Interactive Regret Minimization

Danupon Nanongkai*
University of Vienna
danupon@gmail.com

Ashwin Lall
Denison University
lalla@denison.edu

Atish Das Sarma
Google Research
atish.dassarma@gmail.com

Kazuhisa Makino
University of Tokyo
makino@mist.i.u-tokyo.ac.jp

ABSTRACT

We study the notion of *regret ratio* proposed in [19] to deal with multi-criteria decision making in database systems. The regret minimization query proposed in [19] was shown to have features of both skyline and top-k: it does not need information from the user but still controls the output size. While this approach is suitable for obtaining a reasonably small regret ratio, it is still open whether one can make the regret ratio arbitrarily small. Moreover, it remains open whether reasonable questions can be asked to the users in order to improve efficiency of the process.

In this paper, we study the problem of minimizing regret ratio when the system is enhanced with *interaction*. We assume that when presented with a set of tuples the user can tell which tuple is most preferred. Under this assumption, we develop the problem of *interactive regret minimization* where we fix the number of questions and tuples per question that we can display, and aim at minimizing the regret ratio. We try to answer two questions in this paper: (1) How much does interaction help? That is, how much can we improve the regret ratio when there are interactions? (2) How efficient can interaction be? In particular, we measure how many questions we have to ask the user in order to make her regret ratio small enough.

We answer both questions from both theoretical and practical standpoints. For the first question, we show that interaction can reduce the regret ratio almost *exponentially*. To do this, we prove a lower bound for the previous approach (thereby resolving an open problem from [19]), and develop an almost-optimal upper bound that makes the regret ratio exponentially smaller. Our experiments also confirm that, in practice, interactions help in improving the regret ratio by many orders of magnitude. For the second question, we prove that when our algorithm shows a reasonable number of points per question, it only needs a few questions to make the regret ratio small. Thus, interactive regret minimization seems to be a necessary and sufficient way to deal with multi-criteria decision making in database systems.

*Work done while at Georgia Tech and was supported in part by Georgia Tech ACO fellowship and by Richard J. Lipton.

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

Keywords

Skyline, Top-k, Regret Minimization

1. INTRODUCTION

Assisting the end-users in finding the most desired tuples in the database is an important task in many application domains. Top-k [9] and skyline [1, 6] queries are two well-studied tools in such settings but they have some drawbacks that motivate the proposals of many later queries. In this paper, we adopt the notion of *maximum regret ratio* proposed in [19]. To motivate the notion of maximum regret ratio, consider the following example from [19].

Suppose Alice is searching for a car with high miles per gallon (MPG) and high horse power (HP). Note that horsepower comes at the expense of fuel economy. How does a car dealer's database system best assist Alice in trading off between these two criteria?

One way is by using the *top-k* operator (see [9] for a recent survey). This approach is based on a widely-accepted assumption that users pick tuples based on their *utility functions* (or *ranking functions*). This operator asks the users to provide their utility functions and then finds the best k tuples in the database that maximize the function. For example, Alice could indicate that she gives weights 70% to the MPG and 30% to the HP. In this particular case, where Alice has a linear utility function, the system could use a well-developed system such as PREFER [8] and employ sophisticated techniques such as ONION [4] and Ranked Join Indices [22] to effectively assist her. However, the major drawback of this approach is that *asking Alice for her utility function is unreasonable* since she might not know what weights she wants to give to each criterion.

An alternative for avoiding this problem is the *skyline* operator proposed by Börzsönyi et al. [1]. (For further details, see, e.g., [6] and references therein.) This operator asks the customer only for a set of criteria and outputs everything that the user might be interested in. For example, if Alice

specifies that she wants to maximize the MPG and the HP, then the operator will output *all* the cars based on these criteria. A car will not be shown if there is another with more MPG and HP. However, this approach has a major drawback in that it *cannot control the output size*. In fact, the skyline size is usually large in practice. This poses a big challenge in adopting the skyline operator even in giving users the big picture of the database, leaving almost impossible the task of using it as a stand alone operator.

To deal with the drawbacks of top-k and skyline discussed above, many papers proposed different approaches (e.g., [2, 3, 5, 7, 12, 15, 18–21, 23, 24]). In this paper, we adopt the notion of *maximum regret ratio when users have linear utility functions* proposed in [19] since it makes no additional assumptions about the users. Moreover, it has attractive properties such as scale invariance and stability, and one can guarantee the effectiveness of this approach independent of the database size (see further discussion and details in [19]).

Intuitively, the maximum regret ratio is as follows. 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 she obtains from the best tuple in the whole database. We say that the *happiness ratio* of the user on this list is $x\%$. Since we are hoping to make x large, we will focus on minimizing the smaller quantity called the regret ratio which is simply $(100 - x)\%$. The problem of *minimizing the regret ratio* aims at making the regret ratio as small as possible, given the output size limit of k *without asking the user any questions*. This problem has features of both top-k and skyline operators, i.e., the output size (k) can be controlled (like top-k) and it does not need any feedback from users (like skyline).

In [19], it is shown that a reasonably small regret ratio can be achieved by displaying only a few tuples. For example, to guarantee a maximum regret ratio of 10% in the worst case when there are two criteria, one has to show only ten tuples. Moreover, experiments in [19] show that even for the rare case of ten criteria, showing ten tuples still makes the maximum regret ratio below 35% in practice. This makes the notion of maximum regret ratio suitable for giving a "big picture" of the database.

Despite this good news, it remains open whether interactions with users will help. Intuitively, questions can make the maximum regret ratio smaller even when we display the same number of tuples. Moreover, it might help the user find the most desired tuple in the database (i.e., make the maximum regret ratio zero). Without interactions, the latter task is impossible while the former is very hard to achieve when there are many criteria and one wants a very small maximum regret ratio.

For example, while experiments in [19] suggest that, in practice, a maximum regret ratio of 35% can be guaranteed using only ten tuples for the case of ten criteria, it turns out that one needs more than a thousand tuples to guarantee a 1% maximum regret ratio (as we will show later in this paper). The situation is even worse from the theoretical point of view since, in the worst case, there exists a database such that a maximum regret ratio of 10% cannot be guaranteed without displaying all tuples—and the database could contain billions of tuples! (We will show this fact later in this paper.) Thus, while having no interactions was proved to be sufficient in presenting a "big picture" of the database (by guaranteeing a reasonable regret ratio), it is not suitable for targeting a small maximum regret ratio or dealing with a large number of criteria.

In this light, we study how interaction helps in this paper. The first consideration is what kind of questions should the system ask the users? Since we do not want to run into the issue of asking a question that is too hard to answer by users, as we faced earlier in the case of top-k query, we try to make no additional assumptions about the users. In particular, we observe that in order to make the whole area of multi-criteria decision making well-defined, it is always assumed implicitly that the users know which tuple they desire the most when presented with choices. We therefore make only this assumption: *The users are able to tell which tuple they prefer the most from a set of tuples presented to them.*

The above observation leads us to the following framework. The database system interacts with the end-user in *rounds*. In each round, the system displays some tuples to the user (these tuples do not necessarily come from the database). Then, the user picks the tuple that she prefers the most. At any time, the system can notify the user that there are no more tuples to display or the user can notify the system that she wants to stop answering questions—at this point, the system can only display tuples without asking any further questions. The user is now free to pick one of the tuples presented to her so far as her desired tuple.

We are interested in minimizing the minimum regret ratio of the user on the set of all displayed tuples (in any round). In other words, we compare the happiness of the user in two scenarios: (1) when the user spends the effort to go through and find her favorite tuple in the entire database, and (2) when she picks her favorite tuple among some displayed subset. Our goal is to make the user's happiness in the latter case as close to that in the former case as possible.

Although the above framework is quite flexible in that it allows the system to display any tuple to the user, even those outside the database, its natural applications require the system to have one important property called *truthfulness*. Imagine that Alice is looking for a car from some website that runs our system. Naturally, the website should try to show the best car it has to attract Alice to stay and answer more questions. Now, suppose that Alice sees a car having 60 MPG and 200 HP in one set of the tuples the website displays, and gets excited by this car. This encourages her to put more effort in answering the questions in the hope that the website might be able to find her an even better car. However, if in the end she found out that the best car the website can offer has only 50 MPG and 190 HP, she might think that this website is a fraud (using a bait-and-switch tactic) and never come back. Thus, although the system may want to show tuples that attract users, it must always be *truthful*: The tuple that the user picks (i.e., her favorite tuple among all displayed tuples) must be in the database. We note that previous approaches such as Top-k and skyline are truthful but suffer from other problems as discussed earlier. Displaying only tuples in the database is a stronger form of truthfulness and is crucial in some cases. The algorithm in this paper does not satisfy this stronger property and we leave this as an interesting future direction.

