Efficient k-Regret Query Algorithm with Restriction-free Bound for any Dimensionality

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

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

Jian Li
Tsinghua University
lijian83@mail.tsinghua.edu.cn

Cheng Long
Queen's University Belfast
cheng.long@qub.ac.uk

Ashwin Lall
Denison University
lalla@denison.edu

ABSTRACT

Extracting interesting tuples from a large database is an important problem in multi-criteria decision making. Two representative queries were proposed in the literature: top-k queries and skyline queries. A top-k query requires users to specify their utility functions beforehand and then returns k tuples to the users. A skyline query does not require any utility function from users but it puts no control on the number of tuples returned to users. Recently, a *k-regret query* was proposed and received attention from the community because it does not require any utility function from users and the output size is controllable, and thus it avoids those deficiencies of top-k queries and skyline queries. Specifically, it returns k tuples that minimize a criterion called the *maximum regret ratio*.

In this paper, we present the lower bound of the maximum regret ratio for the k-regret query. Besides, we propose a novel algorithm, called Sphere, whose upper bound on the maximum regret ratio is *asymptotically optimal* and *restriction-free* for any dimensionality, the best-known result in the literature. We conducted extensive experiments to show that Sphere performs better than the state-of-the-art methods for the k-regret query.

CCS CONCEPTS

• **Information systems → Data analytics**;

KEYWORDS

k-regret; multi-criteria decision making; data analytics

ACM Reference Format:
Min Xie, Raymond Chi-Wing Wong, Jian Li, Cheng Long, and Ashwin Lall. 2018. Efficient k-Regret Query Algorithm with Restriction-free Bound for any Dimensionality. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196903

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD'18, June 10–15, 2018, Houston, TX, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196903

1 INTRODUCTION

A database system usually contains millions of tuples nowadays and an end user may be interested in only some of them. It is convenient if the database system can provide some operators for an end user to obtain the tuples s/he is interested in. Consider the following example. Assume that a car is characterized by two attributes, namely horse power (HP) and miles per gallon (MPG). Alice visits a large car database and wants to buy a car with high HP and high MPG. However, it might be impossible for Alice to go through every car tuple in the database. A possible solution is that some representative cars are selected based on some criteria (e.g., cars favored by Alice) and are shown to Alice. In order to decide which car to be shown, we implicitly assume that Alice has a preference function, called a *utility function*, in her mind. Based on the utility function determined by Alice, each car tuple in the database has a *utility*. A high utility means that this car is favored by Alice.

We have two goals in multi-criteria decision making. First, we do not require a user to specify a utility function, which may not be known in advance. Second, we want a set of tuples with a controllable size since it is meaningless if millions of tuples are returned. Many queries were proposed in the literature for multi-criteria decision making: *top-k* queries, *skyline* queries and *k-regret* queries.

In the setting of a *top-k* query [14, 16, 17, 23, 29], we assume that the utility function is given. For each tuple in the database, a utility is computed, and the k tuples with the highest utilities are returned. For example, Alice's utility function can have weight 60% for HP and weight 40% for MPG. Here, a higher weight indicates that the corresponding attribute is more important to Alice. With this utility function, we compute the k cars with the highest utilities and they are shown to Alice. However, the top-k query requires users to specify utility functions, which is hard to most users.

A *skyline* [8, 9, 18, 19, 22] query does not ask a user for any utility function. Instead, a concept, called *domination*, 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 the utility of p is higher than that of q no matter what utility function Alice has. Tuples which are not dominated by any other tuples are returned in a skyline query. However, the output size of a skyline query cannot be controlled. It could happen that all tuples in the database are returned in a skyline query.

Recently, a *k-regret* query [2, 5, 7, 11, 21, 24], was proposed which integrates the merits of top-k queries and skyline queries.

In the setting of a k-regret query, we do not require any utility function from a user and the output size is controllable since only k tuples are returned. When the output size for a particular user is restricted, it is very likely that there is a difference between the highest utility over all tuples in the database and the highest utility over the selected k tuples. We call the ratio of this difference to the highest utility in the database as the *regret ratio*, which is a real number from 0 to 1. The greatest possible ratio (among all users) is quantified by a criterion, called the *maximum regret ratio*. It quantifies how *regretful* a user is if s/he gets the best tuple among the selected k tuples but not the best tuple among all tuples in the database. A k-regret query is to select a set of k tuples such that the maximum regret ratio of the set is minimized.

According to our discussion above, the k-regret query meets our two goals. The applications originally applied to top-k queries and skyline queries could naturally be applied to k-regret queries. Consider our car database application. Different users might have different preferences in their minds. One might think that high HP is more important. The other might think high MPG is more important. A k-regret query on the car database returns a set of k cars, minimizing the "regret" level for all users. That is, no matter what preference the user has, there is a car in the selected set which is favored by the user in a great extent (i.e., a car with a high utility based on the user's preference). Other applications of k-regret queries include Information Retrieval (IR) [27] and Recommendation Systems (RS) [15]. For example, a job recommendation system recommends job positions to the job seekers according to some criteria (e.g., a short travel distance and high salary). Different from the car database where the same type of cars can be recommended and purchased by many customers, each job position will finally hire one or a few people from a large number of candidates. In this case, a job seeker is willing to be recommended with a sufficient number of positions to increase his chance to get some jobs finally.

We consider the following four requirements for evaluating an algorithm A for the k-regret query:

- *Restriction-free Bound* Requirement: There is *no restriction* on the bound of the maximum regret ratio of the set returned by algorithm A. Specifically, if the bound of the maximum regret ratio of the set returned by A is in the range between 0 and 1 for *any* setting, we say that A satisfies the restriction-free bound requirement. Otherwise, the bound is in the range between 0 and 1 in some *restricted* cases and thus this algorithm does not satisfy the requirement.
- *Dimensionality* Requirement: Algorithm A could be executed on datasets of *any* dimensionality.
- *Efficiency* Requirement: Algorithm A is efficient in practice.
- *Quality* Requirement: The maximum regret ratio of the set returned by algorithm A should be very small in practice.

All requirements above are essential to the k-regret query since (1) an algorithm which does not satisfy the restriction-free bound requirement *cannot* give a theoretical bound on the maximum regret ratio in some cases and may give an *invalid* bound (e.g., a bound greater than 1) in other cases, which further implies that this algorithm does not have a useful theoretical bound since the maximum regret ratio itself is a real number from 0 to 1; (2) an algorithm which does not satisfy the dimensionality requirement may not be executed on datasets of some dimensionalities, which reduces the generality of the algorithm in practice; (3) an algorithm which has poor efficiency is undesirable since returning a solution set efficiently is one of the goals for each query; (4) an algorithm whose solution is of low quality (i.e., has a large maximum regret ratio) is also unacceptable since we want a solution set with a small maximum regret ratio for reducing the "regret" level of a user. We say that an algorithm is *elegant* if it satisfies all four requirements.

Algorithm	Requirement			
	Restriction-free Bound	Dimensionality	Efficiency	Quality
Cube [21]	✓	✓	Fast	Low
Greedy [21]		✓	Medium	Medium
GeoGreedy [24]		✓	Slow	Medium
StoredList [24]		✓	Fast	Medium
2d kRMS [11]	✓		Fast	High
RMS_HS [2]		✓	Slow	High
A-IntCov-1 [7]			Fast	High
E-Greedy-1 [7]	✓		Fast	High
ε-kernel [7]		✓	Medium	Medium
2D-RRMS [5]	✓		Fast	High
DMM [5]	✓	✓	Medium	Medium
Sphere (Our Algorithm)	✓	✓	Fast	High

Table 1: Comparison among Different Algorithms

Unfortunately, the existing methods cannot address the k-regret query well since they are not elegant, i.e., they do not satisfy all four requirements simultaneously (see Table 1). Note that the empirical execution time and the maximum regret ratio of an algorithm are affected by many factors, such as the datasets and the running environments. In order to avoid ambiguity, we use relative terms in Table 1 to describe the efficiency requirement (i.e., fast, medium and slow) and the quality requirement (i.e., high, medium and low) for each algorithm. An algorithm is said to satisfy the efficiency requirement if it is *fast* and an algorithm is said to satisfy the quality requirement if it returns a set with *high* quality (i.e., has a small maximum regret ratio). Consider the comparison in Table 1. Some methods [21, 24] do not have provable bounds, and the bounds of some other methods [2, 7] are not restriction-free. The methods [5, 7, 11] work only on 2-dimensional datasets. Some methods [2, 5, 21, 24] have large execution times and the maximum regret ratios of some other methods [5, 7, 21] are of low quality.

Motivated by this, we study the k-regret query and propose a new algorithm called Sphere which meets all four requirements, i.e., Sphere is elegant. Our major contributions are as follows.

- Firstly, we propose an elegant algorithm called Sphere, which (1) has a restriction-free bound on the maximum regret ratio, (2) is executable in datasets of any dimensionality, (3) is asymptotically optimal in terms of the maximum regret ratio, and (4) enjoys a novel greedy strategy which is 20 times faster than the existing greedy strategy. It is the best-known result in the literature when at most k tuples are returned.
- Secondly, we conducted experiments on both real and synthetic datasets, showing that Sphere returns a tuple set with the smallest maximum regret ratio empirically and achieves a significant speedup over most of the existing methods.

The rest of the paper is organized as follows. Section 2 discusses the related work and Section 3 introduces the preliminaries. Section 4 presents the background techniques which are used in algorithm Sphere. Section 5 presents the algorithm Sphere. Section 6 presents the experimental results. Section 7 concludes the paper.

2 RELATED WORK

Motivated by the deficiencies of top-k queries and skyline queries, a number of new queries were proposed. Firstly, various approaches were proposed to control the output size of skyline queries [18, 30]. A representative query is the *k-representative skyline* query, which outputs a set of k skyline tuples which "best" represents all the skyline tuples. Secondly, effort was also spent to improve top-k queries [23, 34], where the utility function is determined implicitly. Besides, [16] requires a partial utility function from a user and [19] proposed an alternative framework to discover the utility function. Lastly, some other approaches were also proposed based on both top-k queries and skyline queries. [14] developed a top-k skyline select query and a top-k skyline join query. [33] studied an ϵ-skyline query whose output size is controlled by ϵ.

Recently, there are some queries [3, 12, 26] which do not heavily depend on top-k queries and skyline queries and one of them is the *k-regret* query. The k-regret query, also known as the *min-error regret query*, was first introduced in [21], where we want a set of at most k tuples whose *maximum regret ratio* is as small as possible. Finding an optimal solution for k-regret queries was proven to be an NP-hard problem [2, 7, 11]. A number of algorithms were proposed to obtain a solution with a small maximum regret ratio.

The first approach called Cube [21] divides the space into multiple hypercubes and constructs the solution by selecting a tuple from each hypercube. Cube has a known upper bound (i.e., $O(k^{-\frac{1}{d-1}})$) on the maximum regret ratio of the set returned, but its empirical performance is quite poor, which indicates that Cube dissatisfies the quality requirement. The second approach (Greedy [21], GeoGreedy [24] and StoredList [24]) is based on the idea of "greedy". Greedy and GeoGreedy iteratively construct the solution by choosing the tuple that contributes to the greatest value of the maximum regret ratio of the current set. StoredList is a materialized version of GeoGreedy. No upper bound on the maximum regret ratio of the set returned by a greedy-based algorithm is known yet and hence they violate the restriction-free bound requirement.

The k-regret query [21] was generalized to the kRMS problem in [11]. Note that the meanings of k are different in the above definitions. In the k-regret query, k represents the maximum size of the solution set while in the kRMS problem, we want a set of tuples whose highest utility is greater than the k-th highest utility in the whole dataset. In this paper, we follow the definition in [21] and study the k-regret query. A geometry-based algorithm (called 2d kRMS) was proposed in [11]. It solves a k-regret query optimally but it is only applicable when the dataset contains two attributes.

Recently, [2] studied a variation of k-regret queries, called *min-size regret queries*, which minimize the output size while keeping the maximum regret ratio at most ε. To answer a min-size regret query, they proposed an ε-kernel algorithm and a hitting-set algorithm. The former will be described in Section 4. The later can be modified into a 2-approximate algorithm called RMS_HS for k-regret queries (the constant 2 in the approximate factor is not important since it can be arbitrarily close to 1 by increasing the running time). However, instead of returning at most k tuples (the maximum allowable output size), RMS_HS returns $\Theta(k \log k)$ tuples which is not desired. Specifically, it sets the maximum output size to be $ck \log k$ for a large constant c. Then, it iteratively constructs a larger (smaller) solution set whenever the output size is at most (at least) $ck \log k$. The running time of RMS_HS depends on the maximum regret ratio of the returned set. If the maximum regret ratio is small, it might take a long time to return the solution.

Moreover, Asudeh et. al [5] transformed the k-regret query in a 2-dimensional dataset into a path search problem in a weighted complete graph, and an optimal algorithm called 2D-RRMS was proposed to solve the problem. Besides, they interpreted the k-regret query in a d-dimensional dataset as a discretized matrix min-max problem (the DMM problem) and the maximum regret ratio of the set returned was loosely upper bounded by $1 - c(1 - \text{mrr}^*)$ ($\leq \text{mrr}^* + 1 - c$, i.e., within a distance from the optimal maximum regret ratio, denoted by mrr^*) where c is a real number in (0,1).

The work [7] is closely related to ours. In 2-dimensional datasets, they presented two algorithms, namely E-Greedy-1 and A-IntCov-1, for the k-regret query. In d-dimensional datasets, they presented the ε-kernel algorithm for the min-size regret query independently, and translated it to an approximate algorithm for the k-regret query with an asymptotically optimal bound on the maximum regret ratio (i.e., $O(k^{-\frac{2}{d-1}})$). However, the bound is not restriction-free. Specifically, its bound is greater than 1 even under typical settings, making it difficult to be applied on real applications. A more detailed discussion on the ε-kernel algorithm [7] is postponed to Section 4.

Comparison with the existing algorithms. Compared with the existing algorithms, Sphere has the following advantages. Firstly, Sphere is elegant. That is, it satisfies all four requirements for the k-regret query while none of the existing algorithm could do. Secondly, Sphere returns at most k tuples while some exiting algorithms (e.g., RMS_HS[2]) might return more tuples than expected. Finally, Sphere is asymptotically optimal in term of the theoretical upper bound on the maximum regret ratio, a stronger result compared with some existing algorithms (e.g., Cube[21] and DMM[5]).

3 PROBLEM DEFINITION

The input to our problem is a tuple set $\mathbb{P}$ with n tuples (i.e., $|\mathbb{P}| = n$) in a d-dimensional space. A positive integer k is used to specify the maximum size of the solution set to be found.

