Regret Minimization with Monotonic Utility Functions

Quang Nguyen
Denison University
Granville, OH, USA

Matthew Rinker
Denison University
Granville, OH, USA

Ashwin Lall
Denison University
Granville, OH, USA

ABSTRACT

While considering data in a representative database users want to look at a small subset of the database that best interest them. Based on this we will expand upon the idea of regret minimization that was conceived by Nanongkai et al. to cover all linear monotonic utility functions. Regret minimization is a method of finding a subset of k points in the database so that the users will be at least $x\%$ happy with the subset. The regret minimization method provides the benefits of both the main methods of multidimensional queries, top-k and skyline, while eliminating their respective drawbacks. Prior work with regret minimization has focused primarily on linear, convex, concave, constant elasticity of substitution utility functions. In this paper we will deal with linear monotonic utility functions. To be more specific, we want to look at increasing and decreasing monotonic utility function. We give an algorithm that facilitates changing any problem using monotonic utility functions into standard linear utility problem. Additionally, experimental evaluation shows that this method performs well on real and synthetic datasets as well as read life dataset.

PVLDB Reference Format:
Quang Nguyen, Matthew Rinker, Ashwin Lall. Regret Minimization with Monotonic Utility Functions. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/TBD

1. INTRODUCTION

When users want to look at a database, they normally only want to look at a small subset that best interest them. Given this problem, many methods have been used in order to find such subset with the smallest size. Two of the most famous methods to handle multi-attribute queries are the top-k [3] and skyline [1] operators. For the top-k operator, we assume that the user has an utility function that can best describe their preference. We will use the user's given utility function to calculate the top k items with the highest utility. In top-k, it is unrealistic to assume that the average user is aware of their utility function, thus many will be unable to give any utility functions at all. For the skyline operator, it will return all items that are not dominated by another item in each attribute and strictly worse in at least one. It defines a dominated item as an item that is equal or worse than some other item in the dataset. However, since we cannot control the size of the output in anyway it could output a large set, especially for datasets with a large number of dimensions. It is even possible for the skyline of a dataset to be the entire dataset.

The k-regret operator solves both of these problems. The k-regret operator allows control over the size of the output set to k tuples as well as ensuring there will be at least one interesting item in the returned set for any user with any utility function.

The k-regret operator can be informally defined with the following example. Suppose Alice wants to buy a new car and in particular she only cares about miles per gallon (MPG) and horse power (HP). Thus, suppose she looks at the cars database in Table 1 and her utility function is $0.2(MPG) + 0.8(HP)$. Suppose Alice see $car1$ with MPG of 60 and HP of 124. Thus $car1$ will gives Alice an utility of $0.2(60) + 0.8(124) = 111.2$. Another $car4$ with MPG of 45 and HP of 188 will give Alice an utility of $0.2(45) + 0.8(188) = 159$. Suppose $car4$ is the best car in the entire database and the first car is the best car in the subset. Thus Alice's regret is given by $159 - 111.2 = 47.8$. Thus, Alice's regret ratio is $\frac{47}{159} = 0.29$.However, instead of only looking at three utility functions, the k-regret relies on the calculation of the maximum regret ratio over all possible utility functions. The aim of k-regret minimization problem is to reduce the maximum regret ratio as much as possible.

In the past a lot of work has been done to flesh out the k-regret operator and its modeling of user preferences. In particular in original k-regret paper, Nanongkai et al. [5] proved that there exists an algorithm which bounds the maximum regret ratio for all linearly increasing utility functions. Later, [2] Kessler Faulkner et al. showed that the algorithm can also be adapted to consider non-linearly increasing utility functions. The functions they expanded the algorithm to included the convex, concave, and CES classes of utility functions. However, there exists a lot of problems which still need to be solved to better model user preferences.

One such problem that we encounter when considering user preferences is what if we have an item that gets better the smaller the value. For example, suppose John is browsing a website such as Yelp. Suppose he is concerned about one of the following attributes:

Proceedings of the VLDB Endowment, Vol. 12, No. xxx

DOI: https://doi.org/TBD

- The number of negative reviews a restaurant has received
- The distance from him
- The price of the restaurant

Suppose he lives in an area with a large number of restaurants necessitating some way of filtering his results. He must use the k-regret operator since he wants a small list to choose from but doesn't know his specific utility function in regards to any of the aforementioned attributes. Up until now the k-regret operator had not been compatible with decreasing utility functions. Our contribution will be to illustrate one way to allow any existing algorithm to consider the Lesser is Better problem. In this paper we will:

- Introduce a transformation to turn a Lesser is Better attribute to the Bigger is Better case. We also handle a mixed case where we both have BIgger is Better and Lesser is Better attributes
- Allow us to use existing algorithm on new transformation dataset
- Introduce an algorithm to handle the Monotonic Linear Utility Function problem
- Prove an upper bound for the regret ratio of the above algorithm
- Show through experimentation on synthetic datasets that the regret ratio for the above algorithm is relatively small

2. RELATED WORK

Many other papers have explored and suggested approaches to solving the issue of multidimensional user preference in relational databases. Originally there were the concepts of the Skyline and Top-k operators. Additionally, there have been several different methods of minimizing regret proposed since the original paper was published.

The skyline operator as described by Borzsonyi et al [1] uses the concept of domination. A point is said to dominate another point in a database if it is better than the point in at least one attribute and is not worse in all others. The points which are said to be dominated can be removed from the query as no user would value a point higher than one that is simply better than it. This query is convenient and requires no effort from the user making it desirable, however the output size of the query cannot be fixed and thus could contain an undesirable number of tuples.

Another query which has been proposed is the top-k query [3]. This query is able to provide a fixed number of tuples which are guaranteed to be satisfactory to the user. This operator asks the user to provide their own utility function and will then calculate what the best k tuples in the database are and return those tuples. This operator is undesirable due to the requirement of the user knowing their own utility function. This requires an amount of effort not reasonable for the common user as well as the issue that the common user often would not know their mathematical utility function.