Moreover, in order to attract users to answer questions as long as they can, it is desirable (although not crucial) that the system is *informative* and *ρ-progressive*. The system is informative if it is able to inform the user how much better

she can expect the best tuple in the database to be, compared to the displayed tuples. This helps the user decide whether she should continue answering questions or stop. The system is ρ-progressive if the user's regret ratio is improved by a factor of ρ in every round. This gives the user a feeling of improvement and thus encourages her to answer more questions. We will define these properties formally in the next section.

The above framework generalizes the problem of minimizing the maximum regret ratio in [19] since it adds interactions between the system and users. The main problem of this paper is how much does this interaction help. In particular, we consider the following problem which comes from [19]:

> How small can the user's regret ratio be made when the system can display at most k tuples to the user?

An equally important question is the following:

> How many tuples does the system need to display in order to make the user's regret ratio at most ϵ, for some $0 \leq \epsilon \leq 1$?

We are interested in comparing the answers to the above questions in the cases when interactions are used or not used (in both cases assuming linear utility preferences). We note that we are also interested in the special case of $\epsilon = 0$, i.e., we want the user to find her favorite tuple in the database.

Besides comparing with the previous approach, we are also interested in measuring the efficiency of the new approach—we would like to know if this minimal assumption we made is enough to make the system efficient or if we need a stronger assumption. In particular, since answering questions needs effort on the part of the user, we are interested in the following question:

> How many rounds do we need to ask the user to get the user's regret ratio below ϵ, for some $0 \leq \epsilon \leq 1$?

The parameter that comes into play in answering the above questions is the number of tuples that we can display per round (since it turns out that the more tuples we display per round, the faster the regret ratio decreases) which can be specified either by website creators or end-users. Therefore, we study the effect of this parameter on the number of rounds.

Our results

In this paper, we show that interaction is necessary and sufficient. From a theoretical point of view, our main results are upper bounds (with interactions) and lower bounds (for the case of no interactions) of the cases where there are and are not interactions:

- When interactions are allowed, we develop an algorithm called UtilityApprox that guarantees a regret ratio of ϵ by displaying $O(sd \log_s(d/\epsilon))$ tuples (equivalently, $O(d \log_s(d/\epsilon))$ rounds), where s is the number of tuples displayed per question and d is the number of criteria chosen (by the user) from a larger set of attributes. Moreover, we show that this algorithm is truthful, informative, and progressive. The algorithm is also efficient as it streams through the database just once in each round.
- We show that UtilityApprox is almost optimal, i.e., any interactive algorithm needs to display $\Omega(sd \log_s(1/\epsilon))$ tuples in order to achieve a regret ratio of ϵ.
- In contrast to the positive result for UtilityApprox, we show that without interaction, an algorithm can achieve a regret ratio of ϵ only when it shows $\Omega((1/8\epsilon)^{\frac{d-1}{2}})$ tuples to the user. This gives the first lower bound for the regret minimization problem for a general value of d, a question that was left open in [19].

Using the above results, we conclude the following answers to the questions previously stated.

- Using UtilityApprox algorithm with constant s, to achieve a regret ratio of ϵ, interaction exponentially reduces the number of tuples required: If $T = (1/8\epsilon)^{\frac{d-1}{2}}$ then we need only $O(\log T + d \log d)$ tuples when there are interactions while we need $\Omega(T)$ tuples otherwise.
- When we fix the total number of tuples k to be displayed, the above guarantee of UtilityApprox can be converted to a regret ratio guarantee of $O(d/s^{\frac{k}{sd}})$ and the lower bound for the case of no interactions can be converted to a regret ratio lower bound of $\Omega(1/k^{\frac{2}{d-1}})$. Using constant s, we again have that interactions help make the regret ratio exponentially smaller.

Thus, in theory, interaction improves the efficiency exponentially both in terms of the number of tuples displayed and the regret ratio achieved. Our extensive experiments also confirm these findings, i.e., although algorithms studied in [19] work well in achieving reasonable regret ratios (e.g., 10%), they are much inferior to UtilityApprox when we target a more accurate solution. For example, to achieve a regret ratio of 1% when there are ten criteria in the case of *anti-correlated data*, the previous non-interactive algorithms need to display more than a thousand tuples while UtilityApprox algorithm needs as few as 46 points (when we use $s = 2$). In fact, even to find the *most desired tuple* in the database, UtilityApprox still needs only 87 tuples! Thus, interactions help improve the efficiency significantly in both theory and practice.

As discussed earlier, although interactions help considerably, they also demand some effort on the user's part. In particular, answering many questions might be tiresome. We thus study how many questions UtilityApprox needs to ask. We show that with slightly larger sized questions, the total number of questions is reasonably small. From a theoretical perspective, we show that, to achieve a regret ratio of ϵ, UtilityApprox needs to ask $O(d \log_s(1/\epsilon))$ questions to the user. While theory, for example, guarantees that we need roughly twenty questions of ten tuples each to get 1% regret ratio, our experiments show that the number of questions needed in practice is often much smaller.

Organization: In Section 2 we discuss the prior work most relevant to this paper. We formally define the problem that we solve and related properties in Section 3. We describe the UtilityApprox algorithm in Section 4. We show the lower bound for the previous approach (no interactions) as well as the fact that UtilityApprox is almost optimal in Section 5. We present experimental results in Section 6. Finally, we discuss possible future work and conclude in Section 7.

2. RELATED WORK

Motivated by the deficiencies of top-k and skylines, many variants have been proposed recently. The work that is most relevant to us is [19] which proposed the notion of maximum regret ratio, as discussed earlier. This work combined features from top-k and skylines, i.e., its output size can be controlled, and it asks no questions. There have been many prior attempts to achieve both features. Lin et al. [15] and Tao et al. [21] consider an operation called representative skyline. Yiu and Mamoulis [20, 24] propose the top-k dominating query which has been further explored in other domains [11, 14, 25]. Goncalves and Vidal [7] propose two operators called top-k skyline select and top-k skyline join. Xia et al. [23] propose ϵ-skyline queries. It is argued in [19] that these approaches lack some important properties such as scale-invariance and stability. Other approaches try to reduce the output size. Lee et al. [12] avoid asking users for utility functions by requesting only partial ranking over attributes. Chan et al. [2, 3] propose the concept of skyline frequency and k-dominance. It is argued in [19] that, while these approaches can reduce the output size, they do not have a full control over it. For further discussion, see [19].

There are also papers that consider alternatives to directly asking for the utility function from the user. Mindolin et al. [18] propose the *p-skyline* query which is a framework that assumes that different attributes have varying levels of importance. This enables the system to rank the tuples and control the output size. To avoid asking users directly for weights, they offer an alternative approach to discover importance from user feedback, i.e., the user has to partition example tuples to *desirable and undesirable* groups. Our paper requires users to pick one tuple, which is less demanding than partitioning items. Moreover, assuming that tuples can be ranked based on importance of attributes is a strong assumption (e.g., it assumes that a car with 51 MPG and 80 HP is better than a car with 50 MPG and 200HP, if MPG is more important to the user). The same user feedback approach has also been used in [10]. Lee et al. [12] also avoid asking users for utility functions by requesting only partial ranking over attributes.

The interactive regret minimization problem studied in this paper, as well as problems studied in some previous work (e.g. [10,12,18]), share some similarities with the *learning to rank* problem studied heavily in machine learning and information retrieval (see, e.g., [13, 16] for a recent survey). However, the settings of both problems are so different that the techniques used for the ranking problem seem to be inapplicable to our problem. Moreover, while ranking is an extremely hard problem, our problem turns out to have some special structures that lead to a stronger result. The first important difficulty in applying the ranking techniques is that learning to rank approaches typically require much more information from the users (or human trainers). For example, the three most popular approaches [16], namely the *pointwise*, *pairwise* and *listwise* approaches, essentially ask the human trainers to rank the given list of items. This makes it hard to adapt these techniques to the regret minimization problem where the users are only required to pick the most interesting tuple. (This seems to be the case even for previous problems that demand more feedback from users [10, 12, 18].) Another difficulty is that most ranking algorithms are developed in the *supervised* or *semi-supervised* setting (we are aware of very few papers in the *active* setting, e.g., [17]) while the minimum regret problem is *interactive* and highly values the number of interactions. Moreover, the usage in the database setting motivates properties such as truthfulness, informativeness, and progressiveness. It is not clear how to modify the existing ranking algorithms to guarantee these properties.