3.1 Terminologies

We use the word "tuple" and "point" interchangeably in the rest of this paper. We denote the i-th dimensional value of a point $p \in \mathbb{P}$ by $p[i]$ where $i \in [1, d]$. Without loss of generality, each dimension is normalized to (0,1] and for each $i \in [1, d]$, there exists at least one point $p \in \mathbb{P}$ such that $p[i] = 1$. We name one of them as the i-th dimensional boundary point, denoted by b_i. We assume that a large value in each dimension is preferable by the users. Recall that in a car database, each car is associated with 2 attributes, namely HP and MPG. Consider the car database $\mathbb{P} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, containing 6 points (with normalized attribute values) in Table 2. Since $p_4[1] = 1$ and $p_1[2] = 1$, we have $b_1 = p_4$ and $b_2 = p_1$.

Assume that the user's happiness is measured by an unknown *utility function*. A utility function f is a mapping $f : \mathbb{R}_+^d \to \mathbb{R}_+$. Denote the *utility* of a point p w.r.t. the function f by $f(p)$. A user wants a point in $\mathbb{P}$ which maximizes his/her utility w.r.t. his/her utility function. Given a utility function f, a point q in $\mathbb{P}$ is said to have the *maximum utility of* $\mathbb{P}$ *w.r.t.* f if $f(q) = \max_{p\in\mathbb{P}} f(p)$, and we call q the *maximum utility point of* $\mathbb{P}$ *w.r.t.* f.

Definition 3.1 ([21]). Given a set $S \subseteq \mathbb{P}$ and a utility function f, the *regret ratio* of S over $\mathbb{P}$ w.r.t. f , denoted by $\mathsf{rr}_{\mathbb{P}}(S, f)$, is defined to be $\frac{\max_{p\in\mathbb{P}} f(p) - \max_{p\in S} f(p)}{\max_{p\in\mathbb{P}} f(p)} = 1 - \frac{\max_{p\in S} f(p)}{\max_{p\in\mathbb{P}} f(p)}$.

Given a utility function f and a set $S \subseteq \mathbb{P}$, we have $\max_{p\in S} f(p) \leq \max_{p\in\mathbb{P}} f(p)$ since S is a subset of $\mathbb{P}$ and thus, the regret ratio ranges from 0 to 1. The user with utility function f will be happy if the regret ratio is close to 0 since the maximum utility of S is close to the maximum utility of $\mathbb{P}$ w.r.t. f. Unfortunately, in real cases, it is difficult to obtain the utility function of a user. Thus, in this paper, we assume that the utility function of a user is in a function class, denoted by $\mathcal{FC}$. The *maximum regret ratio* of a set S is defined over a function class $\mathcal{FC}$ which can be regarded as the worst-case regret ratio w.r.t. a utility function in $\mathcal{FC}$.

Definition 3.2 ([21]). Given a set $S \subseteq \mathbb{P}$ and a function class $\mathcal{FC}$, the *maximum regret ratio* of S over $\mathbb{P}$ w.r.t. $\mathcal{FC}$ is defined to be $\max_{f\in\mathcal{FC}} \mathsf{rr}_{\mathbb{P}}(S, f)$.

To illustrate, we assume that $\mathcal{FC}$ consists of three utility functions, namely $f_{0.4,0.6}$, $f_{0.2,0.8}$ and $f_{0.7,0.3}$ where $f_{a,b}(p) = a\times p[1] + b\times p[2]$. Consider p_1 in Table 2. Its utility w.r.t. $f_{0.4,0.6}$ is $f_{0.4,0.6}(p_1) = 0.4 \times 0.2 + 0.6 \times 1 = 0.68$. The utilities of the remaining points in $\mathbb{P}$ w.r.t. $f_{0.4,0.6}$ are computed in a similar way and they are shown in Table 2. Consider a set $S = \{p_1, p_4\}$. The maximum utility point of S w.r.t. $f_{0.4,0.6}$ is p_1 and its utility is equal to 0.68 while the maximum utility point of $\mathbb{P}$ w.r.t. $f_{0.4,0.6}$ is p_2 and its utility is equal to 0.78. Then, $\mathsf{rr}_{\mathbb{P}}(S, f_{0.4,0.6}) = 1 - \frac{\max_{p\in S} f_{0.4,0.6}(p)}{\max_{p\in\mathbb{P}} f_{0.4,0.6}(p)} = 1 - \frac{0.68}{0.78} = 0.1282$. Similarly, we have $\mathsf{rr}_{\mathbb{P}}(S, f_{0.2,0.8}) = 0$ and $\mathsf{rr}_{\mathbb{P}}(S, f_{0.7,0.3}) = 0.0617$. Then, the maximum regret ratio of S w.r.t. $\mathcal{FC}$ is computed to be $\max_{f\in\mathcal{FC}} \mathsf{rr}_{\mathbb{P}}(S, f) = \max\{0.1282, 0, 0.0617\} = 0.1282$.

In this paper, we focus our analysis on the class of *linear utility functions* which contains infinite number of utility functions and it is very popular in modeling user preferences [10, 11, 17, 20, 21, 24]. We say that 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. For the sake of standardization, we assume that the utility vector u has its norm (the L2-norm) equal to 1 (i.e., $\|u\| = 1$). In the rest of this paper, we denote the maximum regret ratio of S over $\mathbb{P}$ w.r.t. the class of infinite number of linear utility functions by $\mathsf{mrr}_{\mathbb{P}}(S)$. Other classes of utility functions are considered in [25, 32]. The computation of maximum regret ratio is presented in [21, 24] and we omit their details here. The frequently used notations are summarized in the appendix (see Table 4).

3.2 Problem Definition

The k-regret query [21]. Given a positive integer k, the k-regret query is to find a subset $S \subseteq \mathbb{P}$ containing at most k points such that the maximum regret ratio $\mathsf{mrr}_{\mathbb{P}}(S)$ is minimized.

Car	HP	MPG	$f_{0.4,0.6}(p)$	$f_{0.2,0.8}(p)$	$f_{0.7,0.3}(p)$
p_1	0.2	1	0.68	0.84	0.44
p_2	0.6	0.9	0.78	0.84	0.69
p_3	0.9	0.6	0.72	0.66	0.81
p_4	1	0.2	0.52	0.36	0.76
p_5	0.35	0.2	0.26	0.23	0.305
p_6	0.3	0.6	0.48	0.54	0.39

Table 2: Car Database and Car Utilities

We assume that $k \geq d$. Otherwise, the maximum regret ratio of a set can be unbounded [21]. It is shown [2, 7, 11] that solving the k-regret query *optimally* is NP-hard. Instead, we propose an algorithm which returns an *asymptotically optimal* solution with a restriction-free bound for datasets of any dimensionality efficiently.

3.3 Lower Bound of Maximum Regret Ratio

In this section, we present a lower bound of the maximum regret ratio in the following theorem, which is $\Omega(k^{-2/(d-1)})$.

Theorem 3.3. *There is a d-dimensional database such that the maximum regret ratio of any set of k points is at least $\frac{1}{8}(2k)^{-\frac{2}{d-1}}$.*

Proof. The theorem follows from Theorem 8 in [20], which proved that, given a real number $\varepsilon \in (0, 1]$, we should return at least $\frac{1}{2}(\frac{1}{8\varepsilon})^{\frac{d-1}{2}}$ points to guarantee a regret ratio ε. In other words, by returning at most k points, the regret ratio is at least $\frac{1}{8}(2k)^{-\frac{2}{d-1}}$. □

In later sections, we present our algorithm Sphere, which returns a set with a maximum regret ratio of $O(k^{-2/(d-1)})$. Based on the lower bound in Theorem 3.3, Sphere is asymptotically optimal.

4 BACKGROUND TECHNIQUES: ε-KERNEL

As described in Section 2, we first describe the details of the ε-kernel algorithm in this section. Then, we elaborate why it does not satisfy the restriction-free bound requirement and why it is difficult to be applied in real cases. The reason why we need to describe this algorithm in detail is that a variation of this algorithm will be used in our Sphere algorithm (to be described in Section 5).

The concept, ε-kernel, was first introduced by Agarwal et al. [1]. Specifically, $S \subseteq \mathbb{P}$ is an ε-kernel of $\mathbb{P}$ if $\frac{\max_{p\in S} v\cdot p - \min_{p\in S} v\cdot p}{\max_{p\in\mathbb{P}} v\cdot p - \min_{p\in\mathbb{P}} v\cdot p} \geq 1 - \varepsilon$ for each non-zero vector v. Intuitively, an ε-kernel preserves the "width" of $\mathbb{P}$ for each direction. It was shown in [2, 7] that the concept "ε-kernel" is closely related to the k-regret query. Specifically, if $S \subseteq \mathbb{P}$ is an ε-kernel of $\mathbb{P}$, $\mathsf{mrr}_{\mathbb{P}}(S) \leq \varepsilon$. Besides, it is well-known that an ε-kernel of size $O(\varepsilon^{-\frac{d-1}{2}})$ can be computed in $O(n + \frac{1}{\varepsilon^d})$ time [2, 7]. Then, the following theorem follows directly.

Theorem 4.1 ([2, 7]). *Given a real value $\varepsilon > 0$, one can compute a set $S \subseteq \mathbb{P}$ of size $O(\varepsilon^{-\frac{d-1}{2}})$ in $O(n + \frac{1}{\varepsilon^d})$ time with $\mathsf{mrr}_{\mathbb{P}}(S) \leq \varepsilon$.*

Wei et. al [7] translated Theorem 4.1 to an approximate algorithm for the k-regret query by setting a proper value of ε to obtain an ε-kernel of at most k points. Informally, given $\varepsilon > 0$, one can obtain a set S with at most $c\varepsilon^{-\frac{d-1}{2}}$ points and $\mathsf{mrr}_{\mathbb{P}}(S) \leq \varepsilon$ where c is a (sufficiently large) constant depending on d according to Theorem 4.1. It suffices to let $c\varepsilon^{-\frac{d-1}{2}} = k$. Then, $\mathsf{mrr}_{\mathbb{P}}(S) \leq \varepsilon = (\frac{c}{k})^{\frac{2}{d-1}} = O(k^{-\frac{2}{d-1}})$. However, the bound above is not restriction-free. In a typical setting of d, c can be much larger than k . In case

that $k < c$, the bound $(\frac{c}{k})^{\frac{2}{d-1}} > 1$ becomes useless. Even worse, some algorithms for computing ε-kernel have more restricted conditions for the bound to be valid (e.g., ε must be at most $\frac{1}{4}$ [35] and thus, k must be at least $\frac{c}{\frac{1}{2}^{\frac{2}{d-1}}}$). We show how large c can be next.

We define a few terminologies first. The *convex hull* of a set $\mathbb{P}$ of points in a d-dimensional space, denoted by $\mathcal{CH}(\mathbb{P})$, is the smallest convex set containing $\mathbb{P}$. A *facet* of the convex hull, denoted by F, is a bounded flat surface that forms a part of the boundary of the convex hull. The *diameter* of a facet F is defined to be the maximum distance between any two points on F. Let $\mathbb{C}$ be the d-dimensional hypercube $[-1,+1]^d$ (which is a convex hull and has $2d$ facets). Let $\mathbb{S}_R$ be the surface of the sphere of radius R centered at the origin O.

Example 4.2. Consider the example in Table 2. For the ease of presentation, we visualize the points $\mathbb{P} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ in Figure 1 where the X_1 coordinate and the X_2 coordinate represent HP and MPG, respectively. The convex hull of $\mathbb{P}$, denoted by $\mathcal{CH}(\mathbb{P})$, is shown in Figure 2. In this 2-dimensional example, the line segment between p_1 and p_2 is one of the facets of $\mathcal{CH}(\mathbb{P})$. □

It is well-known that one can compute a set $\mathcal{I}_R$ of points in $\mathbb{S}_R$ such that for each s in $\mathbb{S}_R$, there is a point s' in $\mathcal{I}_R$ and $\text{dist}(s,s')$ is bounded where $\text{dist}(s,s')$ denotes the Euclidean distance between s and s'. Formally, we have the following lemma when $R = 1$ [1].

Lemma 4.3 ([1]). *Given $\delta > 0$, one can compute a set $\mathcal{I}_1$ of $O(\frac{1}{\delta^{d-1}})$ points in $\mathbb{S}_1$ such that for each s in $\mathbb{S}_1$, there is a point s' in $\mathcal{I}_1$ and $\text{dist}(s,s') \le \delta$. Specifically, $|\mathcal{I}_1| = 2dm^{d-1}$ where $m = \lceil \frac{2\sqrt{d-1}}{\delta} \rceil$.*

This is done by partitioning each of the $2d$ facets of $\mathbb{C}$ into m^{d-1} $(d-1)$-dimensional "hypercubes" of diameter δ where m is computed to be $\lceil \frac{2\sqrt{d-1}}{\delta} \rceil$, resulting in $2dm^{d-1}$ $(d-1)$-dimensional hypercubes in total. For each $(d-1)$-dimensional hypercube, one point is computed and included into $\mathcal{I}_1$ (i.e., $|\mathcal{I}_1| = 2dm^{d-1}$). We refer the readers who do no familiar with the above process to [1].

Corollary 4.4. *Given $\delta > 0$, one can compute a set $\mathcal{I}_R$ of $O(\frac{1}{\delta^{d-1}})$ points in $\mathbb{S}_R$ such that for each s in $\mathbb{S}_R$, there is a point s' in $\mathcal{I}_R$ and $\text{dist}(s,s') \le \delta$. Specifically, $|\mathcal{I}_R| = 2dm^{d-1}$ where $m = \lceil \frac{2\sqrt{d-1}R}{\delta} \rceil$.*

The algorithm for computing an ε-kernel [35]. We compute the set $\mathcal{I}_R$ according to Corollary 4.4 by setting $R = 1 + \sqrt{d}$ and $\delta = \sqrt{\varepsilon\alpha}$ where α is a constant in (0,1] depending only on d [1]. For each point s' in $\mathcal{I}_R$, we compute its *ε-approximate nearest-neighbor* [4], denoted by $\varphi(s')$, in $\mathbb{P}$. $S = \{\varphi(s') | s' \in \mathcal{I}_R\}$ is an ε-kernel of $\mathbb{P}$.

According to the procedure above, $|S| = |\mathcal{I}_R| = 2dm^{d-1} \approx 2d(2\sqrt{d-1}R/\delta)^{d-1} = c\varepsilon^{-\frac{d-1}{2}}$ where $c = 2d(2\sqrt{d-1}(1+\sqrt{d})/\sqrt{\alpha})^{d-1}$. Consider a typical setting with $d = 6$. $c \ge 2d(2\sqrt{d-1}(1+\sqrt{d}))^{d-1} \approx 10^7$. In other words, the ε-kernel algorithm provides a valid bound on the maximum regret ratio only when $k \ge 10^7$ on a 6-dimensional dataset. This requirement is very restricted and makes the algorithm difficult to be applied in real cases.

In later sections, we introduce our algorithm, Sphere, for solving the k-regret query. Unlike the ε-kernel algorithm which has restricted applicability, the bound of Sphere, which is also asymptotically optimal, is restriction-free, i.e., it is valid for any setting.

