret ratio $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ for the given p and, by Lemma 1, $\mathsf{mrr}_D(S,\mathsf{L})$ is the maximum such x^* value over all points in D.

2.6 SQL Extensions

Similar to the SQL extension for the skyline query (i.e., the `SKYLINE OF` clause in [6]), SQL's `SELECT` statement can be extended by an optional `REGRET-SET OF... WITH...` clause for the regret minimization query (RMS) as follows.

```
SELECT ... FROM ... WHERE ...
GROUP BY ... HAVING ...
REGRET-SET OF A_1 [MIN|MAX], ..., A_d [MIN|MAX]
WITH [SIZE r | ERROR ε]
ORDER BY ...
```

where $A_1,\ldots,A_d$ denote the d attributes selected by the user, e.g., HP, MPG and price. `MIN` and `MAX` specify whether a smaller or a larger value is preferable in the corresponding dimension. For example, a larger HP is preferred (`MAX` annotation) whereas a lower price is preferred (`MIN` annotation). Besides, r and ε are the parameters we constrain in RMS (see Problem 1), which represent the output size and the required maximum regret ratio, respectively. The query below is a SQL query for RMS, which finds at most r cars from a car database `CARS` with high HP, high MPG and low price.

```
SELECT *  FROM CARS
REGRET-SET OF HP MAX, MPG MAX, price MIN
WITH SIZE r
```

The semantics of `REGRET-SET OF` clause are very straightforward. The implementation of `REGRET-SET OF` clause can be encapsulated by a new logical operator in a database system, say the *regret* operator, which is typically executed after `SELECT... FROM... WHERE... GROUP BY... HAVING...` but before the `ORDER BY` clause. In other words, the implementation of existing logical operators (e.g., *scan* and *join*) of a database system does not need to be changed and we can easily integrate the *regret* operator into a traditional SQL query processor with some minor modifications on the existing parser and query optimizer. Same as most of other logical operators of a database system (e.g., *scan* and *join*), the *regret* operator can be implemented in different (physical) ways, which will be discussed shortly in Section 3.

3 RMS Algorithms

In this section, we survey the existing algorithms for RMS and they can be classified into three categories according to the dimensionality and whether they provide theoretical guarantees on the solutions, as summarized in Table 4. Specifically, when $d = 2$, RMS can be solved optimally in polynomial time. The 2-dimensional exact RMS algorithms are presented in Section 3.1. The heuristic-based algorithms and the algorithms with theoretical guarantees in d-dimensional spaces are discussed in Section 3.2 and Section 3.3, respectively. Finally, a theoretical comparison among all existing RMS algorithms is provided in Section 3.4 while the experimental comparison appears later in Section 5.

Recall that an algorithm can control either the maximum output size r or the required maximum regret ratio ε (or both) for solving RMS. Let r_ε denote the smallest size of any solution set in the dataset whose maximum regret ratio is at most ε and ε_r denote the smallest maximum regret ratio of any solution set in the dataset with at most r points.

3.1 2-dimensional Approaches

In this section, we present the algorithms, which solve RMS optimally in 2-dimensional spaces (i.e., $d = 2$). Some existing algorithms are properly renamed to avoid confusion.

2d-SweepDP (denoted as 2d-kRMS in [11]). Chester et al. [11] offered the first exact algorithm for RMS in 2-dimensional spaces. Specifically, they worked on a *dual* space where each point in D is represented by a line and then, they showed that solving RMS in the original space is equivalent to finding a subset of lines in the dual space whose *lower envelope* is *close* to the lower envelope of the dual lines of all points in D. Note that the *lower envelope* of a set of lines in the dual space is a piecewise linear convex chain, which is a sequence of line segments with decreasing slopes where any two consecutive line segments have a common end point. Thus, the proximity between two lower envelopes can be computed by evaluating the endpoints of each line segments in lower envelopes. They proposed a plane sweeping algorithm, which computes the desired lower envelope by rotating a line L from the positive x-axis to positive y-axis. When L encounters a new intersection point (of two lines in the dual space), it checks whether the lower envelope of the current selection set of lines can be improved using dynamic programming. The optimality of 2d-SweepDP is shown as follows.

Theorem 5 ([11]) *Given an integer r, the 2d-SweepDP algorithm returns an optimal solution set of at most r points for RMS ($d = 2$) in $O(rn^2)$ time.*

For example, given a dataset D with 3 dual lines (a, f), (b, e) and (c, d) in Figure 2, their lower envelope is (a, j, h, d) (shown in red). Assume that initially, the solution set has a single line (a, f) (whose lower envelope is (a, f) itself) and the rotating line L encounters the intersection point, namely j, between (a, f) and (b, e). If we include (b, e) into the solution set, the lower envelope of the updated solution set becomes (a, j, e), which is closer to the target lower envelope (a, j, h, d). Then, the dynamic programming data structure in 2d-SweepDP will be updated by adding the line (b, e). Similar process continues until L reaches the positive y-axis.

2d-BiSearch (denoted as E-Greedy-1 in [7]). Cao et al. proposed the 2d-BiSearch algorithm [7] for solving RMS optimally, which improves the efficiency of 2d-SweepDP.

2d-BiSearch is a randomized binary search algorithm and it uses the solutions of Decision-RMS (i.e., Problem 2) as subroutines. Specifically, given the maximum output size r, it maintains a finite number of candidate values of the optimal ε and determines the smallest possible value of ε such that there is a solution whose size is at most r and maximum regret ratio is at most ε (which is a Decision-RMS problem) by performing a binary search on different values of ε.

To solve a Decision-RMS problem, Cao et al. also transformed the dataset D into a set of lines in a dual space and solved it in a geometric way. Specifically, given a point p in D, they defined a dual line in the parametric form $f_p(\lambda) = p[1]\lambda + p[2](1-\lambda)$ with $\lambda \in [0, 1]$. Given a set $S \subseteq D$, the *upper envelope* of S in the dual space can be expressed as $\max_{p\in S} f_p(\lambda)$ for $\lambda \in [0, 1]$. Then, given a real value ε and an integer r, it solves the Decision-RMS problem by computing a set S of at most r points such that the upper envelop of S lies entirely *above* the *scaled* upper envelop of D in the dual space where the scaling factor is $1-\varepsilon$ (i.e., $\max_{p\in S} f_p(\lambda) \geq \max_{p\in D}(1-\varepsilon) f_p(\lambda)$ for $\lambda \in [0, 1]$). The main result of 2d-BiSearch is shown in the following.

Theorem 6 ([7]) *Given an integer r, the 2d-BiSearch algorithm returns an optimal solution set of at most r points for RMS ($d = 2$) in $O(n \log n)$ time.*

To illustrate, assume that there are four dual lines of D in Figure 3. The upper envelope of D is shown in solid red while the $(1-\varepsilon)$-scaled upper envelop of D is drawn in dashed red, which lies entirely below the line (a, b) in Figure 4. If p is the corresponding point of (a, b) in the original space, $S = \{p\}$ is a valid solution for this Decision-RMS.

2d-GraphDP (denoted as 2d-RRMS in [4]). Asudeh et al. [4] transformed RMS in a 2-dimensional dataset into a path search problem in a weighted complete graph $G = (V, E)$ where V is the set of all skyline points $p_1, p_2, \ldots, p_{s-1}, p_s$ in D and two dummy points p_0 and p_{s+1}, sorted in the "clockwise" order and E is the set of edges between every pair of points in V. In particular, for each edge e_{ij} between p_i and p_j in E, the edge weight, denoted by $w_{i,j}$, is defined to be the regret ratio of removing all skyline points between p_i and

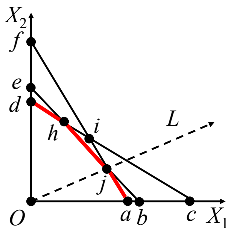
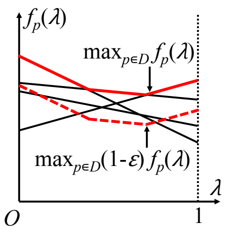
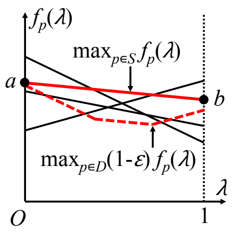


Fig. 2 2d-SweepDP Example



$f_p(\lambda)$
$\max_{p\in D} f_p(\lambda)$
$\max_{p\in D}(1\text{-}\varepsilon) f_p(\lambda)$
O
1
λ

Fig. 3 Upper Envelope



$f_p(\lambda)$
$\max_{p\in S} f_p(\lambda)$
a
b
$\max_{p\in D}(1\text{-}\varepsilon) f_p(\lambda)$
O
1
λ

Fig. 4 2d-BiSearch Example

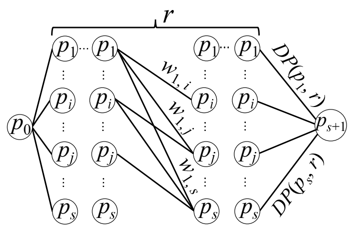


Fig. 5 2d-GraphDP Example

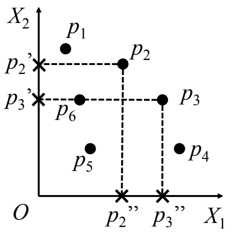


Fig. 6 Orthotope Set

	Algorithm	Output Size	Maximum Regret Ratio	Time Complexity	Remark	Related Material
2-Dimensional Exact Algorithms	2d-SWEEPDP [11]	r	ε_r	$O(rn^2)$		Theorem 5
	2d-BISEARCH [7]			$O(n \log n)$		Theorem 6
	2d-GRAPHDP [4]			$O(rs \log s \log c)$	s is # of skyline points, c is # of convex hull points	Theorem 7
d-Dimensional Heuristic Algorithms	GREEDY [28]	$\lvert S\rvert = r$ or $\mathsf{mrr}_D(S,\mathsf{L}) \le \varepsilon$		$O(nr^2)$		
	IMPGREEDY [44]			$O(nr^2)$	support pruning	Lemma 2&3
	GEOGREEDY [31]			$O(nr^{O(d)})$		Lemma 4
	STOREDLIST [31]			$O(r)$	require pre-processing	
d-Dimensional Theoretical Algorithms	CUBE [28]	r	$O(r^{-1/(d-1)})$	$O(n)$		Theorem 8
	ε-KERNEL [2,7]	$O(\frac{1}{\varepsilon^{(d-1)/2}})$	ε	$O(n+\frac{1}{\varepsilon^d})$	there is a large hidden constant in big-O notations	Theorem 9
		r	$O(r^{-2/(d-1)})$	$O(n+r^{2d/(d-1)})$		
	SPHERE [44]	r	$O(r^{-2/(d-1)})$	$O(nr^2)$		Theorem 10
	HITTINGSET [2,21]	$O(r_\varepsilon)$ for $d \le 3$ and $O(r_\varepsilon \log r_\varepsilon)$ for $d \ge 4$	$(1-\gamma)\varepsilon+\gamma$	$O(n+\frac{1}{\gamma^{d-1}}+\frac{\log^2 \frac{1}{\gamma}}{\gamma^{3(d-1)/2}})$	γ is a user controlled parameter ($0 \le \gamma \le 1$)	Theorem 11
		$O(r)$ for $d \le 3$ and $O(r \log r)$ for $d \ge 4$	$(1-\gamma)\varepsilon_r+\gamma$	$O(n+\frac{1}{\gamma^{d-1}}+\frac{\log^3 \frac{1}{\gamma}}{\gamma^{3(d-1)/2}})$		
	DMM [4]	r	$c\varepsilon_r+(1-c)$	$O(\log(n\gamma^d)\cdot(n\gamma^d +(2^{\min\{2^{\gamma m},n\}}\gamma^d)))$	γ, c are user controlled parameters ($0 \le c \le 1$, $\gamma \ge 1$)	Theorem 12
		$rd \log \gamma$	$c\varepsilon_r+(1-c)$	$O(2n\gamma^d \log(n\gamma^d))$		

Table 4 Summary of Existing RMS Algorithms in terms of (1) Output Size, (2) Maximum Regret Ratio and (3) Time Complexity

p_j. Then, given the output size r, the goal is to find a path from p_0 to p_{s+1} with at most r intermediate points whose subscripts are in an increasing order so that the maximum of the edge weights is minimized, which can be efficiently computed based on dynamic programming (see Figure 5).

Formally, let $DP(p_i, r')$ be the optimal solution of a path starting from p_i to p_{s+1} with at most $r' \le r$ intermediate points which minimizes the maximum edge weights. Clearly, $DP(p_0, r)$ is the desired solution for RMS. The recursive formula for the dynamic programming is given as follows:

$$DP(p_i, r') = \min_{j>i}\{\max\{w_{i,j}, DP(p_j, r'-1)\}\}$$

where $DP(p_i, 0)$ is initialized to be $w_{i,s+1}$. Note that the pairwise regret ratios (i.e., the edge weights in $G = (V, E)$) are efficiently computed in [4] by simply checking the end points of each edge (instead of solving the LPs in Section 2). The performance of 2d-GRAPHDP is summarized as follows.

Theorem 7 **([4])** *Given an integer r, the 2d-GRAPHDP algorithm returns an optimal set of at most r points for RMS ($d = 2$) in $O(rs \log s \log c)$ time where s is the number of skyline points in D and c is the number of points in D, which are also on the boundary of the convex hull of D (i.e., the smallest convex set containing D).*

3.2 d-dimensional Heuristic Approaches

In this section, we summarize the heuristic-based approaches for RMS in d-dimensional spaces, including the Linear Programming (LP) algorithms, namely GREEDY [28] and IMPGREEDY [44], and the geometric algorithms, namely GEOGREEDY [31] and STOREDLIST [31]. Note that all heuristic algorithms presented here mainly differ in implementations and thus, they produce exactly the same solution sets.

Greedy [28]. GREEDY is the first heuristic algorithm for RMS, which performs well by returning a set S with a small maximum regret ratio empirically. Initially, S can be initialized to be the point with the highest first dimensional value [28] (or d points where the i-th point has the highest i-th dimensional value [31]). Then, GREEDY iteratively adds more points into S until $|S| = r$ or $\mathsf{mrr}_D(S,\mathsf{L}) \le \varepsilon$ (depending on which parameters we are controlling). At each iteration, the point in D that *realizes the current maximum regret ratio* $\mathsf{mrr}_D(S,\mathsf{L})$ is included into the current set S. We say that a point q *realizes* the maximum regret ratio $\mathsf{mrr}_D(S,\mathsf{L})$ if $\mathsf{mrr}_D(S,\mathsf{L}) = \mathsf{mrr}_{S\cup\{q\}}(S,\mathsf{L})$ [28]. Such a point q is determined by computing $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ using the LP (1) for each $p \in D$ and then, we have $q = \arg\max_{p\in D} \mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$.

