K-Regret Queries Using Multiplicative Utility Functions

JIANZHONG QI, The University of Melbourne
FEI ZUO, The University of Melbourne
HANAN SAMET, University of Maryland
JIA CHENG YAO, The University of Melbourne

The k-regret query aims to return a size-k subset $\mathcal{S}$ of a database $\mathcal{D}$ such that, for any query user that selects a data object from this size-k subset $\mathcal{S}$ rather than from database $\mathcal{D}$, her regret ratio is minimized. The regret ratio here is modeled by the relative difference in the optimality between the locally optimal object in $\mathcal{S}$ and the globally optimal object in $\mathcal{D}$. The optimality of a data object in turn is modeled by a utility function of the query user. Unlike traditional top-k queries, the k-regret query does not minimize the regret ratio for a specific utility function. Instead, it considers a family of infinite utility functions $\mathcal{F}$, and aims to find a size-k subset that minimizes the maximum regret ratio of any utility function in $\mathcal{F}$.

Studies on k-regret queries have focused on the family of additive utility functions, which have limitations in modeling individuals' preferences and decision making processes, especially for a common observation called the diminishing marginal rate of substitution (DMRS). We introduce k-regret queries with multiplicative utility functions, which are more expressive in modeling the DMRS, to overcome those limitations. We propose a query algorithm with bounded regret ratios. To showcase the applicability of the algorithm, we apply it to a special family of multiplicative utility functions, the Cobb-Douglas family of utility functions, and a closely related family of utility functions, the Constant Elasticity of Substitution family of utility functions, both of which are frequently used utility functions in microeconomics. After a further study of the query properties, we propose a heuristic algorithm that produces even smaller regret ratios in practice. Extensive experiments on the proposed algorithms confirm that they consistently achieve small maximum regret ratios.

1 INTRODUCTION

Ever growing data have produced various databases that are beyond any user's capability to explore them in full. For example, Amazon has a database of over 562 million products [2]; Booking.com has a database of over 1.7 million hotels [1]. *Window queries* [5] (a.k.a. *range queries* [3, 4]),

Table 1. A Computer Database

Computer	CPU ($p_i.c_1$)	Brand recognition ($p_i.c_2$)	$f_1(p_i)$	$f_2(p_i)$	$f_3(p_i)$	$f_4(p_i)$
p_1	2.3	80	41.15	3.08	**13.56**	2.38
p_2	1.7	**90**	**45.85**	2.58	12.37	1.77
p_3	2.8	50	26.40	3.27	11.83	2.88
p_4	2.1	55	28.55	2.63	10.75	2.17
p_5	2.1	50	26.05	2.58	10.25	2.17
p_6	**3.0**	55	29.00	**3.52**	12.85	**3.09**

top-k queries [14, 20, 21, 42, 49], and *skyline queries* [8, 25, 26, 34, 43] have been used traditionally to produce a representative subset $\mathcal{S}$ when a database $\mathcal{D}$ is too large. Window queries return data objects with attribute values falling in certain ranges, which may not represent the distribution of the full database. Top-k and skyline queries, on the other hand, suffer by either requiring a predefined *utility function* to model user preferences over the data objects, or returning an unbounded number of data objects. Recent studies [24, 32, 52] aim to overcome these limitations by a new type of query, the *k-regret query*, which returns a size-k subset $\mathcal{S} \subseteq \mathcal{D}$ that minimizes the *maximum regret ratio* of any query user. The set $\mathcal{S}$ is called the *regret-minimizing set*. The concept of *regret* comes from microeconomics [27]. Intuitively, if a query user had selected the locally optimal object in $\mathcal{S}$, and were later shown the globally optimal object in $\mathcal{D}$, then the query user may have some regret. A k-regret query uses the *regret ratio* to model how regretful the query user may be, which is the relative difference in the *optimality* between the locally optimal object and the globally optimal object. Here, the optimality is computed by a utility function. The k-regret query considers a family of infinite utility functions such as the family of linear summation functions. It aims to find the subset $\mathcal{S}$ that minimizes the maximum regret ratio for any utility function in such a function family.

To illustrate the k-regret query, consider an online computer shop with a database $\mathcal{D}$ of computers as shown in Table 1. There are six computers: $\mathcal{D} = \{p_1, p_2, ..., p_6\}$. Every computer p_i has two attributes: CPU clock speed and brand recognition, denoted as $p_i.c_1$ and $p_i.c_2$, respectively. Here, brand recognition represents how well a brand is recognized by the customers. A larger value means that the brand is better recognized. Since the database may be too large to be shown in its entirety, the shop considers showing only a size-k subset $\mathcal{S} \subseteq \mathcal{D}$ in the front page as a recommendation. Such a subset may be $\mathcal{S} = \{p_1, p_3, p_5\}$ (i.e., $k = 3$). Suppose that there is a customer whose preference can be expressed as a utility function $f_1(p_i) = 0.5 \cdot p_i.c_1 + 0.5 \cdot p_i.c_2$. The customer may purchase p_1 from the recommended subset $\mathcal{S}$ since p_1 has the largest utility value: $f_1(p_1) = 0.5 \cdot p_1.c_1 + 0.5 \cdot p_1.c_2 = 0.5 \times 2.3 + 0.5 \times 80 = 41.15 > f_1(p_3) = 26.40 > f_1(p_5) = 26.05$. Note that another computer $p_2 \in \mathcal{D}$ exists with an even larger utility value $f_1(p_2) = 0.5 \times 1.7 + 0.5 \times 90 = 45.85$. If the customer later sees p_2, she may have some regret. Her regret ratio is computed as $\frac{f_1(p_2) - f_1(p_1)}{f_1(p_2)} \approx 10.25\%$. For another customer with a different utility function $f_2(p_i) = 0.99 \cdot p_i.c_1 + 0.01 \cdot p_i.c_2$, the computer in $\mathcal{S}$ that best suits her preference is p_3: $f_2(p_3) = 0.99 \times 2.8 + 0.01 \times 50 \approx 3.27 > f_2(p_1) \approx 3.08 > f_2(p_5) \approx 2.58$. For this customer, the globally optimal computer in $\mathcal{D}$ is p_6: $f_2(p_6) = 0.99 \times 3.0 + 0.01 \times 55 = 3.52$. If the customer purchases p_3, her regret ratio will be $\frac{f_2(p_6) - f_2(p_3)}{f_2(p_6)} \approx 7.05\%$. Since customers have different preferences and different utility functions, different data objects are needed to minimize their regret ratios. It is

unlikely that a size-k subset can satisfy all the customers. The k-regret query addresses this limitation by finding a subset S that minimizes the *maximum regret ratio* for a family of infinite utility functions.

Existing studies on k-regret queries have focused on the families of *additive utility functions* (AUFs) where the overall utility of an object is computed as the sum of the utility in each attribute of the object. The linear summation functions f_1 and f_2 are examples. They can be written in a more general form:

$$f(p_i) = \sum_{j=1}^{d} \alpha_j \cdot p_i.c_j,$$

where d denotes the number of attributes, and $\alpha_j \in [0, 1]$ is the *weight* of attribute j. Studies [24, 32] have shown that the maximum regret ratio of the k-regret query with AUFs can be bounded.

A significant limitation of AUFs, however, is that the overall utility of an object always increases at the same rate as the utility in an attribute increases. For an AUF $f(p_i) = \sum_{j=1}^{d} \alpha_j \cdot p_i.c_j$, the value of $f(p_i)$ always increases by α_j units for every unit of increase in attribute j. A large increase in the value of an attribute may cause a dramatic change in the value of the overall utility. Objects with the maximum values in certain attributes tend to be favored by AUFs, especially when the value ranges vary for different attributes. Consider again the example above. Utility function f_1 favors p_2 which has the maximum value in c_2 but also the minimum value in c_1. The two attributes c_1 and c_2, however, have the same weight (0.5) in the utility function, indicating that the user has the same preference towards the two attributes. The object favored by f_1 does not suit this preference. Thus, utility functions like f_1 do not model individuals' preferences and decision making processes well.

Intuitively, as the value of an attribute gets larger, adding an extra unit to its value should contribute a smaller increment to the overall utility. For example, adding a 4GB RAM to an old home computer with a 512MB RAM would make a major difference in the user experience; adding a 4GB RAM to a server with 256GB RAM would probably go without notice. This is in fact a common observation in individuals' decision making process called the *diminishing marginal rate of substitution* (DMRS) [11, 17, 45, 46]. The DMRS refers to the principle that, as the utility in an attribute j gets larger, the extent to which this utility can make up (substitute) for the utility in any other attribute j' decreases (diminishes). Thus, as the utility of attribute j gets larger, the increment of the overall utility when adding an extra unit to attribute j decreases.

To overcome the limitation of AUFs, we introduce a new type of k-regret queries with utility functions that are more expressive in modeling the DMRS – the k-regret query with *multiplicative utility functions* (MUFs). An MUF computes the overall utility of an object as the product of the utility in each attribute:

$$f(p_i) = \prod_{j=1}^{d} p_i.c_j^{\alpha_j}$$

An MUF helps deal with exponential-like utility functions and is more expressive in modeling the DMRS for the following reason. Since the attribute value $p_i.c_j$ has been raised to the power equal to the weight α_j, as $p_i.c_j$ gets larger, the increment of $f(p_i)$ when adding an extra unit to $p_i.c_j$ decreases. This is because the function $g(p_i.c_j) = (p_i.c_j + 1)^{\alpha_j} - p_i.c_j^{\alpha_j}$ is monotonically decreasing (i.e., $\forall \alpha_j \in [0, 1] : g'(p_i.c_j) \leq 0$). An MUF models user preferences towards different attributes better. For example, an MUF $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$ has the same weight in the two attributes. It favors p_1 in the example above, where $f_3(p_1) = 2.3^{0.5} \times 80^{0.5} \approx 13.56$ is larger than the function value of any other object. Object p_1 does not have the maximum value in either attribute but is reasonably good in both attributes. It suits the user preference. The MUF is also

more robust to large value changes in an attribute, since a weight with a value between 0 and 1 is in the power computation. The varying value ranges caused by the different types of attributes can be easily handled by an MUF, enabling the use of the attribute values in their natural form without normalization.

A related family of utility functions called the *Constant Elasticity of Substitution (CES) functions* studied earlier [24] can also model the DMRS. CES functions are not MUFs, and they have limitations as discussed below. A CES utility function $f(p_i)$ has the form of $f(p_i) = (\sum_{j=1}^{d} \alpha_j \cdot p_i.c_j^b)^{\frac{1}{b}}$ where b is a system parameter. Raising $p_i.c_j$ to the power of b allows a CES utility function to model the DMRS, making the function a popular utility function [45–47]. However, parameter b is a constant across all attributes. This leads to less flexibility in modeling different diminishing marginal rates of substitution over different attributes [12]. As studies in economics show [7, 29], finding a suitable value of b to fit a CES utility function to real data can be difficult and is sensitive to data construction. In comparison, an MUF raises $p_i.c_j$ to the power of α_j which can have different values for different attributes. This allows different diminishing marginal rates of substitution over different attributes and can be easier to fit user preferences.

The higher expressive power of MUFs brings challenges in bounding the maximum regret ratio for them. It is difficult to tightly bound the product of a series of exponential expressions. To the best of our knowledge, so far, no existing bound has been obtained for the k-regret query with MUFs. We overcome the challenges with a novel algorithm that we call *MinVar*. MinVar is an adaption of the *CUBE* [32] and the *MinWidth* [24] algorithms, which are k-regret query algorithms for AUFs. The MinVar algorithm partitions the data space into multiple buckets. Together the buckets enclose all the data objects, and one object in each bucket is returned to form the answer set $\mathcal{S}$. For any utility function, the corresponding optimal object p^* must be in some bucket. There is an object s^* in this bucket that has been returned in $\mathcal{S}$. The distance between p^* and s^* is bounded by the range of attribute values spanned by the bucket in each attribute, which further bounds the maximum regret ratio. Our contribution in MinVar is a novel space partitioning strategy based on data distribution, which produces tighter buckets in practice. More importantly, we make theoretical contributions by showing that the MinVar algorithm can obtain a maximum regret ratio bounded between $\Omega(\frac{1}{k^2})$ and $O(\ln(1 + \frac{1}{k^{\frac{1}{d-1}}}))$ for k-regret queries with MUFs, where d denotes the number of data attributes. To showcase the applicability of the MinVar algorithm in real world scenarios, we apply it on k-regret queries with a special family of MUFs, the *Cobb-Douglas functions*, which is used extensively in economics studies for modeling the DMRS [16, 18, 48]. As a by-product, we derive a new upper bound $O(\frac{1}{k^{\frac{1}{d-1}}})$ on the maximum regret ratio for k-regret queries with CES functions [48]. This upper bound is tighter than a previously obtained bound [24], while it also applies to the MinWidth algorithm [24] proposed for k-regret queries with CES functions.

MinVar aims to *bound* the maximum regret ratios rather than to *minimize* them. Its bucket-based answer object selection strategy is conservative. It works well when there are a large number of data objects lying in different buckets, each of which is optimal for a different utility function. However, in real data sets, many of the objects may not be optimal for any utility function. Some of the buckets created by MinVar may not contain any data objects that are optimal for any utility function. Returning objects in those buckets does not contribute to lowering the maximum regret ratios. We will show that, for MUFs, the set $\mathcal{P}$ of all the *skyline points* (objects) [8] in a database $\mathcal{D}$ minimizes the maximum regret ratio, which is 0. This is because any non-skyline point p_i must be dominated by at least one skyline point p_j, and hence its utility $f(p_i)$ does not exceed the utility $f(p_j)$ for any MUF f. If $|\mathcal{P}| \leq k$, the entire set of $\mathcal{P}$ should be returned as the query answer set.

Otherwise, we need to select k skyline points to form a size-k answer set. We use the regret ratio to guide the selection of skyline points such that the difference in the utilities of the selected and unselected points is minimized. This leads to a heuristic algorithm named *MaxDif* that produces even smaller maximum regret ratios in practice.

To summarize, our paper makes the following contributions:

- We introduce a novel type of k-regret queries − k-regret queries with multiplicative utility functions, which are more expressive in modeling the diminishing marginal rate of substitution in making decisions.
- We propose an algorithm named MinVar to process the query and to bound the maximum regret ratios. Based on this algorithm, we obtain bounds of the maximum regret ratio for the k-regret query with multiplicative utility functions.
- The MinVar algorithm is deigned for multiplicative utility functions but it can also be applied to non-multiplicative utility functions. We showcase such applicability via two families of utility functions used in economic studies: (i) the Cobb-Douglas family of utility functions, which is a special type of multiplicative utility functions that has not been studied before in the context of k-regret queries, and (ii) the CES family of utility functions, which is a family of non-multiplicative utility functions but is closely related to the Cobb-Douglas family of utility functions. As a by-product, we derive an upper bound on the maximum regret ratio for k-regret queries with CES utility functions that is tighter than an existing bound [24] under the case where the function parameter $b \in (0, 1)$. This bound applies to our MinVar algorithm as well as the MinWidth algorithm [24] proposed for k-regret queries with CES utility functions.
- Since MinVar aims to bound the maximum regret ratios rather than to minimize them, we further propose a heuristic algorithm named MaxDif that computes a size-k subset of skyline points to minimize the maximum regret ratios.
- We perform extensive experiments using both real and synthetic data to verify the effectiveness and efficiency of the proposed algorithms. The results show that the maximum regret ratios obtained by the proposed algorithms are consistently small. Meanwhile, the proposed algorithms are more efficient than the baseline algorithm MaxDom [28], which is a heuristic algorithm that computes top-k representative skyline points.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the basic concepts. Sections 4 and 5 describe the MinVar algorithm and derive bounds on the maximum regret ratio for k-regret queries with MUFs, respectively. Section 6 showcases the applicability of MinVar to both MUFs and non-MUFs. Section 7 presents the heuristic algorithm MaxDif. Section 8 examines the results of our experiments while Section 9 concludes the paper.

2 RELATED WORK

We review two queries: skyline and k-regret.