Fortunately, the interactive regret minimization problem seems to have some features from the database setting which do not exist in the ranking problem. In contrast to machine learning settings where algorithms have to be able to deal with unseen data, we can assume that all data is already available in the database (this type of problem is sometimes called *dynamic search* [12]). Moreover, the notion of regret ratio is concerned with only the *most interesting tuple* in the database while the ranking problem deals with ranking *everything* (even some unseen data). We believe that these differences lead to a more efficient algorithm with several desired properties.

Despite these differences, we believe that insights from machine learning might help to improve the algorithm performance in practice. Moreover, studying the notion of regret ratio in the machine learning setting is interesting in its own right. We leave these as interesting open directions.

3. DEFINITIONS

Recall the following definition of the *regret ratio* defined in [19]. (For readers who are not familiar with the notion of regret ratio, we recommend reading through the detailed explanation of this notion in Section 3.1 of [19].) We are given a database D which is a set of d-dimensional points, i.e., $D \subseteq \mathbb{R}_+^d$ (where $\mathbb{R}_+$ denotes the set of positive reals). For any set $S \subseteq \mathbb{R}_+^d$ and a vector $u \in \mathbb{R}_+^d$, the *regret ratio of* u is defined to be

$$rr_D(S,u) = \max_{p\in D} \max_{q\in S} \frac{\langle u,p\rangle - \langle u,q\rangle}{\langle u,p\rangle} = 1 - \max_{p\in D} \max_{q\in S} \frac{\langle u,q\rangle}{\langle u,p\rangle}$$

where for any $x, y \in \mathbb{R}_+^d$, $\langle x,y\rangle$ is the inner product between x and y, i.e., $\langle x,y\rangle = \sum_{i=1}^d x[i]y[i]$. When D is clear from the context, we simply write $rr(S,u)$ instead of $rr_D(S,u)$. We think of u as a (linear) utility function of a user and p as her most desired point in the database D. (We sometimes call u the *utility vector*.) We think of S as the set of representative points. Thus, for a user with utility function u and desired point p, we try to show the best point q in S so that the utility $\langle u,q\rangle$ is as close to $\langle u,p\rangle$ as possible, i.e., we want to maximize $\frac{\langle u,q\rangle}{\langle u,p\rangle}$ which we sometimes call the happiness ratio. We will focus on minimizing the smaller quantity $\frac{\langle u,p\rangle - \langle u,q\rangle}{\langle u,p\rangle}$ which we call the regret ratio.

We consider the following problem of *interactive regret minimization with r questions of size s*. In this problem, the system can interact with the user with an unknown utility vector for r rounds. In each round, the system displays at most s points to the user and the user returns her favorite point among these s points to the system. After these r rounds, the system can display any number of points to the user. Then, the user picks her favorite point among all points displayed to her. We note that the displayed points might not be in D. However, we require that the algorithm must be *truthful* in the sense that the point that the user picks in the end must be in D. In other words, if S is the set of points displayed to the user in the process and $p = \arg\max_{p'\in S}\langle u,p'\rangle$ is the favorite point of the user, then

the algorithm is truthful if p is in D. As we discussed in Section 1, it is crucial in the database setting that the system is truthful since it guarantees that the users will not select tuples outside D (as our goal is to help the users select tuples *in* D).

We say that any algorithm A *guarantees* ϵ *regret ratio in* r *rounds* if, for any value of u, A always makes the regret ratio at most ϵ within r rounds. Moreover, we say that A *guarantees* ϵ *regret ratio using* k *points* if, for any value of u, A always makes the regret ratio at most ϵ by displaying at most k points (using any number of rounds). We say that A is *informative* if it is able to compute the upper bound of the user's regret ratio in every round. For any $0 < \rho < 1$, we say that A is ρ-*progressive* if it guarantees $O(\rho^r)$ regret ratio in r rounds (for any r). We note that if A is ρ-progressive for some ρ then it is informative since it can output $O(\rho^r)$ as an upper bound of the user's regret ratio in every round r.

Note that the problem considered in [19] is the special case of interactive regret minimization with r questions of size s where $r = 0$ and $s = k$.

In this paper, we are interested in three questions: (1) Given ϵ, how many points do we need to guarantee ϵ regret ratio (given that we can use any number of rounds of questions)? (2) Given k, how small a regret ratio (ϵ) can we guarantee by displaying k points? (3) Given ϵ and s, how many rounds do we need to guarantees ϵ regret ratio if we can display s points in each round?

4. THE UtilityApprox ALGORITHM

In this section we present an algorithm called UtilityApprox (cf. Algorithm 1). We prove a worst case performance guarantee for this algorithm. Moreover we show that it is truthful and progressive (and thus informative), as follows.

Theorem 1. UtilityApprox *algorithm (cf. Algorithm 1) terminates in* $O(d \log_s(d/\epsilon))$ *rounds where in each round it displays at most* s *points. It is truthful. Moreover, it is progressive (and thus informative) with a rate that is exponential in the number of rounds, i.e., in round* t *the maximum regret ratio of the user is* $O(d/s^{\lfloor \frac{t}{d-1} - 1 \rfloor})$.

UtilityApprox is described in Algorithm 1. We now explain the algorithm and intuition behind it along with some lemmas used to prove the theorem above. Throughout, we let u denote the user's utility vector (which is unknown to the algorithm). For simplicity in explaining the algorithm and its intuition, let us assume that $\lambda_i = 1$ for all i (see Line 1 of Algorithm 1). We will show that this can be assumed even in the analysis (as stated in the following lemma). Proofs for all lemmas in this section can be found in Appendix A.

Lemma 2. *We may assume without loss of generality that, for any* $1 \le i \le d$, $\max_{p \in D} p[i] = 1$.

The main idea of this algorithm is to *approximate* the utility vector of the user. We use v to denote this approximated utility vector. The goal of the algorithm is to minimize $\sum_{i=1}^{d} |u[i] - v[i]|$. We will show that if this quantity is small, we can use v to find a point that makes the regret ratio for u small.

It is useful to imagine that there are two types of users, the *real* user with utility vector u and an imaginary user with utility vector v. We abuse notation slightly and denote these users simply by u and v. The algorithm will display points based on the point that v prefers the most and use the answer of u to update the value of v so that the new value of v is a better approximation of u. The process is as follows.

Algorithm 1 UtilityApprox (D, s)

1: For $1 \le i \le d$, let $\lambda_i = \max_{p \in D}, p[i]$.
2: For any $1 \le i \le d$, let e_i be such that $e_i[i] = \lambda_i$ and $e_i[j] = 0$ for all $j \ne i$. Let $e_i = e_d$ for all $i > d$.
3: Let $i^* = 1$ and $i = 2$.
4: **while** $i \le d$ **do**
5: Display e_{i^*} and $e_i, e_{i+1}, \ldots, e_{i+s-2}$.
6: Let i' be the point such that $e_{i'}$ is the point returned by the user. Let $i^* = i'$.
7: Let $i = i + s - 1$.
8: **end while**
9:
10: Let $U_{i^*} = L_{i^*} = 1$. For any $i \ne i^*$, let $U_i = 1$ and $L_i = 0$.
11: Let $t = 1$.
12: **repeat**
13: This is the t^{th} round.
14: Let $i = (t \mod d) + 1$. If $i = i^*$ then, again, let $i = (t \mod d) + 1$.
15: Let v be a vector such that $v[j] = (U_j + L_j)/2$, for all $1 \le j \le d$.
16: Let $p_1, p_2, \ldots, p_s$ be points returned from CreatePoints algorithm (cf. Algorithm 2).
17: Display $p_1, p_2, \ldots, p_s$ to the user.
18: **if** the user does not terminate **then**
19: Let p_α be the point the the user picks.
20: Set $L_i = \chi_{\alpha-1}$ and $U_i = \chi_\alpha$ where $\chi_{\alpha-1}$ and χ_α are defined in Algorithm 2.
21: Let $t = t + 1$.
22: **else**
23: Terminate the loop
24: **end if**
25: **until** the user terminates
26: Let v be as defined in Line 15. Display $p^* = \arg\max_{p \in D} \langle v, p \rangle$.

The algorithm first finds the coordinate with the largest value, i.e., it finds $i^* = \arg\max_i \lambda_i u[i]$. As described from Line 3 to Line 8, this is done by displaying $(\lambda_1, 0, 0, \ldots)$, $(0, \lambda_2, 0, \ldots)$, …, $(0, 0, \ldots, \lambda_d)$. The algorithm then defines U_i and L_i, for every $1 \le i \le d$. These U_i and L_i are used as upper and lower bounds of $u[i]$ respectively. To set the initial values of U_i and L_i, we use the following lemma.