Itr	S	$\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$					
		p_1	p_2	p_3	p_4	p_5	p_6
1	$\{p_4\}$	**0.80**	0.78	0.67	0	0	0.67
2	$\{p_1,p_4\}$	0	**0.20**	0.20	0	0	0
3	$\{p_1,p_2,p_4\}$	0	0	**0.13**	0	0	0
4	$\{p_1,p_2,p_3p_4\}$	0	0	0	0	0	0

Table 5 Greedy & ImpGreedy Example

Example 1 Consider our car database in Table 1. Assume that we want a solution set $S\subseteq D$ with $\mathsf{mrr}_D(S,\mathsf{L})=0$. We illustrate how GREEDY works in Table 5 (where each cell contains a maximum regret ratio $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$). Assume that S is initialized to be $\{p_4\}$ which is the point with the highest first dimensional value. Then, in the first iteration, p_1 is the point realizing the current maximum regret ratio since $p_1=\arg\max_{p\in D}\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ (shown in bold) and p_1 is inserted to S. This process continues until we find that $\mathsf{mrr}_D(S,\mathsf{L})=0$ after 4 iterations and $S=\{p_1,p_2,p_3,p_4\}$. □

ImpGreedy [44,34]. To determine the point realizing the current maximum regret ratio, GREEDY solves the LP (1) for each point in D in every iteration, which is very expensive. IMPGREEDY overcomes this deficiency by identifying the unnecessary LP computations and thus, speeds up the overall process. Specifically, it develops the following punning strategies for reducing the LP computations:

1. **Upper bounding:** Since we want the point with the largest $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ in every iteration, IMPGREEDY maintains an upper bound of $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ for each p during the computation. If the bound is at most the largest maximum regret ratio observed so far, we skip the exact computation of $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ since p cannot be the point to be included. Formally, given a p in D, the upper bound of $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ is presented in the following lemma.

   Lemma 2 ([44,34]) *Given a set S and a point p in D,* $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})\le\mathsf{mrr}_{S'\cup\{p\}}(S',\mathsf{L})$ *where* $S'=S\setminus\{q\}$ *and q is the last point added to S in previous greedy process.*

2. **Invariant checking:** The LP solutions obtained in previous iterations can be re-used directly for computing the $\mathsf{mrr}_{S\cup\{p\}}(S)$ in the current iteration if certain conditions are satisfied. Formally, the lemma is shown as follows.

   Lemma 3 ([44]) *Given a set S and a point p in D, we have* $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})=\mathsf{mrr}_{S'\cup\{p\}}(S',\mathsf{L})$ *if* $(p-q)\cdot u_q\ge\mathsf{mrr}_{S'\cup\{p\}}(S',\mathsf{L})$ *where* $S'=S\setminus\{q\}$*, q is the last point added to S in previous greedy process and* u_q *is the utility vector such that* $\mathsf{rr}_{S'\cup\{p\}}(S',u_q)=\mathsf{mrr}_{S'\cup\{p\}}(S',\mathsf{L})$.

Example 2 Let S_i be the solution set obtained in the i-th iteration in Table 5. According to Lemma 2, we know that $\mathsf{mrr}_{S_i\cup\{p\}}(S_i,\mathsf{L})\le\mathsf{mrr}_{S_{i-1}\cup\{p\}}(S_{i-1},\mathsf{L})$. It conforms with our computations in Table 5 where maximum regret ratios in the same column are non-increasing from top to bottom.

We show how LP computations are reduced in IMPGREEDY. Assume that points in Table 5 are processed from left to right in each iteration of IMPGREEDY. In the first iteration, we computed the maximum regret ratios for all points in D. Just before we process p_5 in the second iteration, we know that the upper bound of $\mathsf{mrr}_{S_2\cup\{p_5\}}(S_2,\mathsf{L})$ is $ub=\mathsf{mrr}_{S_1\cup\{p_5\}}(S_1,\mathsf{L})=0$ and the largest maximum regret ratio observed so far is $mrr^*=\max_{i\in[1,4]}\mathsf{mrr}_{S_2\cup\{p_i\}}(S_2,\mathsf{L})=0.2$. Since $ub<mrr^*$, we can directly conclude that p_5 cannot be the point with the largest maximum regret ratio and skip its LP computation. Similarly, some other LP computations can also be skipped in IMPGREEDY and they are shown shaded in Table 5. □

Note that Qiu et al. [34] also considered a variation of IMPGREEDY by applying a randomized sampling on D before performing the greedy selection to further reduce the number of LP computations. However, they sacrificed the quality of the solution set (e.g., the maximum regret ratio) for better efficiency and there is no theoretical guarantee on the quality of the solution set provided in [34].

GeoGreedy [31]. GEOGREEDY follows the same framework as that in GREEDY. However, it differs from GREEDY by computing $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ using the *critical ratio* of p (whose formal definition is given shortly) instead of the LP (1).

Before we introduce the critical ratio, we present some terminology first. For each point $p\in D$, we define the *orthotope set* of p [31], denoted by $\mathsf{Orth}(p)$, to be a set of 2^d d-dimensional points constructed by $\{0,p[1]\}\times\{0,p[2]\}\times\ldots\times\{0,p[d]\}$. That is, for each $i\in[1,d]$, the i-dimensional value of a point in $\mathsf{Orth}(p)$ is equal to either 0 or $p[i]$. Given a set $S\subseteq D$, we define the orthotope set of S, denoted by $\mathsf{Orth}(S)$, to be $\bigcup_{p\in S}\mathsf{Orth}(p)$ and we let $\mathsf{Conv}(S)$ be the *convex hull*, the smallest convex set, of the orthotope set of S.

Definition 3 ([31]) Given a set $S\subseteq D$ and a point $p\in D$, the critical ratio of p w.r.t. S, denoted by $\mathsf{cRatio}(S,p)$, is defined to be $\min\{\frac{\|p'\|}{\|p\|},1\}$, where p' is the intersection between the ray shooting from O to p and the surface of $\mathsf{Conv}(S)$.

The following lemma shows that the definition of critical ratio $\mathsf{cRatio}(S,p)$ is closely related to $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ and thus, we can compute $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})$ in a geometric way (i.e., by computing the critical ratio using a ray intersection).

Lemma 4 ([31]) $\mathsf{mrr}_{S\cup\{p\}}(S,\mathsf{L})=1-\mathsf{cRatio}(S,p)$.

Example 3 Consider our running example in Table 1 where $D=\{p_1,p_2,p_3,p_4,p_5,p_6\}$. For the ease of presentation, we normalize HP/MPG to (0,1] and visualize the points in Figure 6 where the X_1 and X_2 coordinate represent HP and MPG, respectively. The orthotope set $\mathsf{Orth}(p_2)=\{p_2,\,p_2',\,p_2'',\,(0,0)\}$ is shown in Figure 6 where $p_2'=(0,p_2[2])$ and

$p''_2 = (p_2[1], 0)$. Similarly, $\mathsf{Orth}(p_3)$ is shown in the same figure. Given $S = \{p_2, p_3\}$, we define $\mathsf{Orth}(S)$ to be $\mathsf{Orth}(p_2) \cup \mathsf{Orth}(p_3)$. The convex hull $\mathsf{Conv}(S)$ is shown in Figure 7. Given p_1 and $S = \{p_2, p_3\}$, the intersection between Op_1 and the surface of $\mathsf{Conv}(S)$ is denoted by p'_1. By Lemma 4, $\mathsf{mrr}_{S\cup\{p_1\}}(S, \mathsf{L}) = 1 - \mathsf{cRatio}(S, p_1) = \frac{\|p'_1\|}{\|p_1\|} = 0.9$. □

StoredList [31]. STOREDLIST, proposed by Peng et al. [31], is a materialized version of GEOGREEDY. Specifically, it pre-computes a set of candidate points for RMS, called *happy points*, in D, based on which it runs GEOGREEDY and materializes the results. Then, when the user issues a query, RMS can be answered efficiently with the materialized results.

3.3 d-dimensional Theoretical Approaches

In this section, we summarize the d-dimensional algorithms ($d \geq 2$), namely CUBE [28], ε-KERNEL [2,7], SPHERE [44], HITTINGSET [2,21] and DMM [4], which provide theoretical guarantees on the solutions returned for RMS.

Cube [28]. CUBE is the first algorithm which provides a provable theoretical guarantee on solutions returned for RMS. Specifically, after some initialization steps, CUBE constructs the solution set S by first, dividing the data space into multiple hypercubes based on the first $d-1$ dimensions of the data space, and second, picking a point from each hypercube, which has the largest d-dimensional value in that hypercube and inserting that point into S. Since CUBE picks one point from each hypercube, the number of hypercubes constructed in CUBE has to be determined appropriately according to the maximum output size. For example, in the 3-dimensional example in Figure 8, the data space is divided into four hypercubes based on the first two dimensions and the points, namely s_1, s_2, s_3 and s_4, which have the largest third dimensional value in each hypercube, are inserted to the solution set S. According to the construction above, no matter which hypercube the user's favorite point is in, there is a point p in S which is in the same hypercube and thus, the utility of p is close to the utility of the user's favorite point. For example, if a user's favorite point is p^* as indicated in Figure 8, there exists a point, say s_1, which is in S and is in the same hypercube as p^*. Thus, s_1 has its utility close to the utility of p^*. Since s_1 has been included into S, the user will be satisfied with S and we can bound the regret ratio. Formally, we provide its theoretical guarantee as follows.

Theorem 8 ([28]) *Given an integer r,* CUBE *returns a set S of at most r points such that* $\mathsf{mrr}_D(S, \mathsf{L}) \leq \frac{d-1}{\lfloor r-d+1 \rfloor^{\frac{1}{d-1}} + d - 1}$. *Specifically, for a fixed dimensionality,* $\mathsf{mrr}_D(S, \mathsf{L}) = O(r^{-\frac{1}{d-1}})$.

ε-Kernel [2,7]. ε-KERNEL improves the upper bound in CUBE by utilizing the concept of "ε-kernel", which was first introduced by Agarwal et al. in [1]. Specifically, a set $S \subseteq D$ is said to be an ε-kernel of D if $\frac{\max_{p\in S} v\cdot p - \min_{p\in S} v\cdot p}{\max_{p\in D} v\cdot p - \min_{p\in D} v\cdot p} \geq 1 - \varepsilon$ for each non-zero vector v. Intuitively, an ε-kernel of D preserves the "width" of D for each direction; e.g., Figure 9 shows a set D (dot points), its ε-kernel S (points enclosed by circles), the width of D (denoted by w) and the width of S (which is $(1-\varepsilon)w$) along one particular direction v.

It was shown in [2,7] that the definition of ε-kernel is closely related to RMS. Specifically, if S is an ε-kernel of D, $\mathsf{mrr}_D(S, \mathsf{L}) \leq \varepsilon$, which indicates that S can be returned as a solution for RMS. Moreover, it is well-known that one can compute an ε-kernel of size $O(\varepsilon^{-\frac{d-1}{2}})$ according to the procedure in [1,45]. The following theorem follows directly.

Theorem 9 ([2,7]) *Given a real value* $\varepsilon > 0$, *one can compute a set* $S \subseteq D$ *of size* $O(\varepsilon^{-\frac{d-1}{2}})$ *with* $\mathsf{mrr}_D(S) \leq \varepsilon$.

Cao et al. [7] translated Theorem 9 to an approximate algorithm for RMS for obtaining an ε-kernel of at most r points. This is done by setting a proper value of ε in Theorem 9. The result is summarized in the following corollary.

Corollary 2 ([7]) *Given an integer r, one can compute a set* $S \subseteq D$ *of at most r points with* $\mathsf{mrr}_D(S) = O(r^{-\frac{2}{d-1}})$.

Combining the results above with the lower bounds presented in Section 2, ε-KERNEL is the first *asymptotical optimal* algorithm for RMS. Another advantage of ε-KERNEL is that it allows for maintaining the solution efficiently when the dataset is changed by point insertions and deletions without building the entire solution from scratch. However, the hidden constant behind the big-O notations of ε-KERNEL is extremely large (see a more detailed discussion in [44]), making it difficult to be applied in real scenarios.

Sphere [44]. Recently, SPHERE, which is also an asymptotical optimal algorithm for RMS, was proposed by Xie et al. [44] to reduce the hidden constant in ε-KERNEL. The core of SPHERE constructs a small set of "representative" utility functions in L and then, includes the points in D with high utilities w.r.t. those utility functions into the solution set.

Formally, SPHERE computes a small set U of utility vectors such that for each utility vector u in L, there is a utility vector in U, denoted by u', and $\mathsf{dist}(u, u') \leq \delta$ where $\mathsf{dist}(u, u')$ denotes the Euclidean distance between u and u', and δ is a non-negative similarity threshold (i.e., u is similar to u'). Intuitively, U can be regarded as a representative set of utility vectors that are *uniformly* distributed in the utility space such that for *any* utility vector u in L, there is a utility vector in U, which u is similar to. Then, for each utility vector u' in U, SPHERE searches its *D-basis* (to be defined shortly) in D, which is then included into the solution set S.

Given $B \subseteq D$ and a vector u' in U, we define the distance between B and u', denoted by $\mathsf{dist}(B, u')$, to be the minimum distance between the endpoint of vector u' and a point in the *convex hull* of B. Then, we define the "D-basis" as follows.

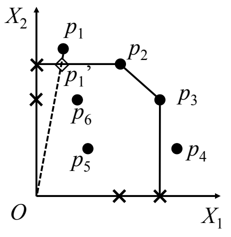


Fig. 7 Critical Ratio

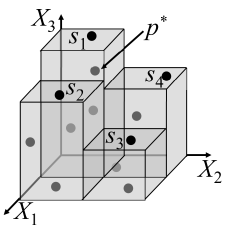


Fig. 8 Cube Example

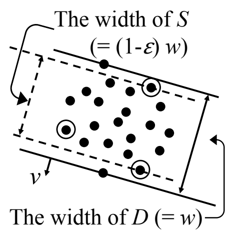


Fig. 9 ε-Kernel Example

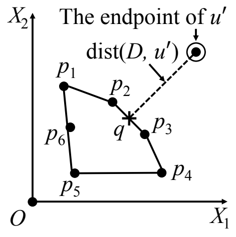


Fig. 10 Sphere Example

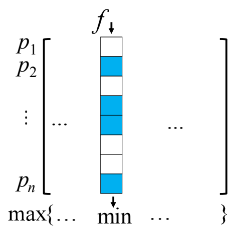


Fig. 11 DMM Example

Definition 4 ([44]) Given a set $B \subseteq D$ and a utility vector u' in U, B is said to be a D-basis of u' if (1) for each proper subset B' of B (i.e., $B' \subset B$), we have $\mathsf{dist}(B,u') < \mathsf{dist}(B',u')$ and (2) we have $\mathsf{dist}(B,u') = \mathsf{dist}(D,u')$.

