Regret Minimization with Monotonic Linear Utility Functions

Quang Nguyen, Matthew Rinker
Denison University, Department of Mathematics and Computer Science
Research Advisor: Dr. Ashwin Lall

Abstract

Suppose you are shopping online to buy a new car. There are an extraordinary amount of options to choose from and it is unreasonable to go through all of them. If only there was some way to only look through a smaller subset of the entire selection and be guaranteed that you would find something to your satisfaction. That is the *k*-regret operator. The *k*-regret operator mathematically picks out a subset of a size of your choosing based upon what attributes you care about. Prior work with the *k*-regret operator has been solely focused on the case where every attribute is better the larger it gets. However, this is commonly not the case. To this end we have expanded the reach of the *k*-regret operator to encompass the cases where some or all attributes have the quality where smaller is better.

Regret Minimization

The *k*-regret operator was proposed as a solution to resolve the deficiencies of two other popular multi-dimensional database operators, top-*k* and skyline, which we will briefly go over. Consider the case of a database of cars where we care about Miles per Gallon (MPG) and Horsepower (HP).

Top-*k*
Requires a specific utility function to be provided.
Suppose our utility function is 0.5(MPG)+0.5(HP)
Top-*k* provides *k* items with the most utility, in this case if *k*=2, we get cars 2 and 3 in the table below.
Skyline
The skyline operator work by removing all "dominated" items. An item is dominated if it is strictly worse than another item. In the below example car 4 would be dominated because it is strictly worse than car 3 in both MPG and HP.

Top-*k* requires the user to know how to mathematically model their preferences and Skyline cannot bound the output size. The *k*-regret operator fixes these flaws. Consider the following example:

	Car #	MPG	HP	$0.5(MPG) + 0.5(HP)$ utility
	1	50	50	$0.5(50) + 0.5(50) = 50$
	2	100	40	$0.5(100) + 0.5(40) = 70$
	3	65	70	$0.5(65) + 0.5(70) = 67.5$
	4	30	30	$0.5(30) + 0.5(30) = 30$

Suppose Alice wants to buy a new car and only cares about MPG and HP. Suppose Alice's utility function is: 0.5(MPG) + 0.7(HP). There is a car (Car 1) with MPG of 50 and a HP of 50. Thus Car 1 will give Alice a utility of 0.5(50) + 0.7(50) = 60. Another car, Car 2 has MPG of 100 and HP of 40, will give Alice a utility of 0.5(100) + 0.7(40) = 78. Suppose car 2 is Alice's best car on the lot and the first car is Alice's best car in the solution set. Thus Alice's regret is given by 78-60 = 18 and her regret ratio is 18/78 = 0.23. Since we do not assume we know the user's utility function beforehand we must choose a set which makes the regret as small as possible for all possible utility functions. This is called the Maximum Regret Ratio.
A surprising result came with the discovery of this method of finding a guaranteed interesting subset. It turns out, there exists an algorithm where the Maximum Regret Ratio can be bounded by a function of the number of attributes and solution set size and is independent of the database size.

Lesser is Better

In everyday life we are constantly making decisions based on multiple attributes. Quite commonly these attributes have the quality where smaller is better. These examples include:

- How far something is from you
- How much an item costs
- Shipping time
- Effort required
- Number of bad reviews
- How something ranks on a top x list

These are all every common things to consider and as such we have developed a way for the *k*-regret operator to incorporate them.

Notation

Symbol	Definition
D	The database
S	The solution set
n	Number of items in the database
k	Number of items in the solution
d	Number attributes considered
d_b	Number of bigger is better attributes.
d_s	Number of smaller is better attributes
M	Smallest possible gain for any item in D

Algorithm

We developed an algorithm to facilitate the mixed bigger is better and smaller is better cases. For this we assume that every utility function can be represented as a vector and order the attributes so that the first d_b attributes are bigger is better and the last d_s attributes are smaller is better. Our algorithm is as follows:

```
Algorithm 1 The LIB algorithm
Input: $D, n, k, d, d_b, |D| = n$
Output: $S \subseteq D, |S| = k$
Our algorithm requires all attributes to be between $[0, 1]$
and all utility attributes to be between $[-1, 1]$
for a in $D$ do
    for a[i] in a do
        a[i] = $\frac{a[i]}{\max(a[i])}$
Do the same for all attributes of every utility vector
Run the transformation to turn D into D'
for a in $D$ do
    for all smaller is better attributes i do
        replace i with 1-i
Run any regret algorithm on $D'$ to get $S'$
Run the reverse transformation to turn S' back into S
for a in $S'$ do:
    for all smaller is better attributes j do
        replace j with 1-j
Return $S$.
```

Additionally, we were able to prove an upper bound for the regret ratio of this algorithm:
Theorem 1: For any database D with n points, d attributes, and d_s lesser is better attributes the maximum regret ratio of the set S after Algorithm 1 has been run can be bounded by the maximum regret ratio of the transformed set, S' times the factor: $(1 + \frac{d_s}{M})$.