In original k-regret paper [5] Nanongkai et al. introduced an algorithm which can evaluate regret based upon linear monotonically increasing utility functions. The major contribution of this paper was the introduction of the concept of regret as well as providing a way to bound regret independently of the size of the dataset.

This concept of regret has been built upon in other papers leading to a variety of algorithms being created which have increased the running speed as well as making the algorithm more efficient. Recently a paper introduced a new algorithm named Sphere [6]. This paper created a new algorithm which tightened the bounds of the regret problem which ultimately provided a better regret ratio.

Additionally, user interaction has been incorporated into regret queries to provide the most accurate answers. This has helped allow for even better regret ratios with minimal user involvement. Nanongkai et al. [4] introduced this concept in their 2012 paper and used it to define novel concepts which helped further the concept of regret.

Work has also been focused on expanding regret so that it can better model user preference. To this end there has been work which expanded the original k-regret algorithm, cube, to evaluate non-linear utility functions [2]. Kessler Faulkner et al. further generalized the format of the original cube algorithm so it could function with more classes of utility functions. Specifically these non-linear functions include the classes of concave, convex, and CES utility functions. In this paper we will introduce a similar solution for the monotonic case. However, we will utilize a transformation so as to not limit the regret ratio by confining it to one algorithm.

Car	MPG	HP	$f_{(0.2,0.8)}$	$f_{(0.4,0.6)}$	$f_{(0.7,0.3)}$
$car1$	60	124	111.2	98.4	79.2
$car2$	50	100	90	80	65
$car3$	51	181	155	129	90
$car4$	45	188	159	130	87.9
$car5$	40	130	112	94	67

Table 1: A cars database and cars Utilities

3. NOTATION

We will now formally define the problem this paper solves. We are given a d-dimensional set D of n points. This D represents our database which contains n tuples. We are also given an integer k which is the user-defined number of output points . Our goal is to return a set S which contains k tuples which have the lowest possible $MaximumRegretRatio$. S will be a subset of D. This set S represents our set of interesting points which the user wishes to see. In Table 2, we will show some of the most important notations that we use throughout the paper.

3.1 Regret

To understand the formal definition of the problem we must first understand the basic notations of the k-regret problem.

A *point* in $\mathbb{R}^d_+$, denoted by a is a $1 \times d$ vector from the database D. A *point* represents a specific item from the database.

A set of *utility functions* is denoted by F. A linear utility function f of the set F is denoted as a $d \times 1$ vector u ,$u \in R^d$.

DEFINITION 1. ***Utility Function*** *A mapping from a vector to its utility s.t.* $f : \mathbb{R}^d_+ \to \mathbb{R}_+$. *The utility for a point a for users with utility function* f *is* $f(a)$.

DEFINITION 2. ***Gain****: Gain represents a user's maximum "utility" or happiness given they see only a set of*

points. Given a specific utility function f and some set A s.t. $A \subseteq D$, we define the gain of the user with utility function f to be the greatest utility given by any one of the point in the set. Gain can be formally define as:
$gain(A, f) = max_{a \in A} f(a)$.

Regret and *regret ratios* are the integral concepts to the k-regret operator. Regret is loosely defined as the quantitative amount that a user would regret seeing a specific subset of the database instead of the overall database. This is calculated given a specific utility function f and some subset $A \subseteq D$.

DEFINITION 3. ***Regret.*** $r_D(A, f) = gain(D, f) - gain(A, f)$.

The *regret ratio* is a way of bounding the *regret* between 0 and 1 so that it can be easier compared to other *regret ratios*.

DEFINITION 4. ***Regret Ratio.*** $rr_D(A, f) = \dfrac{r_D(A, f)}{gain(D, f)}$.

Due to the fact that we are not given which specific utility function the user favors we must therefore accommodate the fact that the user could favor any utility function in the class of utility functions. To solve this we consider the *maximum regret ratio*. That is to say we consider the greatest regret ratio possibly given for a certain subset after considering all the utility functions in the set F of all possible utility function.

DEFINITION 5. ***Maximum Regret Ratio***
$sup_{f \in F} rr_D(A, f) = sup_{f \in F} \dfrac{\max_{a \in D} f(a) - \max_{b \in A} f(b)}{\max_{a \in D} f(a)}$.

Symbol	Definition
D	The set representing the entire database
$rr_D(S, u)$	The regret ratio of S over D
$rr_{D'}(S', u')$	The regret ratio of transformation set S' over D'
S	The set representing the solution set to the algorithm
n	The number of tuples (points) in the database
k	The user specified number of tuples (points) in the solution set
f and u	A utility function from the set of all possible utility functions
d	The number of dimensions (attributes) in the database.
d_b	The number of bigger is better attributes.
d_s	The number of smaller is better attributes.
p	The best point in the database for a given utility function.
p'	The best point in the database for a given utility function after transformation.
q	The best point in the solution set given utility function.
q'	The best point in the transformation solution set given utility function.
f and u'	A utility function and its after transformation.
B_i	The largest value for the ith attribute.
M	The user defined smallest possible value for the gain of any attribute.
α	This represents the upper bound for our regret equation as given by the original k-regret paper

Table 2: Notation table

3.2 Problem Definition

As stated earlier we are given our database D of size n and the user's maximum output size k. The user will also indicate d attributes which they are interested in. These d attributes will be the dimensions of our points and utility vectors. Since the user does not define their utility function, we operate under the assumption that the user does not know their preferred utility function.

The formal definition of this problem is as follows: given some integers k, d, n where $d \leq k$, and a class of utility functions F, find the minimum $\rho(k, d, n, F)$ such that for any set D of n d dimensional points there exists a k sized subset S of D such that $rr_D(S, F) \leq \rho(k, d, n, F)$. This was found to be true for many algorithms, the most recent of which being Sphere [6] which found $\rho(k, d, n, F) = O(\frac{1}{k^{\frac{2}{d-1}}})$, where F is the set of monotonically increasing linear utility functions.