Intuitively, the D-basis of u' is a minimal subset of D whose distance to u' is equal to the distance between D and u'. For example, consider the car database in Figure 10 where the endpoint of a vector u' in U is indicated. The distance between D and u', $\mathsf{dist}(D,u')$, is drawn in dashed, which is the minimum distance between the endpoint of u' and a point in the *convex hull* of D (drawn in solid lines). Point q, represented by a cross point, is the point in the convex hull of D achieving such minimum distance. The D-basis of u' is $B = \{p_2, p_3\}$ since $\mathsf{dist}(B,u') = \mathsf{dist}(D,u') = \mathsf{dist}(\{q\},u')$ and, for each $B' \subset B$, $\mathsf{dist}(B,u') < \mathsf{dist}(B',u')$ (i.e., $\mathsf{dist}(B,u') < \mathsf{dist}(\{p_2\},u')$ and $\mathsf{dist}(B,u') < \mathsf{dist}(\{p_3\},u')$).

It was shown in [44] that, given a u' in U, the points in the D-basis of u' has high utilities w.r.t. u'. For each utility vector u in L, since the D-basis of u' has been included into the solution set S and u and u' are similar, the points in S also have high utilities w.r.t. u and thus, the regret ratio can be bounded. Formally, we have the following theorem.

Theorem 10 ([44]) *Given an integer r,* SPHERE *returns a set S of at most r points such that* $\mathsf{mrr}_D(S,\mathsf{L}) \leq$

$$\min\left\{1-\frac{1}{d}, \frac{(d-1)d}{\max\left\{1/4, \left\lfloor\left(\frac{r-d}{d^2}\right)^{\frac{1}{d-1}}\right\rfloor^2\right\}+(d-1)d}\right\}$$

Specifically, for a fixed dimensionality, $\mathsf{mrr}_D(S,\mathsf{L}) = O(r^{-\frac{2}{d-1}})$.

HittingSet [2,21]. RMS was first formulated as a hitting set problem in [2]. Specifically, given the set D, [2] constructs a set system (or a range system) $\sum = (D,\mathsf{R})$ where R is a family of subsets of D. Each subset R in R is created based on a particular utility function f in L and R is defined to be $\{q \in D \mid f(q) \geq (1-\varepsilon)\max_{p\in D} f(p)\}$. That is, the utility of any point in R is at least $(1-\varepsilon)$ of the utility of user's (whose utility function is f) favorite point in the whole D.

To illustrate, assume that ε is set to be 0.1 and we construct the set system $\sum = (D,\mathsf{R})$ based on the dataset D and three particular functions $f_{0.4,0.6}$, $f_{0.2,0.8}$ and $f_{0.7,0.3}$ shown in Table 1. Take the utility function $f_{0.4,0.6}$ as an example. We define the set $R_{0.4,0.6}$ to be $\{q \in D \mid f_{0.4,0.6}(q) \geq (1-\varepsilon)\max_{p\in D} f_{0.4,0.6}(p)\} = \{q \in D \mid f_{0.4,0.6}(q) \geq 0.9 \times 86.4 = 77.76\} = \{p_3, p_4\}$. Similarly, we have $R_{0.2,0.8} = \{p_2, p_3\}$, $R_{0.7,0.3} = \{p_3, p_4\}$ and thus, $\mathsf{R} = \{R_{0.4,0.6}, R_{0.2,0.8}, R_{0.7,0.3}\}$.

According to the way we define $\sum$, it can be easily verified that a set $S \subseteq D$ is a *hitting set* of $\sum$ (i.e., $S \cap R \neq \emptyset$ for all $R \in \mathsf{R}$) if and only if $\mathsf{mrr}_D(S,\mathsf{L}) \leq \varepsilon$. For example, given $\varepsilon = 0.1$, the set $S = \{p_3\}$ is a hitting set of $\sum = (D,\mathsf{R})$ where $\mathsf{R} = \{R_{0.4,0.6}, R_{0.2,0.8}, R_{0.7,0.3}\}$ (defined above) and thus, $\mathsf{mrr}_D(S, \{f_{0.4,0.6}, f_{0.2,0.8}, f_{0.7,0.3}\}) \leq \varepsilon = 0.1$. By utilizing the well-known approximate algorithm for the hitting set problem [20] and allowing approximations on both the maximum regret ratio and the output size simultaneously, HITTINGSET solves RMS by (1) sampling a finite number of utility functions in L, (2) constructing the corresponding set system and (3) solving the resulting hitting set problem. Formally, the result is summarized as follows.

Theorem 11 ([2]) *Given ε and a user-controlled parameter $0 \leq \gamma \leq 1$,* HITTINGSET *returns a set S such that* $\mathsf{mrr}_D(S,\mathsf{L}) \leq (1-\gamma)\varepsilon + \gamma$ *and $|S| = O(r_\varepsilon)$ for $d \leq 3$ and $|S| = (r_\varepsilon \log r_\varepsilon)$ for $d \geq 4$ where r_ε is the smallest size of any solution set in the dataset whose maximum regret ratio is at most ε.*

Note that the bound $(1-\gamma)\varepsilon + \gamma$ on the maximum regret ratio in Theorem 11 can be made arbitrarily close to ε by increasing the execution time (i.e., sampling more utility functions). Kumar et al. [21] improved the execution time of HITTINGSET by applying it on a pre-computed ε-kernel of D. Besides, Agarwal et al. [2] extended the HITTINGSET algorithm to find a solution set for RMS with size at most $cr\log r$ (for a given output size constraint r) where c is an appropriate constant by running HITTINGSET multiple times in a binary search manner on different values of maximum regret ratios. Formally, the result is summarized below.

Corollary 3 ([2]) *Given r and a user controlled parameter $0 \leq \gamma \leq 1$,* HITTINGSET *returns a set S of points such that* $\mathsf{mrr}_D(S,\mathsf{L}) \leq (1-\gamma)\varepsilon_r + \gamma$ *and $|S| = O(r)$ for $d \leq 3$ and $|S| = (r\log r)$ for $d \geq 4$ where ε_r is the smallest maximum regret ratio of any set in the dataset with at most r points.*

DMM [4]. DMM works similarly as HITTINGSET by discretizing the utility space based on a user-controlled parameter and formulating RMS as a *matrix min-max* problem [36]. Specifically, consider a matrix M where each row corresponds to a point in D and each column corresponds to a utility function in L (see Figure 11 as an example). Each cell $M[p,f]$ of the matrix is the regret ratio of p w.r.t. f. Given a set S of r points and a utility function f, the regret ratio $\mathsf{rr}_D(S,f)$ can be computed to be the minimum value (among the selected r rows of points in S, shown in shaded in Figure 11) on the corresponding column of f, and the maximum regret ratio $\mathsf{mrr}_D(S,\mathsf{L})$ is estimated to be the maximum assigned regret ratio among all columns in M. Then, RMS is transformed to a min-max problem on M, which can be solved as a number of set-cover problems in a binary search manner. Its theoretical performance is shown as follows.

Theorem 12 ([4]) *Given r and a user-controlled parameter* $\alpha \in [0, \frac{\pi}{2}]$, DMM *returns a set S of at most r points such that* $\mathsf{mrr}_D(S,\mathsf{L}) \leq c\varepsilon_r + (1-c)$ *where* ε_r *is the smallest maximum regret ratio of any solution set in the dataset with at most r points,* $c = \frac{\cos(\alpha'/2)\cos(\pi/4)}{\cos(\pi/4-\alpha'/2)}$ *and* $\alpha' = 2\arcsin(\sqrt{\frac{1-\cos^{d-1}\alpha}{2}})$.

While DMM runs in $O(n\log n)$ time in theory, its dependence on the parameter α is exponential. To improve its efficiency, we can solve the matrix min-max problem approximately by solving set-cover problems using the well-known greedy strategy, which, however, adds another level of approximation and increases the output size by a log-factor.

3.4 Theoretical Comparison

After surveying different RMS algorithms, we provide a brief theoretical comparison among them. In particular, in addition to the complexity analysis shown in Table 4, we also consider the following aspects which were considered in the literature [7,44] for evaluating the theoretical performance of an algorithm A for RMS (see the summary in Table 6):

– *Deterministic?* Algorithm A is a deterministic algorithm.
– *Has Bounds?* Algorithm A provides theoretical bounds on the size/maximum regret ratio of the returned set.
– *Restriction-free MRR bound?* The definition of restriction-free MRR bound was first proposed in [44]. Specifically, it means that when there is a bound on the maximum regret ratio, there is no restriction on the bound. Recall that the maximum regret ratio is a real value between 0 and 1. If the bound of the maximum regret ratio of the solution set returned by A is in the range between 0 and 1 for *any* setting, we say that algorithm A has a restriction-free MRR bound. Otherwise, the bound is in the range between 0 and 1 in some *restricted* cases and thus, we say that A does not have a restriction-free MRR bound.
– *Asymptotically Optimal?* Algorithm A returns an asymptotically optimal solution for RMS.
– *Optimal?* Algorithm A returns an optimal solution.
– *Arbitrary Dimensionality?* Algorithm A could be executed on datasets with an arbitrary dimensionality.
– *Parameter-free?* Algorithm A does not require users to specify additional parameters for executing the algorithm.

All aspects are important to RMS since (1) a deterministic algorithm could be more desirable than a randomized algorithm in some applications since it returns stable solutions; (2) an algorithm which returns a solution with theoretical bounds is more useful than an algorithm which does not. In particular, the tighter the bound, the more desirable the algorithm. For example, an optimal algorithm is better than an asymptotically optimal one, which is then better than a theoretically bounded (but not asymptotically optimal) algorithm; (3) an algorithm which does not have a restriction-free MRR bound may give an *invalid* bound (e.g., a bound greater than 1) on the maximum regret ratio, which implies that this algorithm does not have a useful bound since the maximum regret ratio itself is a real number from 0 to 1; (4) an algorithm which could not be executed on datasets of some dimensionalities could have limited generality; and (5) a parameter-free algorithm is user-friendly since setting appropriate parameters requires additional user effort.

Consider the comparison summarized in Table 4 and Table 6. Due to the NP-harness of the problem, only the 2-dimensional exact algorithms returns optimal solutions for RMS. Among them, 2d-BISEARCH has a clearly better time complexity ($O(n\log n)$) than 2d-SWEEPDP ($O(rn^2)$) while 2d-GRAPHDP is the best algorithm when the number of skyline/convex hull points in the dataset is much smaller than the dataset size. Unfortunately, the 2-dimensional exact algorithms are restricted when the datasets have two attributes only. In contrast, the heuristic algorithms can be executed on datasets of any dimensionality. However, they fail to provide any theoretical guarantee on the solutions. Among them, GREEDY and IMPGREEDY scale better than GEOGREEDY, whose performance degrades when the dimensionality is large due to its exponential dependency on d. Finally, among all d-dimensional theoretical algorithms, CUBE has the smallest time complexity since it constructs the solution set by scanning the database once while the time complexities of most of the other theoretical algorithms exponentially depend on d. Meanwhile, CUBE is the first theoretically bounded algorithm for RMS, whose bound is improved later by SPHERE and ε-KERNEL. Although both SPHERE and ε-KERNEL provide asymptotically optimal guarantees on the solutions, the large hidden constant in the bound of ε-KERNEL prohibits it from being a restriction-free algorithm. Moreover, in practice, ε-KERNEL and HITTINGSET are usually implemented in a randomized manner. HITTINGSET and DMM are not parameter-free algorithms since they require additional parameters from users and they relax both the output size and maximum regret ratio simultaneously.

	Algorithm	Determin-istic?	Has Theoretical Guarantees?				Arbitrary Dimen-sionality?	Parameter-free
			Has Bounds?	Restriction-free MRR Bound?	Asymptotically Optimal?	Optimal?		
2-Dimen-sional Exact Algorithms	2d-SWEEPDP [11]	✓	✓	✓	✓	✓		✓
	2d-BISEARCH [7]	✓	✓	✓	✓	✓		✓
	2d-GRAPHDP [4]	✓	✓	✓	✓	✓		✓
d-Dimen-sional Heuristic Algorithms	GREEDY [28]	✓					✓	✓
	IMPGREEDY [44]	✓					✓	✓
	GEOGREEDY [31]	✓					✓	✓
	STOREDLIST [31]	✓					✓	✓
d-Dimen-sional Theoretical Algorithms	CUBE [28]	✓	✓	✓			✓	✓
	ε-KERNEL [2,7]		✓		✓		✓	✓
	SPHERE [44]	✓	✓	✓	✓		✓	✓
	HITTINGSET [2,21]		✓	✓			✓	
	DMM [4]	✓	✓	✓			✓	

Table 6 Theoretical Comparison among Existing RMS Algorithms

4 Variants

In this section, we summarize the variants of RMS studied in the literature. In particular, we present the generalized kRMS problem in Section 4.1 and RMS over non-linear utility function class in Section 4.2, which are two major variants of RMS. Other variants are shown in Section 4.3.

4.1 The kRMS Problem

A major variant of RMS is the kRMS problem proposed by Chester et al. [11], which can be regarded as a generalization of the traditional RMS problem. Denote the k-th highest utility among points in D by $k\text{-}\max_{p\in D} f(p)$. In this variant, the "regret ratio" ("maximum regret ratio") is generalized to the "k-regret ratio" ("maximum k-regret ratio").

Definition 5 ([11]) Given a set $S \subseteq D$, an integer k and a utility function f, the *k-regret ratio* of S over D w.r.t. f, denoted by $k\text{-rr}_D(S, f)$, is defined to be $\max\{0, 1 - \frac{\max_{p\in S} f(p)}{k\text{-}\max_{p\in D} f(p)}\}$.

Definition 6 ([11]) Given a set $S \subseteq D$ and a function class FC, the *maximum k-regret ratio* of S over D w.r.t. FC, denoted by $k\text{-mrr}_D(S, \mathsf{FC})$, is defined to be $\sup_{f\in \mathsf{FC}} k\text{-rr}_D(S, f)$.

Different from RMS where a user is happy with S if the highest utility in S is close to the highest utility in D, a user will be happy with S in kRMS and his/her k-regret ratio is 0 if the highest utility in S is at least the k-th highest utility in D. Similar to RMS, the goal of kRMS is to optimize the worst-case k-regret ratio, i.e., we want a set $S \subseteq D$ such that the maximum k-regret ratio $k\text{-mrr}_D(S, \mathsf{L})$ is minimized.

When k is set to be 1, kRMS is reduced to the original RMS problem. Besides, if S is a solution of RMS, it is also a solution of kRMS. However, there can be another solution for kRMS whose size and maximum k-regret ratio are much smaller. Next, we show how to extend some algorithms originally designed for RMS to find a better solution for kRMS.