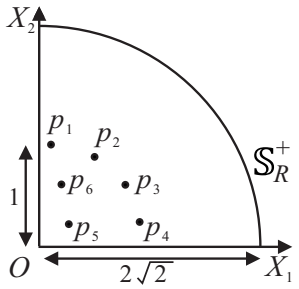


Figure 1: Car database

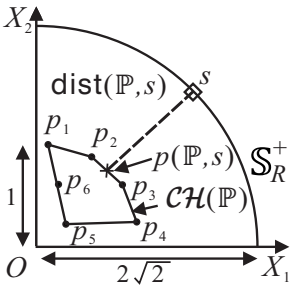


Figure 2: Convex hull

5 ALGORITHM

We are ready to present the algorithm, Sphere, which is both elegant and asymptotically optimal. We present some preliminaries in Section 5.1, introduce Sphere in Section 5.2, and then analyze the time complexity and the theoretical guarantee in Section 5.3.

5.1 Preliminaries

We define $\mathbb{S}_R^+ \subseteq \mathbb{S}_R$ to be the set of points in $\mathbb{S}_R$ (defined in Section 4) which are in the positive quadrant. (i.e., $\mathbb{S}_R^+ = \{s \in \mathbb{R}^d | s[i] \ge 0,\ \forall i \in [1,d] \text{ and } \|s\| = R\}$). In this section, we set $R = 2\sqrt{d}$. We will discuss why we set this value later.

Given a set $P \subseteq \mathbb{P}$ and a point $s \in \mathbb{S}_R^+$, we define the distance between P and s, denoted by $\text{dist}(P,s)$, to be the minimum distance between a point in $\mathcal{CH}(P)$ and s. Formally, we write $\text{dist}(P,s) = \min_{p \in \mathcal{CH}(P)} \text{dist}(p,s)$. We denote the point that *realizes* the distance $\text{dist}(P,s)$ by $p(P,s)$. That is, $p(P,s) = \arg\min_{p \in \mathcal{CH}(P)} \text{dist}(p,s)$.

Given a point $s \in \mathbb{S}_R^+$, a *basis* B of s is a set of at most d points such that for each proper subset B' of B, we have $\text{dist}(B,s) < \text{dist}(B',s)$. Given a set $P \subset \mathbb{P}$ and a point $s \in \mathbb{S}_R^+$, we say that a set $B \subseteq P$ is a *P-basis* of s if (1) B is a basis (i.e., $\text{dist}(B,s) < \text{dist}(B',s), \forall B' \subset B$), and (2) we have $\text{dist}(B,s) = \text{dist}(P,s)$. Intuitively, a P-basis of s is a minimal subset of P whose distance to s is equal to the distance between P and s. Note that a basis $B \subseteq P$ of s is not necessary a P-basis of s if $\text{dist}(B,s) \ne \text{dist}(P,s)$.

Example 5.1. Consider the set $\mathbb{P}$ and a point s in $\mathbb{S}_R^+$ in Figure 2. The distance between $\mathbb{P}$ and s, namely $\text{dist}(\mathbb{P},s)$, is drawn in a dashed line. The point realizing the distance $\text{dist}(\mathbb{P},s)$ is indicated by $p(\mathbb{P},s)$. Consider $B \subseteq \mathbb{P}$ where $B = \{p_2, p_3\}$. Firstly, B is a basis of s since for each $B' \subset B$, $\text{dist}(B,s) < \text{dist}(B',s)$ (e.g., $\text{dist}(B,s) < \text{dist}(\{p_2\},s)$ and $\text{dist}(B,s) < \text{dist}(\{p_3\},s)$) where $\text{dist}(B,s)$ is the minimum distance between a point in $\mathcal{CH}(B)$ (i.e., the line segment connected by p_2 and p_3) and s. Secondly, $\text{dist}(B,s) = \text{dist}(\mathbb{P},s)$ (and indeed, the points realizing both $\text{dist}(B,s)$ and $\text{dist}(\mathbb{P},s)$, labeled as $p(\mathbb{P},s)$ in the figure, are the same). In other words, $B = \{p_2, p_3\}$ is a $\mathbb{P}$-basis of s. □

5.2 Algorithm Sphere

We want to bound $\text{mrr}_{\mathbb{P}}(S)$. Intuitively, given a utility function f, even if the maximum utility point p of $\mathbb{P}$ w.r.t. f is not in S, we want to guarantee that there is a point q in S such that the difference between the utility of q and the utility of p w.r.t. f (i.e., $f(p) - f(q)$) is bounded and thus the regret ratio of S w.r.t. f is bounded.

Based on this idea, we develop Algorithm Sphere. Intuitively, Sphere constructs a set $\mathcal{I}_R^+ \subseteq \mathbb{S}_R^+$ which can be regarded as a

set of points "uniformly" distributed on the sphere that $\mathbb{S}_R^+$ lies on. Then, for each point s in $\mathbb{S}_R^+$, there exists a point $s' \in \mathcal{I}_R^+$ such that $\text{dist}(s, s')$ is bounded (analogous to the relation between $\mathcal{I}_R$ and $\mathbb{S}_R$ in Section 4). Given a point $s \in \mathbb{S}_R^+$, we denote the utility function whose utility vector is in the same direction of s by f_s. Ideally, for each point $s' \in \mathcal{I}_R^+$, we find the point q in $\mathbb{P}$ with a "high" utility w.r.t. $f_{s'}$ and include q to S. Since the distance $\text{dist}(s, s')$ is bounded, the utility of q (which achieves high utility w.r.t. $f_{s'}$) in S w.r.t. f_s is also bounded. This is true for all s in $\mathbb{S}_R^+$.

Note that the radius R of the sphere that $\mathbb{S}_R^+$ lies on is set intentionally to be $2\sqrt{d}$. Let s be a point in $\mathbb{S}_R^+$ and s' be the point in $\mathcal{I}_R^+$ such that $\text{dist}(s, s')$ is bounded. Intuitively, if R is too large, the bound of the distance $\text{dist}(s, s')$ is large (since s and s' have large norms), which will affect tightness of the final bound on the maximum regret ratio; if R is too small, the norm of s' is close to the norms of points in $\mathbb{P}$ and thus, the "discriminant ability" of s' is bad. In particular, the point q, determined by s' to be inserted into S, which is supposed to have a "high" utility w.r.t. $f_{s'}$, might in fact have a poor utility, and thus, gives a loose bound on its utility w.r.t. f_s and hence a loose bound on the maximum regret ratio. We will formally present the discussion on the trade-off of R in Section 5.3.

Formally, our algorithm Sphere involves four steps. Note that we integrate a greedy step at the end of the algorithm. The intuition is that there is no harm to include more points into S as long as S does not violate the size constraint in the k-regret query. The pseudocode of the Sphere algorithm is presented in Algorithm 1.

- **Step 1 (Initialization):** S is initialized to $\{b_1, b_2, \ldots, b_d\}$ where b_i is the i-th dimensional boundary point.
- **Step 2 (Constructing set $\mathcal{I}_R^+$):** We construct a set $\mathcal{I}_R^+$ of k' points (where k' is an integer computed based on k and d) in $\mathbb{S}_R^+$ such that for each s in $\mathbb{S}_R^+$, there is a point $s' \in \mathcal{I}_R^+$ and $\text{dist}(s, s') \leq \delta$ where δ is a non-negative number computed based on the way we construct $\mathcal{I}_R^+$ (see Section 5.2.1).
- **Step 3 (Finding $\mathbb{P}$-basis):** For each $s' \in \mathcal{I}_R^+$, we search its $\mathbb{P}$-basis, denoted by $B(s')$. The basis search can be done efficiently [13, 28]. Intuitively, a basis $B \subseteq \mathbb{P}$ (not necessarily a $\mathbb{P}$-basis) is maintained during the basis search. We iteratively examine each point p in $\mathbb{P}$ and check if we can incorporate p to B to reduce the distance $\text{dist}(B, s)$. Eventually, no point in $\mathbb{P}$ can be used to reduce the distance and the resulting basis B is indeed the desired $\mathbb{P}$-basis. The interested readers can find the detailed algorithm in [13, 28]. For each basis found, we include all points in the basis into S. Note that the number of points in S after this step is at most k. We will discuss this issue in more detail in Section 5.2.2.
- **Step 4 (Inserting additional points):** If $|S| < k$ after the third step, we greedily include points into S until S contains k points. The greedy strategy (including the computation of $\text{mrr}_{\mathbb{P}}(S)$) is described in Section 5.2.3.

Example 5.2. Consider the example in Figure 2. Assume that k is set to 4. Firstly, S is initialized to be $\{p_1, p_4\}$ which are the boundary points in $\mathbb{P}$. Secondly, based on a formula (to be shown later), we compute k' to be 1. We construct the set $\mathcal{I}_R^+ = \{s'\}$ where $s' = s$ which is shown in Figure 2. Thirdly, we find its $\mathbb{P}$-basis which is $B = \{p_2, p_3\}$. Then, $S = S \cup B = \{p_1, p_2, p_3, p_4\}$. Since $|S| = 4 (= k)$ in this case, we do not need to perform Step 4. Thus, S is $\{p_1, p_2, p_3, p_4\}$. □

Algorithm 1 Algorithm Sphere

Require: A tuple set $\mathbb{P} \subseteq \mathbb{R}^d$ ($|\mathbb{P}| = n$) and an integer k
Ensure: A subset $S \subseteq \mathbb{P}$ with $|S| \leq k$

```
/* Step 1: initialization */
S ← {b_1, b_2, ..., b_d}
/* Step 2: constructing I_R^+ */
Construct a set I_R^+ with the desired property.
/* Step 3: finding the P-basis of each s' ∈ I_R^+ */
for each s' ∈ I_R^+ do
    B(s') ← the P-basis of s'
    S ← S ∪ B(s')
/* Step 4: inserting points greedily into S until |S| = k. */
while |S| < k do
    q ← the point that realizes the current mrr_P(S).
    S ← S ∪ {q}
return S
```

Comparison with the existing algorithms: Compared with the existing algorithms [2, 5, 7, 11, 21, 24] for the k-regret query, our algorithm is both elegant and asymptotically optimal in term of maximum regret ratio which none of the existing algorithms is.

5.2.1 The procedure of constructing $\mathcal{I}_R^+$. We define a few terminologies first. Let $\mathbb{C}^+ \subseteq \mathbb{C}$ to be the set of points in $\mathbb{C}$ (defined in Section 4) which are in the positive quadrant (i.e., $\mathbb{C}^+ = [0, 1]^d$). *A front facet* of $\mathbb{C}^+$ is defined to be a facet of $\mathbb{C}^+$ that does not pass through the origin. There are d front facets of $\mathbb{C}^+$. In the following, when we say a facet of $\mathbb{C}^+$, we mean the front facet of $\mathbb{C}^+$.

We describe our procedure of constructing $\mathcal{I}_R^+$ in Algorithm 1 as follows. We want the set $\mathcal{I}_R^+$ with the following two properties:

- **Size property.** The set $\mathcal{I}_R^+$ must be designed in a way that the size of S does not exceed k (i.e., the maximum allowable output size) after we include all $\mathbb{P}$-basis of points in $\mathcal{I}_R^+$.
- **Distance property.** For each s in $\mathbb{S}_R^+$, there is a point $s' \in \mathcal{I}_R^+$ and $\text{dist}(s, s') \leq \delta$ (which we call the *distance bound*). This property is crucial to bound the maximum regret ratio.

Depending on the value of k, which constrains the number of points that we can include to S without violating the size property after the initialization (S is initialized to be $\{b_1, \ldots, b_d\}$), we construct $\mathcal{I}_R^+$ in different ways. According to the procedure in Sphere, for each point in $\mathcal{I}_R^+$, we include its $\mathbb{P}$-basis, which is a set of at most d points, into the solution set S. If k is so small that we cannot include one basis into S, $\mathcal{I}_R^+$ is defined to be empty (Case 1).

If we are allowed to include at least one basis into S, we want the set $\mathcal{I}_R^+$ of points to be "uniformly" distributed on $\mathbb{S}_R^+$ so that the distance property is met and δ is as small as possible. We construct such $\mathcal{I}_R^+$ with the help of the front facets of $\mathbb{C}^+$ by adapting Lemma 4.3[1], which allows an efficient implementation and gives the desired distance bound. Intuitively, each front facet of $\mathbb{C}^+$ (which is itself a hypercube) can be divided into several smaller hypercubes so that the distance between two points on the same hypercube is bounded by the *diameter* of that hypercube. By creating a point s' for each hypercube and insert it to $\mathcal{I}_R^+$, we can relate the value of δ with the diameter of the hypercube and guarantee the desired distance property. However, since there are d front facets

of $\mathbb{C}^+$ in total, we have to evenly create points for the hypercubes on each of the d front facets. Otherwise, the distance bound might not be guaranteed on some facets of $\mathbb{C}^+$. If we can only include strictly less than d points in $\mathcal{I}_R^+$ due to the size property (but there are d front facets) (Case 2), we construct $\mathcal{I}_R^+$ with the help of the "corner" point on the front facets of $\mathbb{C}^+$, the only point in $\mathbb{C}^+$ that lies on all front facets of $\mathbb{C}^+$. To be shown later, $\mathcal{I}_R^+$ is then defined to contain the point on the "center" of $\mathbb{S}_R^+$ whose distance to other points is minimized. If we can include at least d points in $\mathcal{I}_R^+$ (Case 3), we naturally adapt Lemma 4.3[1] to create points in $\mathcal{I}_R^+$ for each of the d front facets of $\mathbb{C}^+$. Formally, we have the following cases:

Case 1 ($d \le k < 2d$, i.e., the number of points that we can include into S is in $[0, d)$ after the initialization): We define $\mathcal{I}_R^+$ to be the empty set and thus, $k' = 0$. Step 2 and Step 3 of Sphere are skipped. Intuitively, since S is initialized to the set of d boundary points, we are allowed to include strictly less than d points to S, which might be less than the size of one basis, after the initialization of Sphere. Otherwise, we may violate the size constraint in the k-regret query.

For example, consider the example in Figure 2 where we set k to be 3 and $k < 2d = 4$ (Case 1). S is initialized to be $\{p_1, p_4\}$ as before. Suppose that we construct $\mathcal{I}_R^+ = \{s\}$ as shown in Figure 2. If we include its $\mathbb{P}$-basis, i.e., $B = \{p_2, p_3\}$ into S, $|S| = 4 > k$ violates the size constraint in the k-regret query.

Case 2 ($2d \le k < d^2 + d$, i.e., the number of points that we can include into S is in $[d, d^2)$ after the initialization): We set $k' = 1$ and $\mathcal{I}_R^+ = \{s'\}$ where $s'[i] = 2$ for each $i \in [1, d]$.

When $2d \le k < d^2 + d$, we are allowed to include less than d^2 points into S in Step 2 and Step 3 of Sphere. That is, $\mathcal{I}_R^+$ is allowed to contain strictly less than d points (since each point in $\mathcal{I}_R^+$ can lead to the insertion of d points into S). In other words, we cannot evenly create points into $\mathcal{I}_R^+$ ($|\mathcal{I}_R^+| < d$) for each of the d front facets of $\mathbb{C}^+$. Alternatively, the desired $\mathcal{I}_R^+$ is obtained as follows.