In this paper we will focus on the case in which the set F includes utility functions where lesser is better. We will refer to any thing that's related to the lesser is better algorithm as LIB and anything that related to the bigger is better as BIB. In order to achieve LIB for an item, we allow the coefficient of the utility function to be negative.

Consider a database in which we arrange every point so that the first d_b attributes are BIB attributes and the rest are LIB attributes. Thus for each point a in the database, $a \in D$ can be represented as:

a $= \langle a_1, a_2, ..., a_{d_b}, a_{d_b+1}, ..., a_d \rangle$, where the user prefers larger values for $a_1, ..., a_{d_b}$ and smaller values for $a_{d_b+1}, ..., a_d$.

We suppose every utility function vector u can be represented as: $u = \langle u_1, u_2, ..., u_{d_b}, u_{d_b+1}, ..., u_d \rangle$, where $u_1, ...u_{d_b}$ are BIB coefficients and $u_i > 0$ for $i \in [1, d_b]$, and $u_{d_b+1}, ..., u_d$ are LIB coefficients and $u_i < 0$ for $i \in [d_b+1, d]$. The *gain* of a tuple can be represented as the dot product of the vectors $a \cdot u$.

We will rescale every point in the database so that every attribute ranges within $[0, 1]$. Thus for every point p, $p_i \in [0, 1]$ for $i \in [1, d]$. We will also scale every attribute in our utility vectors such that every attribute ranges within $[-1, 1]$. Thus for every u, $u_i \in [-1, 1]$ for $i \in [1, d]$. The fact that we can do this rescaling for the point and the utility function is because of the *scale invariance* theorem.

The *scale invariance* theorem state that:

THEOREM 1. *Given the two set of point $D = \{a_1, a_2, ..., a_t\}$ and $D'' = \{a''_1, a''_2, .., a''_t\}$. Suppose D and D'' are rescaling of each other, thus there exist positive reals $\lambda_1, \lambda_2, ..., \lambda_d$ such that $D'' = \{(a_i[1]\lambda_1, a_i[2]\lambda_2, ..., a_i[d]\lambda_d) | 1 \leq i \leq t\}$*

Consider two sets $S \subseteq D$ and $S'' \subseteq D''$ where S'' is the rescaled set of S. Here $rr_D(S, \mathcal{L}) = rr_{D''}(S'', \mathcal{L})$.

This property is a modified version of scale invariance proposed by Nanongkai et al. [5]. We extend it to the class of

monotonic utility functions in the appendix.

After rescaling u and p, we define M, which is the smallest possible utility of a users. This is based on the assumption that when an user looks at the database, no matter how bad an item is, the user will assign some minimum value for that item's utility. Thus, for every item in the database D, there will exist a minimum value M such that it is the minimum value of the utility of that product. We can formally define the assumption as, there must exist a $M > 0$ such that $gain(a, u) = max(M, a \cdot u)$, where a is any item in the database D and u is the utility function. It is really important to define an M value because since our utility function coefficient can be negative, the value $a \cdot u$ can be negative, thus leading to the calculation of the regret ratio goes above 1. (this means some users will regret over 100% for some item)

4. TECHNICAL RESULTS

We will introduce a transformation which, combined with a user default value to represent the minimum worth of any given point M, will allow us to convert any monotonic linear utility functions to BIB utility functions with a bound for our regret ratio. Our result is that the upper bound of any monotonic utility function is $\alpha(1+\frac{d_s}{M})$, where α is the bound of the regret ratio for the BIB linear utility function.

In the following section we will present our algorithm for the transformation as well as our proof showing the regret ratio bound of this transformation.

4.1 The Algorithm

Algorithm 1 The LIB algorithm

Input: A database D, The number of points in the database n, The size of the output set k. The number of attributes in each point d, The number of BIB attributes d_b.
Output: $S \subseteq D$

```
Normalize all points to [0, 1]
Normalize all utility vectors to [−1, 1]
for a in D do
    for i = db; i < d do
        a[i] = 1-a[i]
    end for
end for

Run regret algorithm on D' to get S'

for a in S' do:
    for i = db; i<d do
        a[i] = 1-a[i]
    end for
end for
Return S.
```

Lets define how the algorithm 1 works. First, we scale all points and utility vectors. We rescale every point and utility vector so that the points' attributes range within $[0, 1]$ and the utility vectors' attributes range within $[-1.1]$. This has been proven by our scale invariance that it does not affect the maximum regret ratio.

Suppose we come up with a transformation that converts a point a to a' where a' doesn't have any LIB attribute.

For each of the point a in the database , we define the transformation to a' as the we replace all of the LIB attributes with the difference between the maximum value of that attribute and its value. After the transformation, all of our LIB will become bigger is BIB. Thus, we can do a regret minimization on this new set D'.

Finally, after we get the result set S', we transform it back to the set S in the original set D.

4.2 Upper bound

We will restate the our theorem in a more formal way.

THEOREM 2. *For any database D with n points and d attributes and d_s the number of LIB attributes, the regret ratio of the set S after running Algorithm 1 can be bounded. $rr_D(S, \mathcal{L})$ is bounded by $rr'_D(S', \mathcal{L})(1 + \frac{d_s}{M})$, where M is the minimum utility value for any users, $rr_D(S, \mathcal{L})$ is the regret ratio of the set S and $rr'_D(S', \mathcal{L})$ is the regret ratio of set S'.*

PROOF. Let p be the best point in the database and p' be the p after transformation. Let u be some utility vector. Let u' be a transformation vector of the utility vector u where u' has all positive coefficient.

Suppose p is defined as: $p = \langle p_1, p_2, .., p_{d_b}, ...p_d \rangle$ and u is defined as: $u = \langle u_1, u_2, ..., u_{d_b}, ..., u_d \rangle$.

Thus, According to the algorithm , p' after the transformation can be represented as: $p' = \langle p_1, ...p_{d_b}, 1-p_{d_b+1}, ..., 1-p_d \rangle$ and u' after then transformation can be defined as: $u' = \langle u_1, u_2, ..., u_{d_b}, -u_{d_b+1}, ..., -u_d \rangle$.