2d-kRMS [11,4]. 2d-SWEEPDP [11] and 2d-BISEARCH [4] can be extended to handling kRMS in 2-dimensional spaces. Specifically, 2d-SWEEPDP can be modified to solve kRMS by finding a set S of lines in the dual space whose lower envelope (which corresponds to the top-ranked points in S) is close to *the top-k rank contour* of the dual lines of all points in D (which corresponds to the k-ranked points in D). Similarly, 2d-BISEARCH can solve kRMS optimally by determining the candidate values of the optimal ε (here, ε is the maximum k-regret ratio rather than the maximum regret ratio in RMS) implicitly based on a line sweeping algorithm since there are much more such values than those in RMS.

kRMS-Greedy [11]. Chester et al. extended GREEDY to a randomized algorithm for the more general kRMS problem. Intuitively, it decomposes each iteration in the greedy process, which identifies the point realizing the current maximum k-regret ratio, into a set of 2RMS problems and looks for a common solution. Specifically, given a utility function f, if p is the k-ranked point in D w.r.t. f, D must be able to be divided into $k-1$ partitions, namely $D_1, \ldots, D_{k-1}$, so that p is the 2-ranked point on each of these $k-1$ partitions (i.e., there is exactly one point in each partition with a higher utility than p). However, it is difficult to find such a partition without the knowledge of f. They used a random partitioning approach to construct candidate partitions. In particular, they modified the LP (1) to tell whether the partitioning is successful and whether they need to try new partitions.

kRMS-HittingSet [2,21]. HITTINGSET [2,21] can be easily extended to handling the kRMS problem by re-defining the set system $\Sigma = (D, \mathsf{R})$ where each set R in R is defined to be $\{q \in D \mid f(q) \geq (1-\varepsilon)k\text{-}\max_{p\in D} f(p)\}$. In other words, the utility of any point in the redefined set R is at least $(1-\varepsilon)$ of the k-th highest utility among all points in D. The remaining procedure of kRMS-HITTINGSET is kept unchanged.

When k is large, Kumar et al. [21] further improved the efficiency by sampling a smaller subset D' of D. It was proven in [21] that, given any function f in L, we can approximate the k-ranked point in D by the k'-ranked point in D' with a high probability where $k' \ll k$ and $|D'| \ll |D|$ so that we can solve the original kRMS problem by solving an alternative k'RMS problem with a smaller input size in a shorter time.

***k*RMS-DMM.** Similar to HittingSet [2,21], DMM [4] can also be extended to support kRMS with a minor modification. This is done by re-defining each cell $M[p, f]$ of M to be the k-regret ratio of p w.r.t. f (instead of the regret ratio). The remaining steps of kRMS-DMM are kept unchanged.

Remark. kRMS can be further generalized to the top-k RMS problem [21] where we want a set S such that the i-th highest utility in S is close to the i-th highest utility in D for *every* $i \in [1,k]$. Intuitively, the goal of top-k RMS is to find a set S approximating the top-k query well. A multi-hitting set based algorithm was proposed in [21] to solve top-k RMS.

4.2 RMS over Non-Linear Utility Functions

In Section 3, we focus on RMS where $\mathsf{FC} = \mathsf{L}$, the class of linear utility functions. Now, we relax this assumption by considering different types of non-linear utility functions.

Definition 7 (Convex Function) A function f is said to be convex over $\mathbb{R}_+$ if for all $x_1, x_2 \geq 0$ and $\lambda \in [0,1]$, we have $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$.

Definition 8 (Concave Function) A function f is said to be concave over $\mathbb{R}_+$ if $-f$ is a convex function.

Definition 9 (CES Function) A function f is said to be a Constant Elasticity of Substitution (CES) function over $\mathbb{R}_+^d$ if f is in the form $f(p) = (\sum_{i=1}^{d} a_i p[i]^b)^{\frac{1}{b}}$ where $b > 0, a_i > 0$.

Definition 10 (MUF) A function f is said to be a Multiplicative function (MUF) over $\mathbb{R}_+^d$ if f is in the form $f(p) = \prod_{i=1}^{d} p[i]^{\alpha_i}$ where each $\alpha_i \geq 0$ and $\sum_{i=1}^{d} \alpha_i \leq 1$.

Given a function f, consider the *marginal gain* on its utility $f(p)$ caused by every unit increment on a particular dimensional value of point p. If f is a linear function where $f(p) = u \cdot p = \sum u[i]p[i]$, it corresponds to a *constant* marginal gain since $f(p)$ always increases $u[i]$ units for every unit increment on the i-th dimensional value of p (i.e., $p[i]$). In comparison, non-linear functions correspond to other types of marginal gains; e.g., a convex (concave) function corresponds to an increasing (decreasing) marginal gain.

Based on the definitions above, we summarize the non-linear function classes commonly studied in the literature:

- **Convex Function Class.** FC is said to be a convex function class if $\mathsf{FC} = \{f \mid f(p) = \sum_{i=1}^{d} f_i(p[i])$ where each f_i is a *convex* function over $\mathbb{R}_+\}$. For the purpose of illustration, we stick to a particular convex function class $\mathsf{FC} = \{f \mid f(p) = \sum_{i=1}^{d} a_i p[i]^b$ where $a_i \geq 0$ and $b \geq 1\}$; e.g., $f(p) = \sum_{i=1}^{d} p[i]^2$ is in the convex function class.

FC	Lower Bound	Upper Bound	
		MinWidth [13]	MinVar [32]
Convex	$\Omega(1/r^{2b})$	$O(1/r^{\frac{1}{d-1}})$	-
Concave	$\Omega(1/r^2)$	$O(1/r^{\frac{1}{d-1}})$	-
CES ($b < 1$)	$\Omega(1/br^2)$	$O(1/br^{\frac{b}{d-1}})$	$O(1/r^{\frac{1}{d-1}})$
CES ($b \geq 1$)	$\Omega(1/br^2)$	$O(1/r^{\frac{1}{b(d-1)}})$	-
MUF	$\Omega(1/r^2)$	-	$O(\ln(1+1/r^{\frac{1}{d-1}}))$

Table 7 RMS over Non-linear Utility Function Classes

- **Concave Function Class.** FC is a concave function class if $\mathsf{FC} = \{f \mid f(p) = \sum_{i=1}^{d} f_i(p[i])$ where each f_i is a *concave* function over $\mathbb{R}_+\}$. For the purpose of illustration, we stick to a particular concave function class $\mathsf{FC} = \{f \mid f(p) = \sum_{i=1}^{d} a_i p[i]^b$ where $a_i \geq 0$ and $0 < b < 1\}$; e.g., $f(p) = \sum_{i=1}^{d} \sqrt{p[i]}$ is in the concave function class.
- **CES Function Class.** FC is said to be a CES function class if $\mathsf{FC} = \{f \mid f$ is a CES function$\}$. The CES function class is a popular function class in Economics.
- **Multiplicative Function (MUF) Class.** FC is said to be a MUF class if $\mathsf{FC} = \{f \mid f$ is a MUF$\}$. The MUF class is a function class that has more expressive power in modeling the *diminishing marginal rate of substitution (DMRS)* [41] (a popular economic concept).

Note that according to [13,32], the scale-invariance of RMS is preserved under all non-linear utility function classes defined above. In the following, we summarize the known lower bounds on RMS when considering non-linear utility function classes and the best-known algorithms proposed (both theoretical and heuristic) for solving non-linear RMS.

Lower bound. Assume that the maximum output size is fixed to be r. The authors in [13,32] derived the lower bounds on the maximum regret ratio over each of the non-linear function classes described above in 2-dimensional spaces and their main results are summarized in Table 7.

Theoretical Algorithms. Cube [28] was extended to handling non-linear function classes with provable guarantees in [13,32]. Specifically, Kessler Faulkner et al. [13] proposed MinWidth, which omits empty hypercubes in Cube so that sparse datasets can be better handled. Qi et al. [32] proposed MinVar, which performs well even when the dataset is skewed. Their corresponding bounds on the maximum regret ratio for a fixed output size r are shown in Table 7.

Heuristic Algorithms. Algorithms were also proposed for solving non-linear RMS heuristically. Specifically, Area-Greedy [13] constructs a solution iteratively by including the point that greatest increases the *area* under the current set at each iteration. Angle [13] computes a set of directions discretizing the polar space and identifies the farthest point in each direction, which is added to the solution. MaxDif [32] greedily selects points according to an upper bound on the maximum regret ratio for each point in D.

4.3 Other Variants

Many other variants of RMS were also studied in the literature and they are summarized in this section.

Interactive RMS. Nanongkai et al. [27] enhanced traditional RMS with *user interactions*. Intuitively, instead of asking the user for the exact utility functions directly, they implicitly learned the user's utility function by asking the user to provide some "hints". Specifically, at each interaction, a user is presented with a short list of points and s/he is asked to indicate the point s/he favors the most among them. Based on the user feedback, the utility function is learned implicitly and finally, the user's favorite point can be identified. User interactions are shown to be very useful in [27]: they reduce both the user regret and the output size *exponentially*. The main result known for interactive RMS is shown as follows.

Theorem 13 ([27]) *Given a real value $\varepsilon > 0$, one can guarantee an ε regret ratio by displaying $O(s \log_s \frac{1}{\varepsilon})$ points to the user where s is the number of points displayed at each interaction (i.e., the number of rounds of interactions is $O(\log_s \frac{1}{\varepsilon})$).*

In most cases, s is small and it can be regarded as a fixed constant. Compared with the traditional RMS algorithms presented in Section 3 (e.g., Theorem 9), Theorem 13 shows an exponential improvement in the output size when user interactions are allowed. Moreover, by combining Theorem 13 with the following lower bound on interactive RMS, we know that the algorithm in [27] is almost optimal.

Theorem 14 (Lower Bound on Interactive RMS[27]) *For any dimensionality d and $\varepsilon \in (0,1]$, there is a d-dimensional database such that any algorithm needs to present $\Omega(s \log_s \frac{1}{\varepsilon})$ points from the database (i.e., to interact with the user for $\Omega(\log_s \frac{1}{\varepsilon})$ rounds) to guarantee a regret ratio at most ε.*

However, [27] has two major disadvantages. Firstly, it performs poorly in the number of rounds of interactions when a user wants to find the point with a 0 regret ratio (i.e., $\varepsilon = 0$). Secondly, during interaction, it presents users with some fake/artifical points (i.e., points not in the database). Fortunately, Xie et al. [43] proposed algorithms which overcome these deficiencies. Specifically, they used a concept, called the *utility hyperplane*, to model the user preference and two effective pruning strategies to locate the user's favorite tuple in the database. Moreover, the algorithms in [43] always display true tuples in the database during interaction, and thus, they are said to be *strongly truthful* algorithms.

Average RMS. [46, 47, 33] studied the regret ratios in the average case, rather than the worst case. In this setting, it is assumed that the probability distribution of utility functions in FC is given. Then, the average regret ratio is defined to be the integral of regret ratios over this probability distribution, which is also known as the expected regret ratio. To improve the computational efficiency, they used sampling to estimate the average regret ratio, which is within an additive distance to its true value with a high probability. It was proven in [46, 47, 33] that the average regret ratio is a monotonically non-increasing supermodular set function. Thus, they find a set with small average regret ratio using the well-known greedy algorithm for minimizing a supermodular set function [18]. Specifically, the solution S is initialized to be the whole database D and then they iteratively remove points from S until there are at most r points in S. At each iteration, the point which minimizes the average regret ratio of S is removed. Unfortunately, the above algorithm is very inefficient and it has a cubic execution time in the dataset size.

Various techniques were proposed to improve its empirical performance. For example, lazy evaluations, which maintain a list of lower bounds on the average regret ratios, were considered in [33] to remove unnecessary computations. Pre-computations and re-used computations were also utilized in [47] to improve the efficiency. In particular, when only considering linear utility functions on a 2-dimensional dataset, a dynamic programming based algorithm was proposed in [47] to solve the average RMS problem optimally.

Diversified RMS. Hussain et al. [17] examined how user regret can be minimized while maximizing the *diversity* of the solution set. In their context, diversity is measured as the average distance (e.g., Euclidean distance) between every pairs of points in the returned set. They aimed at optimizing an objective function which is a linear combination of appropriately scaled diversity and regret metrics. Specifically, they proposed a greedy-based algorithm, which incrementally constructs the solution by adding one point at a time, and a swap-based algorithm, which iteratively updates the solution by swapping points to improve the objective value.

RMS in Multi-Objective Submodular Function Maximization (Multi-RMS). Instead of optimizing over a single utility function in RMS, it is assumed in [39] that there are multiple submodular objective functions in the user's mind and they studied the regret minimization in the context of multi-objective submodular function maximization (Multi-RMS). In this setting, the approximate algorithm for each single objective function maximization is taken as an input. To solve Multi-RMS, a coordinate-wise maximum method [39] was proposed to output a fixed size solution and a polytope method [39] was presented to enable users to control the output size. In particular, in the biobjective case, the polytope method provides a provable guarantee on the regret ratio which can not be improved significantly according to the lower bound $\Omega(1/r^2)$ proven for Multi-RMS in [39].

Rank RMS. While RMS measures the user regret based on the utility difference between the points in the selected set and in the whole database, Asudeh et al. [5] measured the user regret based on their rank difference, which is also known as rank RMS. In a 2-dimensional space, they proposed a 2-approximation algorithm based on angular sweeping. In a d-dimensional space, they modeled rank RMS by a geometric hitting set problem based on the *k-set enumeration* (a well-known concept in computational geometry) and solve it with a logarithmic approximation factor. A function space partition based algorithm was also proposed in [5], which provides a fixed approximation on rank RMS.

Dataset	d	$\|D\|$	$\|D_{sky}\|$
AL	2	5,810,462	37
IL	2	63,383	206
EN	5	178,080	483
NBA	6	16,916	130
HH	7	1,048,578	57
CL	9	68,040	3,460

Table 8 Real Datasets

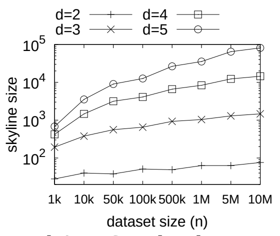


Fig. 12 Preprocessed Anti-Correlated Datasets

Candidate Set for RMS. A problem, which is orthogonal to RMS, was raised in [31], which aims at reducing the set of candidate points that we need to consider for RMS. It it well-known [32] that when we are constructing a solution set S for RMS, it suffices to consider the set of all skyline points in D, denoted by D_{sky}, since the maximum regret ratio of S will not be larger if we replace any non-skyline point p in S with a skyline point that dominates p in D_{sky}. Peng and Wong [31] further reduced the candidate set to be the set of *happy points*, denoted by D_{happy}. In particular, they proved that $D_{happy} \subseteq D_{sky} \subseteq D$ and the optimal solution of RMS must be a subset of D_{happy}, which is summarized below.

Lemma 5 ([31]) *Given an integer r, $D_{happy} \subseteq D_{sky} \subseteq D$ and there exists a set $S \subseteq D_{happy}$ such that $\mathsf{mrr}_D(S,\mathsf{L}) = \mathsf{mrr}_D(S^*,\mathsf{L})$ and $|S| = |S^*| \leq r$ where S^* is the optimal solution of RMS.*

Apart from [31], [15, 16] computed the candidate set for RMS by considering skyline points with high *priority* and *frequency*. However, these approaches are heuristic-based, and there is no known guarantees on their effectiveness.

