Regret-Minimizing Representative Databases

Danupon Nanongkai Atish Das Sarma Ashwin Lall
Richard J. Lipton Jun Xu *
Georgia Institute of Technology
{danupon, atish, alall, rjl, jx}@cc.gatech.edu

ABSTRACT

We propose the k-representative regret minimization query (k-regret) as an operation to support multi-criteria decision making. Like top-k, the k-regret query assumes that users have some utility or scoring functions; however, it never asks the users to provide such functions. Like skyline, it filters out a set of interesting points from a potentially large database based on the users' criteria; however, it never overwhelms the users by outputting too many tuples.

In particular, for any number k and any class of utility functions, the k-regret query outputs k tuples from the database and tries to minimize the *maximum regret ratio*. This captures how disappointed a user could be had she seen k representative tuples instead of the whole database. We focus on the class of linear utility functions, which is widely applicable.

The first challenge of this approach is that it is not clear if the maximum regret ratio would be small, or even bounded. We answer this question affirmatively. Theoretically, we prove that the maximum regret ratio can be bounded and this bound is independent of the database size. Moreover, our extensive experiments on real and synthetic datasets suggest that in practice the maximum regret ratio is reasonably small. Additionally, algorithms developed in this paper are practical as they run in linear time in the size of the database and the experiments show that their running time is small when they run on top of the skyline operation which means that these algorithm could be integrated into current database systems.

1. INTRODUCTION

Extracting a few tuples from the database to support multi-criteria decision making is an important functionality for database systems. This is important in many application domains where the end-users are more interested in the most important query answers in the potentially huge answer space. Top-k [17] and skyline [5, 13] queries are two well-studied tools in such settings. Consider the following example.

Alice is looking for a car that has a high miles per gallon (MPG) and a high horse power (HP). However, the horsepower comes at the expense of the fuel economy. How can a car dealer's database system best assist Alice in trading off between these two goals?

One approach is the *top-k* operator (see [17] for a recent survey). This approach is based on a widely-accepted assumption that utility functions exist. It allows the users to input their utility functions. In particular, for the case of linear utility functions, we could use a well-developed system such as PREFER [16] and employ sophisticated techniques such as ONION [8] and Ranked Join Indices [28] to assist the customers. For example, in this case, Alice could indicate that she gives weights 70% to the MPG and 30% to the HP. However, she does not always know what weights she wants to give to each criterion. Moreover, it might be too much of an effort for her to input all these weights.

An alternative operator that helps in avoiding this problem is the *skyline* operator proposed by Börzsönyi et al. [5]. (For further details, see, e.g., [13] and references therein.) This operator asks the customers only for the sets of criteria they want and outputs *everything* that the users might be interested in. For example, with this operator, Alice can specify that she wants to maximize the MPG and the HP. Then, the operator will output *all* the cars based on these criteria. Some car may not be shown if there is another car with more MPG and HP. After this, Alice can either go through the list of these "skyline cars" and find the car that she likes the most, or she can use this list to get the "big picture" and then apply other queries such as top-k to get more specific results (as envisioned by the founders of skyline operator in their paper [5]). However, the size of the skyline is usually too large to even get the big picture of the database, let alone the use of it as a stand alone operator. Theoretically, the size of the skyline could be arbitrarily large even when there are only two criteria involved. Moreover, when there are many criteria involved, the skyline size grows exponentially in the number of criteria even when the values are randomly generated [3, 12].

In this paper, we consider the setting where users have unknown utility functions. Given a list of k tuples, we say that a user is x% happy with the list if the utility she obtains from the best tuple in this list is at least x% of the utility

*Supported in part by NSF grant CNS-0905169, funded under the American Recovery and Reinvestment Act of 2009 (Public Law 111-5), and by NSF grant CNS-0716423.

Proceedings of the VLDB Endowment, Vol. 3, No. 1

she obtains from the best tuple in the whole database.

We attempt for an operator that has features from both top-k and skyline. That is, an operator that outputs a small set of k tuples without asking for users' utility functions. Since it is impossible to display every user's optimal tuple, we instead ask the following question.

> Does there exist one set of k tuples that makes *every* user at least x% happy?

In this paper, we answer this question affirmatively for linear utility functions. Even more surprisingly, the happiness guarantee is independent of the size of the database.

In particular, we propose the k-representative regret minimization operator (k-regret), as follows. Define the regret ratio of a user to be one minus the happiness of the user's utility function for the k tuples. This operator aims at finding k tuples to minimize the maximum regret ratio, where the maximum is taken over a (possibly infinite) class of functions. Our theoretical and experimental results for the class of non-decreasing linear utility functions are stated below.

- We prove that, in the worst case, the maximum regret ratio is at most
$$\frac{d-1}{(k-d+1)^{\frac{1}{d-1}}+d-1},$$
where d is the number of criteria.

  In particular, for $d = 2$, this corresponds to a maximum regret ratio of $\frac{1}{k}$. Therefore, with $k = 10$ the bound is at most 10% (i.e., everyone is 90% happy). Observe that the regret ratio guarantee is independent of the size of the database. To complement this, for $d = 2$, we show a theoretical lower bound on the maximum regret to be $\Omega(1/k^2)$.

- Our extensive experiments on many datasets and algorithms suggest that, in practice, the regret ratio might be much lower than the worst case guarantee. For example, in the case of two criteria we found that the regret ratio is at most 1% for all the datasets we experimented on with just $k = 10$. Moreover, even when we increase the number of criteria to six, the regret ratio is still at most 16% in all the datasets when $k = 20$.

We show that the k-regret operator has the following nice properties. First, it is *scale invariant*. That is, even if we multiply values in some attribute with a constant, k-regret still outputs the same set of tuples. For example, whether the cars' efficiencies are in miles per gallon or kilometers per liter, k-regret's output is the same.

Secondly, the k-regret operator is *stable*. Informally, this means that adding unimportant cars (i.e., cars no user will be interested in) to the database does not change the solution of k-regret. This avoids, e.g., car companies from adding cars to manipulate the database in the hope that their main products will be shown by the k-regret operator.

We also note that the the algorithms developed in this paper run in time linear in n. Moreover, our experiments show that their running time is small when they are run on top of the skyline operator. This means that these algorithms can be immediately integrated into a system where skyline operator is available, while it is possible to completely remove skyline operator from the system in the future when faster algorithms for the k-regret operator, which might be faster than computing skyline, are available.

Organization: In Section 2 we discuss previous work that is most relevant to this paper. We formally define the problem that we solve and highlight some properties of this definition in Section 3. We give worst case upper and lower bounds in Section 4. In Section 5, we develop a heuristic which seems to work well in practice. We present experimental results in Section 6. We discuss possible future work and conclude in Section 7.

2. RELATED WORK

Motivated by the deficiencies of top-k and skyline discussed above, there are many variations of top-k and skyline proposed recently to assist multi-criteria decision making.

Lin et al. [22] and Tao et al. [27] consider finding k skyline tuples that best represent the contour of the entire skyline, called k representative skyline. [22] propose finding a set of k skyline tuples that dominate the most number of tuples. This approach is scale invariant but not stable. [27] illustrates that the approach of [22] could lead to an undesirable solution. This is essentially because of the fact that it is unstable. They propose an alternative distance-based solution which is to solve the *k-center* problem on the skyline tuples. This approach is stable in the sense that adding a non-skyline tuple will not change the solution but it is not scale invariant since their definition is based on distances.