Let $u'^+ = \frac{1}{2}s'$ where s' is defined above. Note that u'^+ is the "corner" point on the facets of $\mathbb{C}^+$ (which is the only point in $\mathbb{C}^+$ that lies in all front facets of $\mathbb{C}^+$). For each s in $\mathbb{S}_R^+$, let u^+ be the intersection between the facets of $\mathbb{C}^+$ and the ray shooting from the origin to s. Let $\beta = \sqrt{d-1}$ which is the diameter of a facet of $\mathbb{C}^+$. Since u^+ and u'^+ are on the same facet, we have $\text{dist}(u^+, u'^+) \le \beta$. Note that s' and s are the "scaled" points of u'^+ and u^+. Since $\text{dist}(u^+, u'^+)$ is bounded, the distance $\text{dist}(s, s')$ is also bounded. Intuitively, s' is the point on the "center" of $\mathbb{S}_R^+$ so that the maximum distance between it and a point in $\mathbb{S}_R^+$ is as small as possible (i.e., δ is small). Note that we can insert some additional points into $\mathcal{I}_R^+$ if the size property allows. However, it might not guarantee a smaller distance bound according to our discussion before. We left it as a future direction to obtain an asymptotically smaller distance bound by creating more points into $|\mathcal{I}_R^+|$. We formally prove the value of δ next and the result is summarized in the following lemma.

Lemma 5.3. *Given $\mathcal{I}_R^+ = \{s'\}$ where $s'[i] = 2$ for each $i \in [1, d]$, for each s in $\mathbb{S}_R^+$, we have $\text{dist}(s, s') \le \delta$ where $\delta = R\sqrt{d-1}$.*

Proof. Let u and u' be the normalized vector of s and s' respectively. That is, $s = Ru$ and $s' = Ru'$ where $\|u\| = \|u'\| = 1$. According to the discussion in [1], $\text{dist}(u, u') \le \text{dist}(u^+, u'^+) \le \beta$. Then, we have $\text{dist}(s, s') = R\text{dist}(u, u') \le R\beta = R\sqrt{d-1} = \delta$. □

For example, consider Figure 2 again. We construct $\mathcal{I}_R^+ = \{s'\}$ where $s'[1] = 2$ and $s'[2] = 2$ (the squared point shown in Figure 2). It can be verified easily that s' is the point on the "center" of $\mathbb{S}_R^+$ which is the position minimizing its distance to other points in $\mathbb{S}_R^+$.

Case 3 ($k \ge d^2 + d$, i.e., the number of points that we can include into S is at least d^2 after the initialization): We set k' to be dm^{d-1} where $m = \lfloor (\frac{k-d}{d^2})^{\frac{1}{d-1}} \rfloor \ge 1$. Note that the value of m is determined intentionally. If m is large, the size of the final set might be larger than k (the maximum allowable output size). If m is small, the maximum regret ratio obtained might be loosely bounded.

The major idea of the construction of $\mathcal{I}_R^+$ is similar to the process introduced in Lemma 4.3[1]. That is, we partition the front facets of $\mathbb{C}^+$ into a number of $(d-1)$-dimensional hypercubes and construct $\mathcal{I}_R^+$ from each of the $(d-1)$-dimensional hypercubes. We summarize the result in the following lemma.

Lemma 5.4. *Given an integer k' which can be written as $k' = dm^{d-1}$ where m is a positive integer, one can compute a set $\mathcal{I}_R^+$ of k' points in $\mathbb{S}_R^+$ such that for each s in $\mathbb{S}_R^+$, there is a point s' in $\mathcal{I}_R^+$ and $\text{dist}(s, s') \le \delta$ where $\delta = \frac{\sqrt{d-1}R}{2m}$.*

Proof Sketch. We modify Lemma 4.3 to construct $\mathcal{I}_R^+$. The difference is that we focus on the sphere $\mathbb{S}_R^+$ in the first quadrant instead of $\mathbb{S}_R$. The complete proof and a detailed example illustrating Lemma 5.4 is presented in the appendix due to the lack of space. □

The value of δ in Case 3 is smaller than that in Case 2 which is intuitive since we are allowed to include more points in Case 3.

5.2.2 The number of points in S. We verify in the following lemma that after Step 3, S has at most k points.

Lemma 5.5. *$|S| \le k$ after Step 3 of Sphere.*

Proof. We prove the lemma case by case.
Case 1 ($d < k < 2d$): Since $|\mathcal{I}_R^+| = 0$, we skip Step 2 and Step 3 in Sphere. S contains d points after the initialization and $|S| = d < k$.
Case 2 ($2d \le k < d^2 + d$): Let $\mathcal{I}_R^+ = \{s'\}$. According to Sphere, we search the $\mathbb{P}$-basis of s', namely $B(s')$, and include all points in $B(s')$ into S, initialized to be the set of d boundary points. That is, after the first three steps, we have $|S| = d + |B(s')| \le d + d = 2d \le k$.
Case 3 ($k \ge d^2 + d$): According to Step 2, $|\mathcal{I}_R^+| = dm^{d-1}$ where $m^{d-1} = \lfloor (\frac{k-d}{d^2})^{\frac{1}{d-1}} \rfloor^{d-1} \le \frac{k-d}{d^2}$. Consider Step 3 ($S = \bigcup_{s' \in \mathcal{I}_R^+} B(s') \cup S$). $|S| \le d + \sum_{s' \in \mathcal{I}_R^+} |B(s')| \le d + d|\mathcal{I}_R^+| \le d + d \times d\frac{k-d}{d^2} = k$. □

5.2.3 The greedy strategy. Denote the set returned after the first three steps of Sphere by S. The greedy strategy performs in iterations. In each iteration, the point in $\mathbb{P}$ that *realizes the current maximum regret ratio* $\text{mrr}_{\mathbb{P}}(S)$ is included into the current set S until S contains k points or $\text{mrr}_{\mathbb{P}}(S)$ is 0. We say that a point q realizes the maximum regret ratio $\text{mrr}_{\mathbb{P}}(S)$ if $\text{mrr}_{\mathbb{P}}(S) = \text{mrr}_{S \cup \{q\}}(S)$ [21]. Such a point q is determined by computing $\text{mrr}_{S \cup \{p\}}(S)$ for each $p \in \mathbb{P}$ and $q = \arg\max \text{mrr}_{S \cup \{p\}}(S)$. We compute $\text{mrr}_{S \cup \{p\}}(S)$ by formulating it as a linear programming (LP) problem [21]:

$$\begin{array}{lrcll} \max & x & & & \\ s.t. & (p-q) \cdot u & \ge & x & \forall q \in S \\ & p \cdot u & = & 1 & \\ & u[j] & \ge & 0 & \forall 1 \le j \le d \end{array} \quad (1)$$

where the optimal solution x^* is the desired $\mathsf{mrr}_{S\cup\{p\}}(S)$ and u^* is the utility vector of f^* such that $\mathsf{rr}_{S_{i-1}\cup\{p\}}(S_{i-1}, f^*) = x^*$.

In order to determine the point realizing the current maximum regret ratio, we need to solve the LP (1) for each point in $\mathbb{P}$ in each iteration, which is very expensive. In the following, we introduce a number of punning strategies for reducing the number of expensive LP computations. We make the following observations:

(1) **Upper bounding:** We want the point with the largest maximum regret ratio $\mathsf{mrr}_{S\cup\{p\}}(S)$. Before computing the exact $\mathsf{mrr}_{S\cup\{p\}}(S)$ for each p in $\mathbb{P}$, we can first compute an upper bound of $\mathsf{mrr}_{S\cup\{p\}}(S)$. If the bound is at most the largest maximum regret ratio observed so far, it cannot be the point realizing the current maximum regret ratio.
(2) **Invariant checking:** The results of the LP problems computed in previous iterations can be re-used directly for computing the $\mathsf{mrr}_{S\cup\{p\}}(S)$ in the current iteration provided that certain conditions are satisfied (to be shown shortly).

In the following, we denote the set after the i-th iteration by S_i. Consider a function f with utility vector u and a point p in $\mathbb{P}$. We denote the maximum regret ratio $\mathsf{mrr}_{S_{i-1}\cup\{p\}}(S_{i-1})$ by $\mathsf{mrr}_{i-1}(p)$. We say that u is *the worst utility vector* for $\mathsf{mrr}_{i-1}(p)$, denoted by $u_{i-1}(p)$, if $\mathsf{rr}_{S_{i-1}\cup\{p\}}(S_{i-1}, f) = \mathsf{mrr}_{i-1}(p)$. A point $q_i \in \mathbb{P}$ is said to be *the worst point for the i-th iteration* if $q_i = \arg\max_{p\in\mathbb{P}} \mathsf{mrr}_{i-1}(p)$ and q_i is the point to be added in the i-th iteration, i.e., $S_i = S_{i-1} \cup \{q_i\}$. The LP (1) for computing $\mathsf{mrr}_{i-1}(p)$ is denoted by $\mathsf{LP}_{i-1}(p)$.

Upper bounding: We present an upper bound of $\mathsf{mrr}_{i-1}(p)$. If the upper bound of $\mathsf{mrr}_{i-1}(p)$ is at most the largest maximum regret ratio observed so far, we skip the LP computation of $\mathsf{mrr}_{i-1}(p)$.

Lemma 5.6. *Given p in $\mathbb{P}$, we have $\mathsf{mrr}_{i-1}(p) \le \mathsf{mrr}_{i-2}(p)$.*

Proof. It is easy to be verified by the LP (1). $\mathsf{LP}_{i-2}(p)$ is same as $\mathsf{LP}_{i-1}(p)$ except that in $\mathsf{LP}_{i-1}(p)$, we have one more constraint $(p - q_{i-1}) \cdot u \ge x$ since $S_{i-1} = S_{i-2} \cup \{q_{i-1}\}$. Since we have more constraints in $\mathsf{LP}_{i-1}(p)$ than in $\mathsf{LP}_{i-2}(p)$, its objective value ($= \mathsf{mrr}_{i-1}(p)$) is at most that of $\mathsf{LP}_{i-2}(p)$ ($= \mathsf{mrr}_{i-2}(p)$). □

Corollary 5.7. *Given p in $\mathbb{P}$ and two integers i and j ($0 \le j < i-1$), we have $\mathsf{mrr}_{i-1}(p) \le \mathsf{mrr}_j(p)$.*

Invariant checking: We show how to re-use the solutions of the LP problem computed in previous iterations for computing the $\mathsf{mrr}_{i-1}(p)$ in the i-th iteration. The lemma is shown as follows.

Lemma 5.8. *Given a point p in $\mathbb{P}$, if $(p-q_{i-1})\cdot u_{i-2}(p) \ge \mathsf{mrr}_{i-2}(p)$ where q_{i-1} is the worst point for the $(i-1)$-th iteration, we have $\mathsf{mrr}_{i-1}(p) = \mathsf{mrr}_{i-2}(p)$ and $u_{i-1}(p) = u_{i-2}(p)$.*

Proof. Obviously, $x = \mathsf{mrr}_{i-2}(p)$ and $u = u_{i-2}(p)$ satisfy all constraints except $(p - q_{i-1}) \cdot u \ge x$ of $\mathsf{LP}_{i-1}(p)$ since they are feasible solutions of $\mathsf{LP}_{i-2}(p)$. In addition, they satisfy the above additional constraint since $(p - q_{i-1}) \cdot u_{i-2} \ge \mathsf{mrr}_{i-2}(p)$. Thus, $x = \mathsf{mrr}_{i-2}(p)$ and $u = u_{i-2}(p)$ are feasible solutions of $\mathsf{LP}_{i-1}(p)$. On the other hand, according to Corollary 5.7, we have the optimal solution x^* of $\mathsf{LP}_{i-1}(p)$ is at most $\mathsf{mrr}_{i-2}(p)$, Thus, $x = \mathsf{mrr}_{i-2}(p)$ and $u = u_{i-2}(p)$ are indeed the optimal solutions of $\mathsf{LP}_{i-1}(p)$. That is, $\mathsf{mrr}_{i-1}(p) = \mathsf{mrr}_{i-2}(p)$ and $u_{i-1}(p) = u_{i-2}(p)$. □

Corollary 5.9. *Given p in $\mathbb{P}$, two integers i and j ($0 \le j < i-1$), if $(p-q_l)\cdot u_j(p) \ge \mathsf{mrr}_j(p)$, $\forall l \in (j, i-1]$ where q_l is the worst point for the l-th iteration, $\mathsf{mrr}_{i-1}(p) = \mathsf{mrr}_j(p)$ and $u_{i-1}(p) = u_j(p)$.*

Now, we are ready to describe our complete greedy strategy in Sphere. It works in iterations. In the i-th iteration where the current solution set is S_{i-1}, we aim at identifying the point p with the largest $\mathsf{mrr}_{i-1}(p)$ and insert it to S_{i-1}. With a slight abuse of notations, we assume that the largest maximum regret ratio observed so far and the point achieving such a maximum regret ratio are stored in mrr and q_i, respectively. For each point p in $\mathbb{P}$, we process it as follows. Let $j \in [0, i-1)$ be the largest integer such that $\mathsf{mrr}_j(p)$ is computed in previous iterations by either the LP (1) or Corollary 5.9. There are three cases:

Case 1 ($\mathsf{mrr}_j(p) \le$ mrr): According to Corollary 5.7, $\mathsf{mrr}_j(p)$ is an upper bound of $\mathsf{mrr}_{i-1}(p)$. Given the fact that $\mathsf{mrr}_j(p) \le$ mrr, we have $\mathsf{mrr}_{i-1}(p) \le$ mrr and thus, p cannot be the worst point q_i for the i-th iteration. Then, we skip the remaining computation of p.

Case 2 ($(p - q_l) \cdot u_j(p) \ge \mathsf{mrr}_j(p)$, $\forall j < l \le i-1$): According to Corollary 5.9 and the fact that $(p - q_l) \cdot u_j(p) \ge \mathsf{mrr}_j(p)$, $\forall j < l \le i-1$, we know that $\mathsf{mrr}_{i-1}(p) = \mathsf{mrr}_j(p)$ and $u_{i-1}(p) = u_j(p)$ (without computing it using the LP (1))

Case 3 (otherwise): We have no choice but to compute $\mathsf{mrr}_{i-1}(p_j)$ and $u_{i-1}(p)$ by the LP (1).

After $\mathsf{mrr}_{i-1}(p)$ is computed, it is compared with the largest maximum regret ratio mrr observed so far. If mrr $< \mathsf{mrr}_{i-1}(p)$, mrr and q_i are updated accordingly. At the end of the i-th iteration, the point q_i is the worst point for the i-th iteration. q_i is inserted into the current solution set and $S_i = S_{i-1} \cup \{q_i\}$.