RMS with Binary Constraints. [12] augmented traditional RMS with *binary constraints*. Examples of binary constraints include "the HP of this car is among top 10% in the database" and "it is a limousine". Heuristic algorithms were proposed in [12] to find a set with a small maximum regret ratio while maximizing the number of binary constraints it satisfies.

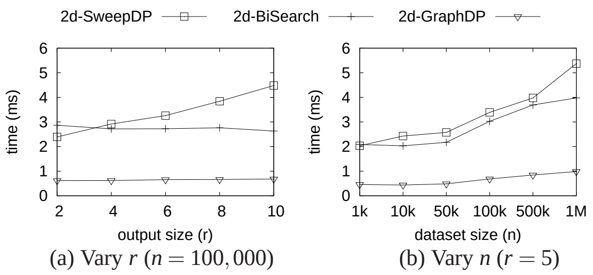
5 Experiments

We conducted experiments on a machine with 1.60GHz CPU and 8GB RAM. All programs were implemented in C/C++. Most of the experimental settings follow those in [28, 4, 44]. Both *synthetic* and *real datasets* are used in our experiments.

Synthetic datasets were generated using a dataset generator developed for skyline queries in [6]. Three types of synthetic datasets with diverse characteristics were considered: (1) anti-correlated datasets (points which are good in one attribute are bad in some of other attributes); (2) correlated datasets (points which are good in one attribute are also good in other attributes); and (3) independent datasets (all attributes are generated independently). Unless stated explicitly, for each synthetic dataset, the number of tuples is set to be 100,000 (i.e., n = 100,000). Note that anti-correlated datasets are the most interesting synthetic datasets where the skyline set is large and cannot be returned as a whole. Thus, we used anti-correlated datasets as our default synthetic datasets. Real datasets contain six datasets commonly used in existing studies [28, 4, 44, 11, 27]. *Airline (AL)* [4] and *Island (IL)* [28] are 2-dimensional datasets, containing the information of 5,810,462 flights and 63,383 geographic locations, respectively, and they are used for evaluating 2-dimensional algorithms. *EL Nino (EN)* [11] consists of 178,080 tuples with five oceanographic attributes taken at the Pacific Ocean. *NBA* [44] contains 16,916 tuples for each player/season combination from 1946 to 2009. Six attributes are selected to represent the performance of each player. *Household (HH)* [44] contains 1,048,576 family tuples with 7 attributes, showing economic characteristics of each family. *Color (CL)* [28, 27] contains the color histograms of 68,040 images. The statistics about real datasets are summarized in Table 8.

For all datasets, each attribute is normalized to (0, 1]. We preprocessed each dataset such that the preprocessed dataset contains skyline points only. The sizes of preprocessed anti-correlated datasets are shown in Figure 12. Note that some RMS algorithms constrain the output size and some other RMS algorithms constrain the maximum regret ratio during execution. Unless specified explicitly, the default output size is set to be 30 (i.e., $r = 30$) if we constrain the output size, and the default maximum regret ratio is set to be 0.05 (i.e., $\varepsilon = 0.05$) if we constrain the maximum regret ratio. The performance of each algorithm is measured by its *query time*, *output size* and *maximum regret ratio*. The query time of an algorithm is the execution time of the algorithm. The output size of an algorithm is the number of points returned by the algorithm. The maximum regret ratio of an algorithm is the maximum regret ratio of the set returned by the algorithm. Some results are plotted in log-scale for better visualization.

We compared the following three sets of algorithms. Firstly, we compared the 2-dimensional algorithms 2d-SWEEPDP [11], 2d-BISEARCH [7] and 2d-GRAPHDP [4], which solve RMS optimally. Secondly, we evaluated the d-dimensional heuristic algorithms GREEDY [28], IMPGREEDY [44] and GEOGREEDY [31]. Note that STOREDLIST [31] is a materialized version of GEOGREEDY and thus, it is excluded. Thirdly, we studied the performance of d-dimensional theoretical algorithms, which can be further divided into two sub-categories according to their primary purposes: (1) *min-*



(a) Vary r ($n = 100,000$) (b) Vary n ($r = 5$)

Fig. 13 2-dimensional Algorithms on Anti-Correlated Datasets

error algorithms which minimizes the maximum regret ratio while fixing the output size (i.e., CUBE [28], SPHERE [44] and DMM [4]) and (2) *min-size* algorithms which minimizes the output size while fixing the maximum regret ratio (i.e., HITTINGSET [2,21] and ε-KERNEL [2,7]). Note that some algorithms (e.g., HITTINGSET) could be applied in both cases (with some modifications). We postpone the detailed description of these variations to later sections. We optimize the performance of each algorithm and the parameters are set following the setting reported in existing studies.

We proceed with the experiments on synthetic and real datasets in Section 5.1 and Section 5.2. In Section 5.3, we evaluated some existing algorithms when they are extended to handling different variants of RMS. A user study about RMS can be found in Section 5.4. Finally, we summarize our findings and the empirical guideline for RMS in Section 5.5.

5.1 Results on Synthetic Datasets

5.1.1 2-Dimensional Exact Algorithms

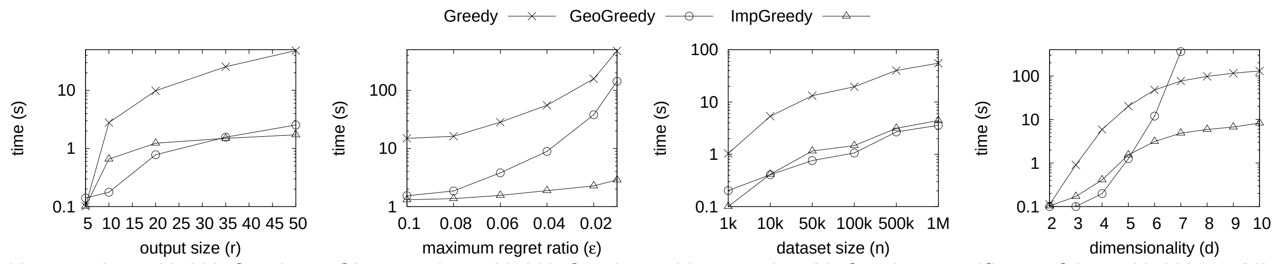
We start with the performance evaluation of 2-dimensional algorithms (2d-SWEEPDP, 2d-BISEARCH and 2d-GRAPHDP) and the results are summarized in Figure 13. Since RMS can be solved optimally in 2-dimensional spaces, all 2-dimensional algorithms produce the same solutions and thus, we only report their query times. Figure 13(a) depicts the query time by varying the output size r. All algorithms are fast and they take only a few milliseconds to return the optimal solutions. However, 2d-SWEEPDP and 2d-BISEARCH are slightly slower than 2d-GRAPHDP, which is consistent with the results reported in [4]. This is because both 2d-SWEEPDP and 2d-BISEARCH have to compute the lower/upper envelope of a given set of lines while 2d-GRAPHDP avoids the envelope computation and solves RMS from a graph perspective. We also show the query time of each 2-dimensional algorithm by varying the dataset size n in Figure 13(b). Since each point is only described by 2 attributes, all algorithms are fast and not very sensitive to the dataset size n. Similar to the result in Figure 13(a), 2d-GRAPHDP achieves the smallest running time in all cases due to its efficient computations on the regret ratios and its concise graph representation.

5.1.2 d-Dimensional Heuristic Algorithms

We studied the performance of d-dimensional heuristic algorithms in Figure 14 on 5-dimensional anti-correlated datasets. Note that all heuristic algorithms differ in implementation and they produce the same solutions. Thus, we only compared their query times (their output sizes and maximum regret ratios will be shown later). Firstly, we varied the output size r in Figure 14(a). In general, IMPGREEDY runs faster than GREEDY since IMPGREEDY avoids the unnecessary LP computations in GREEDY while guaranteeing the correctness. When r is small, GEOGREEDY is the fastest algorithm. However, when r is larger, its performance degrades and becomes slower than IMPGREEDY. This is because GEOGREEDY heavily relies on the convex hull computation to obtain the critical ratios and the next point to be included in the greedy process. Unfortunately, computing the convex hull of r points in GEOGREEDY takes $O(r^{O(d)})$ time, which is expensive for large r, while other algorithms are quadratic in r. Secondly, we proceed with the experiments by varying the maximum regret ratio ε in Figure 14(b). In most cases, when a user specifies a smaller ε, the problem becomes more challenging and an RMS algorithm needs to return more points to guarantee the required maximum regret ratio. Thus, when ε is smaller, the running times of all heuristic algorithms increase. Among all algorithms, IMPGREEDY achieves the best performance by being faster and less sensitive to ε. Thirdly, in Figure 14(c), we evaluated the scalability of the heuristic algorithms by varying the dataset size n. When there are more points in the database, both LP-based algorithms and geometric-based algorithms spend more time to execute, which conforms with human intuition. However, when considering the scalability by varying the dimensionality d in Figure 14(d), LP-based approaches (GREEDY and IMPGREEDY) scale better than the geometric-based approach, GEOGREEDY, since the time of computing convex hulls in GEOGREEDY exponentially depends on d. According to the experiments above, we obtain the following useful observations. Firstly, the geometric-based approach, i.e., GEOGREEDY does not scale well with large dimensionality and large output size since its operation is expensive in these cases. Secondly, IMPGREEDY achieves a superior performance among all heuristic algorithms with a shorter query time and better scalability in most cases since it avoids a large number of redundant computations. Motivated by this, in the rest experiments, we only compare IMPGREEDY and omit the results of other heuristic algorithms.

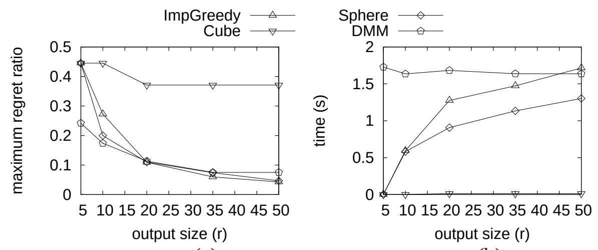
5.1.3 d-Dimensional Theoretical Algorithms

In the following, we conducted two sets of experiments for evaluating the d-dimensional theoretical algorithms. Specifically, we experimentally compared the min-error algorithms



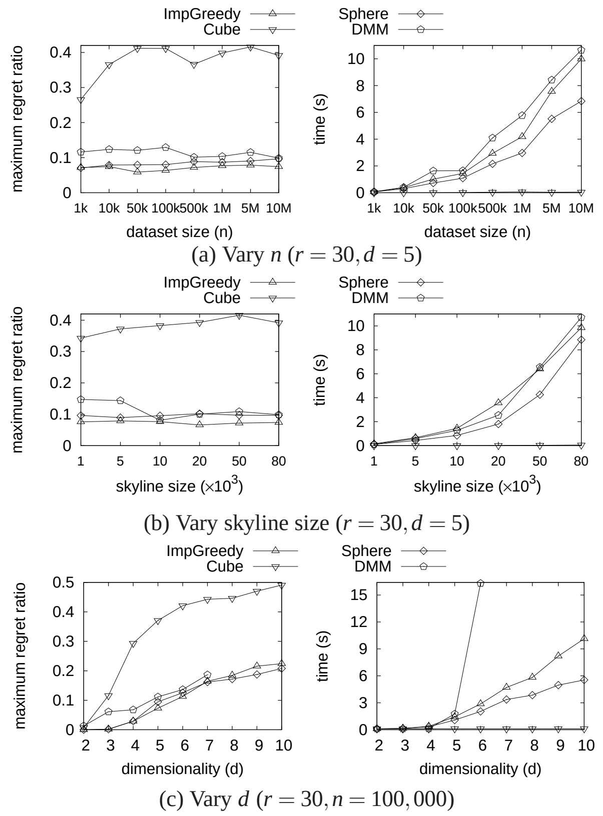
(a) Vary r ($n = 100,000, d = 5$) (b) Vary ε ($n = 100,000, d = 5$) (c) Vary n ($r = 30, d = 5$) (d) Vary d ($n = 100,000, r = 30$)

Fig. 14 Heuristic Algorithms on Anti-Correlated Datasets



(a) (b)

Fig. 15 Min-Error - Vary r on Anti-Correlated ($n = 100,000, d = 5$)



(a) Vary n ($r = 30, d = 5$)

(b) Vary skyline size ($r = 30, d = 5$)

(c) Vary d ($r = 30, n = 100,000$)

Fig. 16 Min-Error - Scalability Test on Anti-Correlated Datasets

CUBE, SPHERE and DMM (Figures 15 and 16) , and the min-size algorithms, HITTINGSET and ε-KERNEL (Figures 17 and 18). For completeness, we also compared the best heuristic algorithm, IMPGREEDY, in the experiments.

Min-Error. In Figure 15, we show the performance of the min-error algorithms by varying the output sizes r on a 5-dimensional anti-correlated dataset. Note that although HITTINGSET and ε-KERNEL, which are primarily designed as min-size algorithms, can be modified to answer min-error RMS in theory, their empirical performances on solving min-error RMS are poor due to the large running time (e.g., $O(n^d)$ in HITTINGSET) and the large maximum regret ratio (e.g., ε-KERNEL). Thus, we did not include these two variations in the figure for better visualization. We measured the maximum regret ratio in Figure 15(a). CUBE produces the worst maximum regret ratio while other algorithms return solutions with smaller maximum regret ratios. For example, when $r = 30$, the maximum regret ratio of CUBE is around 0.4, which is 4 times larger than the maximum regret ratios of other algorithms. Moreover, according to our results in Figure 15(a), none of the algorithms dominates the others in terms of maximum regret ratio. Specifically, when r is smaller, the maximum regret ratio of DMM is smaller while when r is larger, the maximum regret ratios of SPHERE and IMPGREEDY are smaller. Similarly, we plotted the running time in Figure 15(b). Although the maximum regret ratio of CUBE is large, it is extremely fast compared with other algorithms since it constructs the solution by simply scanning the database once, which can be efficiently implemented. Apart from CUBE, SPHERE is the most efficient algorithm and it is faster than both IMPGREEDY and DMM in all values of r. However, unlike CUBE which performs poorly in maximum regret ratio, the maximum regret ratio of SPHERE is not only asymptotically optimal, but also small empirically. Another interesting phenomenon that we can observe from the experiments is that it takes more time for SPHERE and IMPGREEDY to construct a solution with a larger size while the execution time of DMM is less sensitive to the output size. Specifically, SPHERE and IMPGREEDY construct solutions *incrementally*: they start with an empty solution set and construct the solution set by gradually adding more points to it. In particular, when a larger output size is required, an incremental algorithm takes more time to execute. In comparison, DMM solves RMS by solving a number of set cover problems in a binary search manner and thus, its performance is less dependent on the output size (but it is slower than SPHERE and IMPGREEDY). Note that in some scenarios, a user might want a large output size. For example, when a job seeker is looking for a job in a job recommendation system, s/he is willing to be recommended with a sufficient number of positions to increase his/her chance to get some jobs finally since each job position will only hire one or a few people from a large number of candidates. The existing min-error algorithms are not efficient in these scenarios. It is not clear whether RMS can be solved *decrementally* with theoretical guarantees; that is, it starts with the entire dataset as the solution and constructs the solution set by gradually removing points from it. In particular, when a larger output size is required, it takes *less* time to construct the solution.