Lemma 3. *We may assume without loss of generality that* $\max_{1 \le i \le d} u[i] = 1$.

In other words, we can assume that $u[i^*] = 1$ and $0 \le u[i] \le 1$. Thus we initially set the values of U_i and L_i as stated in Line 10.

In the next step, from Lines 12 to 25, we repeatedly reduce the gap between U_i and L_i, for all $i \ne i^*$. We do this by displaying s carefully chosen points, $p_1, \ldots, p_s$, so that after one of these points is chosen by the user, we can reduce the gap between U_i and L_i by a factor of s. (We will explain shortly how to pick these points.) In particular, we show the following lemma.

Algorithm 2 CREATEPOINTS

1: Find $q' = \arg\max_{q' \in D} \langle v, q' \rangle$.
2: Define q as follows. For $1 \le i \le d$, let $q[i] = q'[i]/\lambda_i$.
3: For any $0 \le \alpha \le s$, let $\chi_\alpha = L_i + \alpha(U_i - L_i)/s$.
4: Let $\delta = \min(10^{-5}, \frac{1}{s^{\lfloor \frac{t}{d-1} \rfloor} \sum_{\alpha'=1}^{s-1} (\chi_{\alpha'} - L_i)})$. (Note that, in practice, the smaller δ is, the better.)
5: Let $\gamma = q[i^*]/q[i]$.
6: Define $q_1, \ldots, q_s$ as follows. Let $q_s = q$. For $\alpha = 1, ..., s-1$, let q_α be such that $q_\alpha[j] = q_{\alpha+1}[j]$ for all $j \notin \{i, i^*\}$, $q_\alpha[i^*] = q_{\alpha+1}[i^*] + \chi_\alpha \delta \gamma q_{\alpha+1}[i]$, and $q_\alpha[i] = q_{\alpha+1}[i] - \delta\gamma q_{\alpha+1}[i]$.
7: Let $\beta = \frac{1}{1+\delta \sum_{\alpha'=1}^{s-1} (\chi_{\alpha'} - L_i)}$.
8: Define $p_1, \ldots, p_s$ as follows. For $1 \le \alpha \le s$, and $1 \le i \le d$ let $p_\alpha[i] = \beta \lambda_i q_\alpha[i]$.

LEMMA 4. *At the* $(\lceil (d-1)/(s-1) \rceil + t)^{th}$ *round, for any* $1 \le i \le d$, $|U_i - L_i| = O(1/s^{\frac{t}{d-1}})$.

We note that the number $(\lceil (d-1)/(s-1) \rceil + t)$ comes from the fact that we need this many rounds to find i^* (cf. Lines 3 to 8).

Finally, when the user terminates the process, the algorithm computes a vector v which approximates the user's utility vector u. It then outputs a point in the database D that v prefers the most (cf. Line 26). We will show later that this point also makes u have small regret ratio. In particular, if v approximates u well, giving q to u will not make the regret ratio of u too high, as formalized in the following lemma.

LEMMA 5. *If* $q = \arg\max_{q' \in D} \langle v, q' \rangle$ *then* $rr_D(q, u) \le 2 \sum_{i=1}^{d} |u[i] - v[i]|$.

Picking points to display. We now explain how to pick the points $p_1, \ldots, p_s$ to maintain some desired properties. We use CREATEPOINTS algorithm (cf. Algorithm 2) to do this. First, we pick a point q using an approximated vector v, as in Lines 1-2. In the case where $\lambda_i = 1$ for all i, q is simply the favorite point of v in D. As v will be an approximation of u, we will be able to guarantee that q makes the regret ratio of the real user (with utility vector u) small, as in Lemma 5.

Then, we construct s points from point q, denoted by $q_1, \ldots, q_s$ (cf. Line 6). These points are constructed by slightly changing the values in the i^{th} and $(i^*)^{th}$ coordinate of q, where i is the current coordinate where we are trying to reduce the gap between U_i and L_i (as in Algorithm 1). How "slightly" we change these values is controlled by a parameter δ (cf. Line 4). The amount that these values change is different when we define different q_α. The parameter that controls this amount is χ_α (cf. Line 3).

The value of χ_α is picked in such a way that, for any α, $\chi_\alpha - \chi_{\alpha-1} = (U_i - L_i)/s$. Now suppose that we display points $q_1, \ldots, q_d$ to the user. We show the following.

LEMMA 6. *If the user picks point* q_α *among* $q_1, \ldots, q_s$ *then* $\chi_{\alpha-1} \le u[i] \le \chi_\alpha$.

Thus, our goal of reducing $U_i - L_i$ by a factor of s can be achieved by setting $U_i = \chi_\alpha$ and $L_i = \chi_{\alpha-1}$. Moreover, since $q_s = q$ is claimed to make the regret ratio of the user small, the regret ratio of the user (for the displayed points) will be small. In other words, the algorithm is progressive when we display $q_1, \ldots, q_s$.

However, displaying $q_1, \ldots, q_s$ makes the algorithm untruthful. This is because the user might prefer one of $q_1, \ldots, q_s$ (which are fake points) more than any point from the database. A simple way to resolve this problem is to scale down $q_1, \ldots, q_s$ by a large factor so that the user's utility on them decreases dramatically and thus the user prefers some point in D more. However, the displayed points will not make the regret ratio of the user small and thus the algorithm will not be progressive anymore. Therefore, we have to scale these points down only slightly so that the user's regret ratio is still small. Fortunately, this delicate balancing can be done using the fact that the user's utility on $q_1, \ldots, q_s$ is not much larger than her favorite point in the database. This is formalized below.

LEMMA 7. *Let* q *and* $q_1, \ldots, q_s$ *be as in Algorithm 2. For any* α,

$$\begin{aligned}\langle u, q_\alpha \rangle &\le (1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])) \langle u, q \rangle \\ &\le (1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])) \max_{p \in D} \langle u, p \rangle .\end{aligned}$$

We use a parameter β (cf. Line 7) to control how much we scale the displayed points down. Its value depends on the quantity in Lemma 7. The points resulting from scaling $q_1, \ldots, q_s$ down are $p_1, \ldots, p_s$ (cf. Line 8).

We are ready to prove the main theorem of this section, using the lemmas stated above.

PROOF OF THEOREM 1. **Truthfulness:** Let p be the point that the user prefers the most among points displayed to her. We claim that $p = p^*$ where p^* is as in Line 26 of Algorithm 1. To see this, let S_t be the the set of displayed points in round t. First observe that among points in $\bigcup_t S_t$, the favorite point of the user is in S_{t^*} where t^* is the last round of the algorithm before the user terminates. This is because this is the first time that the user's regret ratio is below ϵ. Now, let p_α be this point in S_{t^*}. Let q^* be the point q defined in the last round. By Lemma 7 and the definition of p_α,

$$\begin{aligned}\langle u, p_\alpha \rangle &= \beta \langle u, q_\alpha \rangle \\ &\le \beta (1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])) \langle u, q^* \rangle \\ &\le \langle u, q^* \rangle\end{aligned}$$

where β is as in Line 7 of Algorithm 2 and the last inequality follows from

$$\beta = \frac{1}{1 + \delta \sum_{\alpha'=1}^{s-1} (\chi_{\alpha'} - L_i)} \le \frac{1}{1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])} .$$

Thus, the user's favorite point, among $q_1, \ldots q_s$ is q^*. Therefore her favorite point is p^*.