Yiu and Mamoulis [24, 31] propose the *top-k dominating* query. Briefly, this query is the top-k query with scoring function of each tuple as the number of tuples it dominates. The authors argue that this query has the features of both top-k and skyline; i.e., the output size can be controlled, and no scoring functions need to be specified by the users. Further, it is scale invariant. However, like [22], it is not stable. This query has been explored further in the domains of uncertain databases [21, 32] and continuous processing [19].

Goncalves and Vidal [14] propose two operators called top-k skyline select and top-k skyline join. The idea is to let the user specify criteria and a ranking function. The operator then uses the criteria to get skyline tuples and output the top k skyline tuples based on the ranking function. Therefore, the size of the skyline is controlled. However, the users still have to specify their utility functions.

Several other approaches have been considered for controlling the size of the output. In particular, some of these works control the size of the output without explicitly specifying a bound. Xia et al. [29] propose ϵ-skyline queries where users specify weights on attributes, and use the parameter ϵ to control the size of the output. Mindolin et al. [23] propose *p-skyline* queries which is a framework that allows attributes to have different importance. To avoid asking users for weights explicitly, they offer an alternative approach to discover importance from user feedback. Lee et al. [20] avoid asking users for utility functions by requesting only partial ranking over attributes. Chan et al. [7] introduce the concept of skyline frequency as a way to measure the importance of a point based on the number of subsets of dimensions for which it is in the skyline. A related concept of k-dominance proposed by Chan et al. [6] relaxes the definition of dominance to a point dominating another if it dominates on at least k dimensions. All these approaches are used to reduce the size of the output.

Car	MPG	HP
(p_1) Toyota Prius	51	134
(p_2) Honda Civic Hybrid	40	110
(p_3) Ford Fusion	41	191
(p_4) Nissan Altima Hybrid	35	198
(p_5) Volkswagen Jetta TDI	30	140

Figure 1: Car database

Car	$f_{(0.2,0.8)}$	$f_{(0.4,0.6)}$	$f_{(0.6,0.4)}$	$f_{(0.8,0.2)}$
p_1	117.4	100.8	84.2	67.6
p_2	96	82	68	54
p_3	161	131	101	71
p_4	165.4	132.8	100.2	67.6
p_5	118	96	74	52

Figure 2: Car utilities

3. PROBLEM DEFINITION

We now define the problem considered in this paper formally. The input to our problem is a set of n d-dimensional points over positive reals, denoted by D. This set corresponds to the database of n tuples over d attributes. Additionally, an integer k which specifies the output size is given. The goal is to find a set of tuples, denoted by S, of size at most k that minimizes the *maximum regret ratio*. Informally, S corresponds to the set of representative tuples that will be displayed to the users and the maximum regret ratio measures how bad the users could feel if they have to choose from these k tuples instead of the entire database. We now define this important concept formally before giving the precise problem statement.

3.1 Maximum regret ratio

In order to define this notion, we need to develop the notions of *utility function*, *gain*, *regret*, and *regret ratio* as follows. (Throughout, we use the car database in Figure 1 as an example.) First, users' happiness is measured by unknown utility functions.

Definition 1. **Utility Function.** A utility function f is a mapping $f : \mathbb{R}_+^d \to \mathbb{R}_+$. The utility of a point p for a user with utility function f is $f(p)$.

Each user attempts at getting an item that maximizes her utility. For example, consider the car database in Figure 1. If Alice has utility function $f(p) = 0.2 \cdot MPG + 0.8 \cdot HP$ then her utility on Toyota Prius is $f(p_1) = 0.2 \cdot 51 + 0.8 \cdot 134 = 117.4$. Her utilities on other items are shown in the second column of Figure 2 and thus Nissan Altima Hybrid (p_4) maximizes her utility function.

Next, we use the notion of *gain* to capture the user's utility when she sees only a set of tuples. For a user with a utility function f and a subset of points $S \subseteq D$, the gain of the user is defined to be the maximum utility derived from this subset of points.

Definition 2. **Gain.** Define $gain(S, f) = \max_{p \in S} f(p)$.

For example, using Alice's utility function $f(p) = 0.2 \cdot MPG + 0.8 \cdot HP$, $gain(D, f) = 165.4$ (achieved by p_4) while $gain(\{p_1, p_2\}, f) = 117.4$ (achieved by p_1). This means that if we represent the database by a set of two points $S = \{p_1, p_2\}$ then Alice's utility will be 117.4 while her utility would be as large as 165.4 if she sees the whole database. We say that she has *regret* $r_D(S, f) = 165.4 - 117.4 = 48$ and *regret ratio* $rr_D(S, f) = 48/165.4 = 0.29$. In other words, by looking at two representative points, her happiness is about 71% of what she can get when she goes through the whole database. Regret and regret ratio are defined as follows.

Definition 3. **Regret and regret ratio.** Define regret $r_D(S, f) = gain(D, f) - gain(S, f)$ and regret ratio $rr_D(S, f) = \frac{r_D(S,f)}{gain(D,f)}$.

Notice that regret ratio is in the range $[0, 1]$. When the regret ratio is close to 0, the user is very happy and when the regret ratio is close to 1 the user is very unhappy.

Since we do not know users' utility functions (even users themselves do not know their own utility functions), we assume that every user has some utility function in a broad class. Three examples of classes of utility functions include monotone, linear, and convex functions. Let the class of utility functions being considered be $\mathcal{F}$. We now define the worst possible regret for any user with a utility function in $\mathcal{F}$.[1]

Definition 4. **Maximum Regret Ratio.** $rr_D(S, \mathcal{F}) = \sup_{f \in \mathcal{F}} rr_D(S, f) = \sup_{f \in \mathcal{F}} \frac{\max_{p \in D} f(p) - \max_{p \in S} f(p)}{\max_{p \in D} f(p)}$.

To illustrate this definition, let us consider the car database in Figure 1 and suppose that we know that users' utility functions are in a contrived class $\mathcal{F} = \{ f_{(0.2,0.8)}, f_{(0.4,0.6)}, f_{(0.6,0.4)}, f_{(0.8,0.2)}\}$, where $f_{(x,y)} = x \cdot MPG + y \cdot HP$. If we represent the database by $S = \{p_1, p_2\}$ then the maximum regret ratio is 0.29. To see this, use values in Figure 2 to get the regret ratios $rr_D(S, f_{(0.2,0.8)}) = 0.29$, $rr_D(S, f_{(0.4,0.6)}) = 0.24$, $rr_D(S, f_{(0.6,0.4)}) = 0.17$, and $rr_D(S, f_{(0.8,0.2)}) = 0.05$.

3.2 Problem

Given a database D and the desired output size k, we are interested in handling situations where no information on users is available. In absence of any distributional information on the utility functions, a natural approach is to assume a broad class of functions $\mathcal{F}$ and maximize over the worst (i.e., the maximum regret ratio). This guarantees the happiness of *all* users.

As motivated in Section 1, this approach would be futile if we cannot guarantee that the optimal maximum regret ratio is always small. After all, it is possible that even when we show users half of all tuples in the database, the maximum regret ratio is still 80% which means that there is at least one user who cannot find anything close to what she is interested most in the database. Motivated by this, the main problem in this paper is whether we can keep the maximum regret ratio low.