Min-Error (Scalability Test). We studied the scalability of the min-error algorithms in Figure 16 where r is fixed to be 30. When the dataset size n (Figure 16 (a)) or the skyline size (Figure 16 (b)) increases, the maximum regret ratios of all algorithms are stable and are not sensitive to the increasing dataset/skyline size. It conforms with the lower bounds in Section 2, which is independent of the dataset/skyline size. In particular, SPHERE and IMPGREEDY return the set with the smallest maximum regret ratios in most cases. Different from the stable performance in maximum regret ratio, all algorithms take longer execution times when the dataset contains more points. CUBE is still the fastest one while DMM and IMPGREEDY are slower. For example, when the dataset contains 1 million points, DMM is 0.5∼10 times slower than other algorithms. The increasing trend of execution time w.r.t. the dataset size is very intuitive. However, in the era of big data, the sizes of datasets are increasing at an unprecedented rate (e.g., the whole dataset cannot be loaded into the main memory). Moreover, data nowadays is distributed over different data centers and thus, it is important to design RMS algorithms in a distributed environment so that large datasets can be handled more efficiently. However, these issues are not considered in existing RMS algorithms, limiting their applicability in real applications.

Similarly, when the dimensionality d increases (Figure 16 (c)), the maximum regret ratios of most algorithms increase slightly. This conforms with the lower bounds in Section 2 and it is intuitive since it is more difficult to guarantee the same regret with the same number of points on datasets with larger dimensionalities where each point is described by more attributes. On datasets with large dimensionalities, all algorithms spend more time to execute. In particular, the running time of DMM increases rapidly when $d \geq 7$. For example, when $d = 7$, DMM takes more than 300 seconds to determine a solution while other algorithms finish in seconds. This is because the operations of DMM are exponentially dependent on d and thus, its execution time is sensitive to d. For better visualization, we omit its results when $d \geq 7$.

Min-Size. We evaluated the min-size algorithms by varying ε on a 3-dimensional anti-correlated dataset in Figure 17.

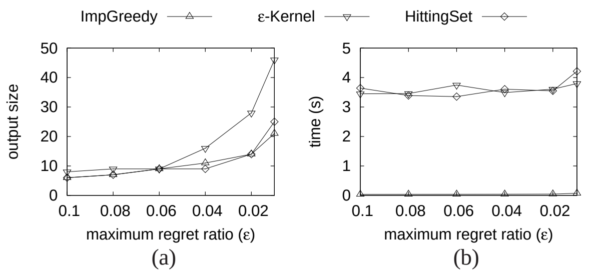
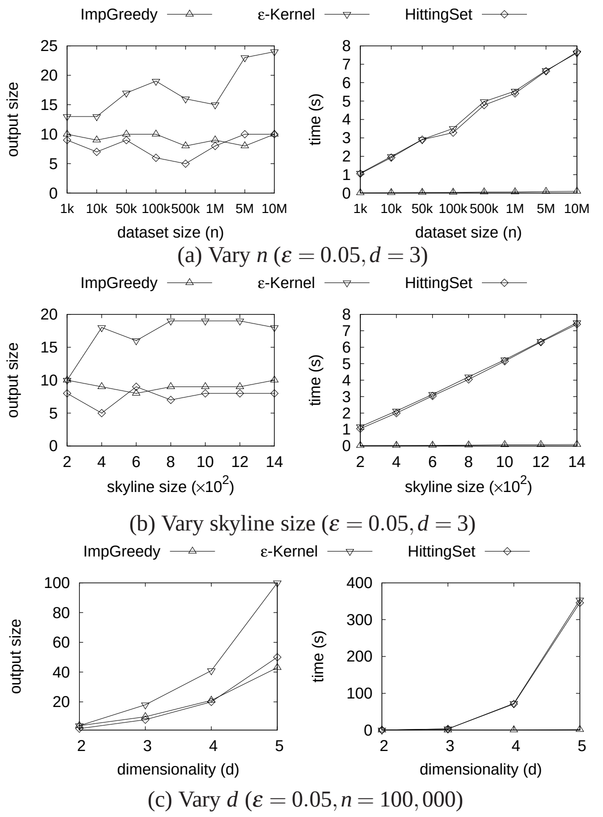


Fig. 17 Min-Size - Vary ε on Anti-Correlated ($n = 100,000, d = 3$)



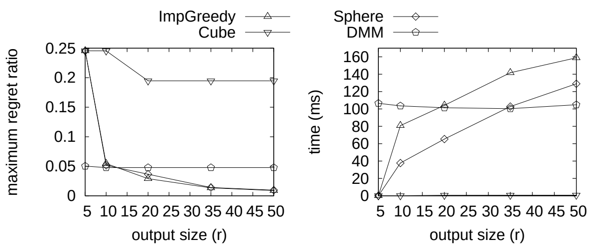
(a) Vary n ($\varepsilon = 0.05, d = 3$)

(b) Vary skyline size ($\varepsilon = 0.05, d = 3$)

(c) Vary d ($\varepsilon = 0.05, n = 100,000$)

Fig. 18 Min-Size - Scalability Test on Anti-Correlated Datasets

Note that it is possible to modify DMM [4] to be a min-size algorithm by solving a so called *Minimum Rows Satisfying the given Threshold (MRST) problem* [4]. However, due to its large output size, we did not plot its result for better visualization. Figure 17(a) depicts the output size of each algorithm. When the user requires a smaller maximum regret ratio, all algorithms tend to return more points to the user. In particular, ε-KERNEL has the largest output size in all cases, while the output size of IMPGREEDY and HITTINGSET are comparably smaller. For example, IMPGREEDY and HITTINGSET returns less than 30 points, which is half of the points needed by ε-KERNEL to guarantee a 0.01 regret ratio. This also justifies that the notion of maximum regret ratio is useful in giving a "big picture" of the database and helping users to find the points that they are interested in: instead of asking the user to examine the entire dataset containing


(a) Independent Datasets ($n = 100,000, d = 5$)

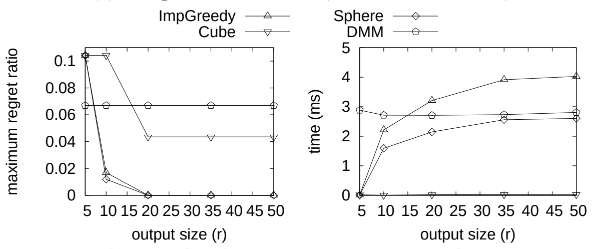
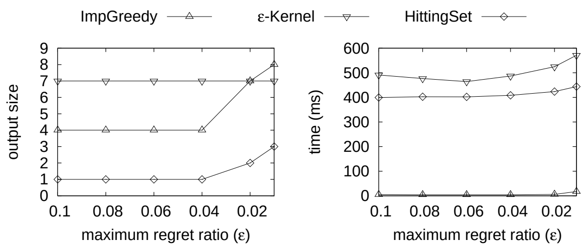

(b) Correlated Datasets ($n = 100,000, d = 5$)

Fig. 19 Min-Error - Vary r on Other Synthetic Datasets


(a) Independent Datasets ($n = 100,000, d = 3$)

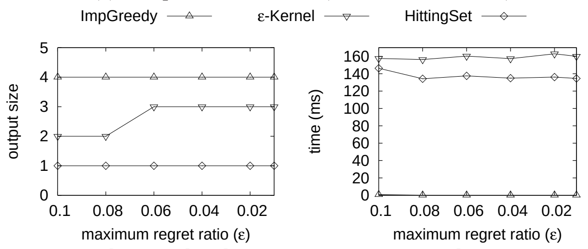

(b) Correlated Datasets ($n = 100,000, d = 3$)

Fig. 20 Min-Size - Vary ε on Other Synthetic Datasets

100,000 points, a user only needs to examine as few as 30 options to get a "good" point (only with 0.01 regret), without providing his/her exact utility function. Figure 17(b) shows the execution time of each algorithm. Both HITTINGSET and ε-KERNEL have comparable execution times, but they are slower than the heuristic algorithm, IMPGREEDY.

Min-Size (Scalability Test). The scalability test of the min-size algorithms is provided in Figure 18 where ε is fixed to be 0.05. According to the results, the output size of HITTINGSET is smaller than ε-KERNEL and IMPGREEDY in most of the cases while its running time is comparable to ε-KERNEL, but slower than that of IMPGREEDY. In particular, compared with IMPGREEDY, the running times of HITTINGSET and ε-KERNEL are more sensitive to the dimensionality d since their operations are exponentially dependent on d. This indicates the insufficiencies of the existing RMS algorithms (especially, the min-size RMS algorithms) in handling datasets with large dimensionalities, which might be the case in real scenarios. This claim also conforms with our theoretical results summarized in Table 4 where the execution times of HITTINGSET and ε-KERNEL are exponentially dependent on the dataset dimensionality.

Theoretical Algorithms on Other Synthetic Datasets. Finally, we studied the performance of the RMS algorithms on other synthetic datasets with different characteristics, (i.e., independent datasets and correlated datasets) in Figures 19 and 20 (results on the anti-correlated datasets have been presented in Figures 16 and 18). Each algorithm follows a similar trend as it behaves on anti-correlated datasets. However, the maximum regret ratios / output size of each algorithm on correlated datasets is smaller than those on independent datasets, which is then smaller that those on anti-correlated datasets. For example, to guarantee a 0.01 regret ratio, IMPGREEDY has to return more than 20 points on anti-correlated datasets, while on independent and correlated datasets, it needs as few as 8 and 4 points, respectively, to guarantee the same regret ratio. This observation is consistent with dataset characteristics. For example, correlated datasets contain points with high values in all attributes and thus, only a small number of points is needed to guarantee a small maximum regret ratio. Similar phenomenon can also be observed in the running time of each algorithm. For example, points in anti-correlated datasets which have high values in some dimensions might have low values in other dimensions, making the trade-off among dimensions more difficult and making it more time-consuming to construct the final solution.

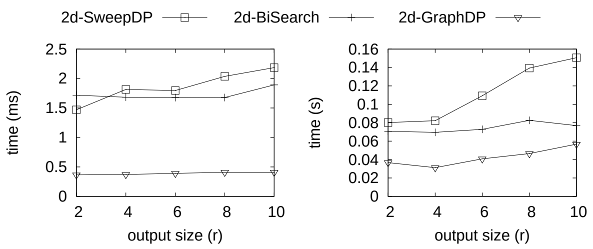
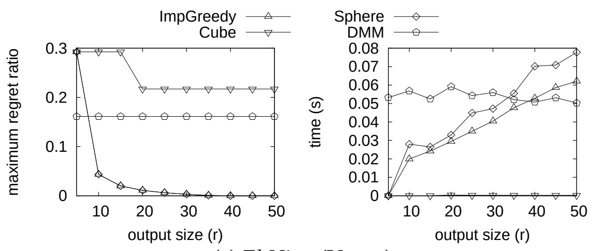

(a) Airline (b) Island

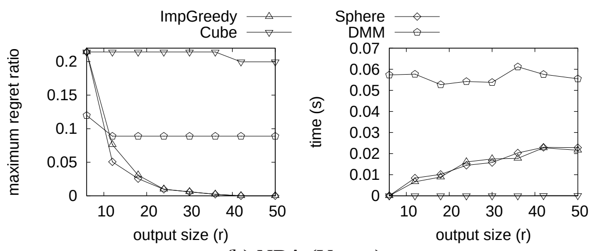
Fig. 21 2D Algorithms on Real Datasets

5.2 Results on Real Datasets

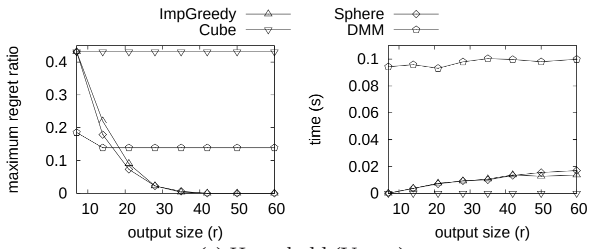
On 2-dimensional real datasets, Airline and Island, in Figure 21, we evaluated the performance of the 2-dimensional algorithms by varying the output size r. Similar to the results observed on synthetic datasets, 2d-SWEEPDP is the slowest algorithm and 2d-GRAPHDP is consistently faster than both 2d-SWEEPDP and 2d-BISEARCH while being not sensitive to r. This is because 2d-GRAPHDP avoids the expensive envelope computation and it computes regret ratios (i.e., the edge weights in the graph representation) efficiently.



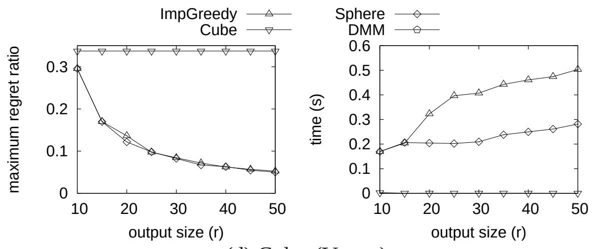
(a) El Nino (Vary r)



(b) NBA (Vary r)

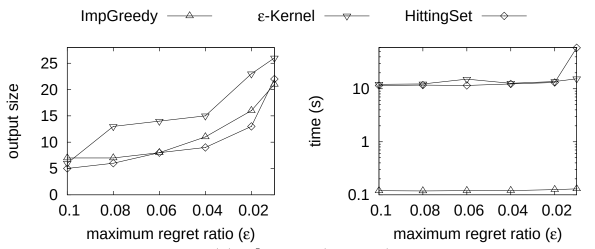


(c) Household (Vary r)

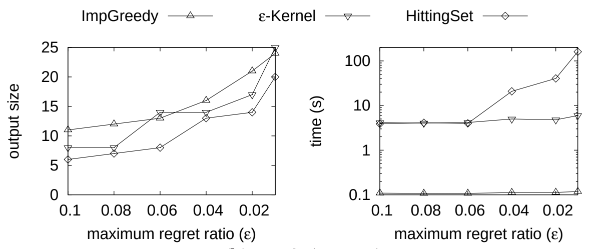


(d) Color (Vary r)

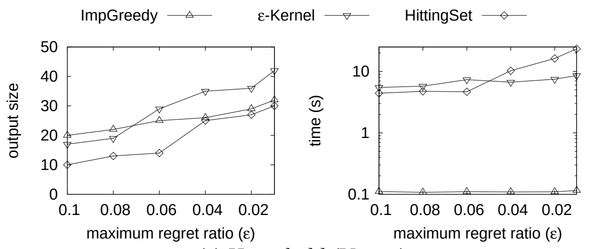
Fig. 22 Min-Error Algorithms on Real Datasets