Progressiveness: We prove here that for any t, $rr_D(S_t, u) = O(d/s^{\lfloor \frac{t}{d-1} \rfloor})$. Since we need $\lceil (d-1)/(s-1) \rceil$ rounds to find i^*, it follows that after $t = \lceil (d-1)/(s-1) \rceil + t'$ rounds, the regret ratio of the user is

$$O(d/s^{\lfloor \frac{t'}{d-1} \rfloor}) = O(d/s^{\lfloor \frac{t - \lceil (d-1)/(s-1) \rceil}{d-1} \rfloor}) = O(d/s^{\lfloor \frac{t}{d-1} - 1 \rfloor})$$

as claimed. It is thus left to show that $rr_D(S_t, u) = O(d/s^{\lfloor \frac{t}{d-1} \rfloor})$. Observe that in the t^{th} round (as defined in Line 13 of Algorithm 1), the value of $U_i - L_i$ is decreased to $\chi_\alpha - \chi_{\alpha-1}$, for some $1 \leq \alpha \leq s$. By definition, for any $1 \leq \alpha \leq s$, $\chi_\alpha - \chi_{\alpha-1} = (U_i - L_i)/s$. Thus, we conclude that after the t^{th} round,

$$\sum_{i=1}^{d}(U_i - L_i) = \frac{d - \lfloor t/(d-1) \rfloor}{s^{\lfloor \frac{t}{d-1} \rfloor}} + \frac{\lfloor t/(d-1) \rfloor}{s^{\lfloor \frac{t}{d-1} \rfloor + 1}} = O(d/s^{\lfloor \frac{t}{d-1} \rfloor}) .$$

Now using Lemma 5, for the value of v and q defined in round t, we have $rr_D(q, u) \leq 2\sum_{i=1}^{d} |u[i] - v[i]|$. In other words,

$$\langle u, q \rangle \geq (1 - 2\sum_{i=1}^{d} |u[i] - v[i]|) \max_{p \in D} \langle u, p \rangle .$$

By Lemma 6, $L_i \leq u[i] \leq U_i$. Since $v[i] = ((U_i + L_i)/2)e_i$, $|u[i] - v[i]| \leq (U_i - L_i)/2$, for all $i \leq i^*$. It follows that

$$\langle u, q \rangle \geq (1 - O(d/s^{\lfloor \frac{t}{d-1} \rfloor})) \max_{p \in D} \langle u, p \rangle .$$

Observe further that

$$\begin{aligned}
\langle u, p_s \rangle &= \beta \langle u, q \rangle \\
&\geq \frac{(1 - O(d/s^{\lfloor \frac{t}{d-1} \rfloor})) \max_{p \in D} \langle u, p \rangle}{1 + \delta \sum_{\alpha'=1}^{s-1} (\chi_{\alpha'} - L_i)} \\
&\geq (1 - \delta \sum_{\alpha'=1}^{s-1} (\chi_{\alpha'} - L_i) - O(d/s^{\lfloor \frac{t}{d-1} \rfloor})) \max_{p \in D} \langle u, p \rangle \\
&\geq (1 - O(d/s^{\lfloor \frac{t}{d-1} \rfloor})) \max_{p \in D} \langle u, p \rangle
\end{aligned}$$

where we use $1/(1+c) \geq 1 - c$ for any $0 \leq c < 1$ and $\delta \sum_{\alpha'=1}^{s-1} (\chi_{\alpha'} - L_i) \leq d/s^{\lfloor \frac{t}{d-1} \rfloor}$. In other words, $rr_D(p_k, u) = O(d/s^{\lfloor \frac{t}{d-1} \rfloor})$. It follows that

$$rr_D(S_t, u) \leq rr_D(p_s, u) = O(d/s^{\lfloor \frac{t}{d-1} \rfloor})$$

as desired.

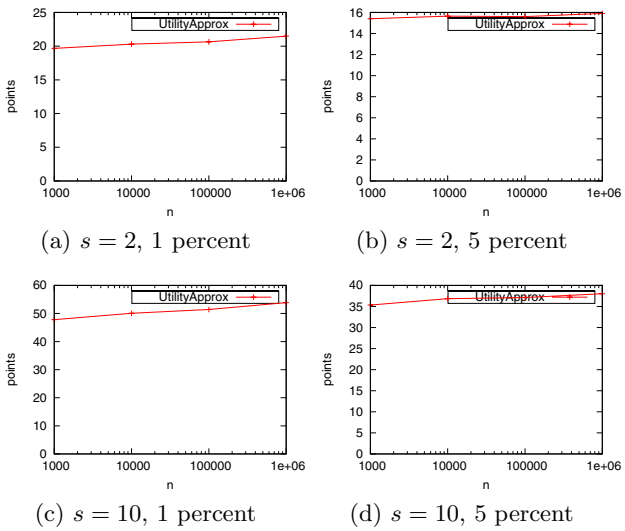
Termination time guarantee: Since, for any t, $rr_D(S_t, u) = O(d/s^{\lfloor \frac{t}{d-1} \rfloor})$, the user's regret ratio in round $t = d \log_s(d/\epsilon)$ is $O(\epsilon)$. By showing points for constant additional rounds, the user's regret ratio will be below ϵ. Thus, the total number of rounds we need before the user terminates is $O(d \log_s(d/\epsilon))$, as claimed. □

5. LOWER BOUNDS

In this section, we study the lower bounds of both the cases of whether or not there are interactions. Proofs of theorems in this section can be found in Appendix A. First, we show that when there are no interactions, we need to display a large number of points in order to achieve a small regret ratio.

THEOREM 8. *For any d and $0 < \epsilon \leq 1$, there is a database of d-dimensional points such that any single-round algorithm needs to display at least $\frac{1}{2}(\frac{1}{8\epsilon})^{\frac{d-1}{2}}$ points in order to guarantee that the regret ratio is at most ϵ.*

Second, we show that the algorithm we propose is almost optimal. The theorem below generalizes the above theorem.



(a) $s = 2$, 1 percent (b) $s = 2$, 5 percent

(c) $s = 10$, 1 percent (d) $s = 10$, 5 percent

Figure 1: Number of points needed by UtilityApprox to achieve target regret ratio (1% and 5%) using $s = 2$ and $s = 10$ points per page with varying database size

THEOREM 9. *For any d, $s \geq 2$, and t, the regret ratio after asking t rounds of questions of s points is at least $\frac{1}{8(4s^t)^{\frac{2}{d-1}}}$. In other words, for any ϵ, to get to a regret ratio of ϵ, any algorithm needs to ask at least $\Omega(d \log_s(1/\epsilon))$ questions.*

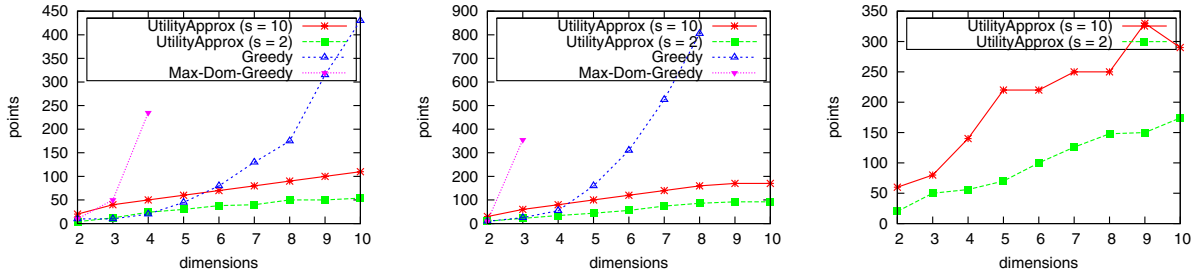
Note that the above theorem holds even for algorithms that do not guarantee the properties defined in this paper (such as truthfulness).

6. EXPERIMENTAL EVALUATION

In this section we experimentally demonstrate the large gains that interactivity adds to this problem, both in terms of giving smaller regret for the same number of points, and requiring smaller number of points for the same regret guarantee. We evaluated UtilityApprox (Algorithm 1) against the best known single-round algorithms [19], varying many different parameters to understand when it is significantly beneficial to query the user over multiple rounds. All the algorithms were implemented in C and run on an Intel Core 2 Duo running Ubuntu 10.04.1.

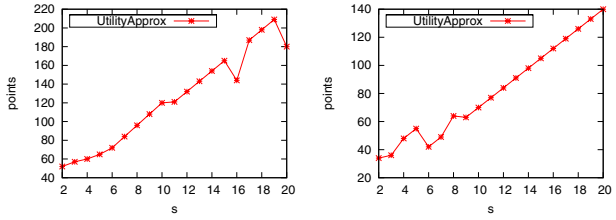
We performed experiments on both synthetic and real data sets. For the synthetic data we used anti-correlated data using the data set generator of [1]. As observed in [19], the anti-correlated case is most interesting for this problem (as compared with, say, correlated or independent data) since this is when the skyline is large and cannot be shown to the user in its entirety. In all these experiments, unless otherwise specified, we used default values of ten thousand points ($n = 10000$), six dimensions ($d = 6$), and regret ratio guarantees of 1% and 5%. We also used real data sets available from Dr. Yufei Tao's homepage [1].