Comparison with the existing the greedy strategies: The best-known greedy-based strategies which return a set with a small maximum regret ratio empirically are Greedy and GeoGreedy. GeoGreedy is only restricted in a low dimensional space since its operation is not affordable in a high dimensional space. Greedy is the only applicable algorithm in this case. However, Greedy does not distinguish the cases discussed above and always solves the time-consuming LP (1) for each point in $\mathbb{P}$ in each iteration, resulting in a poor performance in term of the running time. In comparison, our strategy identifies those unnecessary LP computations and thus speeds up the overall process.

5.3 Theoretical Analysis

In this section, we analyze the time complexity of algorithm Sphere (Section 5.3.1), and prove an upper bound on $\mathsf{mrr}_{\mathbb{P}}(S)$ (Section 5.3.2) where S is the solution set returned by Sphere.

5.3.1 Time Complexity. We analyze the time complexity of Sphere step-by-step. Note that $|\mathcal{I}_R^+| = k' = O(\frac{k}{d})$ in all cases.

- The initialization of the solution set S takes $O(nd)$ time.
- Consider the construction of $\mathcal{I}_R^+$. Due to the regularity of the constructing procedure, each coordinate value of the point in $\mathcal{I}_R^+$ can be computed with the help of the Cartesian product, and all coordinate values of the point could be computed in $O(d)$ time. Since there are k' points in $\mathcal{I}_R^+$, constructing $\mathcal{I}_R^+$ takes $O(k'd) = O(k)$ time.
- Consider the basis search. For each s' in $\mathcal{I}_R^+$, we find its $\mathbb{P}$-basis in $O(ne^{O(\sqrt{d \ln n})})$ time [13] or in $O(nd^3 2^d)$ time [28]. Then, we insert all points in the basis into S, which takes

$O(d)$ time. Since there are $|\mathcal{I}_R^+| = k'$ points, the third step takes $O(nke^{O(\sqrt{d \ln n})})$ time [13] or $O(nkd^2 2^d)$ time [28].
- Let the number of points to be included greedily in the last step be k_G. It takes $O(nk_G^3 d)$ time in practice and $O(nk_G^2 d^{4.5})$ time in the worst case [21] to include k_G points greedily.

In summary, the time complexity of Sphere is $O(ne^{O(\sqrt{d \ln n})} + nk^3 d)$ (if we use the basis search techniques in [13]) or $O(nkd^2 2^d + nk^3 d)$ (if we use the basis search techniques in [28]).

Sphere has the following advantages in term of running time. Firstly, compared with some existing algorithms (e.g., GeoGreedy [24] and DMM [5]) which exponentially depend on the dimensionality d, Sphere can be implemented subexponentially in term of d [13]. Secondly, compared with the algorithm (e.g., RMS_HS [2]) whose performance degrades rapidly when the maximum regret ratio is small, Sphere is not sensitive to the maximum regret ratio of the set returned. Thirdly, compared with some algorithms (e.g., GeoGreedy [24] and RMS_HS [2]) which might take $O(n^{O(d)})$ to execute, Sphere significantly reduces the time needed (e.g., linear in n [28]). Finally, the pruning strategies presented in Section 5.2.3 reduce a large number of unnecessary LP computations empirically compared with Greedy [21] and thus speedup the whole process.

For the other algorithms which have comparable time complexities as Sphere, our Sphere algorithm does not only enjoy a restriction-free and asymptotically optimal bound for any dimensionality, but also returns a set with a much smaller maximum regret ratio empirically as will be shown shortly in the experimental evaluation.

5.3.2 Upper Bound. We are ready to discuss the theoretical guarantee on the maximum regret ratio of the solution set returned by Sphere. We summarize the result in the following theorem.

Theorem 5.10. *Sphere returns a set $S \subseteq \mathbb{P}$ such that*

$$\mathrm{mrr}_{\mathbb{P}}(S) \leq \min\left\{1 - \frac{1}{d}, \frac{(d-1)d}{\max\left\{1/4, \left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor^2\right\} + (d-1)d}\right\}$$

Specifically, for a fixed dimensionality d, $\mathrm{mrr}_{\mathbb{P}}(S) = O(k^{-\frac{2}{d-1}})$.

Proof Sketch. We provide the detailed proof of Theorem 5.10 in the appendix. The intuitions behind our restriction-free bound on the maximum regret ratio $\mathrm{mrr}_{\mathbb{P}}(S)$ are briefly illustrated as follows.

In order to bound the maximum regret ratio, we want to make the utilities of the selected points as large as possible and as close to the maximum utility of the whole dataset as possible.

Firstly, the solution set S contains the set of all boundary points, which guarantees that the maximum utility of S (i.e., $\max_{q \in S} f(q)$) is not too small even under some extreme utility functions. Secondly, the maximum utility of S is indeed close to the maximum utility of the whole dataset $\mathbb{P}$, i.e., $\max_{p \in \mathbb{P}} f(p) - \max_{q \in S} f(q)$ is not too large. This is because that we include all points in the basis $B(s')$ for each s' in $\mathcal{I}_R^+$ into S and, as will be shown in the proof, the points in the basis $B(s')$ enjoy high utilities. Finally, we improve the tightness of the upper bound on $\mathrm{mrr}_{\mathbb{P}}(S)$ by carefully determining a proper value of R (i.e., the radius of the sphere that $\mathbb{S}_R^+$ lies on), whose intuition has been discussed in Section 5.2. □

Remark: Combining the upper bound in Theorem 5.10 with the lower bound in Theorem 3.3, we conclude that the bound $O(k^{-2/(d-1)})$ is *asymptotically optimal* for a fixed d. Besides, it can be verified that *each* bound in Theorem 5.10 ranges from 0 to 1 for all settings, i.e., Sphere satisfies the restriction-free bound requirement.

6 EXPERIMENTAL RESULTS

We conducted experiments on a machine with 2.26GHz CPU and 32GB RAM. All programs were implemented in C/C++. Most of the experimental settings follow those in [5, 11, 21, 31]. Experiments were conducted on both *synthetic datasets* and *real datasets*.

For synthetic datasets, we adopt the dataset generator [6] to generate anti-correlated datasets (additional experiments on correlated and independent datasets are shown in the appendix). Unless specified explicitly, the number of tuples in each synthetic dataset is set to 100,000 (i.e., $|\mathbb{P}| = n = 100,000$), d is set to 6 and k is set to 30. In addition, we adopt 4 real datasets in our experiments, namely *NBA* (http://www.basketballreference.com), *Household* (http://www.ipums.org), *Movie* (https://movielens.umn.edu) and *Airline* (http://kt.ijs.si). NBA contains 21,961 tuples for each player/season combination from 1946 to 2009. Six attributes are selected to represent the performance of each player, such as *total scores* and *blocks* etc. Household consists of 1,048,576 family tuples in US in 2012. Each tuple is associated with seven attributes, showing the economic characteristic of each family, such as *annual property insurance cost* and *annual home heating fuel cost* etc. Movie is a 19-dimensional dataset with 100,000 ratings (1-5) from 943 users on 1682 movies. Each user rated at least 20 movies and each movie was rated by at least 20 users. The users are grouped according to their occupations, resulting in 19 groups of users. Each group of users gives an overall rating on each movie. Airline contains 5,810,463 records with 3 characterizing attributes, namely *the actual elapsed time, the distance and the arrival delay*. The information about the real datasets is summarized in Table 3.

For all the datasets, each attribute is normalized to (0, 1]. Missing values are replaced with the smallest value found in the corresponding attributes. We preprocessed each dataset such that the preprocessed dataset contains skyline points only. Unless specified explicitly, we vary the value of k in our experiments and the range of the k values covers all three subcases of the algorithm which are discussed in Section 5.2. The performance of each algorithm is measured in terms of the *query time* and the *maximum regret ratio (*mrr*)*. The query time of an algorithm is the execution time of the algorithm. The maximum regret ratio of an algorithm is the maximum regret ratio of the set returned by the algorithm. For each experiment on synthetic datasets, five dataset instances are generated independently. We ran the experiment 5 times and the averaged performance was reported. Similarly, each experiment on real datasets was run 5 times and we report the averaged performance.

We compared the performance of the following algorithms: Sphere, Cube [21], Greedy [21], GeoGreedy [24], RMS_HS [2], DMM [5] and ε-kernel [7] where Sphere is our complete algorithm. We exclude the algorithms which fail to satisfy the dimensionality requirement and work only on 2-dimensional datasets in the experiments. Note that the ε-kernel algorithm [7] does not work under typical settings (see Section 4) since it does not satisfy the

Dataset	Dimensionality	$\|\mathbb{P}\|$
NBA	6	21,961
Household	7	1,048,576
Movie	19	1693
Airline	3	5,810,463

Table 3: Statistics about Real Datasets

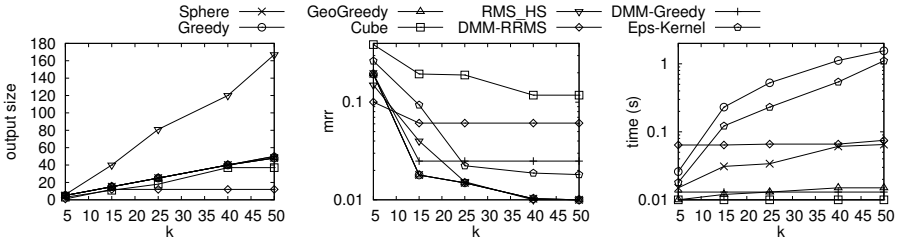


Figure 3: Experiments on 3d Anti-correlated Datasets

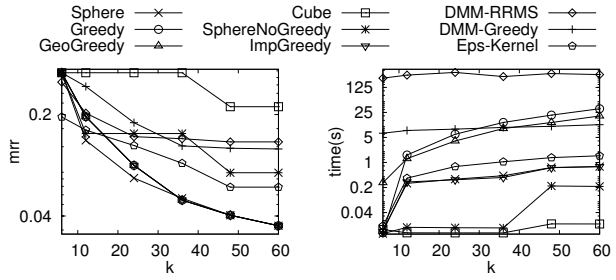
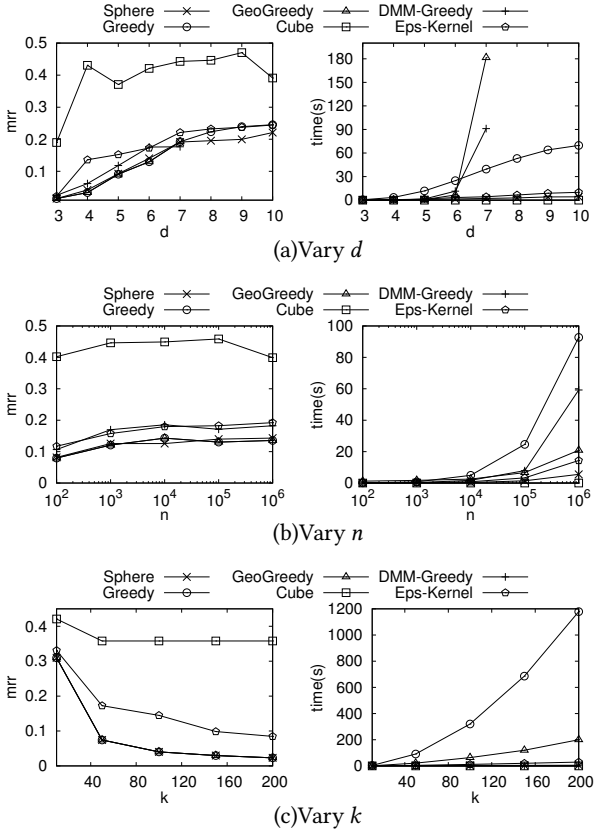


Figure 4: Experiments on 6d Anti-correlated Datasets



(a)Vary d

(b)Vary n

(c)Vary k

Figure 5: Scalability Test

restriction-free bound requirement. Nevertheless, we implemented the ε-kernel algorithm, denoted by Eps-Kernel in the figure, for the purpose of comparison, but its bound on the maximum regret ratio does not hold anymore. In practice, the size of an ε-kernel can be much larger than k. If this is the case, we randomly return k points in the ε-kernel as the solution. RMS_HS is the min-error 2-approximate algorithm presented in [2]. We note that RMS_HS is different from the HS algorithm presented in the experimental evaluation in [2] where algorithm HS [2] is used to solve the min-size regret query (we have discussed the difference between the min-error query and the min-size query in Section 2). RMS_HS invokes HS multiple times to obtain a set that satisfies the size constraint. As it was pointed out in [2], RMS_HS might take exponential time to return the solution if the maximum regret ratio is small. The DMM algorithm proposed in [5] has two practical implementations: the set-cover based implementation DMM-RRMS with theoretical upper bounds (which is denoted as HD-RRMS in [5]), and the greedy-based heuristic implementation DMM-Greedy (which is denoted as HD-Greedy in [5]). We compared both of their performances in the experiments where the number of partitions of the discretized matrix (i.e., γ) is set to 4 (the default setting in [5]).

6.1 Results on Synthetic Datasets

We conducted experiments on synthetic datasets in this section. In particular, we study the effects of k, d and n on synthetic datasets.

The results by varying k on a 3-dimensional dataset are shown in Figure 3 where the maximum regret ratio and running time are plotted in the log-scale for better visualization. Consider the left sub-figure of Figure 3. We show the solution set size of each algorithm. Clearly, Sphere outperforms RMS_HS in term of the output size. RMS_HS returns more points (i.e., $\Theta(k \log k)$) than other algorithms, which return at most k points to the users. The middle sub-figure of Figure 3 depicts the maximum regret ratio of each algorithm. Cube has the largest maximum regret ratio. The maximum regret ratios of the DMM-based algorithms are larger than Sphere and the greedy-based algorithms in most cases. Note that the maximum regret ratio of DMM-RRMS is loosely bounded and it can only guarantee maximum regret ratios within constant distances to the optimal regret ratio (i.e., the bound is not asymptotically optimal). The maximum regret ratio of ε-kernel is also large while the maximum regret ratio of RMS_HS is comparable to Sphere. Clearly, Sphere performs better than DMM-RRMS, DMM-Greedy, ε-kernel and Cube, which do not satisfy the quality requirement, in term of the maximum regret ratio. We proceed with the query time evaluation in the right sub-figure of Figure 3. RMS_HS, which might need to solve a large number of hitting set problems, takes more time to execute compared with other algorithms. For example, RMS_HS takes several minutes to return the solution when $k = 50$ while the other algorithms finish in seconds. For the ease

of presentation, its query time is not shown in the figure. The poor performance of RMS_HS for a min-error k-regret query was also observed in [2] and the situation is even worse when d is large. For example, on a 6-dimensional dataset (which will be shown shortly), RMS_HS takes a couple of hours to return the solution while other algorithms finish in minutes. Due to the poor performance of RMS_HS, we omit its experimental results in the rest of this section. Besides, Greedy and ε-kernel consume larger amounts of time compared with other algorithms. Greedy, ε-kernel and RMS_HS fail to satisfy the efficiency requirement and Sphere clearly beats them in term of the query time. The remaining algorithms are relatively fast in this 3-dimensional dataset (i.e., return the solutions within 0.1 seconds). Note that GeoGreedy and DMM-RRMS achieve reasonable performances in this particular dataset, but their performances degrade rapidly when the dimensionality is large (to be shown later). In short, Sphere, which is elegant and satisfies all four requirements, is the one which achieves a good performance in term of all measurements, namely the output size, the running time and the maximum regret ratio, while most existing algorithms might be good in one aspect but be quite poor in the other aspect.