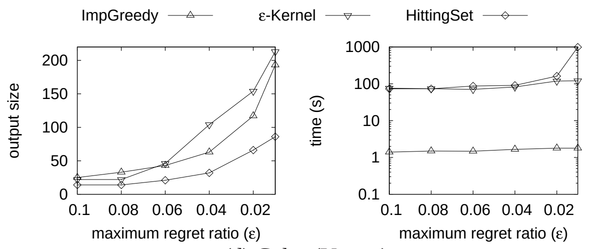
(a) El Nino (Vary ε)



(b) NBA (Vary ε)



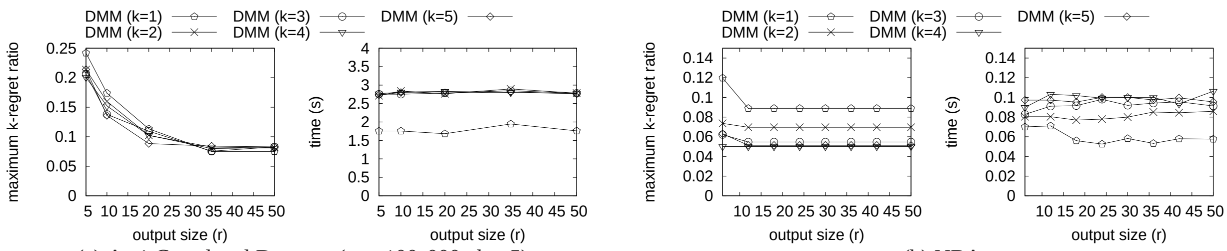
(c) Household (Vary ε)



(d) Color (Vary ε)

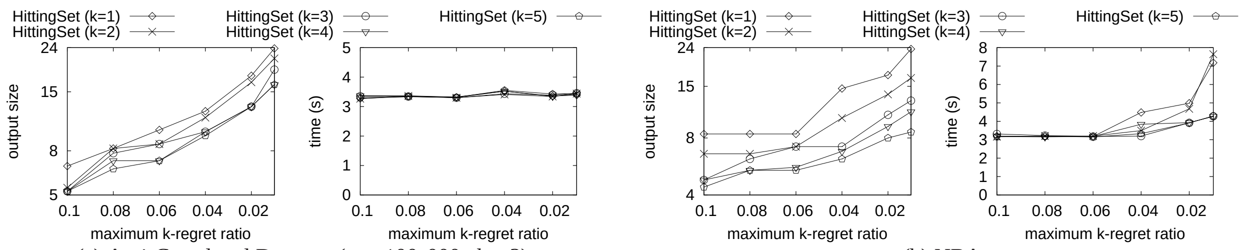
Fig. 23 Min-Size Algorithms on Real Datasets

In Figures 22 and 23, we evaluated the performance of each d-dimensional algorithm on the El Nino, NBA, Household and Color datasets. In particular, we studied min-error algorithms and min-size algorithms in Figure 22 and Figure 23, respectively. Firstly, consider the results of min-error algorithms in Figure 22 where we varied the output size r. Similar to what we observed on synthetic datasets, DMM does not scale well w.r.t. the dimensionality and thus, its results on Color are omitted due to the large execution time. Except for $r \geq 40$ on El Nino, DMM has the largest execution time and its maximum regret ratio is much worse than those of SPHERE and IMPGREEDY; e.g., when $r = 50$ on NBA and El Nino, both SPHERE and IMPGREEDY achieve 0 regret while the maximum regret ratio of DMM is greater than 0.1. In addition, though SPHERE and IMPGREEDY have comparably small running times, SPHERE gives a smaller empirical maximum regret ratio than IMPGREEDY, which is also observed in [44]. For example, when $r = 12$ on NBA, the maximum regret ratio of IMPGREEDY is 0.075 while the maximum regret ratio of SPHERE is 0.05, which is a 30% improvement over IMPGREEDY. Secondly, consider the results of min-size algorithms in 23 where we varied the maximum regret ratio ε. HITTINGSET and ε-KERNEL take more time to execute compared with IMPGREEDY. Nevertheless, HITTINGSET consistently returns the smallest solution set in all setting and thus, it is suitable in providing a small representative set of the database in multi-criteria decision making. Note that the evaluation on real datasets is consistent with our observations on synthetic datasets and it supports the claims we make in Section 5.1. In Section 5.5, we will formally summarize those claims/observations, which also motivates the open problems introduced in Section 6.



(a) Anti-Correlated Datasets ($n = 100,000, d = 5$) (b) NBA

Fig. 24 Results on Min-Error kRMS



(a) Anti-Correlated Datasets ($n = 100,000, d = 3$) (b) NBA

Fig. 25 Results on Min-Size kRMS

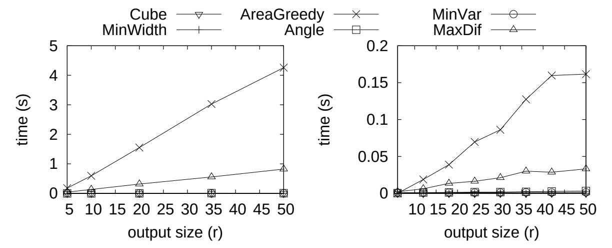
5.3 Results on Variants of RMS

In this section, we demonstrate the experimental performance of some existing RMS algorithms when they are extended to solving different variants of RMS. In particular, two major variants of RMS, namely *k*RMS and non-linear RMS, are studied in Section 5.3.1 and Section 5.3.2, respectively. Besides, although none of the existing RMS algorithms can be directly extended to handling interactive RMS where user interaction is involved, we also conducted experiments in Section 5.3.3 comparing the best performing RMS algorithms against those interactive RMS algorithms, demonstrating the effectiveness of user interactions in reducing the user regret and the output size. Due to the limited space, we only report the experimental results of each RMS variant on the anti-correlated dataset and the NBA dataset in this section. Results on other datasets are similar and thus are omitted.

5.3.1 Results on kRMS

We evaluated the performance of DMM [7] and HITTINGSET [2, 21] when they are extended to handling *k*RMS for min-error RMS and min-size RMS, respectively. Although GREEDY [28] is extended to solving *k*RMS in [11] as *k*RMS-GREEDY, its execution time is much worse than those of DMM and HITTINGSET since the number of LPs and the size of each LP in *k*RMS-GREEDY are very large. For the ease of presentation, results of *k*RMS-GREEDY are not reported.

Results on the anti-correlated and NBA dataset are shown in Figures 24 and 25. In general, when the parameter *k* in *k*RMS increases from 1 to 5, the running time of each algorithm increases. Meanwhile, we also observe that the maximum *k*-regret ratio and the size of the solution set returned for large *k* tend to be smaller than those returned for small *k*. This conforms with our intuition that we can guarantee the same *k*-regret ratio with a smaller set when *k* is larger since *k*RMS can regarded as a relaxation of traditional RMS.



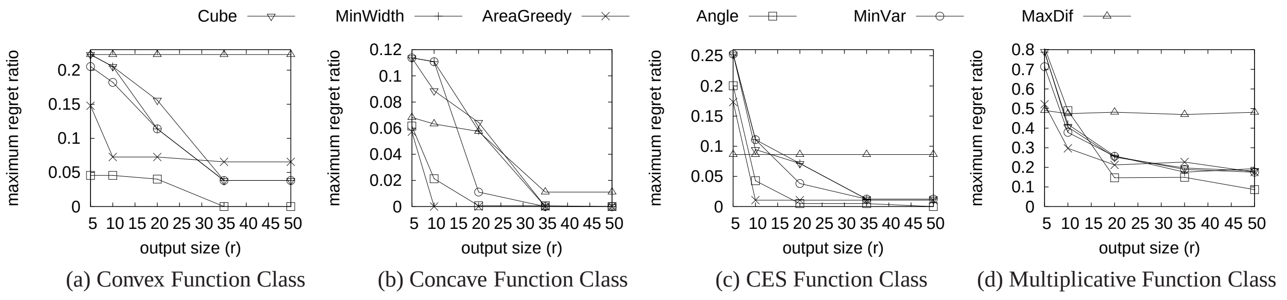
(a) Anti-Correlated Datasets ($n = 100,000, d = 3$) (b) NBA

Fig. 26 Running times on Non-Linear RMS

5.3.2 Results on Non-linear RMS

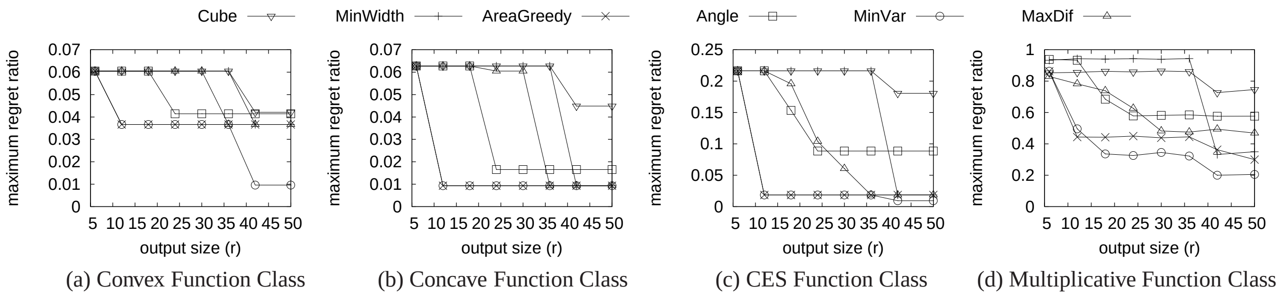
We performed the experimental evaluation on non-linear RMS by comparing the following algorithms: (1) the original CUBE algorithm [28] and its extensions for non-linear RMS, MINWIDTH [13] and MINVAR [32]; and (2) the heuristic algorithms for non-linear RMS, AREAGREEDY [13], ANGLE [13] and MAXDIF [32]. Both MINWIDTH and MINVAR provide guarantees on non-linear RMS (see Table 7).

In Figure 26, we plotted the running time of each algorithm by varying the output size *r* on the anti-correlated dataset and the NBA dataset. According to the results, AREAGREEDY is the most time-consuming non-linear RMS algorithm since it requires expensive area computation. MAXDIF is faster than AREAGREEDY, but is slower than ANGLE and cube-based algorithms, which only scans the database once.



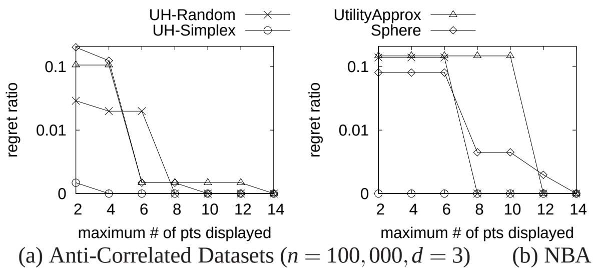
(a) Convex Function Class (b) Concave Function Class (c) CES Function Class (d) Multiplicative Function Class

Fig. 27 Maximum Regret Ratios on Non-Linear RMS (Anti-Correlated Datasets where $n = 100,000, d = 3$)



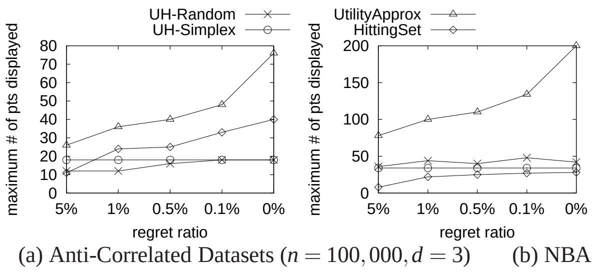
(a) Convex Function Class (b) Concave Function Class (c) CES Function Class (d) Multiplicative Function Class

Fig. 28 Maximum Regret Ratios on Non-Linear RMS (NBA)



(a) Anti-Correlated Datasets ($n = 100,000, d = 3$) (b) NBA

Fig. 29 Vary Number of Points Displayed on Interactive RMS



(a) Anti-Correlated Datasets ($n = 100,000, d = 3$) (b) NBA

Fig. 30 Vary Regret Ratio on Interactive RMS

We also evaluated the maximum regret ratio of each algorithm on four non-linear utility function classes defined in Section 4.2: convex function class, concave function class, CES function class and multiplicative function (MUF) class in Figures 27 and 28. On the anti-correlated dataset, AREAGREEDY and ANGLE perform the best by returning the set with the smallest maximum regret ratio over most of the non-linear function classes. In contrast, MINVAR performs better on NBA where it has the smallest maximum regret ratio over both convex and multiplicative function classes.

5.3.3 Results on Interactive RMS

We proceed with the experiments on interactive RMS. Recall that in interactive RMS, a user interacts with the database system for *rounds*. At each round, the system displays s points and the user is asked to select the point that s/he favors the most among them. Based on the feedback, the system learns the user's preference implicitly and finally, identifies the user's favorite point and returns that point to the user.

We implemented the best-known algorithms for interactive RMS: UH-SIMPLEX [43], UH-RANDOM [43] and UTILITYAPPROX [27]. We set s (i.e., the number of points displayed at each round) to be 2. Then, we compared the above algorithms against the single round algorithms, SPHERE [44] and HITTINGSET [2,21] to demonstrate the effectiveness of user interactions in reducing the regret ratio and the output size, respectively. Since user feedback is required in interactive algorithms, we modeled the users' behavior by randomly generating their utility vectors. The exact utility vectors we generated were not disclosed to any algorithms. Different from traditional RMS, the performance of each algorithm in interactive RMS is evaluated using two measurements: (1) *Regret Ratio.* The regret ratio of an interactive algorithm (a single round algorithm) is the regret ratio (w.r.t. the generated utility vector) of the final point suggested (the solution set returned); and (2) *The maximum number of points displayed.* For a single round algorithm, the number of points displayed is the size of the solution set returned. For an interactive algorithm, the number of points displayed is at most the number of rounds multiplied by s (i.e., the number of points displayed at each round).

We first constrain the number of points that an algorithm can display in Figure 29 (similar to the min-error RMS setting). It shows that user interactions are very useful since they can guarantee a smaller regret ratio compared with the single round algorithm, SPHERE, where no user interaction is allowed. Specifically, by only displaying 4 points, UH-SIMPLEX can guarantee a 0 regret ratio while the regret ratio of SPHERE is around 0.1. Similarly, consider Figure 30 where we constrain the regret ratio of each algorithm (similar to the min-size RMS setting). Due to the small skyline size in NBA, HITTINGSET performs slightly better than the interactive algorithms on NBA. However, on the anti-correlated dataset where the skyline size is larger, its output size is twice more than that of UH-RANDOM when the regret ratio is at most 0.01, which also verifies the usefulness of interactions in reducing the output size in some scenarios.

5.4 User Study

To verify the effectiveness of RMS in real scenarios, we conducted a user study on statistics of the 2018-19 NBA regular season. After removing players who played less than 40 games during this season, there were 386 players remaining. Six popular attributes (game played, minutes played, rebound/assist/steal/points per game) were used to describe the statistical performance of each player. We compared the players returned by min-error RMS (we used IMPGREEDY) with three existing skyline variants, which are compared in [28] where RMS was first proposed: distance-skyline [40], MaxDom [24] and r-dominance [8]. Distance-skyline picks r players that admit the *best r-center clustering*. MaxDom picks r players that dominate the largest number of players. k-dominance relaxes the concept "domination" to "k-domination" and finds r players that best *k-dominate* others. Each query returns a set of 5 players, as shown in Table 9.