Problem Definition. Given any integers k, d and n and a class of functions $\mathcal{F}$ of interest, find minimum $\rho(k, d, n, \mathcal{F})$ such that the following holds. For any set D of n d-dimensional points there exists a set of k points $K \subseteq D$ such that $rr_D(K, \mathcal{F}) \leq \rho(k, d, n, \mathcal{F})$.

[1]Recall that $\sup(S)$ denotes the supremum of S. For simplicity, the reader can think of it as the maximum. Since $\mathcal{F}$ is allowed to be an infinite class of functions, we need the supremum as a maximum may not exist.

Throughout this paper, we implicitly assume that k is at least d; if this is not the case, the maximum regret ratio cannot be bounded. In this paper, we focus on the class of linear utility functions, denoted by $\mathcal{L}$.

Definition 5. **Linear Utility Function.** A utility function f is linear if there exist non-negative reals $a_1, \ldots, a_d$ such that $f(p) = \sum_{i=1}^{d} a_i p[i]$ for any d-dimensional point p. Alternatively, a linear utility function can be represented by a vector $v = \langle a_1, \ldots, a_d \rangle$; i.e., $f(p)$ is the dot product $v \cdot p$.

For linear utility functions, it is more natural to think of gain in term of vectors: observe that $gain(S, v) = max_{p \in S} v \cdot p$. All other notions can be similarly defined using v. We now state two desirable properties of our formulation.

3.3 Properties

We now discuss the *scale invariance* and *stability* properties which are defined on any measure and show that the regret ratio measure satisfies both properties. Throughout this section, we let h be a function on any subset S of D. Essentially, $h(S)$ measures how well a set S represents the database.

Scale Invariance. Informally, scale invariant means that rescaling the attribute values (scaling by a positive value to maintain monotonicity), does not change the relative values of the solutions. Formally, we consider two databases $D = \{p_1, p_2, ..., p_n\}$ and $D' = \{p'_1, p'_2, ..., p'_n\}$. We assume that D' is a *rescaling* of D, i.e., there exists reals $\lambda_1, \lambda_2, ..., \lambda_d$ such that $p'_i[j] = \lambda_j p_i[j]$ for all $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, d$. (For example, to convert MPG to kilometers per liter and HP to watts, we use $\lambda_1 = 0.425$ and $\lambda_2 = 750$, since 1 MPG = 0.425 kilometers per liter and 1 HP = 750 watts.) For any set $S \subseteq D$, we define S' to be the corresponding subsets in D', i.e., if $S = \{p_{i_1}, p_{i_2}, \ldots\}$ then $S' = \{p'_{i_1}, p'_{i_2}, \ldots\}$. Finally, we say that h is *scale invariant* if $h(S_1)/h(S_2) = h(S'_1)/h(S'_2)$ for any $S_1, S_2 \subseteq D$.

THEOREM 1. *The function $h(S) = rr_D(S, \mathcal{L})$ is scale invariant.*

We in fact prove a stronger fact: $rr_D(S, \mathcal{L}) = rr_D(S', \mathcal{L})$ for any set S. The formal proof is in Appendix A.

Stability. Informally, we say that a function h is stable if, for any $S \subseteq D$, $h(S)$ is insensitive to adding or deleting *junk* points, i.e., points that are not optimal for any linear utility function. This captures the intuition that a car dealer should not be able to strategically insert cars not liked by anyone into the database to manipulate the solution.

Formally, we say that a point p is a *junk* point if, for any linear utility function h, there is a point $q \neq p$ in D such that $h(p) \leq h(q)$. (This means that p is not desired by any user with a linear utility function.) We say that a function h is *stable on* S if $h(S)$ outputs the same value even after we add any junk point p to D. Finally, we say that h is *stable* if it is stable on any set $S \subseteq D$. The following theorem follows naturally from definitions (see Appendix B).

THEOREM 2. *The function $h(S) = rr_D(S, \mathcal{L})$ is stable.*

4. WORST CASE BOUNDS

In this section, we answer the question from a theoretical standpoint. We show that in the worst case, the maximum regret ratio is at most $\frac{d-1}{\lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor + d - 1}$. (Note that this does not depend on the database size n.) This is done by developing an algorithm called CUBE. (See Section 4.1.) Additionally, we complement this with a lower bound; the optimal maximum regret ratio could be as bad as $\Omega(1/k^2)$. (See Section 4.2.) Simplifying (assuming d is constant), $\rho(k, d, n, \mathcal{L})$ (cf. Section 3) has the following upper and lower bounds:

$$\Omega\left(\frac{1}{k^2}\right) \leq \rho(k, d, n, \mathcal{L}) \leq O\left(\frac{1}{k^{\frac{1}{d-1}}}\right).$$

Algorithm 4.1 CUBE algorithm(D, k)

Input: A database of d-dimensional points $D = \{p_1, p_2, ..., p_n\}$ and an integer k, the desired output size.
Output: A subset of D of size k, denoted by S, that guarantees that $rr_D(S, \mathcal{L}) \leq \frac{d-1}{t+d-1}$ where $t = \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$.

1: For $i = 1, 2, \ldots, d-1$, let p_i^* be the point with maximum value in the i^{th} coordinate, and let c_i be such value; i.e., $p_i^* = \arg\max_{p \in D}(p[i])$ and $c_i = \max_{p \in D}(p[i])$.[2]
2: Let $S = \{p_1^*, p_2^*, ..., p_{d-1}^*\}$.
3: Let $t = \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$.
4: **for** each group of (not necessarily distinct) integers $j_1, j_2, \ldots j_{d-1}$ such that $0 \leq j_1 < t, 0 \leq j_2 < t, \ldots, 0 \leq j_{d-1} < t$ **do**
5: Let $S_{j_1, j_2, \ldots, j_{d-1}} = \{p \in D : j_i \frac{c_i}{t} < p[i] \leq (j_i + 1)\frac{c_i}{t}$ for all $1 \leq i < d\}$.
6: Let $s_{j_1, j_2, \ldots, j_{d-1}}$ be the point in $S_{j_1, j_2, \ldots, j_{d-1}}$ with maximum value in the d^{th} coordinate; i.e., $s_{j_1, j_2, \ldots, j_{d-1}} = \arg\max_{p \in S_{j_1, j_2, \ldots, j_{d-1}}}(p[d])$. Add $s_{j_1, j_2, \ldots, j_{d-1}}$ to S.
7: **end for**
8: **return** S.

4.1 Upper bound

In this section, we develop an algorithm called CUBE (cf. Algorithm 4.1). We then show that for any database D of d-dimensional points and a parameter k, CUBE outputs a set S of size at most k such that

$$rr_D(S, \mathcal{L}) \leq \frac{d-1}{\lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor + d - 1}$$

(cf. Theorem 3). The upper bound of the optimal maximum regret ratio thus follows.