Skyline queries. The skyline query [8] is an earlier attempt to generate a representative subset $\mathcal{S}$ of a database $\mathcal{D}$ without specifying any utility functions. This query is defined based on the *dominance* relationship. It considers a database $\mathcal{D}$ of d-dimensional points ($d \in \mathbb{N}_+$). Let p_i and p_j be two points in $\mathcal{D}$. Point p_i is said to *dominate* point p_j if and only if $\forall l \in [1..d], p_i.c_l \geq p_j.c_l \wedge \exists l \in [1..d], p_i.c_l > p_j.c_l$, where $p_i.c_l$ $(p_j.c_l)$ denotes the coordinate of p_i (p_j) in dimension l. Here, the "$\geq$" and "$>$" operators represent the preference relationship. A point with a larger coordinate in dimension l is preferable in that dimension. The skyline query returns the subset $\mathcal{S} \subseteq \mathcal{D}$ where each point is *not* dominated by any other point in $\mathcal{D}$. It is interesting to observe that the attributes in the domain over which the skyline query is executed do not have to be spatial [38], as is the

case when they are embedded in a spatial database (e.g., [19, 39, 40]), nor is there a requirement for a distance function to exist between the objects (e.g., Euclidean or Hausdorff [33]).

The skyline query can be answered by a two-layer nested loop over the points in $\mathcal{D}$ and another layer of loop over the d dimensions to filter out the points dominated. The remaining points are *skyline points* which are the query answer. More efficient algorithms have been proposed in the literature [34, 43] but are not the focus of our study.

While the skyline query does not require a utility function, it suffers in lacking control over the size of the answer set. In the worst case, the entire database may be returned. Studies have tried to overcome this limitation by combining the skyline query with the top-k query. For example, Xia et al. [50] introduce the ε-*skyline* which adds a *weight* to each dimension of the data points to reflect user preference towards the dimension. The weights create a built-in rank for the points which can be used to answer the *top-k skyline query*. Chan et al. [13] rank the points by the *skyline frequency*, i.e., how frequently a point appears as a skyline point when different numbers of dimensions are considered. A few other studies extract a representative subset of the skyline points. Lin et al. [28] propose to return the k points that together dominate the most non-skyline points as the *k most representative skyline subset*. Tao et al. [44] select k *representative skyline points* based on the distance between the skyline points instead. These studies bound the size of the answer set. However, they do not bound the maximum regret ratio of the set.

K**-regret queries.** Nanongkai et al. [32] introduce the concept of *regret minimization* to top-k query processing and propose the k-*regret query*. This query does not require query users to specify their utility functions. Instead, it considers a family of infinite utility functions, and finds the subset $\mathcal{S}$ that minimizes the *maximum regret ratio* of the entire family of utility functions. Nanongkai et al. propose the *CUBE* algorithm to process the k-regret query with the family of linear summation utility functions, i.e., each utility function f is in the form of $f(p_i) = \sum_{j=1}^{d} \alpha_j \cdot p_i.c_j$ where α_j denotes the *weight* of dimension j. The CUBE algorithm is efficient, but the maximum regret ratio it obtains is quite large in practice. To obtain a smaller maximum regret ratio, in a different paper [31], Nanongkai et al. propose an interactive algorithm where query users are involved in guiding the search for answers with smaller regret ratios. Peng and Wong [36] advance the k-regret query studies by utilizing geometric properties to improve the query efficiency. Asudeh et al. [6] use the *convex hull* to find the data points that minimize the maximum regret ratio for linear summation utility functions. They propose an algorithm that can approximate the maximum regret ratio to within a user given threshold. Cao et al. [10] and Chester et al. [15] also consider linear summation utility functions but compute the k-*regret minimizing sets*, which is NP-hard.

Kessler Faulkner et al. [24] build on top of CUBE and propose three algorithms, *MinWidth*, *Area-Greedy*, and *Angle*. These three algorithms can process k-regret queries with the "concave", "convex", and CES utility functions. Nevertheless, the "concave" and "convex" utility functions considered have focused on *additive forms* (See Braziunas and Boutilier [9] and Keeney and Raiffa [23] for more details on *additive utilities* and *additive independence*). They are summations over a set of concave and convex functions. The CES utility functions also sum over a set of terms. In this paper, we introduce the use of the family of multiplicative utility functions to overcome the linearity limitation of the additive utility functions. We present an algorithm that can produce answers with bounded maximum regret ratios for k-regret queries with multiplicative utility functions. As a by-product, we also derive a new upper bound on the maximum regret ratio for k-regret queries with CES utility functions which is tighter than a previously obtained upper bound [24], while the bound also applies to the MinWidth algorithm.

Table 2. Frequently Used Symbols

Symbol	Description
$\mathcal{D}$	A database
n	The cardinality of $\mathcal{D}$
d	The dimensionality of $\mathcal{D}$
k	The k-regret query parameter
$\mathcal{S}$	A size-k subset selected from $\mathcal{D}$
$\mathcal{P}$	The set of all the skyline points in $\mathcal{D}$
p_i	A point in $\mathcal{D}$
$p_i.c_j$	The coordinate value of p_i in dimension j
t	The number of intervals into which the data domain is partitioned in a dimension

Zeighami and Wong [52] propose to compute the *average regret ratio*. They do not assume any particular type of utility functions, but use sampling to obtain a few utility functions for the computation. This study is less relevant to our work and is not discussed further.

Note that Chester et al. [15] have used the term *k-regret minimizing set* to denote a subset $\mathcal{S}$ of size r that minimizes the *maximum k-regret ratio*, where the regret is measured by the utility difference between the optimal point in $\mathcal{S}$ and the kth optimal point in the database $\mathcal{D}$. We use the term *k-regret query* following the closest related work [24] to denote a query that finds the *regret-minimizing set*, which is a subset $\mathcal{S}$ of size k that minimizes the *maximum regret ratio*, where the regret is measured by the utility difference between the optimal points in $\mathcal{S}$ and $\mathcal{D}$.

The concept of regret has also been used in a classic problem in operations research – the *multi-armed bandit* (MAB) problem [22]. The MAB problem assumes N *arms* each associated with an unknown reward distribution. During a multi-round process, in each round, an agent chooses an arm and collects a reward generated by the corresponding reward distribution. Let the reward collected at round i be r_i and the largest expected reward of any arm be μ^*. The regret after T rounds is $T\mu^* - \sum_{i=1}^{T} \mathbb{E}(r_i)$. The key question in the MAB problem is how to balance the exploitation on the arm with the largest expected reward observed so far (to maximize r_i for the current round) and the exploration to find the arm with the globally largest expected reward μ^* (to maximize r_i for future rounds). This is less relevant to our problem, and we will not discuss this question further.

3 PRELIMINARIES

We present basic concepts and a problem definition in this section. The symbols frequently used in the discussion are summarized in Table 2.

We consider a database $\mathcal{D}$ of n data objects. Every data object $p_i \in \mathcal{D}$ is a d-dimensional point in $\mathbb{R}_+^d$, where d is a positive integer and the coordinate values of the points are all positive numbers. We use $p_i.c_j$ to denote the coordinate value of p_i in dimension j. This coordinate value represents the *utility* of p_i in dimension j. A larger coordinate value denotes a larger utility and is preferable. A query parameter k is given. It specifies the size of the answer set $\mathcal{S}$ ($\mathcal{S} \subseteq \mathcal{D}$) to be returned. We assume $d \leq k \leq n$.

Gain. Let $f : \mathcal{D} \to \mathbb{R}_+$ be a function that models the utility of a data object, i.e., how preferable the data object is by a query user. The *gain* of a query user over a set $\mathcal{S}$, denoted by *gain*($\mathcal{S}, f$), is

the maximum utility of any object in $\mathcal{S}$, i.e.,

$$gain(\mathcal{S}, f) = \max_{p_i \in \mathcal{S}} f(p_i) \tag{1}$$

Continuing with the example shown in Table 1, if $\mathcal{S} = \{p_1, p_3, p_5\}$ and $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$,

$$gain(\mathcal{S}, f_3) = \max_{p_i \in \mathcal{S}} f_3(p_i) = f_3(p_1) = 2.3^{0.5} \times 80^{0.5} \approx 13.56$$

Regret. For a subset $\mathcal{S}$ of $\mathcal{D}$, the gain over $\mathcal{S}$ may be smaller than that of $\mathcal{D}$. The difference between $gain(\mathcal{D}, f)$ and $gain(\mathcal{S}, f)$ is the *regret* of a query user if she selects the locally optimal object from $\mathcal{S}$ and is later shown the globally optimal object in $\mathcal{D}$, denoted by $regret_{\mathcal{D}}(\mathcal{S}, f)$:

$$regret_{\mathcal{D}}(\mathcal{S}, f) = gain(\mathcal{D}, f) - gain(\mathcal{S}, f) \tag{2}$$

Regret ratio. The *regret ratio*, $r\_ratio_{\mathcal{D}}(\mathcal{S}, f)$, is a relative measure of the regret. It is computed as the regret $regret_{\mathcal{D}}(\mathcal{S}, f)$ over the gain $gain(\mathcal{D}, f)$, i.e.,

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, f) = \frac{regret_{\mathcal{D}}(\mathcal{S}, f)}{gain(\mathcal{D}, f)} = \frac{max_{p_i \in \mathcal{D}} f(p_i) - max_{p_j \in \mathcal{S}} f(p_j)}{max_{p_i \in \mathcal{D}} f(p_i)} \tag{3}$$

Continuing with the example in Table 1, given $\mathcal{S} = \{p_1, p_3, p_5\}$ and $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$, $gain(\mathcal{S}, f_3) = gain(\mathcal{D}, f_3) = f_3(p_1) \approx 13.56$. We have $regret_{\mathcal{D}}(\mathcal{S}, f_3) = 0$ and $r\_ratio_{\mathcal{D}}(\mathcal{S}, f_3) = 0\%$. Given a different utility function $f_4(p_i) = p_i.c_1^{0.99} \cdot p_i.c_2^{0.01}$, $gain(\mathcal{S}, f_4) = f_4(p_3) \approx 2.88$ and $gain(\mathcal{D}, f_4) = f_4(p_6) \approx 3.09$. Then, $regret_{\mathcal{D}}(\mathcal{S}, f_4) \approx 0.21$ and $r\_ratio_{\mathcal{D}}(\mathcal{S}, f_4) \approx \frac{0.21}{3.09} \approx 6.80\%$.

Maximum regret ratio. Given a set $\mathcal{S}$ and a family of utility functions $\mathcal{F}$, the *maximum regret ratio* formulates how regretful a query user can be if her utility function is in $\mathcal{F}$. It is the supremum of the regret ratio of a query user with any utility function in $\mathcal{F}$, i.e.,

$$mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) = \sup_{f \in \mathcal{F}} r\_ratio_{\mathcal{D}}(\mathcal{S}, f) = \sup_{f \in \mathcal{F}} \frac{max_{p_i \in \mathcal{D}} f(p_i) - max_{p_j \in \mathcal{S}} f(p_j)}{max_{p_i \in \mathcal{D}} f(p_i)} \tag{4}$$

Here, the supremum is used instead of the maximum because we consider an infinite set $\mathcal{F}$.

Continuing with the example above, if $\mathcal{F} = \{f_3, f_4\}$,

$$mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) = \max\{0\%, 6.80\%\} = 6.80\%$$

The *k-regret* query aims to return the size-k subset $\mathcal{S} \subseteq \mathcal{D}$ that *minimizes* the maximum regret ratio for a family of utility functions.

DEFINITION 1 (K-REGRET QUERY). *Given a family of utility functions $\mathcal{F}$, the k-regret query returns a size-k subset $\mathcal{S} \subseteq \mathcal{D}$, such that the maximum regret ratio over $\mathcal{S}$ is smaller than or equal to that over any other size-k subset $\mathcal{S}' \subseteq \mathcal{D}$. Formally,*

$$\forall \mathcal{S}' \subseteq \mathcal{D} \cap |\mathcal{S}'| = k : mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq mr\_ratio_{\mathcal{D}}(\mathcal{S}', \mathcal{F})$$

Specific utility functions are not always available because the query users are not usually known in advance and their utility functions may not be specified precisely. The *k-regret* query does not require any specific utility functions to be given. Instead, the query considers a family of infinite functions such as the family of linear functions [32], i.e., $f(p_i) = \sum_{j=1}^{d} \alpha_j \cdot p_i.c_j$ where α_j is the *weight* of dimension j. The *k-regret* query minimizes the maximum regret ratio of any utility function in such a family of utility functions, without knowing the value of the α_j's.

Our contribution to the study of *k-regret* queries is the incorporation of a family of *multiplicative utility functions* (MUFs).

DEFINITION 2 (MULTIPLICATIVE UTILITY FUNCTION). *A multiplicative utility function (MUF) f is defined to be a utility function of the following form:*

$$f(p_i) = \prod_{j=1}^{d} p_i.c_j^{\alpha_j},$$

where $\alpha_j \geq 0$ is a function parameter and $\sum_{j=1}^{d} \alpha_j \leq 1$.

DEFINITION 3 (K-REGRET QUERY WITH MUFs). *The k-regret query with MUFs takes a database $\mathcal{D}$ of d-dimensional points and a family of MUFs $\mathcal{F}$ as the input. It returns a size-k subset $\mathcal{S} \subseteq \mathcal{D}$, such that the maximum regret ratio $mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$ is minimized.*

We note that a k-regret query to find the size-k subset $\mathcal{S}$ that minimizes the maximum regret ratio is NP-hard, as shown by Chester et al. [15]. In this study, we first focus on bounding the maximum regret ratio for k-regret queries with MUFs. We compute a subset $\mathcal{S}$ with a maximum regret ratio that is bounded by a decreasing function of k. This subset, however, may not minimize the maximum regret ratio. We thus further design a greedy algorithm to compute a subset to (heuristically) minimize the maximum regret ratio.

Scale invariance. It has been shown [24, 32] that k-regret queries with additive utility functions are *scale invariant*, i.e., scaling the data domain in any dimension does not change the maximum regret ratio of a set $\mathcal{S}$. This property also holds for k-regret queries with MUFs. For an MUF $f(p_i) = \prod_{d=1}^{n} p_i.c_j^{\alpha_j}$, we can scale each dimension by a factor $\lambda_j > 0$, resulting in a new MUF $f'(p_i) = \prod_{d=1}^{n} (\lambda_j \cdot p_i.c_j)^{\alpha_j} = \prod_{d=1}^{n} \lambda_j^{\alpha_j} \cdot \prod_{d=1}^{n} p_i.c_j^{\alpha_j} = (\prod_{d=1}^{n} \lambda_j^{\alpha_j}) f(p_i)$. Such scaling does not affect the regret ratio (and hence the maximum regret ratio), i.e., $r\_ratio_{\mathcal{D}}(\mathcal{S}, f') = r\_ratio_{\mathcal{D}}(\mathcal{S}, f)$:

$$\begin{aligned}
r\_ratio_{\mathcal{D}}(\mathcal{S}, f') &= \frac{max_{p_i \in \mathcal{D}} f'(p_i) - max_{p_j \in \mathcal{S}} f'(p_j)}{max_{p_i \in \mathcal{D}} f'(p_i)} \\
&= \frac{(\prod_{d=1}^{n} \lambda_j^{\alpha_j})(max_{p_i \in \mathcal{D}} f(p_i) - max_{p_j \in \mathcal{S}} f(p_j))}{(\prod_{d=1}^{n} \lambda_j^{\alpha_j}) max_{p_i \in \mathcal{D}} f(p_i)} \\
&= \frac{max_{p_i \in \mathcal{D}} f(p_i) - max_{p_j \in \mathcal{S}} f(p_j)}{max_{p_i \in \mathcal{D}} f(p_i)} \\
&= r\_ratio_{\mathcal{D}}(\mathcal{S}, f)
\end{aligned}$$

In what follows, for conciseness, we refer to the regret of a query user as the regret when the context is clear. The same applies to the regret ratio and the maximum regret ratio of a query user.

4 THE MINVAR ALGORITHM

We propose an algorithm named *MinVar* to process k-regret queries with MUFs. MinVar shares a similar overall algorithmic approach with that of CUBE [32] and MinWidth [24] which were proposed to process k-regret queries with additive utility functions. The core idea of the algorithm is to partition the data space into multiple buckets that together enclose all data points, and return one data point in each bucket to form the answer set $\mathcal{S}$. For any utility function, the corresponding optimal data point p^* must be in some bucket. There is a point s^* in this bucket that has been returned in $\mathcal{S}$. The distance between p^* and s^* is bounded by the range of attribute values spanned by the bucket in each attribute, which further bounds the maximum regret ratio.