Following [47], we conducted a survey on "Amazon Mechanical Turk". We asked participants with NBA knowledge to indicate the set of players they prefer among four candidates considering the *statistical performance* of each player. If the statistical performance of a player (e.g., rebound) in a set is better than another in another set, the former set is better. In other words, we want a set such that the statistical performance of each player is as good as possible. We paid each participant \$0.05 and there were 104 responses in total.

According to the responses, 44.55% of participants thought that the set returned by RMS has better overall performance, while 22.72%, 13.63% and 19.09% prefer the sets returned by distance-skyline, MaxDom and k-dominance, respectively. Besides, we observed that the players returned by RMS play in different positions such as point guard, small forward and shooting guard and they enjoy diverse statistical performance. For example, Andre Drummond plays in the center position and he has high rebounds while James Harden plays as a shooting guard and he achieves the highest points. In other words, the players returned by RMS not only have good statistical performance, but also satisfy different NBA fans who are interested in diverse positions and statistics.

As another reference on whether the players returned by each query is useful in real world, we compared the players returned by each query to the top-10 NBA MVP award candidates[2], as shown in Table 10. 4 out of 5 players returned by RMS are among the top-10 candidates (bold in Table 9), which is more than those in other queries; e.g., only 1 player returned by distance-skyline appears in the MVP list.

5.5 Summary

We conducted comprehensive experiments in this section on both real and synthetic datasets, comparing the existing algorithms for RMS under various parameter settings. The ability of existing RMS algorithms on handling different variants of RMS and the usefulness of RMS over other variants of the skyline query are also clearly demonstrated.

Specifically, we make the following observations and they provide an empirical guideline to users on choosing the best algorithm when solving RMS. Firstly, none of the existing algorithms dominates others in all aspects. Specifically, some algorithms might be good in one aspect (e.g., execution time) while being poor in other aspects (e.g., maximum regret ratio). Secondly, some RMS algorithms can be extended to handling different variants of RMS. For example, DMM and HITTINGSET can be extended to solving kRMS while CUBE can be extended to solving non-linear RMS. Thirdly, on 2-dimensional datasets where RMS can be solved optimally, 2d-GRAPHDP achieves the best performance by returning the optimal solution in the shortest amount of time. Thus, if the dataset only contains two attributes and the skyline size is small, it is good to use 2d-GRAPHDP to find the optimal solution for RMS. Fourthly, IMPGREEDY scales better and requires a shorter execution time compared with other heuristic algorithms in most cases. If the users want a solution for RMS which (1) is fast; (2) has a good empirical performance; and (3) does not require theoretical guarantees, IMPGREEDY is a good option. Finally, different theoretical algorithms have different advantages and they can be applied in different scenarios. Specifically, among min-error algorithms which optimize over the maximum regret ratio, CUBE is the fastest one while SPHERE guarantees a small maximum regret ratio in most cases; among min-size algorithms which optimize over the output size, HITTINGSET returns a small number of points in most cases while guaranteeing the maximum regret ratio and thus, it provides a good representative subset of the database. The best choice of algorithms depends on application needs.

[2] https://www.basketball-reference.com/friv/mvp.html

RMS	Distance-Skyline [40]	MaxDom [24]	*k*-Dominance [8]
James Harden	**James Harden**	**Nikola Jokic**	Bradley Beal
Andre Drummond	DeAndre Jordan	Bradley Beal	Andre Drummond
Russell Westbrook	P.J. Tucker	**Russell Westbrook**	**Paul George**
Joel Embiid	Chris Paul	**James Harden**	**James Harden**
Paul George	Karl-Anthony Towns	Rudy Gobert	**Russell Westbrook**

Table 9 Five NBA Players Returned by Different Queries (MVP Candidates are in Bold)

Top 1 to 5	Top 6 to 10
Giannis Antetokounmpo	Joel Embiid
James Harden	Damian Lillard
Nikola Jokic	Stephen Curry
Kawhi Leonard	Paul George
Kevin Durant	Russell Westbrook

Table 10 Top 10 Candidates for MVP Award

6 Open Problems

In this section, we highlight the open problems and some possible future directions for RMS according to our discussion and experimental observations in previous sections.

Optimal Algorithms. While there is an asymptotically tight bound proven on RMS [44, 2, 7] (e.g., Theorem 9), the question of an exact (non-asymptotic) bound remains open. Developing an algorithm that computes an optimal bound efficiently is an open algorithmic problem in this area.

Monotonically Decreasing Utility Functions. We assume that a larger value is preferable to all users and only monotonically increasing utility functions are considered in the existing studies. However, it could happen in reality that a smaller value in some dimensions is better. For example, a lower price is better. Although we can use the trick of subtracting each value from the maximum value in those dimensions (so that the "larger is better" assumption is satisfied in Section 2), it changes the value of those attributes and it is no longer clear if the notation of regret still applies. None of the results so far can be extended easily to this case.

Arbitrary Monotonic Utility Functions. While results on RMS are known for some particular function classes, e.g., the convex and CES function classes presented in Section 4.2, it remains unknown whether we can get a general result that applies for any monotonic utility function class.

High Dimensional RMS. According to our experimental evaluation, some existing algorithms (e.g., GEOGREEDY, ε-KERNEL, HITTINGSET) do not scale well w.r.t. to the dimensionality. Specifically, it takes them a very long time to execute and the maximum regret ratios / the output sizes of the solution sets they return are quite large even when the dimensionality is of a medium value (e.g. $d = 8$). Given the known lower bounds on RMS (e.g., Theorem 3), computing a small set with a small regret on high dimensional datasets is a hard problem. Some additional assumptions would have to be made on the data. Besides, when handling datasets with very large dimensionalities, it is also important to handle them very efficiently. Unfortunately, the execution times of many existing algorithms exponentially depend on the dimensionality. It remains open whether some dimension reduction techniques could help in high dimensional RMS.

Large Size RMS. In the era of big data, the size of dataset is increasing in an unprecedented speed and the data might come in a sequential manner. Most existing RMS algorithms implicitly assume that the entire dataset can be loaded into the main memory. Unfortunately, this assumption hardly holds in real-world applications. Besides, data nowadays is distributed over different data centers. Computing a solution for RMS across distributed databases so that the communication cost is minimized (i.e., do not need to send all the datasets to a single location) remains open.

RMS with Dynamic Updates. Nowadays, the database is updated frequently with point insertions and deletions. How to extend the existing methods when the dataset is changed dynamically is an interesting problem. Only ε-KERNEL has this ability, but the results of other methods are unknown.

Decremental RMS. According to the experimental observations, most existing algorithms construct solutions *incrementally* (i.e., start with an empty set and gradually add points). In particular, to return more points or to guarantee smaller regret, it takes more time for them to execute. However, in reality, users are interested in small maximum regret ratios (e.g., Alice wants a car which is as close to her favorite car as possible) and in some scenarios, a larger output size is desirable (e.g., the job recommendation example in Section 5). Motivated by this, it is interesting to develop some *decremental* RMS algorithms (i.e., start with the entire database and gradually delete points) so that we can output a large number of points or guarantee small regret *efficiently*.

7 Conclusion

In this survey, we comprehensively review existing methods for RMS. Specifically, various methods were proposed for solving RMS optimally, but they are restricted in 2-dimensional spaces. In d-dimensional spaces, RMS was proven to be an NP-hard problem. Heuristic algorithms were proposed to obtain solutions with small regret/output sizes and theoretical algorithms were also studied to provide bounded guarantees on the solutions. Different variants of RMS were also reviewed and experimented in this paper. We conducted a comprehensive experimental evaluation of all state-of-the-art RMS algorithms on both synthetic datasets and real datasets, demonstrating the advantages of different RMS algorithms under various parameter settings. A user study comparing RMS with other skyline variants was also conducted, verifying the usefulness of RMS in real-world scenarios.

References

1. Agarwal, P., Peled, S., Varadarajan, K.: Approximating extent measures of points. In: Journal of the ACM (2004)
2. Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Efficient algorithms for k-regret minimizing sets. In: International Symposium on Experimental Algorithms (SEA) (June 2017)
3. Alhenshiri, A.: Web information retrieval and search engines techniques. Al-Satil Journal (2010)
4. Asudeh, A., Nazi, A., Zhang, N., Das, G.: Efficient computation of regret-ratio minimizing set: A compact maxima representative. In: Proc. of the ACM International Conference on Management of Data (2017)
5. Asudeh, A., Nazi, A., Zhang, N., Dasm, G., Jagadish, H.: Rrr: Rank-regret representative. Proc. of the 2019 ACM International Conference on Management of Data (2019)
6. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. 17th International Conference on Data Engineering (2001)
7. Cao, W., Li, J., Wang, H., Wang, K., Wang, R., Wong, R., Zhan, W.: k-regret minimizing set: Efficient algorithms and hardness. In: ICDT (2017)
8. Chan, C., Jagadish, H., Tan, K., Tung, A., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proc. of the 2006 ACM SIGMOD International Conference on Management of Data (2006)
9. Chan, C., Jagadish, H., Tan, K., Tung, A., Zhang, Z.: On high dimensional skylines. In: Advances in Database Technology-EDBT 2006 (2006)
10. Chang, Y., Bergman, L., Castelli, V., Li, C., Lo, M., Smith, J.: The onion technique: Indexing for linear optimization queries. In: Proc. of the 2000 SIGMOD International Conference on Management of Data (2000)
11. Chester, S., Thomo, A., Venkatesh, S., Whitesides, S.: Computing k-regret minimizing sets. In: Proc. of the VLDB Endowment (2014)
12. Dong, Q., Zheng, J., Qiu, X., Huang, X.: Efficient approximate algorithms for k-regret queries with binary constraints. In: International Conference on Web Information Systems and Applications (2018)
13. Faulkner, T.K., Brackenbury, W., Lall, A.: K-regret queries with nonlinear utilities. In: Proc. of the VLDB Endowment (2015)
14. Goncalves, M., Yidal, M.: Top-k skyline: a unified approach. In: On the Move to Meaningful Internet System 2005: OTM 2005 workshops (2005)
15. Han, S., Zheng, H., Dong, Q.: Efficient processing of k-regret queries via skyline priority. In: International Conference on Web Information Systems and Applications (2018)
16. Han, S., Zheng, J., Dong, Q.: Efficient processing of k-regret queries via skyline frequency. In: International Conference on Web Information Systems and Applications (2018)
17. Hussain, Z., Khan, H., Sharaf, M.: Diversifying with few regrets, but too few to mention. In: Proc. of the Second International Workshop on Exploratory Search in Databases and the Web (2015)
18. Il’ev, V.: An approximation guarantee of the greedy descent algorithm for minimizing a supermodular set function. Discrete Applied Mathematics (2001)
19. Kenthapadi, K., Le, B., Venkataraman, G.: Personalized job recommendation system at linkedin: Practical challenges and lessons learned. In: Proc. of the 11th ACM Conference on Recommender Systems (2017)
20. Kleinberg, J., Tardos, E.: Algorithm design. Addison Wesley (2006)
21. Kumar, N., Sintos, S.: Faster approximation algorithm for the k-regret minimizing set and related problems. In: 2018 Proc. of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX) (2018)
22. Lee, J., You, G., Hwang, S.: Personalized top-k skyline queries in high-dimensional space. In: Information Systems (2009)
23. Lian, X., Chen, L.: Top-k dominating queries in uncertain databases. In: Proc. of International Conference on Extending Database Technology: Advances in Database Technology (2009)
24. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: The k most representative skyline operator. In: Proc. of International Conference on Data Engineering (2007)
25. McDonald, D., Ackerman, M.: Expertise recommender: a flexible recommendation system and architecture. In: Proc. of the 2000 ACM conference on Computer supported cooperative work (2000)
26. Mindolin, D., Chomicki, J.: Discovering relative importance of skyline attributes. In: Proc. of the VLDB Endowment (2009)
27. Nanongkai, D., Lall, A., Sarma, A.D., Makino, K.: Interactive regret minimization. In: Proc. of the 2012 ACM International Conference on Management of Data (2012)
28. Nanongkai, D., Sarma, A., Lall, A., Lipton, R., Xu, J.: Regret-minimizing representative databases. In: Proc. of the VLDB Endowment (2010)
29. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. In: ACM Transactions on Database Systems (TODS) (2005)
30. Papadopoulos, A.N., Lyritsis, A., Nanopoulos, A., Manolopoulos, Y.: Domination mining and querying. In: DaWaK (2007)
31. Peng, P., Wong, R.: Geometry approach for k regret query. In: Proc. of International Conference on Data Engineering (2014)
32. Qi, J., Zuo, F., Samet, H., Yao, J.: K-regret queries using multiplicative utility functions. TODS (2018)
33. Qiu, X., Zheng, J.: An efficient algorithm for computing k-average-regret minimizing sets in databases. In: International Conference on Web Information Systems and Applications (2018)
34. Qiu, X., Zheng, J., Dong, Q., Huang, X.: Speed-up algorithms for happiness-maximizing representative databases. In: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data (2018)
35. Roshdi, A., Roohparvar, A.: Information retrieval techniques and applications. In: International Journal of Computer Networks & Communications Security (2015)
36. Russell, S., Norvig, P.: Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited, (2016)
37. Salton, G., McGill, M.: Introduction to modern information retrieval (1986)
38. Soliman, M., Ilyas, I., Chang, K.C.C.: Top-k query processing in uncertain databases. In: Proc. of International Conference on Data Engineering (2007)
39. Soma, T., Yoshida, Y.: Regret ratio minimization in multi-objective submodular function maximization. In: AAAI (2017)
40. Tao, Y., Ding, L., Pei, J.: Distance-based representative skyline. In: Proc. of International Conference on Data Engineering (2009)
41. Varian, H.: Microeconomic analysis. Norton and Company (1992)
42. Walter, F., Battiston, S., Schweitzer, F.: A model of a trust-based recommendation system on a social network. Autonomous Agents and Multi-Agent Systems (2008)
43. Xie, M., Wong, R., Lall, A.: Strongly truthful interactive regret minimization. In: Proc. of the 2019 ACM International Conference on Management of Data (2019)
44. Xie, M., Wong, R., Li, J., Long, C., Lall, A.: Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In: Proc. of the 2018 ACM International Conference on Management of Data (2018)
45. Yu, H., Agarwal, P., Varadarajan, R.P.K.: Practical methods for shape fitting and kinetic data structures using coresets (2008)
46. Zeighami, S., Wong, R.: Minimizing average regret ratio in database. In: Proc. of the 2016 International Conference on Management of Data (2016)
47. Zeighami, S., Wong, R.: Finding average regret ratio minimizing set in database. In: Proc. of 35th International Conference on Data Engineering (2019)