Informally, CUBE does the following. First, it outputs the point with maximum value in each coordinate, except the last coordinate. Then, it divides every dimension, except the last dimension, into t equal-sized intervals, for some choice of t. That is, if the maximum value in the i-th coordinate is c_i then we divide the i-th dimension into t intervals,

$$\left(0, \frac{c_i}{t}\right], \left(\frac{c_i}{t}, 2\frac{c_i}{t}\right], \ldots \left(j\frac{c_i}{t}, (j+1)\frac{c_i}{t}\right], \ldots, \left((t-1)\frac{c_i}{t}, c_i\right].$$

The algorithm then partitions the points into t^{d-1} "buckets" based on which intervals they are in, in each dimension. These buckets are denoted by $S_{j_1, j_2, \ldots, j_{d-1}}$ where $j_1, j_2, \ldots, j_{d-1}$ are integers from 0 to $t-1$ (see formal definition in Algorithm 4.1). Then, CUBE outputs the point in

[2] *argmax* returns the point for which the value of the expression attains its maximum value. *argmin* is defined similarly.

each bucket with highest value in the last coordinate. Notice that CUBE outputs $t^{d-1}+d-1$ points in total ($d-1$ points initially and t^{d-1} points from the buckets). Figure 8 (Appendix) shows an example of CUBE algorithm.

The intuition behind this algorithm is that the points in the same bucket have gains not far from each other when we look only at coordinates $1, 2, \ldots, d-1$. This is because their values in these coordinates are close to each other. Moreover, since we pick from each bucket the point with maximum value in the last coordinate, the gain of this point is close to the gain of other points in the same bucket. The regret is then bounded. Further, we pick the points $p_1^*, \ldots, p_{d-1}^*$ which are largest in the first $d-1$ coordinates to ensure that the gain of the output is large. We can then bound the regret ratio by considering the gain of these points. This intuition leads to the following main theorem, the proof of which can be found in Appendix C.

THEOREM 3. CUBE *(cf. Algorithm 4.1) returns a set S such that $rr_D(S,\mathcal{L}) \leq \frac{d-1}{t+d-1}$ where $t = \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$.*

Example: If $k = 10$ and $d = 2$, then CUBE guarantees that the regret ratio is at most 10%.

Running Time: We only need to find what bucket each point is in and, in each bucket, find the point with maximum value in the last coordinate. Moreover, there are $t^{d-1} \leq k$ buckets to examine. Therefore, CUBE runs in time $O(nd+k)$.

4.2 Lower Bound

In this section, we present the lower bound theorem for bounding the maximum possible regret ratio on 2-dimensional points. Informally, we show that it is impossible to guarantee a maximum regret ratio better than $O(1/k^2)$. We present the theorem and proof intuition below. The formal proof is presented in Appendix D

THEOREM 4. *For any k, there exists n such that there is a set D of n points such that, for any $S \subseteq D$ of size k, $rr_D(S,\mathcal{L}) = \Omega(1/k^2)$.*

The proof relies on considering infinitely many points spread as a quarter-circle (centered at $(0,0)$) between coordinates $(0,1)$ and $(1,0)$. The proof can be suitably adapted to work for finite number of points as well. If k of these points are picked, consider the line segments formed by joining the origin $(0,0)$ to each of these points and also joining $(0,0)$ to $(0,1)$ and $(1,0)$. The angle between some two consecutive line segments is at least $\pi/2(k+1)$. Let p_{ϕ_i} and $p_{\phi_{i+1}}$ denote the two points corresponding to these segments. We then consider a point p in the center of arc $(p_{\phi_i}, p_{\phi_{i+1}})$. The proof considers the linear utility function f that is maximized by p. The regret ratio for f is then measured, and is determined by projecting p_{ϕ_i} and $p_{\phi_{i+1}}$. This can be computed by considering the L_2-distance of the projected point to origin. We argue (details in Appendix D) that this regret ratio is at least $\Omega(1/k^2)$.

5. GREEDY HEURISTIC

In this section, we present a natural greedy heuristic called GREEDY (cf. Algorithm 5.1). This algorithm essentially follows the framework of Ramer-Douglas-Peucker algorithm [11, 25] for approximating curves and polygons, which, although does not have any theoretical guarantees beyond the case of two dimensions [26, 30], has been widely used and shown to perform well in higher dimensions in various applications with different measures (see, e.g., [9, 10, 15] and references therein). Our greedy algorithm can be thought of as another application of Ramer-Douglas-Peucker algorithm with the maximum regret ratio as a measure.

Algorithm 5.1 GREEDY(D, k)

Input: A set of d-dimensional points $D = \{p_1, p_2, ..., p_n\}$ and an integer k, the desired output size.
Output: A subset of D of size k, denoted by S.

1: Let $S = \{p_1^*\}$ where $p_1^* = \arg\max_{p \in D} p[1]$.
   {In the loop below, we find a point $p \in D \setminus S$ with maximum regret ratio.}
2: **for** i=1 to $k-1$ **do**
3: Let $r^* = 0$ and $p^* = null$.
4: **for** each $p \in D \setminus S$ **do**
5: Compute $rr_{S\cup\{p\}}(S,\mathcal{L})$ using Linear Program (1).
6: **if** $r^* < rr_{S\cup\{p\}}(S,\mathcal{L})$ **then**
7: $r^* = rr_{S\cup\{p\}}(S,\mathcal{L})$
8: $p^* = p$
9: **end if**
10: **end for**
11: **if** $r^* = 0$ **then** return S
12: **else** $S = S \cup \{p^*\}$ **endif**
13: **end for**
14: **return** S

Informally, GREEDY performs the following greedy process: First, it picks the point that maximizes the first coordinate. In subsequent iterations it adds the "worst" point, i.e., the point that currently contributes to the maximum regret ratio. To be precise, for each round it adds the point p to the solution set S where p is the point such that $rr_D(S,\mathcal{L}) = rr_{S\cup\{p\}}(S,\mathcal{L})$. This is done by computing $rr_{S\cup\{p\}}(S,\mathcal{L})$ for each point $p \in D \setminus S$ and keep the point with maximum value (as in Line 4–10), where $rr_{S\cup\{p\}}(S,\mathcal{L})$ is computed using the following linear program (LP).

Finding $rr_{S\cup\{p\}}(S,\mathcal{L})$ using an LP. Given a point p and a set S, one can find $rr_{S\cup\{p\}}(S,\mathcal{L})$ by solving the LP (1) below.[3]

$$\begin{aligned} \max \quad & x \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{d}(p[j]-p'[j])v[j] \geq x \qquad \forall\, p' \in S \\ & \textstyle\sum_{j=1}^{d} p[j]v[j] = 1 \\ & v[j] \geq 0 \qquad \forall\, j \leq d \\ & x \geq 0 \end{aligned} \tag{1}$$

The above LP has a variable x and variables $v[1]$, $v[2]$, …, $v[d]$ which denote a non-negative vector $v = (v[1], v[2], \ldots)$. The idea is that for any fixed value of v, x will be equal to $\min_{p' \in S}(p-p') \cdot v$ (by the first constraint and the fact that we are maximizing x). Therefore, maximizing x is equivalent to finding a vector v such that $\min_{p' \in S}(p-p') \cdot v$ (i.e. the regret ratio in the direction of v) is maximized. See Ap-

[3]Note that GREEDY can be implemented with any linear constraints imposed on the class of utility functions since they can be added to the LP.

pendix E for details on correctness and implementation.

Running Time. GREEDY has to run LP (1) at most nk times. Each LP has $k+d+2$ constraints and $d+1$ variables and thus has running time independent of n. In practice, an LP solver (such as variations of the Simplex method) will result in $O(k^2d)$ running time per LP call [4, Chapter 3], and hence $O(nk^3d)$ overall for GREEDY. In the worst case, the time per LP could be exponential in k and d using these LP solvers. However, by using the interior point method [18] we can guarantee a worst case running time of $O(nk^2d^{4.5})$.