Our contributions in the MinVar algorithm are a novel data space partitioning strategy that follows the data distribution and produces tighter buckets, and theoretical bounds on the maximum

regret ratios obtained. We first discuss how to partition the data space and select the point in each bucket to be returned. We derive the bounds on the maximum regret ratio afterwards.

ALGORITHM 1: MinVar

Input: $\mathcal{D} = \{p_1, p_2, ..., p_n\}$: a d-dimensional database; k: the size of the answer set.
Output: $\mathcal{S}$: a size-k subset of $\mathcal{D}$.

1 $\mathcal{S} \leftarrow \emptyset$;
2 **for** $i = 1, 2, ..., d - 1$ **do**
3 $\quad$ Find p_i^* which has the largest utility $p_i^*.c_i$ in dimension i;
4 $\quad$ $c_i^\tau \leftarrow p_i^*.c_i$;
5 $\quad$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{p_i^*\}$;
6 $itr \leftarrow 0$;
7 **while** $|\mathcal{S}| < k$ and $itr < itr_m$ **do**
8 $\quad$ $t \leftarrow \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor + itr$;
9 $\quad$ **for** $i = 1, 2, ..., d - 1$ **do**
10 $\quad\quad$ $bps[i] \leftarrow FindBreakpoints(\mathcal{D}, t, n, i, c_i^\tau)$;
11 $\quad$ **for** each $(d - 1)$-integer combination $1 \le j_1 \le t, 1 \le j_2 \le t, ..., 1 \le j_{d-1} \le t$ **do**
12 $\quad\quad$ $B \leftarrow \{p \in \mathcal{D} | \forall i \in [1..d - 1] : bps[i][j_i].lo \le p.c_i \le bps[i][j_i].hi\}$;
13 $\quad\quad$ $s^* \leftarrow argmax_{p \in B} \, p.c_d$;
14 $\quad\quad$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{s^*\}$;
15 $\quad\quad$ **if** $|\mathcal{S}| == k$ **then**
16 $\quad\quad\quad$ break;
17 $\quad$ $itr \leftarrow irt + 1$;
18 $\mathcal{S} \leftarrow \mathcal{S} \cup \{k - |\mathcal{S}| \text{ random points in } \mathcal{D} \setminus \mathcal{S}\}$;
19 **return** $\mathcal{S}$;

The MinVar algorithm. Algorithm 1 summarizes the MinVar algorithm. Before partitioning the data space, the algorithm first finds the optimal point p_i^* in each of the first $d - 1$ dimensions, i.e., $p_i^*.c_i$ is the largest in dimension i ($i = 1, 2, ..., d - 1$). These $d - 1$ optimal points are added to $\mathcal{S}$ (Lines 1 to 5). They are intended to minimize the regret ratio for the first $d - 1$ dimensions. Another $k - d + 1$ points are needed to fill up $\mathcal{S}$. These points minimize the regret ratio for dimension d. The algorithm partitions each of the first $d - 1$ dimensions of the data domain into t intervals (Lines 8 to 10), where

$$t = \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor \tag{5}$$

These intervals together partition the data space into t^{d-1} buckets:

$$1 \le t^{d-1} = \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor^{(d-1)} \le k - d + 1 \tag{6}$$

The algorithm selects one point s^* in each bucket that has the largest utility $s^*.c_d$ in dimension d, and adds s^* to $\mathcal{S}$ (Lines 11 to 14). There may be less than $k - d + 1$ buckets, and some of the buckets may be empty. Thus, there may be less than $k - d + 1$ points added in this step. To ensure k points in $\mathcal{S}$, we repeat the partitioning step and increase t by 1 in each iteration (Lines 7, 8, and 17). This creates more buckets and obtains more points to be added to $\mathcal{S}$. The loop terminates when $|\mathcal{S}| = k$ (Lines 15 and 16) or a preset number of iterations itr_m has been reached (Line 7). At this point, if $|\mathcal{S}| < k$, we simply fill it up with randomly selected points and return the set (Line 18). The set $\mathcal{S}$ is then returned (Line 19).

Fig. 1. The MinVar algorithm

Figure 1 gives an example. Suppose $d = 3$ and $k = 6$. Then $t = \lfloor (6-3+1)^{\frac{1}{3-1}} \rfloor = 2$. We first add the two points p_1^* and p_2^* to $\mathcal{S}$ which have the largest utility in dimensions 1 and 2, respectively. Then, the data domain in dimensions 1 and 2 are each partitioned into $t = 2$ intervals, forming $t^{d-1} = 2^{3-1} = 4$ buckets. Four more points s_1^*, s_2^*, s_3^*, and s_4^* are added to $\mathcal{S}$, each has the largest utility in dimension 3 in a different bucket. Now there are 6 points in $\mathcal{S}$, i.e., $\mathcal{S} = \{p_1^*, p_2^*, s_1^*, s_2^*, s_3^*, s_4^*\}$. No further partitioning is needed, and the set $\mathcal{S}$ is returned as the query answer.

The FindBreakpoints algorithm. The novelty of MinVar lies in the sub-algorithm *FindBreakpoints* to find the breakpoints to partition a dimension of the data domain into t intervals (Line 10). The intuition behind the algorithm is as follows. The optimal point p^* for any utility function f must lie in one of the buckets created. Let this bucket be $\mathcal{B}$. The algorithm selects a point s^* from $\mathcal{B}$ to represent this bucket and adds it to $\mathcal{S}$. In the best case, p^* is selected as s^*, and the regret ratio is 0. To maximize the probability of p^* being selected, intuitively, we should partition each dimension such that every interval contains the same number of points. Otherwise, if the intervals are skewed and p^* happens to lie in a dense interval, then its probability of being selected is small.

Fig. 2. Creating the intervals in a dimension

In CUBE [32], the data domain is partitioned evenly, i.e., each interval has the same size of $\frac{c_i^\tau}{t}$, where c_i^τ denotes the largest utility in dimension i (assuming that the data domain starts at 0). When the data points are not uniformly distributed, the probability of p^* being selected to represent its bucket is small. Figure 2 gives an example where $d = 2$. The two intervals created by CUBE in dimension 1 are highly unbalanced in the number of points in each interval. Point p^* has a large utility in both dimensions and may be optimal for many MUFs. However, it will not be

selected by CUBE, since it falls in a dense bucket, and there is another point s^* with a larger utility in dimension 2. In MinWidth [24], the data domain is partitioned with a greedy heuristic. This heuristic leaves out some *empty intervals* with no data points. It uses a binary search to determine the minimal interval width such that t intervals of such width can cover the rest of the data domain. MinWidth handles sparse data better, but the equi-width intervals still do not handle skewed data well. Figure 2 shows two intervals created by MinWidth which are still unbalanced (one has 5 points and the other has 3). The two points p^* and s^* are still in the same bucket.

ALGORITHM 2: FindBreakpoints

Input: $\mathcal{D} = \{p_1, p_2, ..., p_n\}$: a d-dimensional database; t: number of intervals; n : size of $\mathcal{D}$; i: dimension number for which the breakpoints are to be computed; c_i^τ: the largest utility in dimension i.

Output: $bps[i]$: an array of t pairs of breakpoints.

1 Sort $\mathcal{D}$ ascendingly on dimension i; let the sorted point sequence be $p_1', p_2', ..., p_n'$;

2 $hi \leftarrow 0$, $\delta \leftarrow 0$;

3 **while** $hi \neq n$ **do**

4      $lo \leftarrow 1$;

5   **for** $j = 1, 2, ..., t$ **do**

6          $hi' \leftarrow \min\{lo + \lceil n/t \rceil - 1 + \delta, n\}$;

7   Find the largest $hi \in [lo..hi']$ such that $p_{hi}'.c_i - p_{lo}'.c_i \leq \dfrac{c_i^\tau}{t}$;

8          $bps[i][j].lo \leftarrow p_{lo}'.c_i$;

9          $bps[i][j].hi \leftarrow p_{hi}'.c_i$;

10          $lo \leftarrow hi + 1$;

11      $\delta \leftarrow \delta + inc$;

12 **return** $bps[i]$;

To overcome these limitations, our FindBreakpoints algorithm adaptively uses t variable-width intervals such that the number of points in each interval is as close to $\lceil \frac{n}{t} \rceil$ as possible, i.e., the *var*iation of the number of points in each interval is *min*imized, which motivates the name of the MinVar algorithm. As shown in Fig. 2, the two gray intervals created by MinVar contain $\lceil \frac{8}{2} \rceil = 4$ points each; p^* will be selected to represent its bucket. To help derive the maximum regret ratio bounds in the following subsections, we also require that the width of each interval does not exceed $\frac{c_i^\tau}{t}$. Under this constraint, it is not always possible to create t intervals with exactly $\lceil \frac{n}{t} \rceil$ points in each interval. Therefore, we allow $\lceil \frac{n}{t} \rceil + \delta$ data points in each interval, where δ is a parameter that will be adaptively chosen by the algorithm. At the start $\delta = 0$.

Algorithm 2 summarizes the FindBreakpoints algorithm. This algorithm first sorts the data points in ascending order of their coordinate values in dimension i. The sorted points are denoted as $p_1', p_2', ..., p_n'$ (Line 1). The algorithm then creates t intervals, where lo and hi represent the subscript lower and upper bounds of the data points to be put into one interval, respectively. At the start, $lo = 1$ (Line 4). Between lo and $lo + \lceil \frac{n}{t} \rceil - 1 + \delta$, the algorithm finds the largest subscript hi such that $p_{hi}'.c_i - p_{lo}'.c_i$ does not exceed $\frac{c_i^\tau}{t}$. We then have obtained the two breakpoints of the first interval $bps[i][1].lo = p_{lo}'.c_i$ and $bps[i][1].hi = p_{hi}'.c_i$, where $bps[i]$ is an array to store the intervals in dimension i. We update lo to be $hi + 1$, and repeat the above process to create the next interval

(Lines 5 to 10). When t intervals are created, if they cover all the n points, we have successfully created the intervals for dimension i. Otherwise, we need to allow a larger number of points in each interval. We increase δ by inc which is a system parameter (Line 11), and repeat the above procedure to create t intervals until n points are covered. Then, we return the interval array $bps[i]$ (Line 12). Note that the algorithm always terminates, because when δ increases to n, the algorithm will simply create intervals each with width $\frac{c_i^\tau}{t}$. The t intervals must cover the entire data domain and hence cover all n points.

Algorithm correctness. As will be shown in the following subsections, the bounds on maximum regret ratios rely on the fact that the interval size does not exceed $\dfrac{c_i^\tau}{\lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor}$. In Min-Var, even though FindBreakPoints creates variable-width intervals, each interval is still bounded by $\frac{c_i^\tau}{t}$. The value of t starts at $\lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$ and is kept increasing in the loop. Therefore, MinVar creates intervals where the size does not exceed $\dfrac{c_i^\tau}{\lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor}$. This satisfies the requirement of the bounds and guarantees the algorithm correctness. In practice, MinVar can obtain maximum regret ratios smaller than the upper bound derived, since the intervals created by FindBreakPoints may be smaller than $\frac{c_i^\tau}{t}$ and t may be larger than $\lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$.

Algorithm complexity. FindBreakpoints uses a database $\mathcal{D}$ of n points and an array $bps[i]$ to store t intervals. The space complexity is $O(n+t) = O(n)$ where $t = \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor \ll n$. Leaving out the space for storing the input data, the space complexity is $O(t)$. Sorting the points in dimension i takes $O(n \log n)$ time (Line 1). The inner loop of FindBreakpoints (Lines 5 to 10) has t iterations. In each iteration, computing hi requires a binary search between p'_{lo} and $p'_{hi'}$, which takes $O(\log n)$ time. Thus, The inner loop takes $O(t \log n)$ time. The outer loop has $\frac{n}{inc}$ iterations in the worst case. Together, FindBreakpoints takes $O(n \log n + \frac{tn \log n}{inc})$ time.

MinVar uses a database $\mathcal{D}$ of n points, an answer set $\mathcal{S}$ of size k, a $(d-1) \times t$ two dimensional array bps. An array of size $t^{d-1} = k - d + 1$ is also needed to help select the points s^* in the $k - d + 1$ buckets. The space complexity is $O(n + k + dt)$. Leaving out the space for storing the input data, the space complexity is $O(k + dt)$. The first for-loop of MinVar (Lines 2 to 5) takes $O(nd)$ time. The second for-loop (Lines 9 and 10) calls FindBreakpoints $d - 1$ times, which takes $O(dn \log n + \frac{tdn \log n}{inc})$ time. The third for-loop (Lines 11 to 16) finds a point s^* in each of the $k - d + 1$ buckets. A linear scan on the database $\mathcal{D}$ is needed for this task. For each point p visited, we need a binary search on each of the $d - 1$ arrays $bps[i]$ to identify the bucket of p, and to update the point selected s^* in that bucket if needed. This takes $O(nd \log t)$ time. The second and third for-loops are enclosed in a loop to iterate through multiple values of t. The number of iterations is bounded by itr_m. Overall, MinVar takes $O(nd + itr_m(dn \log n + \frac{(t + itr_m)dn \log n}{inc} + nd \log(t + itr_m)))$ time. Here, $\frac{n}{inc}$ and itr_m are controllable parameters of the system. In the experiments, we set $inc = 0.1\% n$. We observe that $itr_{max} = 11$ is sufficient in the data sets tested. The time complexity then simplifies to $O(nd \log n + nd \log t) = O(nd \log(nt))$.

5 MAXIMUM REGRET RATIO BOUNDS

We derive bounds for the maximum regret ratio of k-regret queries with MUFs.

5.1 Upper Bound

We start with an upper bound of a set $\mathcal{S}$ returned by MinVar. The intuition behind the bound is as follows. Given any MUF, its optimal point p^* must be in some bucket created by MinVar. There is also one point s^* returned by MinVar that is at the same bucket as p^*. The point s^* has the largest coordinate value in dimension d. Meanwhile, the difference between s^* and p^* in dimension i ($1 \leq i \leq d-1$) is bounded by the interval size $\dfrac{c_i^{\tau}}{t}$. Thus, the difference between the weighted products of the coordinate values of the two points should be bounded in a certain range. This range yields an upper bound of the maximum regret ratios.

We assume that $p_i.c_j$ has been normalized into the range of $(1, 2]$ to simplify the derivation of the upper bound. This can be done by a normalization function $\mathcal{N}(p_i.c_j) = 1 + \dfrac{p_i.c_j}{\max_{p_i \in \mathcal{D}}\{p_i.c_j\}}$. In fact, it is common to normalize the data domain in different dimensions into the same range, so that utility values of different dimensions become more comparable. *Note that our derivation of the bounds still holds without this assumption, although the bounds may become less concise.* This assumption does not affect the correctness of the MinVar algorithm either, although now the interval size should be bounded by $\dfrac{c_i^{\tau} - 1}{t}$ where 1 is the lower bound of the data domain.

Our upper bound is given by the following theorem.

THEOREM 1. *Let $\mathcal{F} = \{f | f(p_i) = \prod_{j=1}^{d} p_i.c_j^{\alpha_j}\}$ be a set of MUFs, where $\alpha_j \geq 0$, $\sum_{j=1}^{d} \alpha_j \leq 1$, and $1 < p_i.c_j \leq 2$. The maximum regret ratio $mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$ of an answer set $\mathcal{S}$ of MinVar satisfies*

$$mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \ln(1 + \frac{1}{t}) \tag{7}$$

PROOF. We prove the theorem by showing that for each MUF $f \in \mathcal{F}$, the regret ratio $r\_ratio_{\mathcal{D}}(\mathcal{S}, f)$ must be less than or equal to $\ln(1 + \dfrac{1}{t})$. Thus, the maximum regret ratio of $\mathcal{F}$ must also be less than or equal to $\ln(1 + \dfrac{1}{t})$.