Thus, the gain of point p' with linear utility function u' is $gain(p', u') = p' \cdot u' = \sum_i^{d_b} p_i u_i + \sum_{d_b+1}^{d} (p_i - 1)u_i$.

The *gain* of p can be define as $gain(D, u) = \max\{M, p \cdot u\}$.

Case 1: $p \cdot u > M$

Because $gain(D, u) = \max\{M, p \cdot u\}$, thus $gain(D, u) = p \cdot u$.

Thus,

$$\begin{aligned} gain(D,u) &= \sum_{i=1}^{d}(p_i.u_i) \\ &= \sum_{i=1}^{d_b}(p_i u_i) + \sum_{i=d_b+1}^{d}(p_i u_i) \\ &= \sum_{i=1}^{d_b}(p_i u_i) + \sum_{i=d_b+1}^{d}(p_i u_i) - \sum_{i=d_b+1}^{d}(u_i) \\ &\quad + \sum_{i=d_b+1}^{d}(u_i) \\ &= \sum_{i=1}^{d_b}(p_i u_i) + \sum_{i=d_b+1}^{d}(p_i - 1)u_i + \sum_{i=d_b+1}^{d}(u_i) \\ &= p' \cdot u' + \sum_{i=d_b+1}^{d}(u_i). \end{aligned}$$

Consider $-C = \sum_{i=d_b+1}^{d}(u_i)$. Therefore, $p \cdot u = p' \cdot u' - C$.

Case 2: $p \cdot u < M$

Because $gain(D, u) = \max\{M, p \cdot u\}$, thus $gain(D, u) = M$. And we know that from case 1: $p \cdot u = p' \cdot u' - C$. Therefore, $p' \cdot u' - C < M$.

LEMMA 1. *For every $u \in F$ where F is the set of every utility vector, if $p \in D$ such that p maximizes $p \cdot u$, then the transformed version of p, which we call p', is the point that maximizes $p' \cdot u'$ and vice versa.*

The proof for Lemma 1 appears in the Appendix.

According to Lemma 1, we know that if $p \in D$ is the point that maximizes $p \cdot u$, then $p' \in D'$ is the point that maximizes $p' \cdot u'$. We pick $q' \in S'$ such that it is the point that maximizes $q' \cdot u'$. Moreover it follows that q is the point in the set S that maximize $q \cdot u$ according to Lemma 1.

Therefore, $gain(p', u') = gain(D', u')$, $gain(q, u) = gain(S, u)$ and $gain(p', u') = gain(S', u')$.

Thus, the regret ratio of S over D for a utility u is:
$\frac{gain(D,u) - gain(S,u)}{gain(D,u)} = \frac{gain(p,u) - gain(q,u)}{gain(p,u)}$.

Thus, the regret ratio of S' for the database D' for a utility function u' is:
$\frac{gain(D',u') - gain(S',u')}{gain(D',u')} = \frac{gain(p',u') - gain(q',u')}{gain(p',u')} = \frac{p'u' - q'u'}{p'u'}$ and it is bounded by some α.

We want to prove that the regret ratio of S over D can be bounded by the regret of S' over D'. This can be divided into three cases.

Case 1 : *If $p \cdot u < M$, $q \cdot u < M$.* Thus $gain(D, u) = M$ and $gain(S, u) = M$

Therefore $\frac{gain(D,u) - gain(S,u)}{gain(D,u)} = \frac{M - M}{M} = 0$

This must be true since if the best point in the database has utility below the smallest possible utility then all points in the database will be equally unsatisfying for the user.

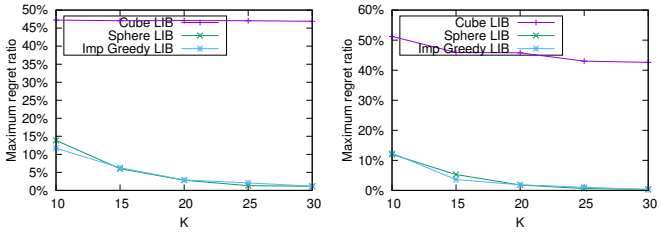
Case 2: *$p \cdot u > M$, $q \cdot u > M$.* Thus $gain(D, u) = p \cdot u$ and, $gain(S, u) = q \cdot u$.

Consider:

$$\begin{aligned}
\frac{gain(D,u) - gain(S,u)}{gain(D,u)} &= \frac{p \cdot u - q \cdot u}{p \cdot u} \\
&= \frac{p' \cdot u' - C - q' \cdot u' + C}{p' \cdot u' - C} \\
&= \frac{p' \cdot u' - q' \cdot u'}{p' \cdot u' - C} \\
&= \frac{p' \cdot u' - q' \cdot u'}{p' \cdot u'} \frac{p' \cdot u'}{p' \cdot u' - C} \\
&= \frac{p' \cdot u' - q' \cdot u'}{p' \cdot u'} \frac{p \cdot u + C}{p \cdot u} \\
&\leq \alpha(1 + \frac{C}{M}). \text{ (as } p \cdot u > M)
\end{aligned}$$

We know that $C = \sum_{i=d_b+1}^{d}(-u_i)$ and because $-u_i \leq 1$. Thus $C \leq d - d_b = d_s$. Therefore, $\frac{p \cdot u - q \cdot u}{p \cdot u} \leq \alpha(1 + \frac{d_s}{M})$.

Case 3: *$p \cdot u > M$ and $q \cdot u < M$.* Thus, $gain(D, u) = p \cdot u$ and $gain(S, u) = M$ And because $M > q \cdot u$, then $M > q'u' - C$, thus $-M < -q' \cdot u' + C$.