To better understand the superiority of Sphere, we conducted the experiments by varying the value of k on a 6-dimensional dataset in Figure 4 where we compared two additional algorithms, namely SphereNoGreedy and ImpGreedy. SphereNoGreedy is the complete Sphere algorithm without the greedy step at the end, to demonstrate the effectiveness of the first three steps of Sphere. ImpGreedy is the algorithm with only the greedy step, to demonstrate the effectiveness of the last step of Sphere. The results in Figure 4 are also plotted in log-scale for better visualization.

Consider the results on the maximum regret ratio in Figure 4. The maximum regret ratio of Sphere is the smallest among all the algorithms on most settings. Cube has the worst empirical maximum regret ratio which verifies our claim that although it has a theoretical bound on the maximum regret ratio, the empirical performance is not satisfiable enough and thus, fail to satisfy the quality requirement. Note that the maximum regret ratio of SphereNoGreedy is also comparable to the complete algorithm Sphere. In particular, when $k = 12$, the maximum regret ratio of SphereNoGreedy is 0.14 and the maximum regret ratio of Sphere is 0.11. Both of the maximum regret ratios of SphereNoGreedy and Sphere are smaller than the maximum regret ratios of all other algorithms. In other words, the simplest set of tuples constructed by Sphere that enjoys the theoretical guarantee also achieves a small maximum regret ratio in practice. Nevertheless, there is no harm to include more points into the solution set as long as it does not contain more than k points. The maximum regret ratios of other algorithms follow a similar trend as in Figure 3. Consider the results on the query time in Figure 4 next. When k increases, the query times of all algorithms increase. Sphere returns the solutions within 1 seconds in all cases. The query times of Sphere and ImpGreedy are much smaller than those of Greedy and GeoGreedy. In particular, when $k = 50$, Sphere and ImpGreedy is 30 times faster than Greedy and 20 times faster than GeoGreedy. Note that GeoGreedy and the DMM-based algorithms consume more time compared with those in Figure 3 and they become slower than Sphere and the greedy-based algorithms due to the fact that their query times exponentially depend on d. In particular, DMM-RRMS, which has to solve the set-cover problems multiple times, takes more than 300 seconds to execute while some other algorithms finishes all the computations in 1 second. For better visualization, DMM-RRMS is also omitted in the rest of experiments.

The complete algorithm, Sphere, achieves the best empirical performance compared with the existing algorithms and at the same time, it enjoys a restriction-free and asymptotically optimal bound on the maximum regret ratio. Compared with the partial algorithm SphereNoGreedy which enjoys the theoretical guarantee, Sphere returns a set with smaller maximum regret ratio empirically. Compared with the partial algorithm ImpGreedy, Sphere not only returns a set with an even smaller maximum regret ratio, but also guarantees a theoretical bound while ImpGreedy does not. Due to the superior performance of Sphere, we focus only on the complete algorithm Sphere in the rest of experiments.

We also test the scalability of Sphere by varying the values of d, n and k. For other parameters, we use the default setting on the synthetic datasets, and the results are shown in Figure 5.

In Figure 5(a), we vary the value of d on syntectic datasets. When the dimensionality increases, the query times and the maximum regret ratios of all the algorithms increase. Sphere consistently consumes fewer time compared than other algorithms except Cube and it returns a set with a comparably small maximum regret ratio in all cases among all the algorithms. Note that GeoGreedy and DMM-Greedy has large query times compared with other algorithms when d is large since their operations (which exponentially depend on d) are not affordable in a high dimensional space (i.e., does not satisfy the efficiency requirement when the dimensionality is large). Specifically, computing a convex hull in GeoGreedy in high dimensional spaces is expensive and the discretized matrix in DMM-Greedy can be large in high dimensional spaces. For example, when $d = 8$, GeoGreedy and DMM-Greedy take a couple of hours to return the solution, but other algorithms finish all computations in minutes. Due to the large query times of GeoGreedy and DMM-Greedy, their results for $d > 7$ are omitted. We vary n on synthetic datasets in Figure 5(b). The maximum regret ratio of Sphere is comparable to the state-of-the-art algorithms, but its running time is significantly smaller than the other algorithms except Cube, which has the worst maximum regret ratio. In particular, when $n = 1,000,000$, Sphere is around 18 times faster than Greedy and DMM-Greedy and 4 times faster than GeoGreedy and ε-kernel. Finally, the results by varying the large value of k are shown in 5(c). By including a sufficiently number of points in the solution, the maximum regret ratios of all algorithms except Cube and ε-kernel are small. However, the speedup of Sphere over Greedy and GeoGreedy is significant, which is the consequence of the efficient basis search technique (in Step 3 of Sphere) and the powerful pruning techniques (in Step 4 of Sphere).

6.2 Results on Real Datasets

We conducted experiments on real datasets. In particular, we study the effect of k (parameter k in a k-regret query) on real datasets. Note that the dimensionalities of most real datasets are large where GeoGreedy and DMM-Greedy have poor running times. For example, GeoGreedy takes several days to return the solution in the

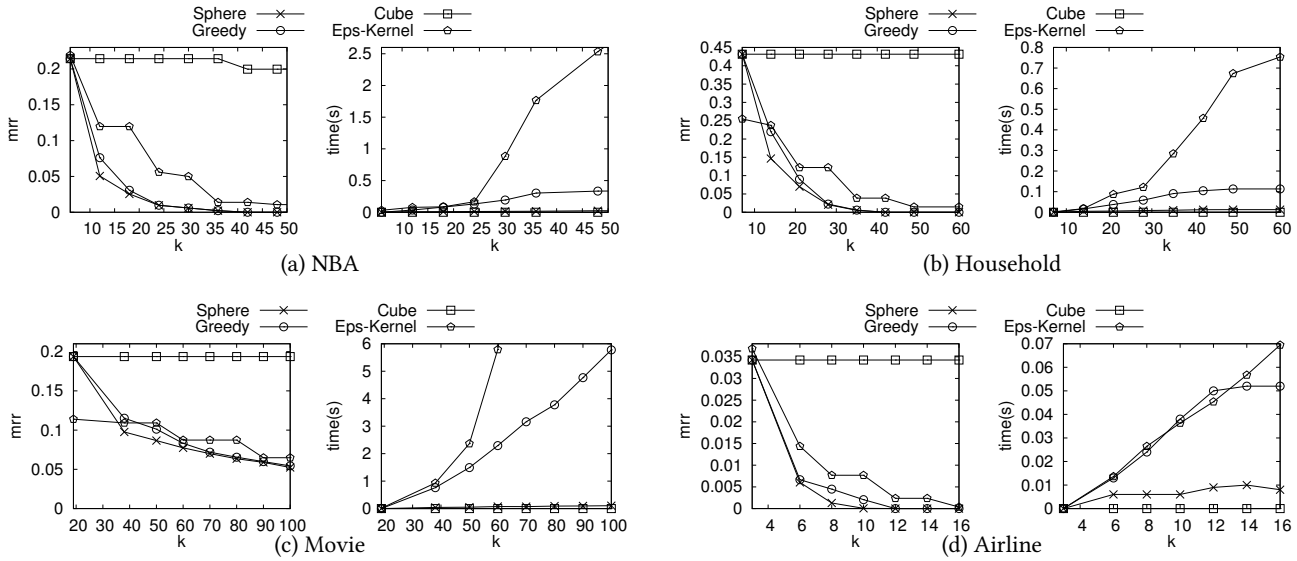


Figure 6: Experiments on Real Datasets

Movie dataset while other algorithms return the solutions in minutes. For the ease of presentation, we omit them in the figures.

Consider the experiment which measures the maximum regret ratio of each algorithm on real datasets in Figure 6. When k increases, the maximum regret ratios of most algorithms decrease, because when more points are selected into the final solution set, the maximum regret ratio is smaller. Compared with ε-kernel and Greedy, Sphere achieves a smaller maximum regret ratio in most of the cases. Specifically, in the Household dataset, Sphere achieves 15% of improvements in term of maximum regret ratio on average. In particular, when $k = 14$, the maximum regret ratio of the set returned by Sphere is 0.14 where the maximum regret ratio of the set returned by Greedy is 0.22. That is, Sphere achieves more than 30% of improvements in term of maximum regret ratio when $k = 14$. The improvements of Sphere are similar in other datasets. Consider the experiment which measures the query time of each algorithm in Figure 6. When k increases, the query times of all algorithm increase. Similar to the results on synthetic datasets, Sphere runs significantly faster than Greedy and ε-kernel in all cases and it achieves an order of improvement in term of the running time. Note that Cube is often slightly faster than Sphere since it constructs the solution set by simply scanning the database once, which can be efficiently implemented. However, its maximum regret ratio is the worst in all cases. For example, in the Household dataset, all other algorithms achieve maximum regret ratios less than 0.1 for $k \geq 20$ while the maximum regret ratio of Cube is around 0.43. In particular, the maximum regret ratios of other algorithms are close to 0 when $k \geq 40$, but the maximum regret ratio of Cube does not reduce significantly (i.e., it is still at least 0.4).

We also conducted a subjective evaluation on the k-regret sets on real datasets to provide an intuitive visualization about the usefulness of our Sphere algorithm. Due to the lack of space, the subjective evaluation is provided in the appendix (see Figure 8).

6.3 Summary

In summary, we conducted various experiments in this section on both real and synthetic datasets, showing the superiority of Sphere over the existing algorithms. Specifically, Sphere has the smallest maximum regret ratio in most of the cases and the running time among the best under many settings (significantly faster than the existing greedy-based algorithms, but slightly slower than the Cube algorithm whose returned set is of poor quality). Moreover, the scalability of Sphere is demonstrated and a subjective evaluation on the k-regret sets is also elaborated in the experiments.

7 CONCLUSION AND FUTURE WORK

We study the k-regret query (i.e., the min-error regret query) in this paper. Specifically, we design an elegant algorithm called Sphere whose upper bound on the maximum regret ratio is asymptotically optimal and restriction-free for datasets of any dimensionality. Comprehensive experiments were conducted on both real datasets and synthetic datasets, which showed that our Sphere algorithm returns a solution set with the best maximum regret ratio efficiently. There are many interesting future directions. We consider preprocessing the dataset such that the basis search can be answered more quickly and constructing the set $\mathcal{I}_R^+$ with an asymptotically smaller distance bound. Another interesting direction is to combine the k-regret query results over changing views.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Min Xie and Raymond Chi-Wing Wong is supported by HKRGC GRF 16214017. Jian Li is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61772297, 61632016, 61761146003, and a grant from Microsoft Research Asia.

REFERENCES

[1] P.K. Agarwal, S.H. Peled, and K.R. Varadarajan. 2004. Approximating Extent Measures of Points. In *Journal of the ACM (JACM)*.

[2] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. 2017. Efficient Algorithms for k-Regret Minimizing Sets. *International Symposium on Experimental Algorithms*.

[3] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. 2009. Diversifying search results. In *Proceedings International Conference on Web Search & Data Mining*.

[4] A. Arya, D.M. Mount, N.S. Netanyahu, and R. Silvermannd A.Y. Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* (1998).

[5] A. Asudeh, A. Nazi, N. Zhang, and G. Das. 2017. Efficient Computation of Regret-ratio Minimizing Set: A Compact Maxima Representative. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 821–834.

[6] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*.

[7] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R.C.W. Wong, and W. Zhan. 2017. k-Regret Minimizing Set: Efficient Algorithms and Hardness. In *20th International Conference on Database Theory*.

[8] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. 2006. Finding k-dominant skylines in high dimensional space. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*.

[9] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. 2006. On high dimensional skylines. In *Advances in Database Technology-EDBT 2006*.

[10] Y. Chang, L. Bergman, V. Castelli, C. Li, M. Lo, and J. Smith. 2000. The Onion Technique: Indexing for Linear Optimization Queries. In *ACM Sigmod Record*.

[11] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. 2014. Computing k-Regret Minimizing Sets. *Proceedings of the VLDB Endowment*.

[12] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. 2012. Top-k bounded diversification. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*.

[13] B. Gartner. 1995. A subexponential algorithm for abstract optimization problems. In *SIAM Journal on Computing*.

[14] M. Goncalves and M. Yidal. 2005. Top-k skyline: a unified approach. In *On the Move to Meaningful Internet Systems 2005: OTM Workshops*. Springer, 790–799.

[15] K. Kenthapadi, B. Le, and G. Venkataraman. 2017. Personalized job recommendation system at linkedin: Practical challenges and lessons learned. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 346–347.

[16] J. Lee, G. won You, and S.W. Hwang. 2009. Personalized top-k skyline queries in high-dimensional space. *Information Systems* (2009).

[17] X. Lian and L. Chen. 2009. Top-k dominating queries in uncertain databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*.

[18] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. 2007. Selecting stars: The k most representative skyline operator. In *Proceedings IEEE Conference on Data Engineering*.

[19] D. Mindolin and J. Chomicki. 2009. Discovering relative importance of skyline attributes. *Proceedings of the VLDB Endowment* 2, 1 (2009), 610–621.

[20] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino. 2012. Interactive Regret Minimization. In *Proc. of the 2012 ACM SIGMOD International Conference on Management of Data, https://dl.acm.org/citation.cfm?id=2213850*.

[21] D. Nanongkai, A.D. Sarma, A. Lall, R.J. Lipton, and J. Xu. 2010. Regret-minimizing representative databases. *Proceedings of VLDB Endowment* (2010).

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. 2005. Progressive skyline computation in database systems. In *ACM Transactions on Database Systems*, Vol. 30. 41–82.

[23] A. N. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos. 2007. Domination mining and querying. In *International Conference on Data Warehousing and Knowledge Discovery*.

[24] P. Peng and R.C.W Wong. 2014. Geometry approach for k regret query. In *Proceedings IEEE Conference on Data Engineering*.

[25] J. Qi, F. Zuo, and J. Yao. 2016. K-Regret Queries: From Additive to Multiplicative Utilities. In *CoRR*. http://arxiv.org/abs/1609.07964

[26] L. Qin, J. Yu, and L. Chang. 2012. Diversifying top-k results. *Proceedings of the VLDB Endowment* 5, 11, 1124–1135.

[27] A. Roshdi and A. Roohparvar. 2015. Information Retrieval Techniques and Applications. *International Journal of Computer Networks & Communications Security*.

[28] M. Sharir and E. Welzl. 1992. A combinatorial bound for linear programming and related problems,. In *Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*. Springer, 569–579.

[29] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *Proceedings IEEE Conference on Data Engineering*.

[30] Y. Tao, L. Ding, and J. Pei. 2009. Distance-based representative skyline. In *Proceedings IEEE Conference on Data Engineering*.

[31] Y. Tao, X. Xiao, and J. Pei. 2007. Efficient skyline and top-k retrieval in subspaces. *IEEE Transactions on Knowledge and Data Engineering* (2007).