Let p^* be the point in $\mathcal{D}$ with the largest utility computed by f, i.e.,

$$p^* = \underset{p_i \in \mathcal{D}}{\mathrm{argmax}} f(p_i)$$

Let s^* be the point in $\mathcal{S}$ that is selected by MinVar in the same bucket in which p^* lies. We have:

$$
\begin{aligned}
regret_{\mathcal{D}}(\mathcal{S}, f) &= \max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_i \in \mathcal{S}} f(p_i) \\
&\leq f(p^*) - f(s^*) \\
&= \prod_{j=1}^{d} p^*.c_j^{\alpha_j} - \prod_{j=1}^{d} s^*.c_j^{\alpha_j} \\
&= \exp\left(\ln \prod_{j=1}^{d} p^*.c_j^{\alpha_j}\right) - \exp\left(\ln \prod_{j=1}^{d} s^*.c_j^{\alpha_j}\right)
\end{aligned}
\tag{8}
$$

Next, we show that $e^x - e^y \leq (x-y)e^x$, which will enable us to simplify the exponential terms in the equation above. Let $g(x) = 1 - e^{-x} - x$. We have $g'(x) = e^{-x} - 1$ and $g''(x) = -e^{-x}$. By letting $g'(x) = 0$, we have $x = 0$, while $g''(0) = -1 < 0$. Thus, the maximum of $g(x)$ is $g(0) = 0$, which means $1 - e^{-x} - x \leq 0$. Therefore, $1 - e^{-x} \leq x$. Replacing x by $x - y$ yields $1 - e^{y-x} \leq x - y$. We multiply e^x to both sides of the inequality and obtain $(1 - e^{y-x})e^x \leq (x-y)e^x$. Thus, $e^x - e^y \leq (x-y)e^x$. Equation 8 is then relaxed as follows.

$$regret_{\mathcal{D}}(\mathcal{S}, f) \leq (\ln \prod_{j=1}^{d} p^*.c_j^{\alpha_j} - \ln \prod_{j=1}^{d} s^*.c_j^{\alpha_j}) \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j}$$

$$= (\sum_{j=1}^{d} \alpha_j \ln p^*.c_j - \sum_{j=1}^{d} \alpha_j \ln s^*.c_j) \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j} \qquad (9)$$

$$= \left[\sum_{j=1}^{d} \alpha_j (\ln p^*.c_j - \ln s^*.c_j) \right] \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j}$$

Since MinVar selects the point in a bucket with the largest value in dimension d, we know that $p^*.c_d \leq s^*.c_d$ and hence $\ln p^*.c_d \leq \ln s^*.c_d$, i.e., $\ln p^*.c_d - \ln s^*.c_d \leq 0$. Thus, we can remove the utility in dimension d from the computation and relax the regret to be:

$$regret_{\mathcal{D}}(\mathcal{S}, f) \leq \left[\sum_{j=1}^{d-1} \alpha_j (\ln p^*.c_j - \ln s^*.c_j) \right] \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j}$$

$$= \left[\sum_{j=1}^{d-1} \alpha_j \ln(\frac{p^*.c_j}{s^*.c_j}) \right] \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j} \qquad (10)$$

Since s^* is selected from the same bucket in which p^* lies, $p^*.c_j - s^*.c_j$ must be constrained by the bucket size in dimension j, which is $\dfrac{c_j^{\tau} - 1}{t}$ where c_j^{τ} and 1 are the largest and smallest utility values in dimension j, i.e.,

$$\forall j \in [1..d-1], p^*.c_j - s^*.c_j \leq \frac{c_j^{\tau} - 1}{t} \qquad (11)$$

Thus,

$$\frac{p^*.c_j}{s^*.c_j} \leq 1 + \frac{c_j^{\tau} - 1}{t \cdot s^*.c_j} \qquad (12)$$

Since $1 < s^*.c_j \leq c_j^{\tau} \leq 2$, we have

$$\frac{p^*.c_j}{s^*.c_j} < 1 + \frac{1}{t} \qquad (13)$$

Therefore,

$$regret_{\mathcal{D}}(\mathcal{S}, f) < \left[\sum_{j=1}^{d-1} \alpha_j \ln(1 + \frac{1}{t}) \right] \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j} \qquad (14)$$

For the regret ratio $r\_ratio_{\mathcal{D}}(\mathcal{S}, f)$, we have

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, f) = \frac{regret_{\mathcal{D}}(\mathcal{S}, f)}{gain(\mathcal{D}, f)}$$

$$< \frac{\left[\sum_{j=1}^{d-1} \alpha_j \ln(1 + \frac{1}{t}) \right] \cdot \prod_{j=1}^{d} p^*.c_j^{\alpha_j}}{\prod_{j=1}^{d} p^*.c_j^{\alpha_j}}$$

$$= \sum_{j=1}^{d-1} \alpha_j \ln(1 + \frac{1}{t}) = \ln\left((1 + \frac{1}{t})^{\sum_{j=1}^{d-1} \alpha_j} \right) \qquad (15)$$

$$\leq \ln(1 + \frac{1}{t})$$

$\square$

In the theorem, $t = \lfloor(k - d + 1)^{\frac{1}{d-1}}\rfloor$. Intuitively, when k increases (i.e., returning more points), the maximum regret ratio is expected to decrease; when d increases (i.e., accumulating regret over more dimensions), the maximum regret ratio is expected to increase. For simplicity, we say that the upper bound grows in the scale of $O(\ln(1 + \frac{1}{k^{\frac{1}{d-1}}}))$.

To give an example, consider a 2-dimensional database, i.e., $d = 2$. Let $k = 3$, which means $t = 2$. The upper bound of the maximum regret ratio is $\ln(1 + \frac{1}{t}) = \ln\frac{3}{2} \approx 40.54\%$. As k increases (e.g., to 20), this upper bound will decrease (e.g., to $\ln\frac{20}{19} \approx 5.13\%$).

We have assumed $p_i.c_j \in (1, 2]$ in the proof above. In a more general case where $p_i.c_j$ lies in a range $(c_\perp, c_\top)$, $0 \leq c_\perp < c_\top$ (positive utilities are considered), the upper bound derived becomes less concise. In particular, Equation 11 becomes $\forall j \in [1..d-1], p^*.c_j - s^*.c_j \leq \dfrac{c_j^\tau - c_\perp}{t}$, since the lower bound of the data space is now $c_\perp$. Dividing both sides of the inequality by $s^*.c_j$ yields:

$$\frac{p^*.c_j}{s^*.c_j} \leq 1 + \frac{c_j^\tau - c_\perp}{t \cdot s^*.c_j} < 1 + \frac{c_\top - c_\perp}{t \cdot c_\perp}$$

The rest of the proof stays the same, except for that $1 + \dfrac{1}{t}$ needs to be replaced by $1 + \dfrac{c_\top - c_\perp}{t \cdot c_\perp}$. Therefore, in the more general case, the maximum regret ratio is bounded by $\ln(1 + \dfrac{c_\top - c_\perp}{t \cdot c_\perp})$. This bound is less tight as it may be greater than 1.

5.2 Lower Bound

We now derive a lower bound of the maximum regret ratio of any k-regret algorithm for MUFs (including but not limited to MinVar), assuming an infinite database $\mathcal{D}$. We show that, given a family of MUFs $\mathcal{F}$, it is impossible to bound the regret ratio to below $\Omega(\dfrac{1}{k^2})$ for a database $\mathcal{D}$ of infinite 2-dimensional points (i.e., $d = 2$). The idea behind the lower bound is as follows. Given sufficient points, for any size-k subset $\mathcal{S}$ returned, we can find an MUF f such that its corresponding optimal point is sufficiently far away from any of the points in $\mathcal{S}$, and the regret ratio $r\_ratio_\mathcal{D}(\mathcal{S}, f)$ is at least $\Omega(\dfrac{1}{k^2})$.

THEOREM 2. *Given $k > 0$, there must be a database $\mathcal{D}$ of 2-dimensional points such that the maximum regret ratio of any size-k subset $\mathcal{S} \subseteq \mathcal{D}$ over a family of MUFs $\mathcal{F}$ is at least $\Omega(\dfrac{1}{k^2})$.*

PROOF. We assume a data space of $(1, e] \times (1, e]$ in this proof. Consider an infinite set $\mathcal{D}$ of 2-dimensional points, where each point p corresponds to an angle θ in a polar coordinate system as illustrated in Fig. 3. The coordinate values of p satisfy:

$$\begin{cases} p.c_1 = e^{\cos\theta} \\ p.c_2 = e^{\sin\theta} \end{cases} \quad 0 < \theta \leq \frac{\pi}{2} \tag{16}$$

Given a size-k subset $\mathcal{S} \subseteq \mathcal{D}$, each point $s_i \in \mathcal{S}$ corresponds to a $\theta_i \in (0, \dfrac{\pi}{2}]$ where $s_i.c_1 = e^{\cos\theta_i}$ and $s_i.c_2 = e^{\sin\theta_i}$. Assume that the points $s_1, s_2, ..., s_k$ in $\mathcal{S}$ are sorted in ascending order of their corresponding θ_i values, i.e., $0 < \theta_1 \leq \theta_2 \leq ... \leq \theta_k \leq \dfrac{\pi}{2}$. Further, let $\theta_0 = 0$ and $\theta_{k+1} = \dfrac{\pi}{2}$. Then θ_i ($i \in [0..k+1]$) can be represented as a point on a unit circle as shown in Fig. 3. There are $k + 2$ points in $[0, \dfrac{\pi}{2}]$. Every pair of adjacent points forms an angle, resulting in a total of $k + 1$ angles.

Fig. 3. Lower bound illustration

One of the adjacent pairs (i.e., for some $j \in [0..k]$) must satisfy:

$$\theta_{j+1} - \theta_j \geq \frac{\pi}{2(k+1)} \tag{17}$$

Otherwise, if all angles are less than $\frac{\pi}{2(k+1)}$, their sum will be less than $\frac{\pi}{2}$. Let θ^* be in the middle of θ_j and θ_{j+1}, i.e.,

$$\theta^* = \frac{\theta_j + \theta_{j+1}}{2} \tag{18}$$

We construct an MUF f where the optimal point p^* corresponds to θ^*, i.e., $p^*.c_1 = e^{\cos \theta^*}$ and $p^*.c_2 = e^{\sin \theta^*}$, and prove the theorem based on the regret ratio of f.

Consider an MUF $f(p) = p.c_1^{\cos \theta^*} \cdot p.c_2^{\sin \theta^*}$.

$$\begin{aligned}
\ln f(p) &= \ln (p.c_1^{\cos \theta^*} \cdot p.c_2^{\sin \theta^*}) \\
&= \cos \theta^* \cdot \ln p.c_1 + \sin \theta^* \cdot \ln p.c_2 \\
&= \cos \theta^* \cdot \cos \theta + \sin \theta^* \cdot \sin \theta
\end{aligned} \tag{19}$$

Let $g(\theta) = \cos \theta^* \cdot \cos \theta + \sin \theta^* \cdot \sin \theta$. By letting $g'(\theta) = 0$, we obtain $\theta = \theta^*$. Meanwhile, $g''(\theta^*) = -1 < 0$. Thus, $\ln f(p)$ is maximized when $\theta = \theta^*$, and $f(p)$ is maximized when $p = p^*$.

$$\ln f(p^*) = \cos^2 \theta^* + \sin^2 \theta^* = 1; \qquad f(p^*) = e. \tag{20}$$

Meanwhile, let s_i be the optimal point for f in $\mathcal{S}$. Since there is no other points in $\mathcal{S}$ that is between θ_j and θ_{j+1},

$$|\theta_i - \theta^*| = \Delta \geq \theta_{j+1} - \theta^* = \theta^* - \theta_j \geq \frac{\pi}{4(k+1)} \tag{21}$$

We consider the case where $\theta_i - \theta^* = \Delta$. The other case where $\theta^* - \theta_i = \Delta$ is symmetric. We omit it for conciseness.

$$\begin{aligned}
\ln f(s_i) &= \ln (s_i.c_1^{\cos \theta^*} \cdot s_i.c_2^{\sin \theta^*}) \\
&= \cos(\theta^* + \Delta) \cdot \cos \theta^* + \sin(\theta^* + \Delta) \cdot \sin \theta^* \\
&= (\cos \theta^* \cos \Delta - \sin \theta^* \sin \Delta) \cdot \cos \theta^* + (\sin \theta^* \cos \Delta + \cos \theta^* \sin \Delta) \cdot \sin \theta^* \\
&= \cos^2 \theta^* \cdot \cos \Delta + \sin^2 \theta^* \cdot \cos \Delta \\
&= \cos \Delta
\end{aligned} \tag{22}$$

Here, the transformation is based on sine and cosine of sum identities. Now we have $f(s_i) = e^{\cos \Delta}$, and $r\_ratio_{\mathcal{D}}(\mathcal{S}, f)$ satisfies

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, f) = \frac{f(p^*) - f(s_i)}{f(p^*)} = \frac{e - e^{\cos \Delta}}{e} = 1 - e^{\cos \Delta - 1} \tag{23}$$

Based on the Maclaurin series, we have

$$e^{\cos \Delta - 1} = 1 - \frac{\Delta^2}{2} + \frac{\Delta^4}{6} - \cdots \tag{24}$$

Thus,

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, f) = \frac{\Delta^2}{2} - \frac{\Delta^4}{6} + \cdots \tag{25}$$

We already know that

$$\Delta \geq \frac{\pi}{4(k + 1)} \tag{26}$$

Therefore,

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, f) \geq \frac{\pi^2}{32(k + 1)^2} - o(\frac{1}{k^4}) \tag{27}$$

This means that $r\_ratio_{\mathcal{D}}(\mathcal{S}, f)$ is at least $\Omega(\frac{1}{k^2})$.   □

6 CASE STUDIES

In this section, we showcase the applicability of the MinVar algorithm to both MUFs and non-MUFs. We derive the maximum regret ratio bounds for applying MinVar on k-regret queries with a real world example of MUFs – the Cobb-Douglas family of utility functions, and a closely related family of non-multiplicative utility functions – the Constant Elasticity of Substitution (CES) family of utility functions.

6.1 The K-Regret Query with Cobb-Douglas Functions

The Cobb-Douglas function was first proposed as a production function to model the relationship between multiple inputs and the amount of output generated [16]. It was later generalized as a utility function. As a real example of MUFs, this utility function has been used extensively in economics studies for modeling the diminishing marginal rate of substitution [16, 18, 48].

DEFINITION 4 (COBB-DOUGLAS FUNCTION). *A generalized* Cobb-Douglas function *[48] with d inputs $x_1, x_2, ..., x_d$ is a mapping $\mathcal{X} : \mathbb{R}_+^d \to \mathbb{R}_+$,*

$$\mathcal{X}(x_1, x_2, ..., x_d) = A \prod_{j=1}^{d} x_j^{\alpha_j}$$

Here, $A > 0$ and $\alpha_j \geq 0$ are the function parameters.

The generalized Cobb-Douglas function is very similar to an MUF. The d inputs here can be seen as a data point of d dimensions where input x_j is the utility in dimension j. MinVar applies to k-regret queries with Cobb-Douglas functions straightforwardly.

To derive an upper bound of the maximum regret ratio for a family of Cobb-Douglas functions

$$\mathcal{F} = \{\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_n\},$$

we transform each function $\mathcal{X}_i$ to an MUF by scaling the parameter A to 1. It can be shown straight-forwardly that this scaling does not affect the regret ratio or the maximum regret ratio. Assume

that x_j has been normalized into the range of $(1, 2)$. Then, the regret ratio upper bound derived in Section 5.1 applies, i.e.,

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{X}_i) \leq \ln\left((1 + \frac{1}{t})^{\sum_{j=1}^{d-1} \alpha_j}\right) \tag{28}$$

Here, each function $\mathcal{X}_i$ has a different set of parameters $\{\alpha_1, \alpha_2, ..., \alpha_d\}$. If $\sum_{j=1}^{d} \alpha_j \leq 1$ holds for every $\mathcal{X}_i \in \mathcal{F}$, the maximum regret ratio is bounded by

$$mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \ln(1 + \frac{1}{t}) \tag{29}$$

Otherwise, the maximum regret ratio is bounded by

$$mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \ln\left((1 + \frac{1}{t})^{\alpha^{\tau}}\right),$$
$$\alpha^{\tau} = \max\{\sum_{j=1}^{d-1} \mathcal{X}_i.\alpha_j | \mathcal{X}_i \in \mathcal{F}, \mathcal{X}_i.\alpha_j \text{ is a parameter of } \mathcal{X}_i\}. \tag{30}$$

Similarly, the lower bound $\Omega(\frac{1}{k^2})$ of the maximum regret ratio derived in Section 5.2 also applies.

6.2 The K-Regret Query with CES Functions

The CES function is a non-MUF that is closely related to the Cobb-Douglas function. It is also used as a production function as well as a utility function [46, 47]. The function provides an alternative model for how well the utility of an attribute makes up for that of another attribute, which is often used in economics studies [45–47] and has been considered previously for k-regret queries [24].