[1] http://www.cse.cuhk.edu.hk/∼taoyf/



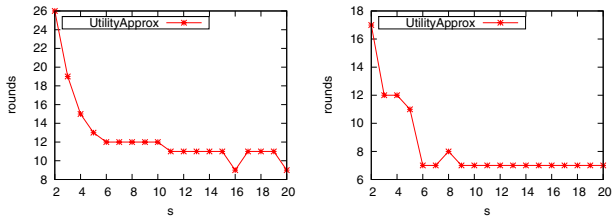
(a) 5 percent (b) 1 percent (c) 0 percent

Figure 4: Number of points needed to achieve a target regret ratio (5%, 1%, and 0%) for $n = 10000$ points



(a) 1 percent (b) 5 percent

Figure 2: Number of points needed by UtilityApprox to achieve target regret ratio (1% and 5%) for varying number of points per page ($n = 10000$ points)



(a) 1 percent (b) 5 percent

Figure 3: Number of rounds needed by UtilityApprox to achieve target regret ratio (1% and 5%) for varying number of points per page ($n = 10000$ points)

We compare the UtilityApprox algorithm against the following single-round algorithms that were shown to give the smallest regret ratio in [19]:

- the Greedy algorithm proposed in [19] that was designed to give a small regret ratio for the single-round case, and
- the Max-Dom-Greedy [15, Algorithm 1] algorithm that was also found to often give small regret even though it was not designed for this purpose.

While we realize that this is an unfair comparison in that our algorithm has the advantage of interactivity, the purpose of these results is to show the benefit of allowing multiple rounds. Another inequity in this comparison is that, while we are able to compute the precise regret guarantee for the single-round algorithms (using techniques described in [19]), we are only able to give an approximation of the maximum regret of UtilityApprox by taking the maximum over many different random preference functions. We found in our experiments that using ten thousand independent preference functions gave a stable estimate for the maximum regret.

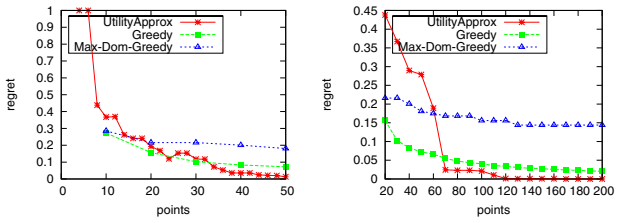
We found that the UtilityApprox algorithm runs very fast since it only performs a single pass over all the points in the database per round; even for one million points and six dimensions it took less than 50 milliseconds to process each round, so we do not show any timing results in this paper.

We began our experiments by varying the size of the database. Our expectation was that the number of points in the database should have little effect on the performance of our algorithm since the database size is not a significant factor in the analyses. This was confirmed in our experiments, as seen in Figure 1. We tried both $s = 2$ and $s = 10$ points per round, and let UtilityApprox run until it went under both 1% and 5% regret. In all cases, the total number of points did not vary significantly with the number of points in the database, n. Hence, for the remainder of the experiments we used a fixed value of $n = 10000$.

Next, we varied the only free parameter for the UtilityApprox algorithm: the number of points that are shown per round (s). The result of this experiment is shown in Figure 2, for the synthetic data set with ten thousand points and with target regret ratios of 1% and 5%. From both graphs we see an increasing trend in which the total number of points needed (i.e., the number of rounds times the number of points per round) grows with the number of points shown in each round. This seems to indicate that using fewer points per round (all the way down to the extreme of $s = 2$) is more beneficial to this algorithm. However, in practice it is sometimes the case that we may need to show more points in each round. Hence, for the remaining experiments we show results for both $s = 2$ and $s = 10$.

Figure 3 shows the number of rounds needed by UtilityApprox when the number of points per round is varied. We see that, in most cases, increasing the number of points shown per page decreases the number of rounds of the algorithm.

In Figure 4, we compare the aforementioned algorithms for various number of dimensions. We made the single-round algorithms comparable to UtilityApprox by showing the total number of points shown by UtilityApprox, as before. We used $n = 10000$ points and ran all the algorithms until they achieved a regret ratio of 5%, 1%, and 0%. For the two single-round algorithms, we picked a ceiling of one thousand



(a) $s = 2$ (b) $s = 10$

Figure 5: Regret of UtilityApprox when the number of rounds was fixed beforehand (with $s = 2$ and 10), compared with other single-round algorithms ($n =$ 10000 points, $d = 6$ dimensions)

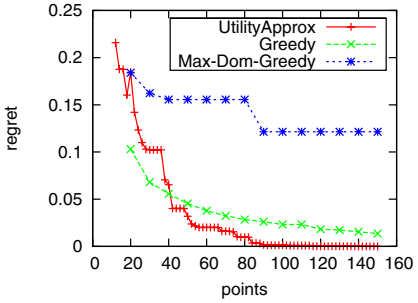


Figure 6: Regret of UtilityApprox when the number of rounds was fixed beforehand (with $s = 2$), compared with other single-round algorithms on the Color data set ($n = 68040$, $d = 9$)

Target Regret Ratio	UtilityApprox	Single-round
5%	42	47
1%	78	190
0%	140	3460

Table 1: Number of points needed to guarantee different regret ratios for Color data set (UtilityApprox had $s = 2$)

Dataset	n	d	UtilityApprox	Single-round
House	127931	6	96	49
NBA	17264	5	68	253
Island	63383	2	14	206

Table 2: Number of points needed to guarantee zero regret for real data sets (UtilityApprox had $s = 2$)

points (i.e., 10% of the total number of points) and didn't run them for larger s. This is why the curves for these two algorithms end prematurely: they very rapidly need more than one thousand points to achieve these regret ratio guarantees. In contrast, UtilityApprox grows only linearly in the number of dimensions and has a small slope. Hence, for higher dimensions there is a very large benefit in having user interactivity. Also, in Figure 4(c) we see that UtilityApprox eventually gets the optimal point for any user (for $s = 2$ and 10), which the single-round algorithms cannot do.

In practice, it may be the case that we cannot have an unlimited number of rounds in which to get information from the user. Figure 5 shows the performance of UtilityApprox when the number of rounds of interaction is fixed beforehand. We measured the total number of points (instead of number of rounds) to make UtilityApprox comparable once again to the single-round algorithms. In both these graphs we see a rapid drop in the regret ratio when the number of rounds allowed goes above some small number. The reason for this lies in the way that UtilityApprox works: it sequentially improves its estimate of each of the user's attribute weights and only begins to give a good estimate once it has some approximation to some (or all) these weights. As a result, UtilityApprox works best when it is allowed at least some minimum number of rounds in which to operate.

Finally, we ran the UtilityApprox algorithm along with the two single-round algorithms on four real data sets commonly used in the skyline literature: House, NBA, Island, and Color. Many papers in the literature use these data sets. [19] studied all these data sets and showed that the single-round algorithms perform very well on House, NBA and Island (the best algorithm achieves the regret ratio of almost 0% by showing just 10 points) while the results are only acceptable for the case of Color data set (the best algorithm achieves the regret ratio of roughly 20% when they show 10 points). This is because Color has highest dimension (thus suffers most from the curse of dimensionality).

In the case of this Color data set, we found that UtilityApprox performs better than single-round algorithms even when the target regret ratio is 5%, as shown in Table 1. Moreover, for smaller target regret ratios, UtilityApprox clearly outperforms the single-round algorithms. The same phenomenon occurs when we increase the number of points shown to the users, as shown in Figure 6. This confirms that interactions are crucial in making the regret ratio small.

We also study other data sets where previous algorithms already perform well. As expected, we found that the UtilityApprox algorithm did not perform as well as the single-round algorithms for small number of points. This is to be expected since the single round algorithms are optimized to give very small regret ratio for very few points displayed in a single round while the UtilityApprox algorithm is designed to sacrifice some points in order to decrease the regret ratio rapidly in the long run. For this reason, when we increased the number of points and the number of rounds of our algorithm, UtilityApprox found the optimal point in the database rapidly, while the single round algorithms usually had positive regret for many more points. (We note an exception for the case of House data set where the number of relevant points are very small.) Table 2 shows how many points the UtilityApprox algorithms had to display to guarantee zero regret as compared with the lower bound for *any* single-round algorithm. The lower bound for single-round algorithms was computed by using a linear program to determine which points were the optimal for *some* utility function; the number of these points was used as the presented lower bound. We conclude that, besides being crucial in making the regret ratio small, interactions can also be used to quickly find users' best tuples in databases, even when the

regret ratios achieved by single-round algorithms are already small.

7. CONCLUSIONS

We study the problem of minimizing the regret ratio for a user seeking her favorite point from a database after rounds of *interaction*. We only assume that the user is able to pick her favorite tuple when presented with a set of tuples. This model generalizes previous work in the domains of top-k and skyline queries. Our paper focuses on the limits of representing databases when no interaction is allowed, and presents significantly improved results in our model of interaction. Theoretical and experimental results show significant improvement over previous work, suggesting that user interaction might be an integral part of multi-criteria decision-making in database systems.