(a) equally distributed (b) anti-correlated

Figure 1: vary K, $n = 10000$, $d = 4$, $M = 0.2$, $d_b = 2$

Consider:

$$\begin{aligned}
\frac{gain(D,u) - gain(S,u)}{gain(D,u)} &= \frac{pu - M}{qu} \\
&< \frac{p' \cdot u' - C - q' \cdot u' + C}{p' \cdot u' - C} \\
&= \frac{p' \cdot u' - q' \cdot u'}{p' \cdot u' - C} \\
&\leq (\frac{p' \cdot u' - q' \cdot u'}{p' \cdot u'})(\frac{p' \cdot u'}{p' \cdot u' - C}) \\
&\leq \alpha(\frac{p \cdot u + C}{p \cdot u}) \\
&\leq \alpha(1 + \frac{C}{M})(\text{as } p \cdot u > M) \\
&\leq \alpha(1 + \frac{d_s}{M}).(\text{as } C \leq d_s)
\end{aligned}$$

We will not consider the case when $p \cdot u < M$ and $q \cdot u > M$. This is because $p \cdot u$ is the gain of the database and $q \cdot u$ is the gain of the subset of the database, thus $p \cdot u \geq q \cdot u$.

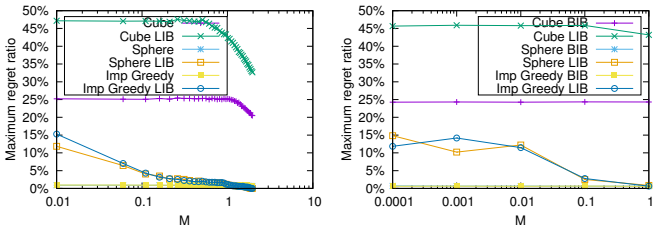
Therefore, the regret ratio of S over D can be bounded by $\alpha(1 + \frac{d_s}{M})$ for all cases. □

Let's do some analyze on the result of the theorem. We know that the $rr_D(S, \mathcal{L})$ is bounded my $rr'_D(S', \mathcal{L})(1 + \frac{d_s}{M})$. Thus, as the value M increase, the regret ratio of S' will become closer and closer to the regret ratio of S. If d_s is smaller, the regret ratio between set S' and set S also closer to each other. To be more specific, one of the most notable analysis is that if $d_s = 0$, which means you don't have any LIB attributes, $rr_D(S, \mathcal{L})$ will be bounded by $rr'_D(S', \mathcal{L})$ which is the same when we only consider increasing linear utility function.

5. EXPERIMENTAL EVALUATION

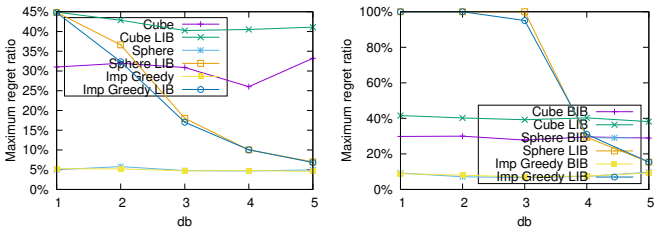
In this section we will show the regret ratios for algorithm 1 through experiments on synthetic datasets and a real life NBA dataset. From section 4, we known that the regret ratio of the LIB dataset (the regret ratio of set S) can be bounded by the regret ratio of the BIB dataset (the regret ratio of set S''). In particular, if the regret ratio of S' is α, then we will expect the regret ratio of S to be $\alpha(1 + \frac{d_s}{M})$. However, in this section, we will show how the regret ratios behave in actual experiments. We want to show that in theory, the upper bound of the regret ratio depends on the value of M and d_s.

Unless otherwise stated, we will pick our default value so that $n = 10000$, $d = 4$, $d_b = 2$ and $k = 20$. Before doing the experiment, we need to pick the value M so that it gives a



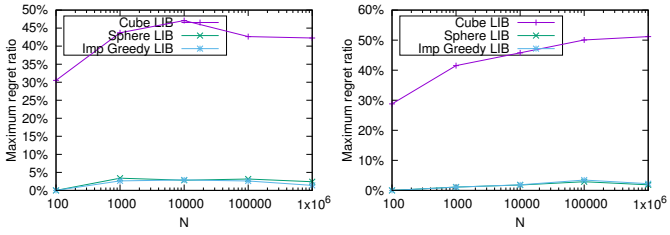
(a) equally distributed (b) anti correlated

Figure 2: vary M, $n = 10000$, $k = 20$, $M = 0.2$, $d_b = 2$



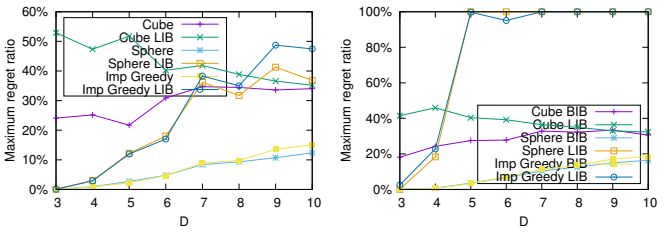
(a) equally distributed (b) anti correlated

Figure 3: vary db, $n = 10000$, $d = 4$, $k = 20$



(a) equally distributed (b) anti correlated

Figure 4: vary-n, $M = 0.2$, $d_b = 2$, $d = 4$, $k = 20$



(a) equally distributed (b) anti correlated

Figure 5: vary-d, $M = 0.2$, $d_b = 2$, $k = 20$, $n = 10000$

reasonable bound. We pick M to be $\frac{d_b}{10}$. This is because for any utility value $p \cdot u$:

$p \cdot u = \sum_{i=1}^{d}(p_i.u_i) = \sum_{i=1}^{d_b}(p_i.u_i) + \sum_{i=d_b+1}^{d}(p_i.u_i) \leq \sum_{i=1}^{d_b}(1) + \sum_{i=1}^{d_b}(0) = d_b$.

Thus, the maximum value of $p \cdot u$ is d_b, the values $p \cdot u$ run from 0 to d_b, therefore we want to pick a M so that it is greater than at least 10% of all the utility value. Therefore, for the default case $M = 0.2$.

We conducted all of our experiment on a 64bit 3.20GHz HP Elite Desk 800 machine which was running Ubuntu 14.0. All of our algorithms were written in C++ and compile using the GNU compiler.