DEFINITION 5 (CES FUNCTION). *A generalized CES function [47] with d inputs $x_1, x_2, ..., x_d$ is a mapping $\mathcal{X} : \mathbb{R}_+^d \to \mathbb{R}_+$,*

$$\mathcal{X}(x_1, x_2, ..., x_d) = A(\sum_{j=1}^{d} \alpha_j x_j^{\rho})^{\frac{\gamma}{\rho}}$$

Here, $A > 0$, $\alpha_j \geq 0$, $\rho < 1$ ($\rho \neq 0$), and $\gamma > 0$ are the function parameters.

When ρ approaches 0 in the limit, the CES function will become a Cobb-Douglas function. When ρ approaches 1, the CES function is very similar to the linear summation utility function. The case where $\rho > 1$ is not considered in the original proposal [51] of the CES utility function. We do not consider this case either, but this case could be an interesting subject for future work.

Algorithm MinVar can also process k-regret queries with CES utility functions. To derive bounds for the maximum regret ratio, we simplify and rewrite the CES function $\mathcal{X}$ as a function f as follows, assuming that $A = \gamma = 1$. Making $A = 1$ can be done by scaling, while the case where $\gamma \neq 1$ is considered as future work.

$$f(p_i) = (\sum_{j=1}^{d} \alpha_j \cdot p_i.c_j^b)^{\frac{1}{b}}$$

Here, $0 < b < 1$ and $\alpha_j \geq 0$.

It has been shown [24] that the maximum regret ratio for k-regret queries with CES utility functions is bounded between $\Omega(\frac{1}{bk^2})$ and $O(\frac{1}{bk^{\frac{b}{d-1}}})$ when $0 < b < 1$ (between $\Omega(\frac{1}{bk^2})$ and $O(\frac{1}{k^{\frac{1}{b(d-1)}}})$ when $b > 1$). The lower bound also applies to our MinVar algorithm. In what follows,

we derive a tighter upper bond for the case where $0 < b < 1$. Note that this bound does not require the data space to be in $(1, 2]$ in each dimension, and it also applies to the MinWidth algorithm [24].

We first derive a new upper bound for the regret ratio for a single CES utility function f. Again, the intuition is to use the bucket size to bound the difference between $f(p^*)$ and $f(s^*)$, where p^* is the optimal point for f and s^* is the point in the same bucket as p^* returned by MinVar.

THEOREM 3. *Let* $f(p_i) = (\sum_{j=1}^{d} \alpha_j \cdot p_i.c_j^b)^{\frac{1}{b}}$ *be a CES utility function, where* $0 < b < 1$ *and* $\alpha_j \geq 0$. *The regret ratio* $r\_regret_{\mathcal{D}}(\mathcal{S}, f)$ *of a set* $\mathcal{S}$ *returned by MinVar satisfies*

$$r\_ratio_{\mathcal{D}}(\mathcal{S}, f) \leq \frac{d^{\frac{1}{b}}}{t + d^{\frac{1}{b}}} \tag{31}$$

PROOF. Let p^* be the point in $\mathcal{D}$ with the largest utility computed by f, and s^* be the point in $\mathcal{S}$ that is selected in the same bucket in which p^* lies. We have:

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &= \max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_i \in \mathcal{S}} f(p_i) \\ &\leq f(p^*) - f(s^*) \\ &= (\sum_{j=1}^{d} \alpha_j \cdot p^*.c_j^b)^{\frac{1}{b}} - (\sum_{j=1}^{d} \alpha_j \cdot s^*.c_j^b)^{\frac{1}{b}} \end{aligned} \tag{32}$$

Since $g(x) = x^{\frac{1}{b}}$ is convex when $0 < b < 1$, we have $g(x) - g(y) \leq (x - y)g'(x)$. Thus,

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq (\sum_{j=1}^{d} \alpha_j \cdot p^*.c_j^b - \sum_{j=1}^{d} \alpha_j \cdot s^*.c_j^b)\frac{1}{b}(\sum_{j=1}^{d} \alpha_j \cdot p^*.c_j^b)^{\frac{1}{b}-1} \\ &= \frac{1}{b}\left[\sum_{j=1}^{d} \alpha_j(p^*.c_j^b - s^*.c_j^b)\right](\sum_{j=1}^{d} \alpha_j \cdot p^*.c_j^b)^{\frac{1}{b}-1} \end{aligned} \tag{33}$$

Consider another function $g(x) = x^b$, which is concave when $0 < b < 1$, and $g'(x)$ is monotonically decreasing ($g''(x) < 0$). According to Lagrange's Mean Value Theorem, there must exist some value ξ between two values x and y, such that $g(x) - g(y) = x^b - y^b = (x - y) \cdot g'(\xi) = (x - y) \cdot b \cdot \xi^{b-1}$. Further, since $g'(x)$ is monotonically decreasing, $b \cdot \xi^{b-1} \leq b \cdot (\min\{x, y\})^{b-1}$. Thus, we have

$$\begin{aligned} p^*.c_j^b - s^*.c_j^b &\leq |p^*.c_j - s^*.c_j| \cdot b \cdot (\min\{p^*.c_j, s^*.c_j\})^{b-1} \\ &\leq \frac{c_j^\tau}{t} \cdot b \cdot c_j^{\tau b-1} = \frac{b}{t} c_j^{\tau b} \end{aligned} \tag{34}$$

Therefore,

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq \frac{1}{b}(\sum_{j=1}^{d} \alpha_j \frac{b}{t} c_j^{\tau b})(\sum_{j=1}^{d} \alpha_j \cdot p^*.c_j^b)^{\frac{1}{b}-1} \\ &\leq \frac{d}{t}(\max_{j \in [1..d]} \alpha_j c_j^{\tau b})\left[d \cdot \max_{j \in [1..d]} \alpha_j c_j^{\tau b}\right]^{\frac{1}{b}-1} \\ &= \frac{d^{\frac{1}{b}}}{t}(\max_{j \in [1..d]} \alpha_j c_j^{\tau b})^{\frac{1}{b}} \end{aligned} \tag{35}$$

Recall that p_j^* has the largest utility in dimension j, i.e., $p_j^*.c_j = c_j^\tau$. This means that $\alpha_j p_j^*.c_j^b = \alpha_j c_j^{\tau b}$. Since $\forall l \in [1..d]$, $\alpha_l \geq 0$ and $p_j^*.c_l > 0$, we have $\sum_{l=1}^d \alpha_l \cdot p_j^*.c_l^b \geq \alpha_j p_j^*.c_j^b = \alpha_j c_j^{\tau b}$. Thus,

$$
\begin{aligned}
regret_{\mathcal{D}}(\mathcal{S}, f) &\leq \frac{d^{\frac{1}{b}}}{t} (\max_{j \in [1..d]} \sum_{l=1}^d \alpha_l \cdot p_j^*.c_l^b)^{\frac{1}{b}} \\
&= \frac{d^{\frac{1}{b}}}{t} \max_{j \in [1..d]} (\sum_{l=1}^d \alpha_l \cdot p_j^*.c_l^b)^{\frac{1}{b}}
\end{aligned}
\tag{36}
$$

Let $\sigma = \dfrac{d^{\frac{1}{b}}}{t}$. Then,

$$
\begin{aligned}
regret_{\mathcal{D}}(\mathcal{S}, f) &\leq \sigma \max_{j \in [1..d]} (\sum_{l=1}^d \alpha_l \cdot p_j^*.c_l^b)^{\frac{1}{b}} \\
\frac{1}{\sigma} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq \max_{j \in [1..d]} (\sum_{l=1}^d \alpha_l \cdot p_j^*.c_l^b)^{\frac{1}{b}}
\end{aligned}
\tag{37}
$$

By the design of the MinVar algorithm, p_j^* ($j \in [1..d-1]$) is in $\mathcal{S}$. Meanwhile, the point with the largest utility in dimension d in each bucket is also in $\mathcal{S}$, which means that p_d^* is also in $\mathcal{S}$. Thus,

$$
\begin{aligned}
\max_{p_i \in \mathcal{S}} f(p_i) &\geq \max_{j \in [1..d]} f(p_j^*) \\
&= \max_{j \in [1..d]} (\sum_{l=1}^d \alpha_l \cdot p_j^*.c_l^b)^{\frac{1}{b}} \\
&\geq \frac{1}{\sigma} regret_{\mathcal{D}}(\mathcal{S}, f)
\end{aligned}
$$

This means $\dfrac{\max_{p_i \in \mathcal{S}} f(p_i)}{regret_{\mathcal{D}}(\mathcal{S}, f)} \geq \dfrac{1}{\sigma}$. The regret ratio $r\_regret_{\mathcal{D}}(\mathcal{S}, f)$ is hence bounded by

$$
\begin{aligned}
r\_regret_{\mathcal{D}}(\mathcal{S}, f) &= \frac{regret_{\mathcal{D}}(\mathcal{S}, f)}{regret_{\mathcal{D}}(\mathcal{S}, f) + \max_{p_i \in \mathcal{S}} f(p_i)} \\
&= \frac{1}{1 + \frac{\max_{p_i \in \mathcal{S}} f(p_i)}{regret_{\mathcal{D}}(\mathcal{S}, f)}} \\
&\leq \frac{1}{1 + \frac{1}{\sigma}} = \frac{\sigma}{1 + \sigma} = \frac{d^{\frac{1}{b}}}{t + d^{\frac{1}{b}}}
\end{aligned}
\tag{38}
$$

$\square$

Therefore, given a set $\mathcal{F}$ of CES functions, the maximum regret ratio $mr\_regret_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$ satisfies:

$$
mr\_regret_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \frac{d^{\frac{1}{b}}}{t + d^{\frac{1}{b}}}
\tag{39}
$$

We can see from this bound that, when k decreases or d increases, the maximum regret ratio is expected to increase. For simplicity, we say that this bound grows in a scale of $O(\frac{1}{k^{\frac{1}{d-1}}})$. This bound is tighter than the bound $O(\frac{1}{bk^{\frac{b}{d-1}}})$ obtained in a previous study [24] since $0 < b < 1$.

To give an example, consider a family of CES functions where $b = 0.5$. Let $d = 2$ and $k = 3$, which means $t = 2$. The upper bound of the maximum regret ratio $\frac{d^{\frac{1}{b}}}{t + d^{\frac{1}{b}}} = \frac{4}{2+4} \approx 66.67\%$. As k increases (e.g., to 20), this upper bound will decrease (e.g., to $\frac{4}{19+4} \approx 17.39\%$).

Fig. 4. Skyline point selection

7 THE MAXDIF ALGORITHM

MinVar aims to *bound* the maximum regret ratios rather than to *minimize* them. Its bucket-based answer point selection strategy is conservative. To minimize the maximum regret ratios, we propose a heuristic based second query algorithm named *MaxDif* that exploits *skyline points* [8]. We first show that the answer set to minimize the maximum regret ratio must be formed by skyline points. When there are more than k skyline points, we need to select k from them to form an answer set. MaxDif makes this selection following a heuristic for regret ratio minimization. If there are no more than k skyline points, the entire set of skyline points should be returned as the answer. The answer set can be padded with randomly selected objects from the database to make it of size-k.

Regret ratio minimization with skyline points. Skyline points are points that are not *dominated* by any other points. Given two points p_i and p_j, p_i is said to dominate p_j if and only if the coordinate values of p_i are no smaller than those of p_j in all dimensions, and there is at least one dimension where the coordinate of p_i is larger than that of p_j, i.e.,

$$\forall l \in [1..d], p_i.c_l \geq p_j.c_l \wedge \exists l \in [1..d], p_i.c_l > p_j.c_l \tag{40}$$

In Fig. 4, the hollow dots p_1, p_2, p_3, p_4, and p_5 are skyline points as they are not dominated, while the black dots are *non-skyline points*, i.e., points dominated by some skyline point.

Let $\mathcal{P}$ be the set of all skyline points. Its regret over any MUF f, $regret_{\mathcal{D}}(\mathcal{P}, f)$, must be 0. This is because, for any non-skyline point $p_j \in \mathcal{D} \setminus \mathcal{P}$, the skyline point $p_i \in \mathcal{P}$ dominating p_j satisfies $f(p_i) \geq f(p_j)$ for any MUF f by definition, and hence $gain(\mathcal{P}, f) \geq gain(\mathcal{D} \setminus \mathcal{P}, f)$. Thus,

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{P}, f) &= gain(\mathcal{D}, f) - gain(\mathcal{P}, f) \\ &= gain(\mathcal{D} \setminus \mathcal{P} \cup \mathcal{P}, f) - gain(\mathcal{P}, f) \\ &= \max\{gain(\mathcal{D} \setminus \mathcal{P}, f), gain(\mathcal{P}, f)\} - gain(\mathcal{P}, f) \\ &= gain(\mathcal{P}, f) - gain(\mathcal{P}, f) \\ &= 0 \end{aligned} \tag{41}$$

Therefore, *the regret ratio of $\mathcal{P}$ and the maximum regret ratio of $\mathcal{P}$ over a family of MUFs must be 0.*

When $|\mathcal{P}| \leq k$, $\mathcal{P}$ is the optimal answer to a k-regret query with MUFs. When $|\mathcal{P}| > k$, a size-k subset of it must be the optimal answer. The reason is as follows. For any size-k subset $\mathcal{S} \in \mathcal{D}$, we can construct a new size-k subset $\mathcal{S}'$ by replacing every non-skyline point $p_j \in \mathcal{S}$ with a skyline point $p_i \in \mathcal{P}$ that dominates p_j. The gain of $\mathcal{S}'$ over any MUF f must be larger than or equal to that of $\mathcal{S}$. Thus, the maximum regret ratio of $\mathcal{S}'$ must be smaller than or equal to that of $\mathcal{S}$. Therefore, the optimal answer set that minimizes the maximum regret ratio must be a set that contains skyline points only, and hence it is a subset of $\mathcal{P}$. This is formulated as the following theorem.

THEOREM 4. *Let $\mathcal{F} = \{f | f(p_i) = \prod_{j=1}^{d} p_i.c_j^{\alpha_j}\}$ be a set of MUFs, where $\alpha_j \geq 0$ and $\sum_{j=1}^{d} \alpha_j \leq 1$. Suppose that there are k or more skyline points in a database $\mathcal{D}$ of size n. Then, there is a size-k subset $\mathcal{S}^* \subseteq \mathcal{D}$ that contains only skyline points, and its maximum regret ratio is less than or equal to that of any other size-k subset $\mathcal{S} \subseteq \mathcal{D}$, i.e., $mr\_ratio_{\mathcal{D}}(\mathcal{S}^*, \mathcal{F}) \leq mr\_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$.*

PROOF. The proof is straightforward as sketched in the paragraph above. We omit the full proof for conciseness.   □

We note that a similar theorem has been proven [6] in parallel to show that skyline points minimize the maximum regret ratio for AUFs. Both AUFs and MUFs are monotonically increasing in each dimension, and their parameters are both non-negative. This allows skyline points to minimize the maximum regret ratio for both types of utility functions.