While the experiment results confirm the efficiency of interactions, they also reveal some weaknesses of our algorithm. First, the improvement is not as dramatic when the data are more correlated (e.g., in some real data sets with small number of attributes). More importantly, our algorithm requires a few rounds of interaction in order to beat the single-round algorithms. We believe that improving our algorithm on these aspects is an interesting open question.

Several other directions also remain open for future work. One interesting unexplored direction is adapting machine learning techniques to our problems. As we mentioned in Section 2, this seems to be a non-trivial task. Another direction is dealing with ambiguities and noise which could occur in reality. For example, a user might not always pick the best tuple if there are two tuples whose qualities are not very different or she might sometimes mistakenly pick a tuple that she is not interested in at all.

Moreover, as we noted in Section 1, one desirable property lacking in our algorithm is displaying only tuples in the database. This property is crucial in some settings (e.g., when some attributes such as "look of the car" cannot be generated by the algorithm), and it would be interesting to develop an algorithm that satisfies this property while guaranteeing an upper bound similar to the one in this paper, or showing that this is impossible. Also, it would be interesting to extend some of these techniques and results to hold for non-linear utility functions of users. Finally, we measure the cost/effort on the user's front mainly in terms of number of points displayed, or number of rounds of interaction; perhaps more elaborate models can be considered in terms of the user's cost for answering various types of interactive questions.

Acknowledgement: The authors would like to thank Bobby Kleinberg for his guidance towards the proofs presented in the lower bounds section. We also thank anonymous referees for many useful comments.

8. REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[2] 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.

[3] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, 2006.

[4] 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.

[5] A. Das Sarma, A. Lall, D. Nanongkai, R. J. Lipton, and J. Xu. Representative skylines using threshold-based preference distributions. In *ICDE*, 2011.

[6] 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.

[7] M. Goncalves and M.-E. Vidal. Top-k skyline: A unified approach. In *OTM Workshops*, 2005.

[8] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, 2001.

[9] 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.

[10] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han. Mining preferences from superior and inferior examples. In *KDD*, pages 390–398, 2008.

[11] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos. Continuous top-k dominating queries in subspaces. In *Panhellenic Conference on Informatics*, 2008.

[12] 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.

[13] H. Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers, Apr. 2011.

[14] X. Lian and L. C. 0002. Top-k dominating queries in uncertain databases. In *EDBT*, 2009.

[15] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.

[16] T.-Y. Liu. *Learning to Rank for Information Retrieval*. Springer-Verlag New York Inc, 2011.

[17] B. Long, O. Chapelle, Y. Zhang, Y. Chang, Z. Zheng, and B. L. Tseng. Active learning for ranking through expected loss optimization. In *SIGIR*, pages 267–274, 2010.

[18] D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *PVLDB*, 2(1):610–621, 2009.

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

[20] A. N. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos. Domination mining and querying. In *DaWaK*, 2007.

[21] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.

[22] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, 2003.

[23] T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *ICDE*, 2008.

[24] M. L. Yiu and N. Mamoulis. Multi-dimensional top-dominating queries. *VLDB J.*, 18(3):695–718, 2009. Also VLDB'07.

[25] 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. OMITTED PROOFS

For the sake of the simplicity of the rest analysis, let us first prove the following lemma. It states that if the algorithm is correct and can guarantee its performance on a database with a certain property (i.e., database D such that for any $1 \leq i \leq d$, $\max_{p\in D} p[i] = 1$), then it is also correct and can guarantee its performance on other types of database.

LEMMA 2 (RESTATED). *We may assume without loss of generality that, for any* $1 \leq i \leq d$, $\max_{p\in D} p[i] = 1$.

PROOF. Let D be the original database and $u \in \mathbb{R}^d_+$ be the user's original utility vector. For any $1 \leq i \leq d$, let $\lambda_i = \max_{p\in D} p[i]$. For any point $p \in D$, let $f(p)$ be the point $(p[1]/\lambda_1, p[2]/\lambda_2, \ldots, p[d]/\lambda_d)$. Let $D' = \{f(p) \mid p \in D\}$. Let u' be such that $u'[i] = \lambda_i u[i]$.

OBSERVATION 10. *For any* $p \in D$, $\langle u, p\rangle = \langle u', f(p)\rangle$.

To see this, observe that $\langle u, p\rangle = \sum_{i=1}^d p[i]u[i] = \langle u', f(p)\rangle$.

Thus, we can assume that the database is D' and the utility vector of the user is u' instead since the user's utility on every point remains the same. □

Thus, from now on, we assume that the input database D is such that, for any $1 \leq i \leq d$, $\max_{p\in D} p[i] = 1$.

We now prove Lemma 3.

LEMMA 3 (RESTATED). *We may assume without loss of generality that* $\max_{1\leq i\leq d} u[i] = 1$

PROOF. Let D be the original database satisfying Lemma 2 and $u \in \mathbb{R}^d_+$ be the user's original utility vector. Let $\alpha = \max_{i\in[1,d]} u[i]$. Let u' be such that $u'[i] = u[i]/\alpha$, for all i.

OBSERVATION 11. *For any points* $p, q \in D$, $\langle u', p\rangle > \langle u', q\rangle$ *if and only if* $\langle u', p\rangle > \langle u', q\rangle$ *and* $\langle u, p\rangle = \langle u, q\rangle$ *if and only if* $\langle u', p\rangle = \langle u', q\rangle$. *Moreover, for any* $S \subseteq D$, $rr_D(S, u) = rr_D(S, u')$.

PROOF. The first observation is obvious. The second observation can be proved as follows.

$$\begin{aligned} rr_D(S,u) &= \frac{\max_{p\in D}\langle u,p\rangle - \max_{q\in S}\langle u,q\rangle}{\max_{p\in D}\langle u,p\rangle} \\ &= \frac{\alpha\max_{p\in D}\langle u',p\rangle - \alpha\max_{q\in S}\langle u',q\rangle}{\alpha\max_{p\in D}\langle u',p\rangle} \\ &= rr_D(S,u') \end{aligned}$$

□

Thus, we can assume that the user's utility vector is u' since for any set $S \subseteq D$, p is the favorite point of u in S if and only if it is the favorite point of u' in S, and $rr_D(S, u) = rr_D(S, u')$. □

We note the following lemma which follows from the above lemmas. It will be useful in proving some lemmas below.

LEMMA 12. *We may assume that* $\max_{p\in D}\langle u, p\rangle \geq 1$.

PROOF. Let i^* be such that $u[i^*] = 1$. (i^* exists by Lemma 3). Note that by Lemma 2, there exists $p \in D$ such that $p[i^*] = 1$. Thus, $\max_{p\in D}\langle u, p\rangle \geq \langle u, p^*\rangle \geq 1$. □

We now prove Lemma 4 stated in Section 4.

LEMMA 4 (RESTATED). *At the* $(\lceil (d-1)/(s-1)\rceil + t)^{th}$ *round, for any* $1 \leq i \leq d$, $|U_i - L_i| = O(1/s^{\frac{t}{d-1}})$.

PROOF. Observe that the user picks p_α among $p_1, \ldots, p_s$ if and only if she picks q_α among $q_1, \ldots, q_s$. Thus, if the user picks q_α then, by Lemma 6, $\chi_{\alpha-1} \leq u[i] \leq \chi_\alpha$ which means that $|U_i - L_i|$ decreases by a factor of s. After t rounds, the value of $U_i - L_i$ is decreased by a factor of s for $\lfloor \frac{t}{d-1} \rfloor$ times and thus $|U_i - L_i| = O(1/s^{\frac{t}{d-1}})$ as claimed. □

LEMMA 5 (RESTATED). *If* $q = \arg\max_{q'\in D}\langle v, q'\rangle$ *then* $rr_D(q, u) \leq 2\sum_{i=1}^d |u[i] - v[i]|$.

PROOF. First, we note the following claim which compares happiness when two users look at the same point.

CLAIM 13. *For any vector* u *and* v *in* $\mathbb{R}_+$, *and any point* $z \in D$, $|\langle v, z\rangle - \langle u, z\rangle| \leq \sum_{i=1}^d |v[i] - u[i]|$.

PROOF. Using the fact that $z[i] \leq 1$ for all i (by Theorem 2), we have

$$|\langle v,z\rangle - \langle u,z\rangle| = \sum_{i=1}^d (v[i]-u[i])z[i] \leq \sum_{i=1}^d |v[i]-u[i]|.$$

□

Let $p = \arg\max_{p'\in D}\langle u, p'\rangle$. Applying the above claim, we have the following two inequalities.

$$\langle v, q\rangle - \langle u, q\rangle \leq \sum_{i=1}^d |v[i] - u[i]| \tag{1}$$

$$\langle u, p\rangle - \langle v, p\rangle \leq \sum_{i=1}^d |v[i] - u[i]| \tag{2}$$

Moreover, by the definition of q, i.e. $q = \arg\max_{p'\in D}\langle v, p'\rangle$, we have that $\langle v, p\rangle \leq \langle v, q\rangle$; in other words,

$$\langle v, p\rangle - \langle v, q\rangle \leq 0. \tag{3}$$

Adding Equations (1), (2), (3) together, the lemma follows. □

Next, we prove Lemma 6.