In our experiments, we measure the performance of *Cube*, *Sphere* and *Greedy* through our Algorithm 1. To be more specific, we use our Algorithm 1 to change the dataset from a LIB problems to a BIB problem. Thus this allows us to run *Cube*, *Sphere* and *Greedy* on the new dataset. After getting the result set S, we will measure the regret ratio computed by the new more is better dataset versus the regret ratio computed by the less is better dataset ($rr_D(S, \mathcal{L})$ versus $rr_{D''}(S', \mathcal{L})$) and see if it matches with the proof.

Since we notice that both anti-correlated and equally distributed dataset behave differently, we will run our algorithm on both of those dataset. We average our experiment dataset over 10 independent datasets. For evaluation, we use a random evaluate function, which generates 10000 random utility functions and finds the maximum regret ratio over those functions. Although this is not the optimal approach for computing the maximum regret ratio since 10000 utility function does not cover every possible utility function. In practice, it is close to the actual maximum regret ratio. Moreover, the LP program that used to evaluate regret ratio only works for more is better problems, which is not applicable since we have to evaluate the regret ratio of the LIB problem.

5.1 Dataset

We use both the anti-correlated and equally distributed dataset. Anti-correlated is when the attribute of the point have a negative correlation with each other. Equally distributed is when the point got spread out equally in the hyperspace. Next, we will run our algorithm on each of the independence dataset, then we will average the regret ratio computed by each algorithm. Depends on the requirement of the experiment, we might have to generate additional dataset (eg: 10 dataset, each with 1000000 data point). We will briefly mention it before each experiment. For real world dataset, we will use the *NBA 2000-2018 seasons player statistic*. Our dataset will have 7 attributes and over 600 points. Our BIB stats consist of point per game, rebound, assist and steal. Our LIB attributes contains turn over percentage, foul per game and age.

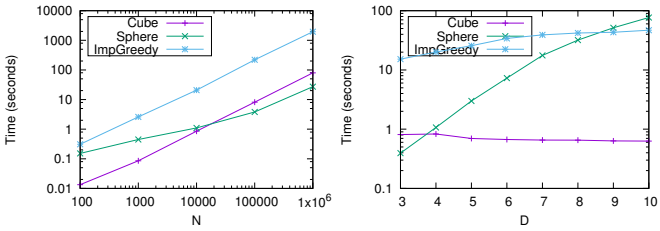
5.2 Algorithm

We will run our experiment on the following algorithm:

1. the original Cube algorithm proposed by Nanongkai[5]
2. the Greedy algorithm proposed by Xie [6]
3. the Sphere algorithm proposed by Xie [6] which worked really good with linear utility function, which gives a lower bound on the regret ratio compared to the original Cube algorithm

5.3 Maximum Regret Ratios

Considering the Theorem 2 in section 4, we know that if the regret ratio of the set S'' is α, then the regret ratio of the set S is $\alpha(1 + \frac{d_s}{M})$. Thus we will expect that the regret ratio of the LIB problem of each algorithm will be greater than the regret ratio of the BIB problems. This can be confirmed by our experiment as we see that from Figure 2 to Figure 1 the LIB algorithm always have a greater regret ratio compare to the BIB regret ratio.



(a) vary-n, d=4, k =20, M =0.2, $d_b = 2$ (b) vary-d, $n = 10000$, $k = 20$, $M = 0.2$

Figure 6: Time analysis

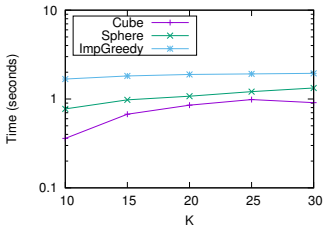


Figure 7: Time analysis, vary-k, $n = 10000$, $d_b = 2$, $M = 0.2$

Now lets look at the regret ratio when we vary db and vary D. According to the proof, as d_b increase, which is increase the number of BIB attributes, we will decrease the number of smaller is better attribute (d_s). As d_b increase, $\frac{d_s}{M}$ will get smaller. Therefore, the regret ratio of the LIB will be closer to the regret ratio of the BIB. This can be confirmed by Figure 3. We still have an exception, which is the Cube algorithm of both anti-correlated and equally distributed data. The regret ratio of Cube does not seems to converge to each other as d_b increases. We also notice that as when d_b close to 0, all of our attributes will be smaller is better attributes. This makes the regret ratio extremely high according to the graph. Moreover, if you look at the regret ratio in Figure 5, we also see an increase in the regret ratio. We expected this increment because as dimension increase, we will deal with high dimension data.Therefore,unless we increase the number of k, we will eventually increase the regret ratio.

Now we look at the regret ratio of all the algorithm after the LIB transformation. In Figure 4 where we vary the number of points. According to the original k-regret paper, we know that the regret ratio of all the algorithms do not depend on the number of point in the database. Thus, we see that the regret ratio of Cube, Sphere and Greedy do not seem to depend on n. As n increases from 100 to 1000000, we see that the regret ratios stay the same for all the algorithm. Thus, we can conclude that our LIB transformation algorithm keeps the property of being independence of n.

More important, another property is that as k increase,it is more likely to get the point that the user prefer. Thus, as k increases, we will expect the regret ratio to decrease. This properties can be confirm by Figure 1, in which we see as k increase, we see a decrease in regret ratio.

For real life dataset analysis, we did an experiment on the real NBA player statistic to see how the maximum regret ratio behaves. Overall in Figure 8, the maximum regret ratios decrease as number of output size k increase. However, it is not perform really well compared to the synthesis dataset.