Selection of k skyline points. Theorem 4 reduces the k-regret query to selecting k skyline points that together minimize the maximum regret ratio. This can be done by finding the size-k subset $\mathcal{S}^* \subseteq \mathcal{P}$ such that, for any MUFs in $\mathcal{F}$, the maximum regret of $\mathcal{S}^*$ over the gain of $\mathcal{P}$ is minimized. Since $\mathcal{P} = (\mathcal{P} \setminus \mathcal{S}^*) \cup \mathcal{S}^*$, the goal translates to minimizing the maximum ratio of gain difference between $\mathcal{P} \setminus \mathcal{S}^*$ and $\mathcal{S}^*$ over the gain of $\mathcal{P} \setminus \mathcal{S}^*$ for any MUFs in $\mathcal{F}$. Formally,

$$
\begin{aligned}
\mathcal{S}^* &= \operatorname*{arg\,min}_{\mathcal{S} \subseteq \mathcal{P}, |\mathcal{S}|=k} \max_{f \in \mathcal{F}} \left\{ \frac{gain(\mathcal{P} \setminus \mathcal{S}, f) - gain(\mathcal{S}, f)}{gain(\mathcal{P} \setminus \mathcal{S}, f)} \right\} \\
&= \operatorname*{arg\,min}_{\mathcal{S} \subseteq \mathcal{P}, |\mathcal{S}|=k} \max_{f \in \mathcal{F}} \left\{ \frac{\max_{p_i \in \mathcal{P} \setminus \mathcal{S}} f(p_i) - \max_{p_j \in \mathcal{S}} f(p_j)}{\max_{p_i \in \mathcal{P} \setminus \mathcal{S}} f(p_i)} \right\}
\end{aligned}
\tag{42}
$$

There are $\binom{|\mathcal{P}|}{k} = \dfrac{|\mathcal{P}|!}{k!(|\mathcal{P}| - k)!}$ size-k subsets of skyline points. Finding the optimal size-k subset from them has been shown to be NP-hard [15]. We use a greedy heuristic to select k skyline points iteratively to form the answer set $\mathcal{S}^*$. In each iteration, the point $p^* \in \mathcal{P}$ is selected such that, for any MUF $f \in \mathcal{F}$, the maximum ratio of difference between $gain(\mathcal{P} \setminus \{p^*\}, f)$ and $f(p^*)$ over $gain(\mathcal{P} \setminus \{p^*\}, f)$ is minimized. Point p^* is added to $\mathcal{S}^*$ and removed from $\mathcal{P}$ before the next point is selected from $\mathcal{P}$. Formally,

$$
p^* = \operatorname*{arg\,min}_{p \in \mathcal{P}} \max_{f \in \mathcal{F}} \left\{ \frac{\max_{p_i \in \mathcal{P} \setminus \{p\}} f(p_i) - f(p)}{\max_{p_i \in \mathcal{P} \setminus \{p\}} f(p_i)} \right\}
\tag{43}
$$

As we can see, Equations 42 and 43 are very similar, except that a size-k subset $\mathcal{S}$ has been replaced by a point p.

To compute point p^*, we need to first rewrite Equation 43. We know that

$$
\max_{p_i \in \mathcal{P} \setminus \{p\}} f(p_i) - f(p) = \max_{p_i \in \mathcal{P} \setminus \{p\}} \{f(p_i) - f(p)\}
\tag{44}
$$

Further,

$$
\frac{\max_{p_i \in \mathcal{P} \setminus \{p\}} \{f(p_i) - f(p)\}}{\max_{p_i \in \mathcal{P} \setminus \{p\}} f(p_i)} = \max_{p_i \in \mathcal{P} \setminus \{p\}} \left\{ \frac{f(p_i) - f(p)}{f(p_i)} \right\}
\tag{45}
$$

Thus,

$$
p^* = \operatorname*{arg\,min}_{p \in \mathcal{P}} \max_{f \in \mathcal{F}} \left\{ \max_{p_i \in \mathcal{P} \setminus \{p\}} \left\{ \frac{f(p_i) - f(p)}{f(p_i)} \right\} \right\}
\tag{46}
$$

The two "max" aggregates in the equation above can be swapped without affecting p^*. They require checking all combinations of points in $\mathcal{P} \setminus \{p\}$ and functions in $\mathcal{F}$. The order of the checking has

no impact on the computation result. Thus,

$$p^* = \underset{p \in \mathcal{P}}{\arg\min} \; \underset{p_i \in \mathcal{P} \setminus \{p\}}{\max} \{ \underset{f \in \mathcal{F}}{\max} \{ \frac{f(p_i) - f(p)}{f(p_i)} \} \} \tag{47}$$

The two aggregates "$\arg\min_{p \in \mathcal{P}}$" and "$\max_{p_i \in \mathcal{P} \setminus \{p\}}$" in this equation can be handled simply by a two-layer loop to examine all the points in $\mathcal{P}$.

MaxDif computation. The only problem remaining is to compute the term $\max_{f \in \mathcal{F}} \{ \frac{f(p_i) - f(p)}{f(p_i)} \}$. This term represents the *max*imum ratio of *dif*ference between the utilities of p_i and p over the utility of p_i for any MUFs in $\mathcal{F}$. We call it the *MaxDif* of p_i over p, and denote it by $maxdif(p_i, p)$.

$$maxdif(p_i, p) = \underset{f \in \mathcal{F}}{\max} \{ \frac{f(p_i) - f(p)}{f(p_i)} \} \tag{48}$$

Without knowing the exact MUFs in $\mathcal{F}$, however, it is infeasible to compute $maxdif(p_i, p)$. We address this problem by computing an upper bound for it instead, which is denoted by $maxdif^*(p_i, p)$:

$$maxdif^*(p_i, p) = \underset{l \in [1..d]}{\max} \ln \frac{p_i.c_l}{p.c_l} \tag{49}$$

We show that $maxdif(p_i, p) \leq maxdif^*(p_i, p)$ holds by considering $\frac{f(p_i) - f(p)}{f(p_i)}$ for any MUF $f \in \mathcal{F}$. Equation 9 in the proof of the maximum regret ratio upper bound in Section 5.1 suggests:

$$f(p_i) - f(p) \leq \left[\sum_{l=1}^{d} \alpha_l (\ln p_i.c_l - \ln p.c_l) \right] \prod_{l=1}^{d} p_i.c_l^{\alpha_l} \tag{50}$$

By definition, $f(p_i) = \prod_{l=1}^{d} p_i.c_l^{\alpha_l} > 0$. Thus,

$$\frac{f(p_i) - f(p)}{f(p_i)} \leq \sum_{l=1}^{d} \alpha_l (\ln p_i.c_l - \ln p.c_l) \tag{51}$$

Since $\ln p_i.c_l - \ln p.c_l \leq \max_{l \in [1..d]} \{ \ln p_i.c_l - \ln p.c_l \}$,

$$\begin{aligned} \frac{f(p_i) - f(p)}{f(p_i)} &\leq \sum_{\gamma=1}^{d} \alpha_\gamma \underset{l \in [1..d]}{\max} \{ \ln p_i.c_l - \ln p.c_l \} \\ &= \sum_{\gamma=1}^{d} \alpha_\gamma \underset{l \in [1..d]}{\max} \ln \frac{p_i.c_l}{p.c_l} \end{aligned} \tag{52}$$

We know that $\sum_{\gamma=1}^{d} \alpha_\gamma \leq 1$. Thus,

$$\frac{f(p_i) - f(p)}{f(p_i)} \leq \underset{l \in [1..d]}{\max} \ln \frac{p_i.c_l}{p.c_l} \tag{53}$$

The right hand side of the inequality is independent of any MUF f. Thus,

$$\begin{aligned} maxdif(p_i, p) &= \underset{f \in \mathcal{F}}{\max} \{ \frac{f(p_i) - f(p)}{f(p_i)} \} \\ &\leq \underset{l \in [1..d]}{\max} \ln \frac{p_i.c_l}{p.c_l} \\ &= maxdif^*(p_i, p) \end{aligned} \tag{54}$$

We can now replace $\max_{f \in \mathcal{F}}\{\frac{f(p_i)-f(p)}{f(p_i)}\}$ with $maxdif^*(p_i, p)$ in Equation 47 for computing p^*:

$$p^* = \underset{p \in \mathcal{P}}{\arg\min} \ \max_{p_i \in \mathcal{P} \setminus \{p\}} \{maxdif^*(p_i, p)\} \tag{55}$$

ALGORITHM 3: MaxDif

Input: $\mathcal{D} = \{p_1, p_2, ..., p_n\}$: a d-dimensional database; k: the size of the answer set.
Output: $\mathcal{S}^*$: a size-k subset of $\mathcal{D}$.

1   Compute skyline points and store them in $\mathcal{P}$;
2   $\mathcal{S}^* \leftarrow \emptyset$;
3   **for** $i = 1, 2, ..., d$ **do**
4   Find $p_i^* \in \mathcal{P}$ which has the largest utility $p_i^*.c_i$ in dimension i;
5      $\mathcal{S}^* \leftarrow \mathcal{S} \cup \{p_i^*\}$;
6      $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_i^*\}$;
7   **while** $|\mathcal{S}| < k$ *and* $|\mathcal{P}| > 0$ **do**
8      $p^* \leftarrow FindMinMaxDifPoint(\mathcal{P})$;
9      $\mathcal{S}^* \leftarrow \mathcal{S}^* \cup \{p^*\}$;
10     $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p^*\}$;
11   $\mathcal{S}^* \leftarrow \mathcal{S}^* \cup \{k - |\mathcal{S}^*| \text{ random points in } \mathcal{D} \setminus \mathcal{S}^*\}$;
12   **return** $\mathcal{S}^*$;

ALGORITHM 4: FindMinMaxDifPoint

Input: $\mathcal{P}$: a skyline point set.
Output: p^*: a MinMaxDif point.

1   $min \leftarrow +\infty$;
2   **for** $p \in \mathcal{P}$ **do**
3      $p.md \leftarrow -\infty$;
4   **for** $p_i \in \mathcal{P} \setminus \{p\}$ **do**
5   **if** $maxdif^*(p_i, p) > p.md$ **then**
6            $p.md \leftarrow maxdif^*(p_i, p)$;
7   **if** $p.md < min$ **then**
8         $p^* \leftarrow p$;
9         $min \leftarrow p.md$;
10   **return** p^*;

The algorithm. We name our query algorithm after the MaxDif metric, i.e., the *MaxDif* algorithm. As summarized in Algorithm 3, the MaxDif algorithm first computes all the skyline points and stores them in a set $\mathcal{P}$ (Line 1). This can be done by an existing skyline query algorithm (e.g., [34, 43]) and is not the focus of our study. A straightforward algorithm is a three-layer nested loop over all the data points and dimensions to look for any non-dominated points. Then, following MinVar, the algorithm adds the skyline point with the largest coordinate value in each dimension to the answer set $\mathcal{S}^*$ (Lines 3 to 6). This serves to cover the extreme case where the MUFs have a weight of 1 in some dimension and 0's in all other dimensions. The algorithm proceeds to add point p^* as defined by Equation 55 into $\mathcal{S}^*$ iteratively (Lines 7 to 10). We call point p^* a *MinMaxDif* point and use a sub-algorithm *FindMinMaxDifPoint* to compute it. Each MinMaxDif point added

to $\mathcal{S}^*$ is removed from $\mathcal{P}$, and the loop terminates when $\mathcal{S}^*$ has k points or $\mathcal{P}$ becomes empty. If the loop terminates and $\mathcal{S}^*$ does not have k points, we fill up $\mathcal{S}^*$ with randomly selected points from $\mathcal{D}$ (Line 11). Then, the set $\mathcal{S}^*$ is returned (Line 12).

The FindMinMaxPoint algorithm loops through the skyline points in $\mathcal{P}$. For every skyline point p, we compute the MaxDif value of every skyline point $p_i \in \mathcal{P} \setminus \{p\}$ over p. The largest MaxDif value is recorded as $p.md$. The skyline point p^* with the smallest $p^*.md$ value is returned as the MinMaxDif point. We summarize this process as Algorithm 4.

Algorithm complexity. The MaxDif algorithm needs to compute and store the set of skyline points $\mathcal{P}$. Leaving out the space for storing the input data, the space complexity of the algorithm is $O(|\mathcal{P}| + k)$. In the worst case, $\mathcal{P}$ has the same size as the entire database, and the worst-case space complexity is $O(n + k)$.

Computing the set $\mathcal{P}$ with a straightforward three-layer nested loop takes $O(n^2 d)$ time (Line 1). There are more advanced skyline query algorithms [34, 43] but these are beyond the scope of the paper. Computing the maximum skyline points in the d dimensions (Lines 3 to 6) takes $O(|\mathcal{P}|d)$ time. The MaxDif algorithm then calls FindMinMaxDifPoint for k times (Lines 7 to 10). Find-MinMaxDifPoint makes a two-layer nested loop pass over $\mathcal{P}$ to compute the MinMaxDif point, where computing the MaxDif value between two points needs to loop through d dimensions. Thus, the time complexity for the k function calls is $O(k|\mathcal{P}|^2 d)$. The overall time complexity is $O(n^2 d + |\mathcal{P}|d + k|\mathcal{P}|^2 d) = O(n^2 d + k|\mathcal{P}|^2 d)$. The worst-case time complexity is $O(kn^2 d)$.

Discussion. The set $\mathcal{S}^*$ returned by the MaxDif algorithm is a heuristic choice to approach the theoretically optimal answer set defined by Equation 42. It is based on a bound $maxdif^*(p_i, p)$ and aims to minimize the maximum regret ratio over the entire set of utility functions in $\mathcal{F}$ which is infinite. In the experiments, we can only test a finite subset $\mathcal{F}' \subset \mathcal{F}$ of utility functions. The set $\mathcal{S}^*$ generated to minimize the maximum regret ratio over $\mathcal{F}$ may not minimize that over $\mathcal{F}'$. The reason is as follows. Given a finite set of utility functions $\mathcal{F}' \subset \mathcal{F}$, there may be a size-$k$ subset $\mathcal{S}$ that contains all the skyline points that maximize the gains over $\mathcal{F}'$, and hence minimizes the maximum regret ratio over $\mathcal{F}'$. This subset $\mathcal{S}$, however, may not minimize the maximum regret ratio over the set in $\mathcal{F} \setminus \mathcal{F}'$. The MaxDif algorithm, which considers both sets of $\mathcal{F}'$ and $\mathcal{F} \setminus \mathcal{F}'$ together, may return a different set $\mathcal{S}^*$. Since $\mathcal{S}^*$ is different from $\mathcal{S}$, and $\mathcal{S}$ minimizes the maximum regret ratio over $\mathcal{F}'$, $\mathcal{S}^*$ may not minimize the maximum regret ratio over $\mathcal{F}'$. Regardless, as the experiments in the next section show, $\mathcal{S}^*$ still has consistently small maximum regret ratios over the set $\mathcal{F}'$. Further, a larger $\mathcal{F}'$ may cause only a small increase in the maximum regret ratio of $\mathcal{S}^*$ as the MaxDif algorithm already considers the infinite set $\mathcal{F}$ when generating $\mathcal{S}^*$.

8 EXPERIMENTS

We evaluate the empirical performance of the two proposed algorithms MinVar and MaxDif.

8.1 Settings

The algorithms are implemented in C++, and the experiments are run on a computer running the OS X 10.12 operating system with a 64-bit 2.7 GHz Intel® Quad-Core$^{(TM)}$ i7 CPU and 16 GB RAM.

Both real and synthetic data sets are used in the experiments. The real data sets used are the *NBA*[1], the *Stocks*[2], and the *Weather*[3] data sets. NBA and Stocks have been used in previous studies on k-regret queries [36, 37]. After filtering out data points with null fields, we obtain 20,640 data points of 7 dimensions in the NBA data set, including 88 skyline points. The Stocks data set

---

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

[2]http://pages.swcp.com/stocks

[3]https://crudata.uea.ac.uk/cru/data/hrg/tmc/

Table 3. Experimental Settings

Parameter	Values	Default
Utility function	AUF, CES, Cobb-Douglas	-
Data set	NBA, Stocks, Weather,	Random
	Anti-correlated, Correlated, Random	
Number of utility functions	10k, 50k, 100k, 500k, 1000k	10k
n	10k, 50k, 100k, 500k, 1000k	100k
d	2, 4, 5, 6, 8, 10, 12	5
k	10, 20, 30, 40, 50	20

contains 122,574 data points of 5 dimensions, including 39 skyline points. Weather is a larger data set, which contains 566,262 data points of 13 dimensions, including 7,947 skyline points. The 13 dimensions of each data point represent the elevation and 12 monthly mean temperature values of a weather observation point. We use absolute values of the temperature data since we assume positive utilities. A value of zero is also allowed as it does not affect the correctness of any algorithm tested.

The synthetic data sets are generated using the *anti-correlated* data set generator [8], which is a popular data generator used in skyline query studies [34, 35, 44]. This data generator can generate points with correlated, anti-correlated, and random coordinate values in different dimensions. Data points with correlated coordinate values have similar coordinate values in different dimensions. This means that a data point p with the largest coordinate value in one dimension is likely to have large coordinate values in other dimensions as well, and p tends to dominate most other points. Only a small number of points like p are needed to dominate all other points in a data set. Such a data set has only a small number of skyline points. In contrast, data points with anti-correlated coordinate values have large coordinate values in some dimensions while small coordinate values in other dimensions. They tend not to dominate or be dominated by other points, which makes them likely skyline points. More skyline points exist in such a data set. Data points with random coordinate values have independent coordinate values in different dimensions. A data set of such points has a relatively moderate number of skyline points. We generate *Correlated*, *Anti-correlated*, and *Random* data sets with these different type of points, respectively.