6. EXPERIMENTAL EVALUATION

In this section, we show via experiments on both real and synthetic data that the maximum regret ratio is small (even for small k) in practice. We measured the maximum regret ratios and running times for the algorithms proposed in this paper as well as three previously proposed algorithms for representing the skyline by a fixed set of points. All our implementations were done in C. The experiments were all performed on a 1.7GHz Intel Xeon machine running Linux 2.6.9.

We used both anti-correlated and independent synthetic data, as is standard in the skyline literature, using the dataset generator of [5]. We averaged the algorithms' performance over 10 independently generated data sets. For our experiments on real data, we made use of some data sets that are available from Dr. Yufei Tao's homepage[4]. Unless otherwise stated, our experiments all used ten thousand points ($n = 10,000$) and picked the top-10 points ($k = 10$). We evaluated the maximum regret ratio for the algorithms using LP (1) implemented in the GNU Linear Programming Kit (GLPK) package [1] (more details in Appendix E).

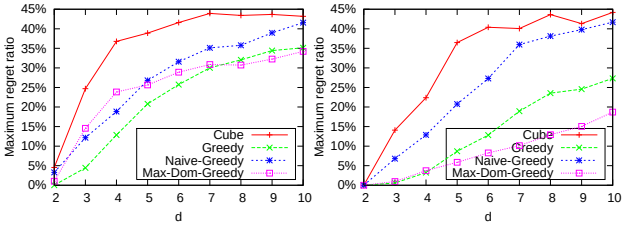
We consider both the CUBE algorithm and the GREEDY algorithm suggested in this paper and also consider Max-Dom-Greedy [22], Naive-Greedy [27], and Ext-Two-Scan [6].

- The CUBE algorithm is our implementation of Algorithm 4.1 with the modification that we increment t until we have at least k distinct points.
- The GREEDY algorithm is our implementation of Algorithm 5.1, which outputs precisely k distinct points.
- The NAIVE-GREEDY algorithm is our implementation of the naive algorithm proposed in [27]. In the Naive-Greedy approach, the goal is to pick k points that admit the best k-center clustering on the skyline set.
- The MAX-DOM-GREEDY algorithm is our implementation of the greedy algorithm in [22, Algorithm 1].[5] The objective of MAX-DOM-GREEDY is to pick k points that together dominate the largest number of points.
- Finally, we implemented the Top-δ Ext-Two-Scan algorithm proposed in [6, Algorithm 5]. However, the maximum regret ratio of this algorithm was so poor in the experiments that we do not include its results in the plots.

In Figure 3, we compare algorithms for various number of dimensions on anti-correlated and independent data, respectively, when the number of points was fixed to $n = 10000$ and $k = 10$. Due to the curse of dimensionality, the regret ratio of the algorithms degrade with the increase in number of dimensions. However, as the number of dimensions increases, the regret ratio of all the algorithms appear to level off much below the theoretical bound.

[4] http://www.cse.cuhk.edu.hk/~taoyf/

[5] We only test the maximum regret ratio yielded by MAX-DOM-GREEDY here and note that, if desired, a more efficient implementation of MAX-DOM-GREEDY (with a slightly worse solution) can be found in [22].



(a) anti-correlated data (b) independent data

Figure 3: Varying d ($n = 10000$ and $k = 10$)

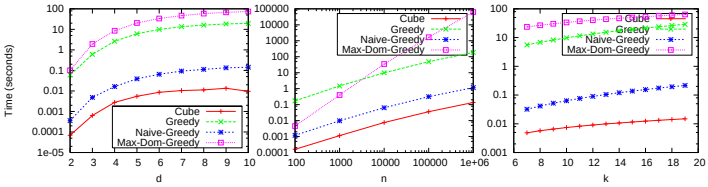


Figure 7: Running time for anti-correlated data (Default: $n = 10000$, $d = 6$, $k = 10$)

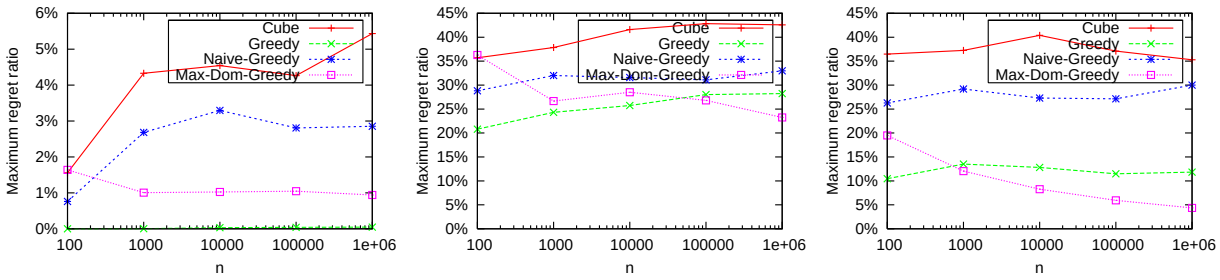
We varied the number of points using anti-correlated data in Figures 4(a) and 4(b), using $d = 2$ and $d = 6$, respectively. For the two-dimensional case, GREEDY gives maximum regret ratio almost zero but for higher dimensions the maximum regret ratio can be much larger (close to 25% for the GREEDY and MAX-DOM-GREEDY algorithms on 6D data using $k = 10$). For 6D independent data (Figure 4(c)), the GREEDY and MAX-DOM-GREEDY algorithms are consistently below 20% for $k = 10$.

In Figure 5 we see the effect of varying k on the four algorithms. As would be expected, the regret-ratio decreases monotonically with k. With only 20 representative points, GREEDY makes the maximum regret ratio at most 16% in all three cases. We observe that the algorithms perform better than predicted by theory on these specific distributions.

The results for other data sets that we considered are summarized in Figure 6. In all cases we used $k = 10$. The algorithms performed extremely well for this data (i.e., the best algorithm was under 1% regret ratio), except for the Color dataset. For this data, the maximum regret ratio went up to almost 17% for the GREEDY algorithm, due to the high dimensionality of the data.

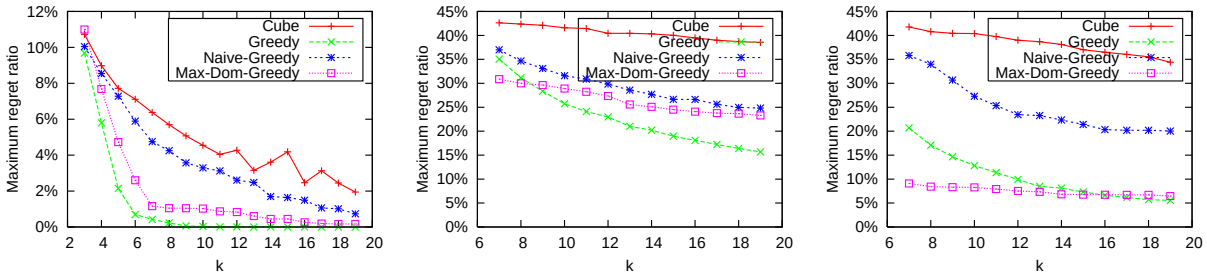
Times

In this section we discuss the running times of various algorithms. We compute the skyline first before running these algorithms as our main objective is to illustrate that minimizing maximum regret ratio takes time negligible as compared to the running time of skyline computation. Also, we compute skyline first to avoid the issue of computation with and without preprocessing (see, e.g., [13] for discussions). It is an interesting future work to explore more on the running time aspect of this problem, especially bypassing the whole skyline computation and considering the computation with and without preprocessing. The running time of skyline



(a) 2D anti-correlated data (b) 6D anti-correlated data (c) 6D independent data

Figure 4: Varying number of points n with $k = 10$



(a) 2D anti-correlated data (b) 6D anti-correlated data (c) 6D independent data

Figure 5: Varying k ($n = 10000$)

computation is not shown here.