[32] K. F. Taylor, W. Brackenbury, and A. Lall. 2015. k-regret queries with nonlinear utilities. *Proceedings of the VLDB Endowment* (2015).

[33] T. Xia, D. Zhang, and Y. Tao. 2008. On skylining with flexible dominance relation. In *Proceedings IEEE Conference on Data Engineering*. IEEE, 1397–1399.

[34] M. L. Yiu and N. Mamoulis. 2009. Multi-dimensional top-k dominating queries. *The VLDB Journal*. (2009).

[35] H. Yu, P.K. Agarwal, and R. Poreddyand K.R. Varadarajan. 2008. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*.

A SUMMARY OF NOTATIONS

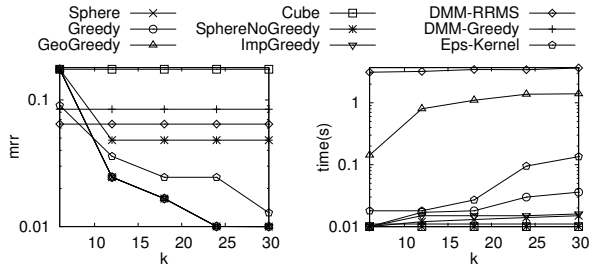
We summarize the notations that are frequently used in Table 4.

Notation	Meaning
n	The size of the dataset $\mathbb{P}$
$\mathsf{rr}_{\mathbb{P}}(S, f)$	The regret ratio of S w.r.t. f where $f(p) = u \cdot p, \|u\| = 1$
$\mathsf{mrr}_{\mathbb{P}}(S)$	The maximum regret ratio of S
b_i	i-th dimensional boundary point ($b_i[i] = 1$)
S_i	The solution set after the i-th iteration in the greedy strategy
$\mathsf{mrr}_{i-1}(p)$	$\mathsf{mrr}_{S_{i-1}\cup\{p\}}(S_{i-1})$
$u_{i-1}(p)$	The worst utility vector for $\mathsf{mrr}_{i-1}(p)$
$\mathsf{LP}_{i-1}(p)$	The LP for computing $\mathsf{mrr}_{i-1}(p)$
$\mathcal{CH}(\mathbb{P})$	The convex hull of a set $\mathbb{P}$
$\mathbb{C}$	A d-dimensional hypercube $[-1, 1]^d$
$\mathbb{S}_R$	$\mathbb{S}_R = \{s \in \mathbb{R}^d \mid \|s\| = R\}$
f_s	The utility function whose utility vector is in the same direction of s
$\mathbb{C}^+$ ($\mathbb{S}_R^+$)	The set of points in $\mathbb{C}(\mathbb{S}_R)$ in $\mathbb{R}_+^d$
$\mathcal{I}_R$ ($\mathcal{I}_R^+$)	A subset of points in $\mathbb{S}_R$ ($\mathbb{S}_R^+$)
$\mathsf{dist}(p, q)$	The Euclidean distance between p and q
$\mathsf{dist}(P, s)$	The minimum distance between a point on the surface of the convex hull of P and s
$B(s)$	A $\mathbb{P}$-basis of s with $B(s) \subseteq \mathbb{P}$, $\|B(s)\| \leq d$, $\mathsf{dist}(B(s), s) = \mathsf{dist}(\mathbb{P}, s)$ and $\mathsf{dist}(B(s), s) < \mathsf{dist}(B', s) \forall B' \subset B(s)$
$p(P, s)$	$p(P, s) = \arg\ \min_{p \in \mathcal{CH}(P)}\ \mathsf{dist}(p, s)$

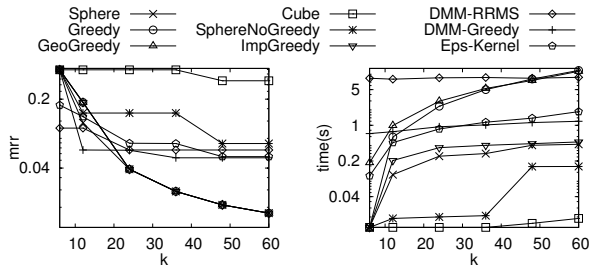
Table 4: Summary of Frequently Used Notations

B ADDITIONAL EXPERIMENTAL RESULTS

We conducted experiments on 6-dimensional correlated and independent datasets in Figure 7 (plotted in log-scale). Consider the results on the maximum regret ratio in Figure 7. The maximum regret ratios of all algorithms on independent datasets are similar to those on anti-correlated datasets, while on correlated datasets, most maximum regret ratios are small since there are points with high values in all dimensions (i.e., the values in all dimensions are correlated). Most maximum regret ratios on correlated datasets drop to 0 when $k > 30$ and the results for $k > 30$ is omitted. The maximum regret ratio of Sphere is the smallest in most cases. Consider the results on the query time in Figure 7. When k increases, the query times of all the algorithms increase. For each algorithm, its query time on correlated datasets is smaller than its query time on independent datasets, which is then smaller than its query time on anti-correlated datasets. This is because that on anti-correlated datasets, points with high values in some dimensions might have low values in other dimensions, making it more time-consuming to determine the points to be included into the final solution set.



(a) 6*d* Correlated Datasets



(b) 6*d* Independent Datasets

Figure 7: Additional Experiments on 6*d* Synthetic Datasets

Finally, we conducted a subjective evaluation on the k-regret sets. Specifically, we selected two attributes, namely the minutes played (minutes) and the scores obtained (scores), for each player in the NBA dataset to visualize the effect of our regret minimizing set. Note that these two attributes are correlated in real scenarios. A coach is less interested in the players who play for a long time and obtain high scores (because it is more likely that players who play for a longer time obtain higher scores). Instead, he may want to find out the players who did not achieve the expected scores given that they are assigned a sufficient time to play so that their team's performance can be improved in the future games. Based on this idea, we inverse the "scores" attribute and thus we are interested in a larger value on each of the two attributes (i.e., "Minutes Played" and "Scores (Inverse)"). The dataset could be visualized in Figure 8 where both attributes are normalized to (0,1]. Note that the players on the top left corner are the players of less interest (long time and high scores). We are interested in the players who obtained low scores in a long play time. We set k to be 10 and plot the set of players returned by both Sphere and Cube in Figure 8. Sphere identifies a lot of "interesting" players which "spreads" over the 2-dimensional space which is reasonable to capture the importance of "different" dimensions/axes. It is worth mentioning that most of these players returned are not boundary points, which are quite reasonable. Besides, Sphere reports the player (indicated with an arrow in the figure) who played 80% of time, but obtained only 20% of scores during the game. In this example, Sphere returns the set with the optimal maximum regret ratio (=0.0028). In contrast, Cube returns fewer "interesting" players which locate uniformly at the boundary of the dataset (so that the maximum regret ratio can be theoretically bounded), and it returns a number of players at the top left corner, which are of less "interest". Besides, its maximum regret ratio is three times larger than that of Sphere.

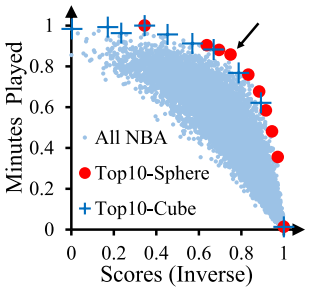


Figure 8: Visualize NBA

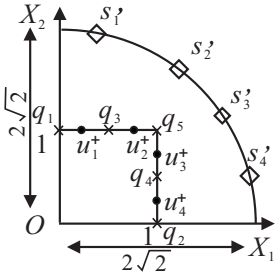


Figure 9: Construct $\mathcal{I}_R^+$

C REMAINING PROOFS

C.1 Proof of Theorem 5.10

Proof. The proof is divided into two parts.

I. We prove the first bound $\text{mrr}_{\mathbb{P}}(S) \leq 1 - \frac{1}{d}$ by considering the boundary points in S. We have the following simple lemma which shows a lower bound on the utilities achieved by the points in S.

Lemma C.1. $\max_{q \in S} f(q) \geq \frac{1}{\sqrt{d}}$.

Proof. For each utility vector u, $\|u\| = 1$. We first prove by contradiction that there exists an integer $i \in [1, d]$ such that $u[i] \geq \frac{1}{\sqrt{d}}$. If $u[i] < \frac{1}{\sqrt{d}}$ for each $i \in [1, d]$, $\|u\|^2 = \sum_{i=1}^{d} u[i]^2 < d \times \frac{1}{d} = 1$, contradicting to $\|u\| = 1$. Let i be the integer such that $u[i] \geq \frac{1}{\sqrt{d}}$. Since b_i is in S, $\max_{q \in S} f(q) \geq f(b_i) = \sum_{j=1}^{d} u[j] b_i[j] \geq u[i] b_i[i] \geq \frac{1}{\sqrt{d}}$. □

Consider the difference between the maximum utility of $\mathbb{P}$ and the maximum utility of S w.r.t. a particular utility function $f(p)(= u \cdot p)$ first. According to Lemma C.1 and the fact that the coordinate value of each point in $\mathbb{P}$ is normalized to (0,1], we have

$$\max_{p \in \mathbb{P}} f(p) - \max_{q \in S} f(q) \leq \max_{p \in \mathbb{P}} u \cdot p - \frac{1}{\sqrt{d}} \leq \max_{p \in \mathbb{P}} \|u\| \|p\| - \frac{1}{\sqrt{d}} \leq \sqrt{d} - \frac{1}{\sqrt{d}}.$$

Utilizing Lemma C.1 again, we have $\frac{\max_{p \in \mathbb{P}} f(p) - \max_{q \in S} f(q)}{\max_{q \in S} f(q)} \leq d - 1$. By rewriting the above inequality, we have $\frac{\max_{q \in S} f(q)}{\max_{p \in \mathbb{P}} f(p)} \geq \frac{1}{d}$. Then we have $\text{mrr}_{\mathbb{P}}(S) = \max_{f \in \mathcal{FC}} \left(1 - \frac{\max_{q \in S} f(q)}{\max_{p \in \mathbb{P}} f(p)}\right) \leq 1 - \frac{1}{d}$.

II. We then prove the second bound. Recall that in Section 5.2.1 (Case 2 and Case 3), we construct the solution set S with the help of a special set $\mathcal{I}_R^+$. Specifically, for each point s in $\mathbb{S}_R^+$, there exists a point $s' \in \mathcal{I}_R^+$ such that $\text{dist}(s, s') \leq \delta$. To provide more insights to the radius of the sphere that $\mathbb{S}_R^+$ lies on, we first denote the radius by R instead of assigning it a concrete value (i.e.,$2\sqrt{d}$). Nevertheless, R must be greater than $\sqrt{d}$. Otherwise, a point s in $\mathbb{S}_R^+$ might lie inside $\mathcal{CH}(\mathbb{P})$, invalidating all the following discussion.

Given a utility function f, we consider the difference between the maximum utility of $\mathbb{P}$ and the maximum utility of S w.r.t. f first. The result is summarized in the following lemma.

Lemma C.2. *Sphere returns a set $S \subseteq \mathbb{P}$ such that, for each utility function f with a utility vector u,*

$$\max_{p\in\mathbb{P}} f(p) - \max_{q\in S} f(q) \leq \frac{\delta^2}{\|s - p(\mathbb{P}, s)\|}$$

where s is the intersection between $\mathbb{S}_R^+$ and the ray shooting from the maximum utility point of $\mathbb{P}$ w.r.t. f in the direction of u.

Proof Sketch. The idea is mainly borrowed from [1, 35], but different from [1, 35], we take the radius of $\mathbb{S}_R^+$ into the consideration. The complete proof is presented in Appendix C.3. □

Note that the i-th dimensional value of $p(\mathbb{P}, s)$ is in (0,1] for each $i \in [1, d]$. That is, $\|p(\mathbb{P}, s)\| = \sqrt{\sum_{i=1}^{d} p(\mathbb{P}, s)[i]^2} \leq \sqrt{d}$. Then, $\|s - p(\mathbb{P}, s)\| \geq \|s\| - \|p(\mathbb{P}, s)\| \geq R - \sqrt{d}$. According to Lemma C.2, $\max_{p\in\mathbb{P}} f(p) - \max_{q\in S} f(q) \leq \frac{\delta^2}{\|s-p(\mathbb{P},s)\|} \leq \frac{\delta^2}{R-\sqrt{d}}$.

Combining the results above with Lemma C.1, we have

$$\frac{\max_{p\in\mathbb{P}} f(p) - \max_{q\in S} f(q)}{\max_{q\in S} f(q)} \leq \frac{\sqrt{d}\delta^2}{R - \sqrt{d}}. \tag{2}$$

According to the discussion in Section 5.2.1, the value of δ depends on the value of k. There are the following two cases which corresponds to Case 2 and Case 3 in Section 5.2.1, respectively.

Case 2 in Section 5.2.1 where $2d \leq k < d^2 + d$: If $2d \leq k < d^2 + d$, $\delta = \sqrt{d-1}R$ according to the discussion in Section 5.2.1 and

$$\frac{\max_{p\in\mathbb{P}} f(p) - \max_{q\in S} f(q)}{\max_{q\in S} f(q)} \leq \frac{\sqrt{d}\delta^2}{R - \sqrt{d}} = \frac{R^2(d-1)\sqrt{d}}{R - \sqrt{d}}.$$

Note that the above inequality is valid for any value of R as long as $R > \sqrt{d}$. However, to obtain an upper bound which is as tight as possible, we want a value of R that minimizes the right hand side of the above inequality.

Lemma C.3. *$\frac{R^2}{R-\sqrt{d}}$ is minimized when $R = 2\sqrt{d}$.*

Proof. Note that $\frac{R^2}{R-\sqrt{d}} = (R+\sqrt{d}) + \frac{d}{R-\sqrt{d}} = 2\sqrt{d} + (R-\sqrt{d}) + \frac{d}{R-\sqrt{d}}$ whose minimum value is $4\sqrt{d}$ when $R = 2\sqrt{d}$. □

According to Lemma C.3, we define $R = 2\sqrt{d}$ and Equation (2) becomes $\frac{\max_{p\in\mathbb{P}} f(p) - \max_{q\in S} f(q)}{\max_{q\in S} f(q)} \leq 4(d-1)d$. That is, $\frac{\max_{q\in S} f(q)}{\max_{p\in\mathbb{P}} f(p)} \geq \frac{1}{4(d-1)d+1}$.

Then, $\mathrm{mrr}_{\mathbb{P}}(S) = \max_{f\in\mathcal{FC}} \left(1 - \frac{\max_{q\in S} f(q)}{\max_{p\in\mathbb{P}} f(p)}\right) \leq 1 - \frac{1}{4(d-1)d+1} = \frac{(d-1)d}{1/4+(d-1)d}$.