We vary the data set cardinality n from 10,000 to 1,000,000, the dimensionality d from 2 to 12, and the query parameter k from 10 to 50. Table 3 summarizes the parameters and their values. By default, we use a Random data set with 100,000 data points of 5 dimensions ($d = 5$), and $k = 20$. Note that both a proposed algorithm MinVar and a baseline algorithm MinWidth [24] divide the data space into $t^{(d-1)}$ buckets and select a single point from each bucket to be added into the query answer, where $t = \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor$. With a default d value of 5, $t = 1$ (i.e., the entire data space is considered as a bucket) for k up to 19. For such small values of k, the performance difference between the two algorithms is minimum, which can be seen from the experimental results in Section 8.2 where the value of k is varied. To observe the performance difference of the two algorithms, we use a default value of $k = 20$. We argue that a representative subset of 20 data points is still manageable by users.

We use three families of utility functions – the generalized Cobb-Douglas functions (denoted by "**Cobb-Douglas**"), the CES functions (denoted by "**CES**"), and the linear summation functions (denoted by "**AUF**"). The involvement of AUFs in the experiments serves to showcase the applicability

of the proposed MinVar and MaxDif algorithms over a wider range of utility functions. MinVar has the same maximum regret ratio bounds for AUFs as those derived by Nanongkai et al. [32], since MinVar also uses parameter t to bound the space partitioning, the values of which are no smaller than those used by Nanongkai et al. MaxDif can also handle k-regret queries with CES functions or AUFs, but may produce suboptimal query answers. This is because the optimization function of MaxDif is designed for MUFs which may not minimize the maximum regret ratios for CES functions or AUFs. In each set of experiments, we randomly generate from 10,000 to 1,000,000 sets of parameters $\{\alpha_i | i \in [1..d], \alpha_i \in [0, 1]\}$ for each family of utility functions, where $\sum_{i=1}^{d} \alpha_i = 1$. The CES function has an extra parameter b. We generate random values of b in the range of $[0.1, 0.9]$. By default, we use 10,000 utility functions in each utility function family for the testing. We run the algorithms on the data sets, and report the running time and the maximum regret ratio (denoted by "**MRR**") on the generated utility functions.

Since no algorithms have been proposed in the past for k-regret queries with MUFs, for comparison purposes, we use four baseline algorithms MinWidth, Angle, AreaGreedy, and MaxDom. MinWidth, Angle, and AreaGreedy have been proposed for k-regret queries with CES functions [24]; MaxDom has been proposed for top-k representative skyline queries [28]. We compare the maximum regret ratios of the answer sets generated by these algorithms over Cobb-Douglas functions, CES functions, and AUFs with those of the answer sets generated by the two proposed algorithms. Together we test six algorithms in our experiments.

- *MinVar* is the algorithm proposed in Section 4. We use $inc = 0.1\%n$ to control the number of iterations for which the sub-algorithm FindBreakpoints is run. We find that $itr_{max} = 11$ is sufficient to handle the data sets tested. The results obtained are based on this setting.
- *MaxDif* is the heuristic algorithm proposed in Section 7.
- *MinWidth* [24] is an algorithm with bounded maximum regret ratios for k-regret queries with CES functions.
- *Angle* [24] is a greedy algorithm (with no bounds on the maximum regret ratios) proposed for k-regret queries with CES functions.
- *AreaGreedy* [24] is another greedy algorithm (with no bounds on the maximum regret ratios) proposed for k-regret queries with CES functions.
- *MaxDom* [28] is a greedy algorithm that returns the k representative skyline points which dominate the largest number of other points. For fairness, both MaxDom and MaxDif use the same straightforward algorithm to compute the skyline points, which checks every pair of points and every dimension for dominance (with early termination once dominance is detected). Skyline computation time is included in the running time reported.

8.2 Results

Effect of k. We first test the effect of varying k. Figures 5 to 11 show the result where k is varied from 10 to 50 on the three real data sets and three synthetic data sets. In general, we can see decreasing maximum regret ratios ("MRR" in the figures) as k increases for all algorithms tested. Meanwhile, the algorithm running times increase. These are expected. A larger k means more points are returned and a higher probability of satisfying more utility functions, which also take more time to compute. Note that the computation process and the answer set of each algorithm are independent of the different types of utility functions. Thus, the running times of the algorithms are independent of the utility function type, and we only report them once for each set of experiments. The same answer set of each algorithm, however, may have different maximum regret ratios for different types of utility functions as shown in Fig. 5, which is consistent with our motivating example in Section 1.

Fig. 5. Varying k (NBA)

Performance on real data sets. Figures 5 to 7 show the results on the NBA, Stocks, and Weather data sets. We see that, between the two algorithms MinVar (denoted by '∘' in the figure; same below) and MinWidth ('+') that have bounded maximum regret ratios, the proposed algorithm MinVar has maximum regret ratios that are consistently lower than or equal to those of MinWidth. The advantage is more significant for larger k values (e.g., up to 56% lower for $k = 50$ with Cobb-Douglas functions, cf. Fig. 5a), as this allows more buckets to be created, and MinVar is designed to increase the probability of catching the optimal points by balancing the numbers of points across different buckets. This advantage comes at the expense of a higher running time than that of MinWidth. However, we argue that the running time of MinVar is manageable. As Fig. 7 shows, MinVar can process the Weather data set which has over half a million data points in just 1.3 seconds. We also notice that MinWidth can process this data set in 0.3 seconds while producing the same maximum regret ratios. This shows that MinWidth is a highly competitive baseline algorithm, and it is not easy to outperform this algorithm in terms of the running time.

For the rest of the algorithms which do not have bounded maximum regret ratios, the proposed algorithm MaxDif ('□') has the most consistent performance in terms of the difference from the smallest maximum regret ratio achieved by any algorithm tested. The maximum regret ratios of MaxDif are no more than 0.24 higher than those of any algorithm tested (i.e., for Cobb-Douglas functions on the Stocks data set where $k = 10$). MaxDif also has the smallest maximum regret ratios for the NBA data set where $k \geq 30$ and for the Stocks data set where $k \geq 40$. For the other heuristic algorithms, AreaGreedy ('△') has maximum regret ratios that can be 0.57 higher than that of MaxDif and MaxDom ('×') on the Stocks data set where $k \geq 40$, although it also has the smallest maximum regret ratios on the Weather data set and for a few k values on the NBA data set. Similarly, Angle ('◇') and MaxDom also suffer on the Stocks data set, where their maximum

(a) MRR (Cobb-Douglas)

(b) MRR (CES)

(c) MRR (AUF)

(d) Running time

Fig. 6. Varying k (Stocks)

(a) MRR (Cobb-Douglas)

(b) MRR (CES)

(c) MRR (AUF)

(d) Running time

Fig. 7. Varying k (Weather)

regret ratios are up to 0.99 and 0.36 higher than those of MaxDif, respectively. The advantage of MaxDif is attributed to its point selection strategy. MaxDif adds the skyline point that differs the least from any of the unselected skyline points into the answer set. By doing so, even if an unselected skyline point turns out to be optimal for some utility function, the optimality of the points in the answers set is not too much worse than that of the optimal point. This point selection strategy is also the reason why MaxDif may have larger maximum regret ratios than those of the other proposed algorithm MinVar in some cases (e.g., $k = 20$ or 30 on the Stocks data set with Cobb-Douglas or CES functions), where MinVar may happen to select an optimal skyline point or some non-skyline point close to it, while MaxDif may select a non-optimal skyline point.

In terms of the running time, all the heuristic algorithms are slower than MinWidth. AreaGreedy and Angle require multiple scans over the data set to find the points that bound the maximum area and are the maximum towards different angles, respectively. MaxDif has a time complexity that is quadratic to the number of skyline points. Its running time is low when the number of skyline points is small, e.g., below 0.15 seconds for the NBA and Stocks data sets where the number of skyline points is below 100. This running time could become higher than those of AreaGreedy and Angle when the number of skyline points becomes larger, e.g., over 500 seconds for the Weather data set where the number of skyline points is over 7,000. MaxDom checks every skyline point against all the data points and its running times are constantly higher than those of MaxDif. It may take over 1,000 seconds to run on the Weather data set.

Performance on skyline points. The proposed algorithm MinVar and three baseline algorithms MinWidth, Angle, and AreaGreedy are designed to run on the entire data set, but they can also run on the set of of skyline points. We examine the performance of these four algorithms on skyline points in this set of experiments. Note that, for MinVar and MinWidth which have bounded maximum regret ratios, running them on skyline points does not impact their maximum regret ratio bounds for the following reason. We have shown that, for any MUF, its corresponding optimal point must be a skyline point. For any skyline point, a point is selected into the query answer from the bucket of the skyline point by MinVar and MinWidth. Further, the bucket size for the set of skyline points $\mathcal{P}$ must not exceed that for the entire data set $\mathcal{D}$, since $\mathcal{P} \subseteq \mathcal{D}$. Thus, the bucket size based maximum regret ratio bounds still hold. We omit the full proof since it is straightforward.

Figure 8 shows the maximum regret ratios for Cobb-Douglas utility functions and running times when the algorithms are run on the skyline points of the three real datasets. We omit the MRR figures for CES and AUF utility functions, since the comparative performance of the algorithms running on skyline points with their counterparts running on the entire data set is similar to those shown in the MRR figures for Cobb-Douglas utility functions in Figs. 5 to 8.

In terms of the maximum regret ratio, we see that MinVar ('○') and MinWidth ('+') benefit the most from running on the skyline points. Their maximum regret ratios are now either the smallest (Fig. 8c) or very close to the smallest maximum regret ratios produced by any algorithm tested (Fig. 8a and Fig. 8e). Among these two algorithms, the proposed algorithm MinVar again obtains smaller maximum regret ratios because it can adaptively shrink the bucket size which leads to smaller regret ratios. Angle ('◇') and AreaGreedy ('△') benefit less. Their maximum regret ratios are still less stable across different data sets than those of the proposed algorithm MaxDif ('□'). They have the largest maximum regret ratios on the Stocks data set where $k \leq 30$ (Fig. 8c), although AreaGreedy also has the smallest maximum regret ratios on the Weather data set (Fig. 8e). Focusing on the two proposed algorithms MaxDif and MinVar, MaxDif still produces smaller maximum regret ratios on the NBA data set where $20 \leq k \leq 40$ (Fig. 8a), while MinVar produces no larger maximum regret ratios in all other cases. This suggests that, while MaxDif is still an effective

(a) MRR (Cobb-Douglas) - NBA

(b) Running time - NBA

(c) MRR (Cobb-Douglas) - Stocks

(d) Running time - Stocks

(e) MRR (Cobb-Douglas) - Weather

(f) Running time - Weather

Fig. 8. Varying k (skyline points of real data sets)

heuristic based algorithm, MinVar could be very competitive if it could be run on the skyline points.

The smaller maximum regret ratios come at the cost of larger algorithm running times. Now all algorithms except MaxDom ('×', which needs to run on the entire data set) have roughly the same running time, which is dominated by the time to compute the skyline points. For example, on the Weather data set (Fig. 8f), the running times of all algorithms except MaxDom are at about 560 seconds, among which 559.31 seconds are taken to compute the skyline points. Once the skyline points are computed, even MaxDif which has a quadratic running time on the number of skyline points only takes 64.88 seconds (i.e., a total of 624.19 seconds) to compute the query answer, whereas MinVar only takes 0.28 seconds to compute the query answer ($k = 50$).

In application scenarios where the data set is dynamic (e.g., online shopping services where new products keep arriving) and precomputing the skyline points is infeasible, the high cost of

computing the skyline points would prevent running the algorithms on them. Thus, in the following experiments, for MinVar, MinWidth, Angle, and AreaGreedy which do not have to run on the skyline points, we focus on their performance over the entire dataset.

Performance on synthetic data sets. Similar performance patterns are observed on the three synthetic data sets, as shown in Figs. 9 to 11. MinVar ('○') has maximum regret ratios that are smaller than or equal to those of MinWidth ('+') in almost all cases, except for when $k = 50$ on the Anti-correlated data set with CES functions (Fig. 10b). The advantage in the maximum regret ratio is most significant on the Correlated data set. This can be explained by noting that the Correlated data set has a more skewed distribution, and MinVar is designed to obtain more balanced buckets for skewed data. The running times of MinVar are again larger than those of MinWidth, but are within 0.2 seconds, which is still reasonably small. Following studies [30, 41] on users' tolerable waiting times, we consider 2 seconds as the threshold of a "reasonable" running time. MaxDif ('□') has the smallest maximum regret ratios (up to 0.019) and running times (up to 0.030 seconds) on the Correlated data set except for when $k = 10$ (cf. Fig. 9). This data set has 26 skyline points, which can be processed by MaxDif with a high efficiency. MaxDom ('×') has the smallest maximum regret ratios on the Anti-correlated (up to 0.238) and Random data sets (up to 0.146), but its running times are also the largest (up to 569.3 and 9.8 seconds on the two data sets, respectively). MaxDif has the second smallest maximum regret ratios on these two data sets (up to 0.326 and 0.224 which are 37.0% and 53.4% higher, respectively) with the exception of AUFs on the Anti-correlated data set. Note that the optimization goal of MaxDif when selecting the data points is designed for MUFs, which may not be optimal for CES functions or AUFs. The running times of MaxDif are lower than those of MaxDom as well. It takes 1.7 seconds (82.7% lower) to process the Random data set (1,068 skyline points) and 92.7 seconds (83.7% lower) to process the Anti-correlated data set (12,710 skyline points) when $k = 50$. For the Anti-correlated data set which has a large number of skyline points, MinVar ('○') and AreaGready ('△') offer the most competitive performance. Their maximum regret ratios are close to those of MaxDif, while their running times are within 0.1 seconds.

Effect of d. Next, we test the algorithm scalability over the number of data dimensions. We vary the number of dimensions d from 2 to 12 with synthetic data. Figure 12 shows the result on Random data sets. As d increases, the maximum regret ratios increase overall for all the algorithms. This confirms the bounds obtained and is expected, since the difference in the utilities of the optimal points in $\mathcal{D}$ and $\mathcal{S}$ accumulates when there are more dimensions. The increase in the maximum regret ratio is not linear to the increase in d, and there are fluctuations, e.g., the maximum regret ratios of MinWidth ('+') drops when d increases from 8 to 10 for the CES functions. This is because adding extra dimensions changes the data distribution, the optimal data points, and the data points selected into the answer set. Such changes may allow a lower maximum regret ratio for a higher dimensional data set, e.g., a point with large utilities in the extra dimensions is selected into the answer set, which compensates the lower utilities in the previous dimensions.

Similar to Fig. 11, on Random data sets, MaxDom ('×') produces the smallest maximum regret ratios, while those of MaxDif ('□') are the closest. The running times of these two algorithms become too high when d reaches 8 (33.9 and 137.0 seconds, respectively). For $d \geq 8$, AreaGreedy becomes the most competitive algorithm, as its running time is within 0.1 seconds while its maximum regret ratios are close to those of MaxDif. MinVar ('○') has larger maximum regret ratios than those of the heuristic algorithms AreaGreedy and (in some cases) Angle ('◇'), but again its maximum regret ratios do not exceed those of MinWidth ('+'). We note that the advantage of MinVar over MinWidth in terms of the maximum regret ratio is less significant on this set of experiments with random data, as MinVar is designed for handling more skewed data.

(a) MRR (Cobb-Douglas)　　　　　　　　　(b) MRR (CES)

(c) MRR (AUF)　　　　　　　　　(d) Running time

Fig. 9.　Varying k (Correlated)

(a) MRR (Cobb-Douglas)　　　　　　　　　(b) MRR (CES)

(c) MRR (AUF)　　　　　　　　　(d) Running time

Fig. 10.　Varying k (Anti-correlated)

Fig. 11. Varying k (Random)

(a) MRR (Cobb-Douglas)

(b) MRR (CES)

(c) MRR (AUF)

(d) Running time

Fig. 12. Varying d (Random)

(a) MRR (Cobb-Douglas)

(b) MRR (CES)

(c) MRR (AUF)

(d) Running time

Fig. 13. Varying n (Random)

Similar patterns are observed when the algorithms are run on Correlated and Anti-correlated data sets with different numbers of dimensions. The comparative performance of the algorithms is similar to those shown in Figs. 9 and 10. On Correlated data sets where the numbers of skyline points are small, MaxDif and MaxDom are both fast and produce small maximum regret ratios. On Anti-correlated data sets where the numbers of skyline points are larger, MinVar and AreaGreeday both have relatively small maximum regret ratios, and they can terminate in a realistic time. We omit the figures for conciseness. The same applies for the rest of the experiments.