The running times on anti-correlated data, with d, n, and k varied, are shown in loglog scale in Figure 7. In all cases, CUBE and NAIVE-GREEDY are very fast. (They finish within one second even when $d = 6$, $n = 1,000,000$ and $k = 10$.) Note, however, that their solution is inferior to GREEDY and MAX-DOM-GREEDY as shown in the previous section. The running time of GREEDY is usually within 10 seconds except in the case of $d = 6$, $n \geq 100,000$, and $k = 10$. We note that its running time is always negligible compared with the time taken to compute skyline. MAX-DOM-GREEDY runs much longer. However, as noted earlier, one can obtain a practical implementation by slightly sacrificing the solution quality (see [22]).

Summary

Both GREEDY and MAX-DOM-GREEDY work well in all data sets (except the Island data set for MAX-DOM-GREEDY), giving solutions with maximum regret ratio much lower than the theoretical guarantee (cf. Section 4.1). Neither algorithm beats the other in all cases. This suggests that GREEDY and an efficient implementation of MAX-DOM-GREEDY (based on a randomized index-based algorithm available in [22]) could be used in practice. Note, however, that MAX-DOM-GREEDY is optimizing a measurement that is unstable (as discussed in Section 2). Finally, we observe that picking k to be about twice of d is sufficient in practice since it bounds the regret ratio below 20%.

7. CONCLUSIONS

In this paper, we consider the problem of finding a few tuples to represent the database such that these representative tuples help users find the best tuple based on their criteria and utility functions. Since it is impossible to display every user's optimal tuple, we consider a natural relaxation where we want to minimize the *maximum regret ratio* which captures how unhappy the users may feel when they see k representative points instead of the whole database. Before considering the computational issue of this problem, it is more important to confirm first that we can always make this quantity small. We confirm that this is possible by showing that for linear utility functions the maximum regret ratio can be bounded in terms of the number of representative points (k) and the dimensionality (d). The bound is independent of the size of the database (n). Moreover, our extensive experiments on various algorithms suggest that the bound could be much lower in practice. This opens a new door to multi-criteria decision support query in terms of regret minimization, and many open problems follow.

The most important open problem is the computational issue of this problem. Although algorithms in this paper run in time linear in n and the experiments suggest that GREEDY and MAX-DOM-GREEDY are practical enough, it would be interesting to explore the running time aspect of this problem further. In particular, the restricted small output size (k) of this problem makes it possible to get an algorithm that runs faster than computing a huge skyline. The issue of external algorithms with and without preprocessing should also be thoroughly explored.

On the theoretical side, it is still open whether we can compute the *optimal solution*. We conjecture that this is NP-hard and it would be interesting to develop algorithms with approximation guarantees. Moreover, for the theoretical guarantee of the maximum regret ratio, there is still

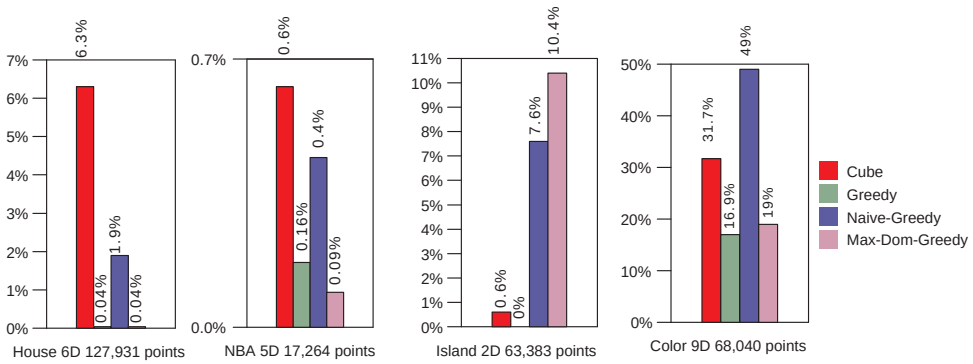


Figure 6: Maximum regret ratios on other data sets

a gap between the lower bound of $\Omega(1/k^2)$ and the upper bound of $O(1/k^{1/(d-1)})$.

Finally, our techniques are very specific to linear utility functions. It would be interesting to bound the maximum regret ratio in other classes of utility functions, such as Cobb-Douglas, weak gross substitutability, submodular, and convex functions. (See, e.g., [2, Chapter 11] for more functions that arise in Economics.) Ideally, we would like to be able to bound the regret ratio for the class of all monotone functions.

Acknowledgement: We thank anonymous reviewers for very helpful comments on the paper.

8. REFERENCES

[1] Gnu linear programming kit, version 4.39,. `http://www.gnu.org/software/glpk/glpk.html`.
[2] *Algorithmic Game Theory*. Cambridge University Press, September 2007.
[3] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.
[4] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
[6] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.
[7] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, 2006.
[8] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *SIGMOD*, 2000.
[9] J. L. Cohon. *Multiobjective programming and planning*. Dover Publications, 2004.
[10] D. L. Craft, T. F. Halabi, H. A. Shih, and T. R. Bortfeld. Approximating convex pareto surfaces in multiobjective radiotherapy planning. *Med. Phys.*, 33(9):3399–3407, 2006.
[11] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
[12] P. Godfrey. Skyline cardinality for relational processing. In *FoIKS*, 2004.
[13] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007. Also appeared in VLDB'05.
[14] M. Goncalves and M.-E. Vidal. Top-k skyline: A unified approach. In *OTM Workshops*, 2005.
[15] P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms, 1997.
[16] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, 2001.
[17] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
[18] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.
[19] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos. Continuous top-k dominating queries in subspaces. In *Panhellenic Conference on Informatics*, 2008.
[20] J. Lee, G. won You, and S. won Hwang. Personalized top-k skyline queries in high-dimensional space. *Inf. Syst.*, 34(1):45–61, 2009.
[21] X. Lian and L. C. 0002. Top-k dominating queries in uncertain databases. In *EDBT*, 2009.
[22] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.
[23] D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *PVLDB*, 2(1):610–621, 2009.
[24] A. N. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos. Domination mining and querying. In *DaWaK*, 2007.
[25] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.
[26] G. Rote. The convergence rate of the sandwich algorithm for approximating convex functions. *Computing*, 48(3-4):337–361, 1992.
[27] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.
[28] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, 2003.
[29] T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *ICDE*, 2008.
[30] X. Q. Yang and C. J. Goh. A method for convex curve approximation. *European Journal of Operational Research*, 97(1):205–212, February 1997.
[31] M. L. Yiu and N. Mamoulis. Multi-dimensional top-dominating queries. *VLDB J.*, 18(3):695–718, 2009. Also VLDB'07.
[32] W. Zhang, X. Lin, Y. Zhang, J. Pei, and W. Wang. Threshold-based Probabilistic Top-k Dominating Queries. *VLDB J.*, 19(2):283–305, 2009.