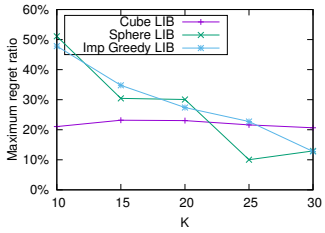


Figure 8: NBA players stats, vary-k, $n = 664$, $d_b = 4$, $M = 0.4$

On the average, the maximum regret ratio for $k = 20$ for synthesis dataset is 5% while the maximum regret ratio for the same k for NBA dataset is 10%. Given these result we agree that there are more works to be done in order to lower the maximum regret ratio

5.3 Times

Figure 6 and Figure 8 shown us the running time of each of the algorithm *Cube*, *Sphere* and *ImpGreedy* by varying the n, k and d variables. We include the time it takes to run the skyline for these algorithm because we want to know the actual time it takes to return a set S given the database size N. We first notice that in all of the three vary plot, the *Cube* has the least running time compare to other two. This is because the *Cube* does really on the linear program, which makes the algorithm significantly faster. This is the trade of of the *Cube* algorithm, we are sacrificing running times for maximum regret ratio. Secondly, we notice that in Figure 6 (b), even though the running times remains the same for Cube and Sphere, the running time of Greedy increase significantly as D increase. As last, the running time of all the three algorithm does not really on the size of output K.

6. CONCLUSION

In this paper, we focus on both increasing and decreasing monotonic utility functions. The k-regret operator is implemented to find the user preferences in a large database D. It wants to find a subset that results in the lowest regret ratio, which is a variable to measure how much the users regret when looking at the whole database versus when looking at the subset. However, the original k-regret operator only works for BIB attributes, it does not consider attribute like price, number of bad reviews, which are more preferable the less of it. In this paper, we focus on dealing with LIB attributes. We find a new bound for the maximum regret ratio. Through numerous of experiments, we confirm these bound by showing that the maximum regret ratio of all the algorithm after running our transformation is reasonably small. Further works include develop our transformation so that it works for non-linear utility functions, picking a reasonable M value, dealing with the case where $d_s = d$. In conclusion, it is possible to use our result to further study the decreasing linear utility functions in representative databases.

7. REFERENCES

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

[2] T. K. Faulkner, W. Brackenbury, and A. Lall. k-regret queries with nonlinear utilities. In *Proceedings of the*

VLDB Endowment, volume 8. VLDB Endowment, 2016.

[3] 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.

[4] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino. Interactive regret minimization. In *Proceedings of SIGMOD*. ACM, 2012.

[5] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2), 2010.

[6] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, 2018.

APPENDIX

A. PROOF FOR LEMMA 1:

PROOF. Lemma 1: *For every $u \in F$ where F is the set of every utility vector, if $p \in D$ such that p maximizes $p \cdot u$, then the transformed version of p, which we call p', is the point that maximizes $p' \cdot u'$ and vice versa*

Because Lemma 1 is an if and only if statement, we have to prove it in two direction. First we need to prove the lemma in the forward direction, that is: if $p \in D$ such that p maximizes $p \cdot u$, then $p' \in D'$ is the point that maximizes $p' \cdot u'$.

Assume by contradiction that p' is not the point that maximize $p' \cdot u'$. Thus, there exist another point $a' \in D'$ such that a' maximize $a' \cdot u'$. Because $a' \in D'$, thus a' will be in the form such that: $a' = \langle a_1, a_2, .., a_{d_b}, 1 - a_{d_b+1}, ..., 1 - a_d \rangle$

Thus there must be exist an $a \in D$ such that $a = \langle a_1, ..., a_{d_b}, a_{d_b+1}, ..., a_d \rangle$. This is because every point in D' must come from a point in D

According to the assumption, we know that a' is the point that maximize $a' \cdot u'$.

Therefore,

$$\begin{aligned} p'u' &< a'u' \\ \sum_{i=1}^{d_b}(p_i u_i) + \sum_{i=d_b+1}^{d}(1-p_i)(-u_i) &< \sum_{i=1}^{d_b}(a_i u_i) + \sum_{i=d_b+1}^{d}(1-a_i)(-u_i) \\ \sum_{i=1}^{d}(p_i u_i) - \sum_{i=d_b+1}^{d}(u_i) &< \sum_{i+1}^{d}(a_i u_i) - \sum_{i=d_b+1}^{d}(u_i) \\ p \cdot u &< a \cdot u. \end{aligned}$$

However, p is the point that maximizes $p \cdot u$. Thus there cannot exist another point $a \in D$ such that $a \cdot u > p \cdot u$. Therefore, the assumption is wrong. Thus if p is the point that maximizes $p \cdot u$ then p' is the point that maximize $p' \cdot u'$

We prove Lemma 1 in the backward direction. That is if p' is in the transformation set D' such that p' maximize $p' \cdot u'$, then $p \in D$ is the point that maximize $p \cdot u$

Assume by contradiction that p is not the point that maximize $p \cdot u$, thus, there exist another point $a \in D$ such that a maximize $a \cdot u$. Because $a \in D$, thus a will be in the form such that $a = \langle a_1, ..., a_{d_b}, a_{d_b+1}, .., a_d \rangle$.

And because every point in D has to be mapped to D'. Thus, there must exist an $a' \in D'$ such that $a' = \langle a_1, a_2, .., a_{d_b}, 1 - a_{d_b+1}, ..., 1 - a_d \rangle$. Because a is the point that maximizes $a \cdot u$.

Thus:

$$\begin{aligned} p \cdot u &< a \cdot u \\ p' \cdot u' + \sum_{i=d_b+1}^{d}(u_i) &< a'u' + \sum_{i=d_b+1}^{d}(u_i) \\ p'u' &< a'u' \end{aligned}$$

However, we know that because p' is the point that maximize $p'u'$. Thus there cannot be exist any other point a' that $a'u > p'u'$. Thus the assumption is incorrect.

Therefore, for every $u \in F$ if $p \in D$ maximizes $p \cdot u$, then the transformed version p' is the point that maximizes $p' \cdot u'$ and vice versa.

□

B. SCALE INVARIANCE

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

Consider two sets $S \subseteq D$ and $S'' \subseteq D''$ where S'' is the rescaled version of S. We will prove that $rr_D(S, \mathcal{L}) = rr_{D''}(S'', \mathcal{L})$.