Effect of n. We further vary the data set cardinality from 10,000 to 1,000,000. Figure 13 shows the result on Random data sets. The comparative performance of the algorithms is again similar to that shown in Fig. 11. MaxDom ('×') has the smallest maximum regret ratios, while its running time (55.3 seconds) is not competitive for data sets with over half a million data points (1,715 skyline points). MaxDif ('□') again has the second smallest maximum regret ratios (except for the AUFs, for reasons discussed earlier), while it can process 1 million data points (2,090 skyline points) within 35.6 seconds. When the number of data points reaches half a million, AreaGreedy ('△') again becomes a more competitive heuristic algorithm as its maximum regret ratios are close to those of MaxDif while its running times are within the 2-second threshold. As for the two algorithms MinVar ('○') and MinWidth ('+') with bounded maximum regret ratios, MinVar again has consistently non-greater maximum regret ratios, and the running times of both algorithms are within the 2-second threshold as well.

We also observe that, while the running times of all the algorithms increase with the data set cardinality, which is natural, the maximum regret ratios do not show a significant increase. This is because the number of points that yield large utilities for most utility functions increases with

the data set cardinality, which leads to a higher probability of selecting and inserting one of such points into the answer set, and hence may produce a smaller maximum regret ratio.

(a) MRR (Cobb-Douglas)   (b) MRR (CES)

(c) MRR (AUF)

Fig. 14. Varying the number of utility functions (Random)

Effect of the number of utility functions. Figure 14 shows the impact of varying the number of utility functions used in computing the maximum regret ratio from 10,000 to 1,000,000 (we generated more utility functions for this set of experiments). Since the algorithms are independent of any specific utility function, their output and running times do not change when the utility functions change. Only the maximum regret ratios may change. Thus, we only show the maximum regret ratios and omit the running times for this set of experiments.

The maximum regret ratios of the algorithms increase with the number of utility functions. This is because, given the same answer set, when there are more utility functions, the probability of satisfying all the utility functions drops, and hence the maximum regret ratio may increase. We see from the figure that, while MaxDom ('×') has the smallest maximum regret ratio, its maximum regret ratio increases the most significantly with the number of utility functions. This is because MaxDom selects the set of skyline points that dominate the most non-skyline points. It does not consider the difference in the utilities of the selected and unselected skyline points. This difference can be large. When there are more utility functions, it is more likely to have an utility function for which an unselected skyline point has the largest utility, which can cause a large regret ratio for MaxDom. Our MaxDif algorithm, on the other hand, is more robust to a larger number of utility functions (for the Cobb-Douglas utility functions), as it already considers the entire family of infinite multiplicative utility functions when computing the answer set. In particular, for the Cobb-Douglas utility functions, the maximum regret ratios of MaxDif ('□') have been kept steady at around 0.2 and become closer to those of MaxDom as more utility functions are used. For the

CES and AUF functions, the maximum regret ratios of MaxDif and MaxDom do not become closer as fast, because the optimization function of MaxDif may not suit CES or AUF functions.

Fig. 15. Varying itr_m (Random)

Effect of itr_m. Parameter itr_m in MinVar controls the maximum number of iterations to try out different numbers of intervals in each dimension. In Fig. 15, we run MinVar by varying itr_m from 1 to 5 on the Random data set to study the impact of itr_m. We see that the maximum regret ratios decrease as itr_m increases from 1 to 2 but become stable when itr_m increases further. This shows that $t + 1$ intervals (i.e., second iteration) is sufficient to produce k points to fill up the answer set, because the buckets are mostly non-empty for the Random data set. Only two iterations are run even if $itr_m > 2$. Note that when $k = 20$, the algorithm running times are close for different values of itr_m. This is because $t = (k - d + 1)^{\frac{1}{d-1}} = 2$ when $k = 20$, which creates $t^{d-1} = 16$ buckets and yields 16 points in one iteration. Adding in the four points with the largest coordinates in the first four dimensions, a total of 20 points are produced, which are sufficient to answer the query. One iteration is needed even for $itr_m > 1$. Experiments on the other data sets reveal that $itr_m = 11$ is the largest needed for all data sets tested. This supports the high efficiency of the MinVar algorithm.

Effect of inc. Parameter inc in MinVar controls the increment step size of the sub-algorithm FindBreakPoints to optimize the interval size given t. We test the impact of inc by varying it from $0.01\%n$ to $1\%n$. In Fig. 16, we present the results on the Correlated data set, since the data space partitioning strategy of MinVar targets skewed data where inc has the most impact. We see that the maximum regret ratios decrease when inc reaches $0.1\%n$. This is because the decrease in inc leads to buckets with more similar numbers of data points, which increases the probability of the optimal points being selected. However, when inc decreases further, the benefit in the maximum regret ratios diminishes, while the running time keeps increasing as more iterations are needed.

Fig. 16. Varying *inc* (Correlated)

In general, when $inc = 0.1\%n$, the MinVar algorithm achieves the best balance between the maximum regret ratio and the running time, which has been used by the default in the experiments.

Discussion. From the experiments, we see that MaxDom often produces small maximum regret ratios. However, *MaxDom does not have a bound on the maximum regret ratio. Its maximum regret ratios grow as more utility functions are used to evaluate the answer set. Further, the running time of MaxDom is quadratic over the number of data points* when the number of skyline points is close to the number of data points, which soon becomes unrealistic. The two proposed algorithms MaxDif and MinVar, on the other hand, have comparable maximum regret ratios, while their running times are lower. MinVar should be used if a bounded maximum regret ratio is needed. MaxDif does not have a bound but is designed to obtain small maximum regret ratios. It is robust to a large number of utility functions (e.g., 1,000,000). MaxDif can produce query answers within 2 seconds when the number of skyline points is within 1,000 (for 5-dimensional data). MinVar can produce query answers within 1 second when the number of data points is within 1,000,000 (for 5-dimensional data). When the data set characteristics are unknown, in order to obtain the best results, a feasible approach is to run MaxDif and MinVar at the same time. If MaxDif terminates soon enough (e.g., in 2 seconds), its answer is returned. Otherwise, the answer produced by MinVar can be used.

9 CONCLUSIONS

We studied k-regret queries with multiplicative utility functions which we showed to be more expressive in modeling the diminishing marginal rate of substitution than the additive utility functions studied in prior work [32]. We presented two algorithms MinVar and MaxDif for such queries. MinVar produces query answers with a bounded maximum regret ratio. When applied on k-regret queries with Cobb-Douglas functions, MinVar achieved a maximum regret ratio not exceeding

$O(\ln(1 + \frac{1}{k^{\frac{1}{d-1}}}))$; when applied on k-regret queries with Constant Elasticity of Substitution functions, MinVar achieved a maximum regret ratio not exceeding $O(\frac{1}{k^{\frac{1}{d-1}}})$. We also showed that, given an infinite set of data points, the maximum regret ratio of any algorithm for k-regret queries with multiplicative utility functions has a lower bound of $\Omega(\frac{1}{k^2})$. When the data set is finite, the set of all skyline points has a maximum regret ratio of 0, which should be returned as the answer set when the number of skyline points does not exceed k. When there are more than k skyline points, MaxDif computes a size-k subset of skyline points to minimize the maximum regret ratio. We performed extensive experiments using both real and synthetic data. The results showed that the regret ratios produced by MinVar and MaxDif are consistently small: MaxDif suits the case where the number of skyline points is not too large, while MinVar can produce small maximum regret ratios in real-time as the number of skyline points becomes larger (e.g., over 1,000).

Future work involves exploring the behavior of k-regret queries with various other types of multiplicative utility functions and parameter settings such as the product of convex (concave) functions and when $\sum_{i=1}^{d} \alpha_i > 1$. It would also be interesting to see how k-regret queries with a mix of both additive and multiplicative utility functions can be answered with a bounded regret ratio.

ACKNOWLEDGMENTS

REFERENCES

[1] 2018. http://booking.com. (2018). Accessed: 2018-03-30.
[2] 2018. How many products does Amazon.com sell - January 2018. https://www.scrapehero.com/many-products-amazon-sell-january-2018/. (2018). Accessed: 2018-03-30.
[3] A. Amir, A. Efrat, P. Indyk, and H. Samet. 2001. Efficient regular data structures and algorithms for dilation, location, and proximity problems. *Algorithmica* 30, 2 (2001), 164–187.
[4] C. H. Ang, H. Samet, and C. A. Shaffer. 1990. A new region expansion for quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, 7 (1990), 682–686.
[5] Walid G. Aref and Hanan Samet. 1997. Efficient window block retrieval in quadtree-based spatial databases. *Geoinformatica* 1, 1 (1997), 59–91.
[6] A. Asudeh, A. Nazi, N. Zhang, and G. Das. 2017. Efficient computation of regret-ratio minimizing set: A compact maxima representative. In *SIGMOD*. Chicago, USA, 821–834.
[7] E. R. Berndt. 1976. Reconciling alternative estimates of the elasticity of substitution. *The Review of Economics and Statistics* 58, 1 (1976), 59–68.
[8] S. Börzsönyi, D. Kossmann, and K. Stocker. 2001. The skyline operator. In *ICDE*. Heidelberg, Germany, 421–430.
[9] D. Braziunas and C. Boutilier. 2007. Minimax regret based elicitation of generalized additive utilities. In *The 23rd Conference on Uncertainty in Artificial Intelligence*. Vancouver, Canada, 25–32.
[10] 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 *ICDT*. Venice, Italy, 11:1–11:19.
[11] D. Cass. 1965. Optimum growth in an aggregative model of capital accumulation. *The Review of Economic Studies* 32, 3 (1965), 233–240.
[12] D. W. Caves and L. R. Christensen. 1980. Global properties of flexible functional forms. *The American Economic Review* 70, 3 (1980), 422–432.
[13] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. 2006. On high dimensional skylines. In *EDBT*. Munich, Germany, 478–495.
[14] L. Chen, G. Cong, X. Cao, and K.-L. Tan. 2015. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*. Seoul, South Korea, 255–266.

[15] S. Chester, A. Thomo, S Venkatesh, and S. Whitesides. 2014. Computing k-regret minimizing sets. *PVLDB* 7, 5 (2014), 389–400.

[16] C. W. Cobb and P. H. Douglas. 1928. A theory of production. *The American Economic Review* 18, 1 (1928), 139–165.

[17] P. A Diamond. 1965. National debt in a neoclassical growth model. *The American Economic Review* 55, 5 (1965), 1126–1150.

[18] P. A. Diamond, L. J. Helms, and J. A. Mirrlees. 1980. Optimal taxation in a stochastic economy. *Journal of Public Economics* 14, 1 (1980), 1–29.

[19] Claudio Esperança and Hanan Samet. 2002. Experience with SAND-Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing* 13, 2 (2002), 229–255.

[20] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. 2011. Top-k most influential locations selection. In *CIKM*. Glasgow, United Kingdom, 2377–2380.

[21] I. F. Ilyas, G. Beskales, and M. A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys* 40, 4 (2008), 11:1–11:58.

[22] M. N. Katehakis and A. F. Veinott. 1987. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research* 12, 2 (1987), 262–268.

[23] R. L. Keeney and H. Raiffa. 1993. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge University Press, Cambridge, United Kingdom.

[24] T. Kessler Faulkner, W. Brackenbury, and A. Lall. 2015. K-regret queries with nonlinear utilities. *PVLDB* 8, 13 (2015), 2098–2109.

[25] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. 2008. Skyline query processing for incomplete data. In *ICDE*. Cancún, México, 556–565.

[26] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. 2010. Skyline query processing for uncertain data. In *CIKM*. Toronto, Canada, 1293–1296.

[27] K. Ligett and G. Piliouras. 2011. Beating the best Nash without regret. *SIGecom Exchanges* 10, 1 (2011), 23–26.

[28] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. 2007. Selecting stars: The k most representative skyline operator. In *ICDE*. Istanbul, Turkey, 86–95.

[29] E. Miller. 2008. An assessment of CES and Cobb-Douglas production functions. *Congressional Budget Office* (2008).

[30] Robert B. Miller. 1968. Response time in man-computer conversational transactions. In *Proceedings of AFIPS Fall Joint Computer Conference*. 267–277.

[31] D. Nanongkai, A. Lall, A. D. Sarma, and K. Makino. 2012. Interactive regret minimization. In *SIGMOD*. Scottsdale, USA, 109–120.

[32] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. 2010. Regret-minimizing representative databases. *PVLDB* 3, 1-2 (2010), 1114–1124.

[33] S. Nutanong, E. H. Jacox, and H. Samet. 2011. An incremental Hausdorff distance calculation algorithm. *PVLDB* 4, 8 (2011), 506–517.

[34] D. Papadias, Y. Tao, G. Fu, and B. Seeger. 2003. An optimal and progressive algorithm for skyline queries. In *SIGMOD*. San Diego, USA, 467–478.

[35] J. Pei, B. Jiang, X. Lin, and Y. Yuan. 2007. Probabilistic skylines on uncertain data. In *VLDB*. Vienna, Austria, 15–26.

[36] P. Peng and R. C.-W. Wong. 2014. Geometry approach for k-regret query. In *ICDE*. Chicago, USA, 772–783.

[37] P. Peng and R. C.-W. Wong. 2015. K-hit query: top-k query with probabilistic utility function. In *SIGMOD*. Melbourne, Australia, 577–592.

[38] H. Samet. 2006. *Foundations of multidimensional and metric data structures*. Morgan-Kaufmann, San Francisco. (Translated to Chinese ISBN 978-7-302-22784-7).

[39] Hanan Samet, Houman Alborzi, František Brabec, Claudio Esperança, Gísli R. Hjaltason, Frank Morgan, and Egemen Tanin. 2003. Use of the SAND spatial browser for digital government applications. *Commun. ACM* 46, 1 (2003), 61–64.

[40] H. Samet, C. A. Shaffer, R. C. Nelson, Y. G. Huang, and A. Rosenfeld. 1987. Recent developments in linear quadtree-based geographic information systems. *Image Vision Comput.* 5, 3 (1987), 187–197.

[41] B. Shneiderman. 1984. Response time and display rate in human performance with computers. *Comput. Surveys* 16, 3 (1984), 265–285.

[42] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. 2008. Probabilistic top-k and ranking-aggregate queries. *ACM Transactions on Database Systems* 33, 3 (2008), 13:1–13:54.

[43] K.-L. Tan, P.-K. Eng, and B. C. Ooi. 2001. Efficient progressive skyline computation. In *VLDB*. Roma, Italy, 301–310.

[44] Y. Tao, L. Ding, X. Lin, and J. Pei. 2009. Distance-based representative skyline. In *ICDE*. Shanghai, China, 892–903.

[45] H. Uzawa. 1962. Production functions with constant elasticities of substitution. *The Review of Economic Studies* 29, 4 (1962), 291–299.

[46] H. R. Varian. 1992. *Microeconomic analysis*. Norton & Company, New York, USA.

[47] A. D. Vîlcu and G. E. Vîlcu. 2011. On some geometric properties of the generalized CES production functions. *Applied Mathematics and Computation* 218, 1 (2011), 124–129.

[48] G. E. Vîlcu. 2011. A geometric perspective on the generalized Cobb-Douglas production functions. *Applied Mathematics Letters* 24, 5 (2011), 777–783.

[49] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. 2012. Joint top-k spatial keyword query processing. *IEEE Transactions on Knowledge and Data Engineering* 24, 10 (2012), 1889–1903.

[50] T. Xia, D. Zhang, and Y. Tao. 2008. On skylining with flexible dominance relation. In *ICDE*. Cancún, México, 1397–1399.

[51] A. Zabalza. 1983. The Ces utility function, non-linear budget constraints and labour supply. Results on female participation and hours. *The Economic Journal* 93, 370 (1983), 312–330.

[52] S. Zeighami and R. C.-W. Wong. 2016. Minimizing average regret ratio in database. In *SIGMOD*. San Francisco, USA, 2265–2266.