APPENDIX

A. SCALE INVARIANCE

PROOF OF THEOREM 1. Consider two sets of t points, $D = \{a_1, a_2, \ldots a_t\}$ and $D' = \{a'_1, a'_2, \ldots, a'_t\}$. Suppose that D and D' are a rescaling of each other, i.e., there exist positive reals $\lambda_1, \lambda_2, \ldots, \lambda_d$ such that for any $1 \leq i \leq t$ and $1 \leq j \leq d$, $a'_i[j] = \lambda_j a_i[j]$.

CLAIM 5. *For any vector v, there exists a vector v' such that $v \cdot a_i = v' \cdot a'_i$ for any $a_i \in D$ and $a'_i \in D'$.*

PROOF. Given a vector v, we define each coordinate of vector v' to be $v'[j] = v[j]/\lambda_j$ for $j = 1, 2, \ldots, d$. Observe that for any a_i and a'_i,

$$\begin{aligned} v \cdot a_i &= \sum_{j=1}^{d} v[j] a_i[j] \\ &= \sum_{j=1}^{d} (\lambda_j v'[j])(a'_i[j]/\lambda_j) \\ &= \sum_{j=1}^{d} v'[j] a'_i[j] \\ &= v' \cdot a'_i \end{aligned}$$

as claimed. □

Now consider any two sets $S \subseteq D$ and $S' \subseteq D'$ that are "equivalent" in the sense that if $S = \{a_{i_1}, a_{i_2}, \ldots, a_{i_s}\}$ then $S' = \{a'_{i_1}, a'_{i_2}, \ldots, a'_{i_s}\}$. We claim that $rr_D(S, \mathcal{L}) = rr_{D'}(S', \mathcal{L})$.

To see this, let v be the "worst" vector for S, i.e., v is such that $rr_D(S, \mathcal{L}) = rr_D(S, v)$. By Claim 5, there exists a vector v' such that $v' \cdot a'_i = v \cdot a_i$ for all $1 \leq i \leq t$. This implies that $rr_D(S, v) = rr_{D'}(S', v')$ because

$$\begin{aligned} rr_D(S, v) &= \frac{\max_{a_i \in D} v \cdot a_i - \max_{p \in S} v \cdot p}{\max_{a_i \in D} v \cdot a_i} \\ &= \frac{\max_{a'_i \in D'} v' \cdot a'_i - \max_{p' \in S'} v' \cdot p'}{\max_{a'_i \in D'} v' \cdot a'_i} \\ &= rr_{D'}(S', v') \,. \end{aligned}$$

Therefore, $rr_D(S, \mathcal{L}) \leq rr_{D'}(S', \mathcal{L})$. (There might be a vector worse than v' for S'.) By the same argument (i.e., start from v' that is "worst" for S'), we get $rr_{D'}(S', \mathcal{L}) \leq rr_D(S, \mathcal{L})$. Therefore, $rr_{D'}(S', \mathcal{L}) = rr_D(S, \mathcal{L})$

Since the regret ratio is the same for every pair of equivalent sets, $rr_D(S_1, \mathcal{L})/rr_D(S_2, \mathcal{L}) = rr_D(S'_1, \mathcal{L})/rr_D(S'_2, \mathcal{L})$ for any sets $S_1, S_2 \subseteq D$ and the equivalent sets $S'_1, S'_2 \subseteq D'$. The theorem thus holds. □

B. STABILITY

PROOF OF THEOREM 2. Let p be any junk point and $D' = D \cup \{p\}$. Let g be any linear utility function. Since p is a junk point, there is a point $q \in D$ such that $g(p) \leq g(q)$. In particular, $\max_{p' \in D'} g(p') = \max_{p' \in D} g(p')$ and thus $gain(D', g) = gain(D, g)$. It follows from the equalities below that $h(S) = rr_D(S, \mathcal{L})$ satisfies the definition of stability:

$$\begin{aligned} rr_{D'}(S, g) &= \frac{gain(D', g) - gain(S, g)}{gain(D', g)} \\ &= \frac{gain(D, g) - gain(S, g)}{gain(D, g)} \\ &= rr_D(S, g) \,. \end{aligned}$$

□

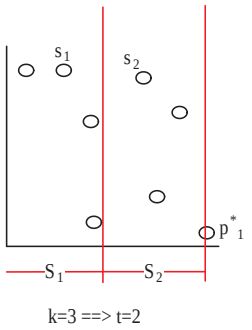


Figure 8: Example of CUBE when $k = 3$ (thus $t = 2$)

C. PROOF OF UPPER BOUND

PROOF OF THEOREM 3. Let v be any vector whose entries are non-negative. Vector v represents a linear utility function of a user. Let p be a point in D that maximizes this utility function; i.e., $gain(D, v) = gain(\{p\}, v)$. Let $j_1, j_2, ..., j_{d-1}$ be integers such that $p \in S_{j_1, \ldots, j_{d-1}}$. To preserve ink, let $s = s_{j_1, \ldots, j_{d-1}}$.

First, we claim that $p \cdot v - s \cdot v \leq \frac{d-1}{t} \max_{i \leq d-1}(p_i^* \cdot v)$. This claim follows from the following inequalities.

$$\begin{aligned} p \cdot v - s \cdot v &= \sum_{i=1}^{d} (p[i] - s[i]) \cdot v[i] \\ &\leq \sum_{i=1}^{d-1} (p[i] - s[i]) \cdot v[i] \qquad (2) \\ &\leq \sum_{i=1}^{d-1} \frac{c_i}{t} v[i] \qquad (3) \\ &\leq \frac{d-1}{t} \max_{i \leq d-1} (c_i v[i]) \\ &\leq \frac{d-1}{t} \max_{i \leq d-1} p_i^* \cdot v. \end{aligned}$$

Note that (2) uses the fact that $s[d] \geq p[d]$ and (3) is because both p and s are in $S_{j_1, j_2, \ldots, j_d}$. Since s is in S, the regret of the user with utility function v is

$$r_D(S, v) \leq p \cdot v - s \cdot v.$$

Moreover, since $p_1^*, p_2^*, ..., p_{d-1}^*$ are in S,

$$gain(S, v) \geq \max_{i \leq d-1} (p_i^* \cdot v) \,.$$

The previous claim implies that

$$\frac{r_D(S, v)}{gain(S, v)} \leq \frac{d-1}{t} \,.$$

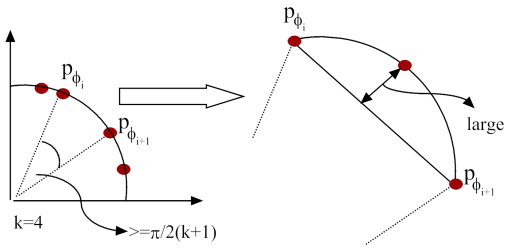


Figure 9: Proof idea of Theorem 4. Notations are defined in the proof.