Case 3 in Section 5.2.1 where $k \geq d^2 + d$: According to the discussion in Section 5.2.1, we substitute the value of $\delta = \frac{\sqrt{d-1}R}{2\left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor}$ into Equation (2) and set R to be $2\sqrt{d}$ similarly to the first case. $\frac{\max_{p\in\mathbb{P}} f(p) - \max_{q\in S} f(q)}{\max_{q\in S} f(q)} \leq \frac{\sqrt{d}\delta^2}{R-\sqrt{d}} \leq \frac{R^2}{R-\sqrt{d}} \frac{(d-1)\sqrt{d}}{4\left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor^2} \leq \frac{(d-1)d}{\left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor^2}$.

That is, $\frac{\max_{q\in S} f(q)}{\max_{p\in\mathbb{P}} f(p)} \geq \left(\frac{(d-1)d}{\left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor^2} + 1 \right)^{-1}$. Then, we have $\mathrm{mrr}_{\mathbb{P}}(S) = \max_{f\in\mathcal{FC}} \left(1 - \frac{\max_{q\in S} f(q)}{\max_{p\in\mathbb{P}} f(p)}\right) \leq 1 - \frac{1}{\frac{(d-1)d}{\left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor^2} + 1} = \frac{(d-1)d}{\left\lfloor \left(\frac{k-d}{d^2}\right)^{\frac{1}{d-1}} \right\rfloor^2 + (d-1)d}$.

By combining the results above, the theorem follows. □

C.2 Proof of Lemma 5.4

Proof. We modify Lemma 4.3 to construct $\mathcal{I}_R^+$. The difference is that we focus on the sphere $\mathbb{S}_R^+$ in the first quadrant instead of $\mathbb{S}_R$.

We construct $\mathcal{I}_R^+$ by the following two-step procedure. Firstly, we partition all front facets of $\mathbb{C}^+$ into a set $\mathcal{F}$ of $k'(= dm^{d-1})$ smaller facets (the $(d-1)$-dimensional hypercubes). The procedure for partitioning the front facets will be presented shortly. Secondly, from each small facet in $\mathcal{F}$, we pick the point at the *center* of the facet, namely u^+, create the point s' which is in the same direction as u^+ and has its norm equal to R and insert it into $\mathcal{I}_R^+$, initialized to be an empty set. Since we pick one point on each facet in $\mathcal{F}$, the total number of points in $\mathcal{I}_R^+$ is $k'(= dm^{d-1})$.

In geometry, a *hyperplane* in a d-dimensional space is a subspace of dimensionality $d-1$. Formally, given a unit d-dimensional vector u and a non-negative constant c, a *hyperplane*, denoted by $h(u, c)$, is defined to $h(u, c) = \{p \in \mathbb{R}^d \mid u \cdot p = c\}$ where u is the *normal* of $h(u, c)$ and c is the *offset* of $h(u, c)$. Two hyperplanes are said to be *parallel* if their normals are the same. A facet F is said to lie on a hyperplane $h(u, c)$ if for each point p on F, we have $u \cdot p = c$. We say that a facet F is *parallel* to a hyperplane $h(u, c)$ if the hyperplane that facet F lies on is parallel to $h(u, c)$.

We focus on the procedure of partitioning a *particular* front facet F of $\mathbb{C}^+$ first. Consider a particular front facet F. We denote the set of remaining $d-1$ front facets by $\{F_1, F_2, \ldots, F_{d-1}\}$. We perform the following two-step procedure. Firstly, we construct a set $\mathcal{H}$ of hyperplanes with the following sub-steps. Let $h_i(u_i, c_i)$ be the hyperplane that the front facet F_i lies on for each $i \in [1, d-1]$. For each front facet F_i, we create m parallel hyperplanes each of which has a common normal u_i. For each $j \in [1, m]$, the j-th hyperplane created, denoted by $h_{ij}(u_i, c_{ij})$, has its offset c_{ij} equal to $\frac{j}{m}$. Since we construct a set of hyperplanes for each front facet, we create $d-1$ sets of hyperplanes. We perform a union operation on all these $d-1$ sets of hyperplanes and obtain the set $\mathcal{H}$. Secondly, for each hyperplane in $\mathcal{H}$, its intersection with F partitions F into two smaller facets each of which is further partitioned by other hyperplanes in $\mathcal{H}$. Thus, we obtain a set of m^{d-1} small facets.

A similar procedure is performed for each of the d front facets of $\mathbb{C}^+$. Thus, there are dm^{d-1} small facets in total. The union of all the small facets forms the desired set $\mathcal{F}$. Then, we relate δ, the distance bound on $\mathrm{dist}(s, s')$, with the diameter, denoted by β, of each small facet in $\mathcal{F}$. Note that $\beta = \frac{\sqrt{d-1}}{m}$. Since we pick the point u^+ at the center of each small facet in $\mathcal{F}$, for each point on the same small facet as u^+, its distance to u^+ is at most $\beta/2$ (We can also pick an arbitrary point in each small facet in $\mathcal{F}$. If this is the case, the distance is bounded by the diameter β instead of $\beta/2$). Following

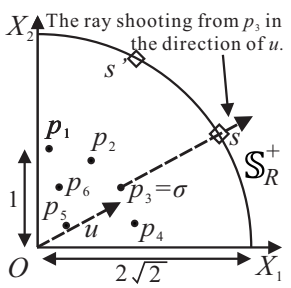


Figure 10: Point σ is the maximum utility point of $\mathbb{P}$ w.r.t. f

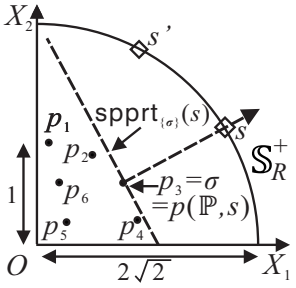


Figure 11: $\text{spprt}_{\{\sigma\}}(s)$ separates s from the rest of points in $\mathbb{P}$

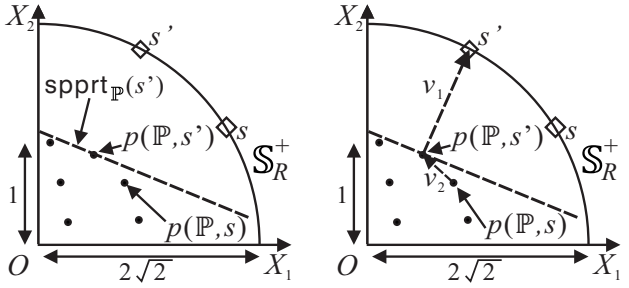


Figure 12: (a) $\text{spprt}_{\mathbb{P}}(s')$ separates s' and $p(\mathbb{P},s)$ (b) $v_1 \cdot v_2 \geq 0$ where $v_1 = s' - p(\mathbb{P},s')$, $v_2 = p(\mathbb{P},s') - p(\mathbb{P},s)$

a similar discussion as in Lemma 5.3, for each point s in $\mathbb{S}_R^+$, there exists a point $s' \in \mathcal{I}_R^+$ such that $\text{dist}(s,s') \leq R\beta/2 = \frac{\sqrt{d-1}R}{2m} = \delta$.

Consider the example in Figure 9 where k' is set to 4, d is 2 and R is $2\sqrt{d}$. Obviously, $k' = dm^{d-1}$ where $m = 2$. The hypercube $\mathbb{C}^+ = [0,1]^2$ is the convex hull formed by q_1, q_2, q_5 and the origin O. We denote the facet formed by the line segment between two points p and q simply by facet $\{p,q\}$. There are 2 front facets of $\mathbb{C}^+$. $\{q_1,q_5\}$ is a front facet since it does not pass through the origin. Similarly, $\{q_2,q_5\}$ is another front facet. We construct a set $\mathcal{I}_R^+$ of 4 points as follows. The front facets (with diameter 1) of $\mathbb{C}^+$, namely $\{q_1,q_5\}$ and $\{q_2,q_5\}$, are partitioned into a set $\mathcal{F} = \{\{q_1,q_3\},\{q_3,q_5\},\{q_4,q_5\},\{q_2,q_4\}\}$ of 4 smaller facets (with diameter 0.5). For facet $\{q_1,q_3\}$, we pick point u_1^+. Similarly, we pick u_2^+, u_3^+ and u_4^+ from the other three facets in $\mathcal{F}$. The points which are in the same direction as u_1^+, u_2^+, u_3^+ and u_4^+ and have their norms equal to $2\sqrt{d}$ are s_1', s_2', s_3' and s_4', respectively. That is, $\mathcal{I}_R^+ = \{s_1', s_2', s_3', s_4'\}$. For each point s in $\mathbb{S}_R^+$, there exists a point $s' \in \mathcal{I}_R^+$ such that $\text{dist}(s,s') \leq \delta = \frac{\sqrt{d-1}R}{2m} = \frac{\sqrt{2}}{2}$. □

C.3 Proof of Lemma C.2

Proof. Let s' be the point in $\mathcal{I}_R^+$ such that $\text{dist}(s,s') \leq \delta$. Consider Figure 10. Let the utility vector u of f be $(0.914, 0.406)$ and u is drawn in a dashed vector in Figure 10. Given the utility vector u, we can compute the utilities for all points in $\mathbb{P}$. It is not difficult to verify that p_3 is indeed the maximum utility point of $\mathbb{P}$ w.r.t. f and we name it as σ. The ray shooting from p_3 in the direction of u, which is also labeled in Figure 10, intersects $\mathbb{S}_R^+$ at s. The point s' in $\mathcal{I}_R^+$ with $\text{dist}(s,s') \leq \delta$ is also drawn in Figure 10 and the value of δ is determined in Section 5.2.1. Given a point $s \in \mathbb{S}_R^+$ and a set $P \subseteq \mathbb{P}$, a *supporting hyperplane* of s given P, denoted by $\text{spprt}_P(s)$, is the hyperplane that passes through $p(P,s)$ and is perpendicular to the vector $p(P,s) - s$. For example, in Figure 11 where $P = \{p_3\}$. The hyperplane $\text{spprt}_P(s)$ is labeled in a dashed line.

In the following, we introduce a few lemmas (ideas are mainly borrowed from [1]), which derive the connection between $\text{dist}(s,s')$ and the utility difference between points in S and points in $\mathbb{P}$.

Lemma C.4. $u = \frac{s - p(\mathbb{P},s)}{\|s - p(\mathbb{P},s)\|}$.

Proof. Note that the hyperplane $\text{spprt}_{\{\sigma\}}(s)$ separates s from the rest of points in $\mathbb{P}$. Otherwise, σ cannot be the maximum utility point of $\mathbb{P}$ w.r.t. f [13]. Thus, $\sigma = p(\mathbb{P},s)$ (because s is the intersection between $\mathbb{S}_R^+$ and the ray shooting from σ in the direction of u). By the definitions of σ and s, the vector $s - p(\mathbb{P},s)$ is in the same direction as u. Since u is a unit vector, the lemma follows. □

In Figure 11, we draw the hyperplane $\text{spprt}_{\{\sigma\}}(s)$ in a dashed line. $\text{spprt}_{\{\sigma\}}(s)$ separates s from the rest of points in $\mathbb{P}$ and thus, $p_3 = \sigma = p(\mathbb{P},s)$ as claimed. Moreover, we have $u = \frac{s - p(\mathbb{P},s)}{\|s - p(\mathbb{P},s)\|}$.

Lemma C.5. $0 \leq (s' - p(\mathbb{P},s')) \cdot (p(\mathbb{P},s') - p(\mathbb{P},s))$.

Proof. Note that the hyperplane $\text{spprt}_{\mathbb{P}}(s')$ (which passes through $p(\mathbb{P},s')$) separates s' and $p(\mathbb{P},s)$. If this is not the case, we have $\text{dist}(p(\mathbb{P},s),s') < \text{dist}(p(\mathbb{P},s'),s')$, which contradicts to the definition of $p(\mathbb{P},s')$ [13]. Since $\text{spprt}_{\mathbb{P}}(s')$ separates s' and $p(\mathbb{P},s)$, the angle between the vector $(s' - p(\mathbb{P},s'))$ and the vector $(p(\mathbb{P},s') - p(\mathbb{P},s))$ is at most 90 degrees and thus their dot product is at least 0. That is, we have $0 \leq (s' - p(\mathbb{P},s')) \cdot (p(\mathbb{P},s') - p(\mathbb{P},s))$. □

Consider Figure 12(a) where $\text{spprt}_{\mathbb{P}}(s')$ is labeled in a dashed line and $p(\mathbb{P},s')$ (in this particular example, $p(\mathbb{P},s') = p_2$) is indicated. Note that $\text{spprt}_{\mathbb{P}}(s')$ separates s' and $p(\mathbb{P},s)$. In Figure 12(b), we label the vector $(s' - p(\mathbb{P},s'))$ and the vector $(p(\mathbb{P},s') - p(\mathbb{P},s))$ as two dashed vectors v_1 and v_2, respectively. $v_1 \cdot v_2 \geq 0$ as claimed.

Lemma C.6. $a \cdot b - \|b\|^2 \leq \|a\|^2$ *where* $a, b \in \mathbb{R}^d$.

Proof. It follows from the fact $2a \cdot b \leq \|a\|^2 + \|b\|^2, \forall a, b \in \mathbb{R}^d$. □

With the help of the above lemmas, we are ready to develop the proof. Let $f(p) = u \cdot p$. By the definition of σ , we have $\max_{p \in \mathbb{P}} f(p) = u \cdot \sigma = u \cdot p(\mathbb{P},s)$. Besides, according to the way of constructing S, we have $\max_{q \in S} f(q) \geq f(p(\mathbb{P},s')) = u \cdot p(\mathbb{P},s')$. The inequality is true because we include all points in the basis $B(s')$ into the solution set S, and $p(\mathbb{P},s')$ is in the convex hull of $B(s')$ [1].

Then, $\|s - p(\mathbb{P},s)\|(\max_{p \in \mathbb{P}} f(p) - \max_{q \in S} f(q)) \leq$

$$
\begin{aligned}
& \|s - p(\mathbb{P},s)\|(u \cdot \sigma - u \cdot p(\mathbb{P},s')) \\
= \; & \|s - p(\mathbb{P},s)\| u \cdot (p(\mathbb{P},s) - p(\mathbb{P},s')) \\
= \; & (s - p(\mathbb{P},s)) \cdot (p(\mathbb{P},s) - p(\mathbb{P},s')) \text{ (by Lemma C.4)} \\
\leq \; & (s - p(\mathbb{P},s)) \cdot (p(\mathbb{P},s) - p(\mathbb{P},s')) \\
& + (s' - p(\mathbb{P},s')) \cdot (p(\mathbb{P},s') - p(\mathbb{P},s)) \text{ (by Lemma C.5)} \\
\leq \; & (s - p(\mathbb{P},s) - (s' - p(\mathbb{P},s'))) \cdot (p(\mathbb{P},s) - p(\mathbb{P},s')) \\
\leq \; & (s - s') \cdot (p(\mathbb{P},s) - p(\mathbb{P},s')) - \|p(\mathbb{P},s) - p(\mathbb{P},s')\|^2 \\
\leq \; & \|s - s'\|^2 = \delta^2 \text{ (by Lemma C.6).}
\end{aligned}
$$

By rewriting the inequality above, the lemma follows. □