LEMMA 6 (RESTATED). *If the user picks point* q_α *among* $q_1, \ldots, q_s$ *then* $\chi_{\alpha-1} \leq u[i] \leq \chi_\alpha$.

PROOF. Let $u \in \mathbb{R}^d$ be the user's utility vector. If q_α is picked then

$$\langle u, q_\alpha\rangle \geq \langle u, q_{\alpha+1}\rangle.$$

Note that

$$\langle u, q_\alpha\rangle = \langle u, q_{\alpha+1}\rangle + \chi_\alpha\delta\gamma q_{\alpha+1}[i]u[i^*] - \delta\gamma q_{\alpha+1}[i]u[i].$$

Thus, we have $\chi_\alpha\delta\gamma q_{\alpha+1}[i]u[i^*] - \delta\gamma q_{\alpha+1}[i]u[i] \geq 0$. Using $u[i^*] = 1$, we have $u[i] \leq \chi_\alpha$.

By a similar argument on the fact that $\langle u, q_\alpha\rangle \geq \langle u, q_{\alpha-1}\rangle$, we have $u[i] \geq \chi_{\alpha-1}$ as desired. □

We now prove Lemma 7 stated in Section 4.

LEMMA 7 (RESTATED). *Let q and $q_1, \ldots, q_s$ be as in Algorithm 2. For any α,*

$$\begin{aligned}\langle u, q_\alpha \rangle &\leq (1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i]))\langle u, q \rangle \\ &\leq (1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])) \max_{p \in D} \langle u, p \rangle .\end{aligned}$$

PROOF. Note that

$$\begin{aligned}\langle u, q_\alpha \rangle &= u[i^*](q_{\alpha+1}[i^*] + \chi_\alpha \delta\gamma q_{\alpha+1}[i]) \\ &\quad + u[i](q_{\alpha+1}[i] - \delta\gamma q_{\alpha+1}[i]) + \sum_{j \neq i, i^*} u[j]q[j] \\ &= \langle u, q_{\alpha+1} \rangle + (\chi_\alpha u[i^*] - u[i])\delta\gamma q_{\alpha+1}[i] \\ &= \langle u, q_{\alpha+1} \rangle + (\chi_\alpha - u[i])\delta\gamma q_{\alpha+1}[i]\end{aligned}$$

where the last equality is because $u[i^*] = 1$. Thus,

$$\begin{aligned}\langle u, q_\alpha \rangle &= \langle u, q \rangle + \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])\delta\gamma q_{\alpha'+1}[i] \\ &\leq \langle u, q \rangle + \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i])\delta\gamma q[i] \\ &\leq (1 + \delta \sum_{\alpha'=\alpha}^{s-1} (\chi_{\alpha'} - u[i]))\langle u, q \rangle\end{aligned}$$

where the second inequality is because $q_\alpha[i] \leq q[i]$ for all i, and the last inequality uses the fact that $\gamma q[i] = q[i^*] \leq \langle u, q \rangle$. The first inequality in the statement follows. The second inequality follows from the fact that $\langle u, q \rangle \leq \max_{p \in D} \langle u, p \rangle$. □

THEOREM 8. *For any d and $0 < \epsilon \leq 1$, there is a database of d-dimensional points such that any algorithm needs to display at least $\frac{1}{2}(\frac{1}{8\epsilon})^{\frac{d-1}{2}}$ points in order to guarantee that the regret ratio is at most ϵ.*

PROOF. For any point p in R^d, let $||p|| = \sqrt{\langle p, p \rangle}$. For any $r \geq 0$ and $r \in \mathbb{R}$, we let $B_r^d(p)$ denote a d-dimensional ball of radius r centered at p, i.e.,

$$B_r^d(p) = \{x \in \mathbb{R} \mid ||x - p|| \leq r\}$$

and $S_r^d(p)$ denote the surface of $B_r^d(p)$, i.e.,

$$S_r^d(p) = \{x \in \mathbb{R} \mid ||x - p|| = r\} .$$

Let D be a set of all points on the surface of a d-sphere of radius one with values in all coordinates positive, i.e., $D = \mathbb{R}_+^d \cap S_1^d[\bar{0}]$ where $\bar{0} = (0, 0, \ldots, 0)$. Let S be any subset of D such that $rr_D(S) \leq \epsilon$.

CLAIM 14. *For any point $p \in D$, there is a point $q \in S$ such that its distance from p is $||p - q|| \leq \sqrt{2\epsilon}$.*

PROOF. Let q be any point such that $||p - q|| > \sqrt{2\epsilon}$. Consider a user with utility vector $u = p$. Note that

$$\begin{aligned}\langle u, q \rangle &= \langle p, q \rangle \\ &= \frac{1}{2}(||p||^2 + ||q||^2 - ||p - q||^2) \\ &< \frac{1}{2}(2 - 2\epsilon) \\ &= 1 - \epsilon\end{aligned}$$

Thus, if, for every point q in S, $||p - q|| > \sqrt{2\epsilon}$ then

$$\begin{aligned}rr_D(S, u) &= 1 - \max_{p' \in D} \max_{q \in S} \frac{\langle u, q \rangle}{\langle u, p' \rangle} \\ &\geq 1 - \max_{q \in S} \frac{\langle u, q \rangle}{\langle u, p \rangle} \\ &> 1 - (1 - \epsilon) \\ &= \epsilon .\end{aligned}$$

This contradicts the fact that $rr_D(S) \leq \epsilon$. □

This claim implies that $\bigcup_{p \in S} B_{\sqrt{2\epsilon}}(p)$ must contain all points in D since otherwise some point in D will have distance more than $\sqrt{2\epsilon}$ from all points in S.

Note the fact that the area (i.e., $(d-1)$-dimensional measure) of $S_r^d(p)$ is $dC_d r^{d-1}$, for some constant C_d which depends on d. Thus, D has area $\frac{dC_d}{2^d}$ and, for any $p \in D$, the area of $B_{\sqrt{2\epsilon}}(p) \cap D$ is at most the area of $S_{\sqrt{2\epsilon}}(p)$ which is $dC_d(2\epsilon)^{(d-1)/2}$. Thus,

$$|S| \geq \frac{dC_d/2^d}{dC_d(2\epsilon)^{(d-1)/2}} = \frac{1}{2}(\frac{1}{8\epsilon})^{(d-1)/2} .$$

The theorem follows.

THEOREM 9. *For any d, $s \geq 2$, and t, the regret ratio after asking t rounds of questions of s points is at least $\frac{1}{8(4s^t)^{\frac{2}{d-1}}}$. In other words, for any ϵ, to get to regret ratio ϵ, every algorithm needs to ask at least $\Omega(d \log_s(1/\epsilon))$ questions.*

PROOF. Consider an s-ary tree of depth t generated by t questions, each of s tuples. In this tree, each node has s children (each children corresponds to each of s points picked by the user) and the tree has depth t (corresponds to t questions). Every node, except the leaf nodes, shows at most s points to the user. (Leaf nodes correspond to the terminating state so nothing is shown.)

There are $1 + s + s^2 + ... + s^{t-1} \leq \frac{s^t - 1}{s-1}$ non-leaf nodes. Thus, there are $s \cdot \frac{s^t-1}{s-1} \leq 2s^t$ points shown in total by all non-leaf nodes. Let A be a set of points shown by any node in this tree. We note that $\bigcup_{p \in A} B^d_{\sqrt{2\epsilon}}(p)$ must contain D; otherwise, by Claim 14, a user with utility vector $u \in D \setminus \bigcup_{p \in A} B^d_{\sqrt{2\epsilon}}(p)$ will not see any point with regret ratio at most ϵ, regardless of which leaf node this user is led to. This implies that this tree has to show at least $\frac{1}{2}(\frac{1}{8\epsilon})^{\frac{d-1}{2}}$ points, by the same calculation as in the proof of Theorem 8. This means that $2s^t \geq \frac{1}{2}(\frac{1}{8\epsilon})^{\frac{d-1}{2}}$ which implies that $\epsilon \geq \frac{1}{8(4s^t)^{\frac{2}{d-1}}}$ as claimed. □