Note a simple observation that

$$\frac{r_D(S,v)}{gain(\{p\},v)} = \frac{r_D(S,v)}{r_D(S,v)+gain(S,v)}\,.$$

The right hand side is bounded by $\frac{d-1}{t+d-1}$. Therefore,

$$\begin{aligned} rr_D(S,\mathcal{L}) &= \max_v \frac{r_D(S,v)}{gain(D,v)} \\ &= \max_{p,v} \frac{r_D(S,v)}{gain(\{p\},v)} \\ &\le \max_{p,v} \frac{r_D(S,v)}{r_D(S,v)+gain(S,v)} \\ &\le \frac{d-1}{t+d-1} \end{aligned}$$

as desired. □

D. PROOF OF LOWER BOUND

Proof of Theorem 4. Consider an infinite set of points in 2-dimensions, $D = \{p_\Theta = (\cos\Theta, \sin\Theta) \mid \Theta \in [0, \pi/2]\}$. This forms quarter circle of unit radius (as in Figure 9). Each of the points in D is maximal for some linear utility function. This is because for $\pi/2 \ge \Theta_1 > \Theta_2 \ge 0$, we have $\sin\Theta_1 > \sin\Theta_2$ and $\cos\Theta_1 < \cos\Theta_2$.

Consider any set S of k representative points. Let these points be defined by angle parameters $\pi/2 \ge \phi_1 > \phi_2 > \ldots > \phi_k \ge 0$; i.e. the points are p_{ϕ_i} for all $1 \le i \le k$. Consider the angles subtended between lines obtained by joining the point $O = (0,0)$ with the points $p_0 = (1,0)$, $p_{\phi_1}, p_{\phi_2}, \ldots, p_{\phi_k}, p_{\pi/2} = (0,1)$; that is, consider angles $p_{\phi_i} - O - p_{\phi_{i+1}}$ for all $1 \le i \le k$ and also consider $p_0 - O - p_{\phi_1}$ and $p_{\phi_k} - O - p_{\pi/2}$. We know that the sum of these $k+1$ angles is exactly $\pi/2$. Therefore, at least one of these $k+1$ angles is at least $\frac{\pi}{2(k+1)}$. Let this angle be defined by $p_{\phi_i} - O - p_{\phi_{i+1}}$ and denote it by γ.

Set $\alpha = \phi_{i+1} + \frac{\gamma}{2}$ and consider a user with utility function $f(x,y) = x\cdot\cos\alpha + y\cdot\sin\alpha$. The utility of any point p along the utility function f is the L_2-distance between the origin $(0,0)$ and the projection of point p onto the line $(0,0) - -(\cos\alpha, \sin\alpha)$.

Consider the point $(\cos\alpha, \sin\alpha)$. The utility of this point for function f is $\cos\alpha\cdot\cos\alpha + \sin\alpha\cdot\sin\alpha = 1$. Notice that among the k selected points $p_{\phi_1}, \ldots, p_{\phi_k}$, the two points that have the largest distance from $(0,0)$ when projected onto the line corresponding to α are p_{ϕ_i} and $p_{\phi_{i+1}}$. This follows from our construction and choice of $\alpha = \phi_{i+1} + \frac{\gamma}{2}$ where γ is the angle subtended at the origin by p_{ϕ_i} and $p_{\phi_{i+1}}$. Therefore, the closer a point p's angle subtended between $(0,0)$ and $(1,0)$ is to α, the larger the L_2 distance of the projection is from $(0,0)$. See Figure 9.

Therefore, the best choice for f, among the representative points in S, is p_{ϕ_i}, giving a gain of $\cos\phi_i\cdot\cos\alpha + \sin\phi_i\cdot\sin\alpha$. (In fact, $p_{\phi_{i+1}}$ is also another best choice which gives the same gain. To see this, observe that the line $(0,0) - (\cos\alpha, \sin\alpha)$ is perpendicular to the line $p_{\phi_i} - p_{\phi_{i+1}}$.) The projection is the point $(\cos(\gamma/2)\cdot\cos\alpha, \cos(\gamma/2)\cdot\sin\alpha)$. The utility therefore is $\cos(\gamma/2)\cdot\cos\alpha\cdot\cos\alpha + \cos(\gamma/2)\cdot\sin\alpha\cdot\sin\alpha = \cos(\gamma/2)\cdot(\cos^2\alpha + \sin^2\alpha) = \cos(\gamma/2)$.

The regret ratio for f therefore is $\frac{1-\cos(\gamma/2)}{1}$. Now using the series expansion for *Cosine*, we get a regret ratio of $1 - (1-(\gamma/2)^2/2! + (\gamma/2)^4/4! - \ldots) = (\gamma/2)^2/2! - (\gamma/2)^4/4! + \ldots$. Now using the fact that $(\gamma) \ge \frac{\pi}{2(k+1)}$, we get that the regret ratio is at least $\frac{\pi^2}{32(k+1)^2} - O(\frac{1}{k^4})$. Therefore, the regret ratio is at least $\frac{\pi^2}{32(k+1)^2} - o(1/k^2)$ which is $\Omega(\frac{1}{k^2})$.

This completes the proof. See Figure 9 for proof idea. □

E. LINEAR PROGRAM

E.1 Computing the regret ratio

Given a set of d-dimensional points S and a point p, we can find the regret ratio when the database has only one point p, i.e. $rr_{S\cup\{p\}}(S,\mathcal{L})$, as follows. First, recall that by definition

$$\begin{aligned} rr_{S\cup\{p\}}(S,\mathcal{L}) &= \max_{v:\ vector} \frac{gain(\{p\},v) - gain(S,v)}{gain(\{p\},v)} \\ &= \max_{v:\ vector} \frac{p\cdot v - max_{p'\in S}(p'\cdot v)}{p\cdot v} \\ &= \max_{v:\ vector} \min_{p'\in S} \frac{(p-p')\cdot v}{p\cdot v}\,. \end{aligned}$$

Instead of maximizing over all vectors, we can restrict to every vector v such that $p\cdot v = 1$. (This is because we can write any vector v as $\lambda v'$ for some positive real scalar λ and vector v' such that $p\cdot v' = 1$. The scalar λ will be canceled out.) Therefore,

$$rr_{S\cup\{p\}}(S,\mathcal{L}) = \max_{v\ :\ p\cdot v=1} \min_{p'\in S} (p-p')\cdot v\,.$$

Letting the coordinates of v be variables $v[1], v[2], \ldots, v[d]$ and introducing a variable x, we can translate the above equation into the following LP.

$$\begin{aligned} \max \quad & x \\ \text{s.t.} \quad & (p-p')\cdot v \ge x && \forall\, p' \in S \\ & p\cdot v = 1 \\ & v[j] \ge 0 && \forall\, j \le d \\ & x \ge 0 \end{aligned}$$

The LP (1) in the paper is the expanded form of the above LP.

E.2 Implementing the Linear Program on GLPK

In this section we show the details of implementing the linear program (1). We use `GNU Linear Programming Kit` (GLPK) [1] in our experiment. To do this, we convert the LP above into the following form. Let $S = \{p_1, p_2, \ldots p_t\}$.

$$
\begin{aligned}
\max \quad & x \\
\text{s.t.} \quad & x - (p - p_i) \cdot v \leq y_i && \forall\ i \leq t \\
& p \cdot v \leq r_1 \\
& -p \cdot v \leq r_2 \\
& 0 \leq v[j] < \infty && \forall\ j \leq d \\
& 0 \leq x \leq \infty \\
& -\infty < y_i \leq 0 && \forall\ i \leq t \\
& -\infty < r_1 \leq 1 \\
& -\infty < r_2 \leq -1
\end{aligned}
$$

For further information, we refer the reader to the GLPK manual.