Before proving the scale invariance, we need to prove two important claims:

CLAIM 1. *We prove that for any vector v, there exists a vector v'' such that $v \cdot a = v'' \cdot a''$ for any $a \in D$ and $a'' \in D''$ where $a''[j] = \lambda_j a[j]$ for $j = 1, 2, ..., d$.*

Proof of Claim 1:

We consider v'' so that $v''[j] = \frac{v[j]}{\lambda_j}$ for every $j = 1, 2, ..., d$.

Thus:

$$\begin{aligned} v \cdot a &= \sum_{j=1}^{d} v[j].a[j] \\ &= \sum_{j=1}^{d} \frac{v[j]}{\lambda_j}\lambda_j a[j] \\ &= \sum_{j=1}^{d} v''[j]a''[j] \\ &= v'' \cdot a'' \end{aligned}$$

CLAIM 2. *We prove that for any vector v, if $a \in D$ is the point that maximize the utility value $a \cdot v$ then $a'' \in D''$, the rescale point of a, is the point that maximize the utility value $a'' \cdot v''$ where v'' is a vector such that $a \cdot v = a'' \cdot v''$.*

Proof of Claim 2: From Claim 1, we know that there exist v'' such that $v''[j] = \frac{v[j]}{\lambda_j}$ for every $j = 1, 2, .., d$ and $a \cdot v = a'' \cdot v''$.

Suppose by contradiction that a'' is not the best point that maximize $a \cdot v$. There exist another point $b'' \in D''$ such that b'' is the vector that maximize $b'' \cdot v''$.Thus $a'' \cdot v'' < b'' \cdot v''$. Because b'' is a point in D'', thus b'' have to be a rescale point of some point $b \in D$. Thus $b''[j] = \lambda_j b[j]$.

However, we know that from Claim 1 there exist vector v such that $v[j] = \lambda_j . v''[j]$ so that $b \cdot v = b'' \cdot v''$.

Thus if $a'' \cdot v'' < b'' \cdot v''$, then $a \cdot v < b \cdot v$ This is contradiction since our statement state that a is the best point that maximize $a \cdot v$. Thus, there cannot be exist another point b such that $b \cdot v > a \cdot v$.

From Claim 1 and Claim 2, we can conclude that $max_{a \in D}(a \cdot v) = max_{a'' \in D''}(a'' \cdot v'')$ for a'' is a rescale point of a and v and v'' is any vector such that $v''[j] = \frac{v[j]}{\lambda_j}$ for every $j = 1, 2, .., d$.

Suppose there is a v such that v is the worst vector of $rr_D(S, \mathcal{L})$ so that $rr_D(S, \mathcal{L}) = rr_D(S, v)$.

Thus:

Case 1: if $max_{a \in D} v \cdot a < M$. We will not consider this case since if the best point has utility less than the standard M, then the users will equally unsatisfied every items in the database.

Case 2: if $max_{a \in D} a \cdot v > M$ and $max_{p \in S} p \cdot v > M$. Thus $gain(D, v) = max_{a \in D} a \cdot v$ and $gain(S, v) = max_{p \in S} p \cdot v$.

Consider:

$$\begin{aligned} rr_D(S, \mathcal{L}) &= \frac{gain(D, v) - gain(S, v)}{gain(D, v)} \\ &= \frac{max_{a \in D} a \cdot v - max_{p \in S} p \cdot v}{max_{a \in D} a \cdot v} \\ &= \frac{max_{a'' \in D''} v'' \cdot a'' - max_{p'' \in S''} v'' \cdot p''}{max_{a'' \in D''} v'' \cdot a''}. \\ &\quad \text{(because of Claim1 and Claim2)} \end{aligned}$$

Because $max_{a \in D}(a \cdot v) > M$ and $max_{p \in S}(p \cdot v) > M$ .Then, $max_{a'' \in D''} v'' \cdot a'' > M$ and $max_{p'' \in S''} v'' \cdot p'' > M$. Thus, $gain(D'', v'') = max_{a'' \in D''} a'' \cdot v''$ and $gain(S'', v'') = max_{p'' \in S''}(p'' \cdot v'')$.

Thus

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

Case 3: if $max_{a \in D} v \cdot a > M$ and $max_{p \in D} v \cdot p < M$. Thus $gain(D, v) = max_{a \in D}(a \cdot v)$ and $gain(S, v) = M$.

Thus

$$\begin{aligned} rr_D(S, \mathcal{L}) &= \frac{gain(D, v) - gain(S, v)}{gain(D, v)} \\ &= \frac{max_{a \in D} v \cdot a - M}{max_{a \in D} v \cdot a} \\ &= \frac{max_{a'' \in D''} v'' \cdot a'' - M}{max_{a \in D''} a'' \cdot v''}. \\ &\quad \text{(Because of Claim1 and Claim2)} \end{aligned}$$

Because $max_{a \in D} v.a > M$ and $max_{p \in S} v.p < M$. Then, $max_{a'' \in D''} v'' \cdot a'' > M$ and $max_{p'' \in S''} v'' \cdot p'' < M$. Therefore, $gain(D, v) = max_{a \in D} a \cdot v$ and $gain(S, v) = M$

Thus,

$$\begin{aligned} rr_D(S, \mathcal{L}) &= \frac{max_{a'' \in D''} v'' \cdot a'' - M}{max_{a'' \in D''} v'' \cdot a''} \\ &= \frac{gain(D'', v'') - gain(S'', v'')}{gain(D'', v'')} \\ &= rr_{D''}(S'', v''). \end{aligned}$$

Thus $rr_D(S, \mathcal{L}) = rr_{D''}(S'', v'')$. And because there might be a vector worst than v'' in a set of all possible utility vector function. Thus $rr_{D''}(S'', v'') \leq rr_{D''}(S'', \mathcal{L})$.

Therefore, $rr_D(S, \mathcal{L}) \leq rr_{D''}(S'', \mathcal{L})$. Similarly, $rr_{D''}(S'', \mathcal{L}) \leq rr_D(S, \mathcal{L})$.

Thus $rr_D(S, \mathcal{L}) = rr_{D''}(S'', \mathcal{L})$. Therefore, rescaling the database does not affect the regret ratio. We have proved the scale invariance properties of the regret ratio. □