Experimental Results

In our experimentation we ran our algorithm on both real and synthetic datasets. Our synthetic datasets were made up of datasets with both uniform and anti-correlated data. Anti-correlated data gave us the worst regret ratio, with uniform distributions being a close second.

These experiments were run on 64bit 3.20Ghz HP Elite Desk 800 machines and were run using 3 different regret algorithms. The default value for our experiments were n = 10,000, d = 4, db = 2, ds = 2, M = 0.2, k = 20.
An interesting aspect of *k*-regret queries is the bound of the maximum regret ratio is independent of dataset size.

- Fig. 1 Shows how the regret ratio is uncorrelated with the size of N
- Fig. 2 Shows how as *k* increases the regret ratio decreases
- Fig. 3 Shows how as *d* increases the regret ratio increases exponentially

This coincides with the algorithm with the optimal bound, Sphere. Sphere has a bound of:

$$O(\frac{1}{k^{\frac{2}{d-1}}}).$$

The behavior we see in our graphs clearly match the expected behavior of that equation.

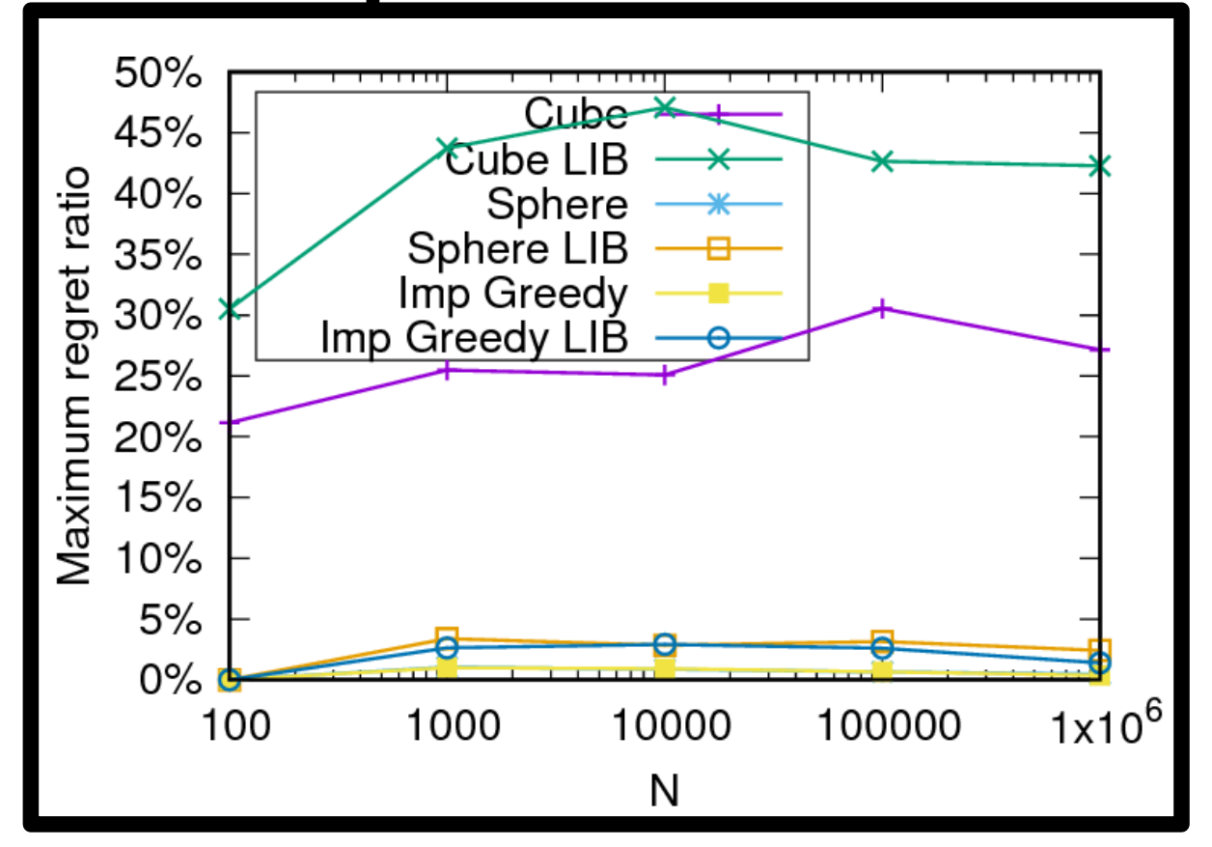

Fig. 1: Varying N

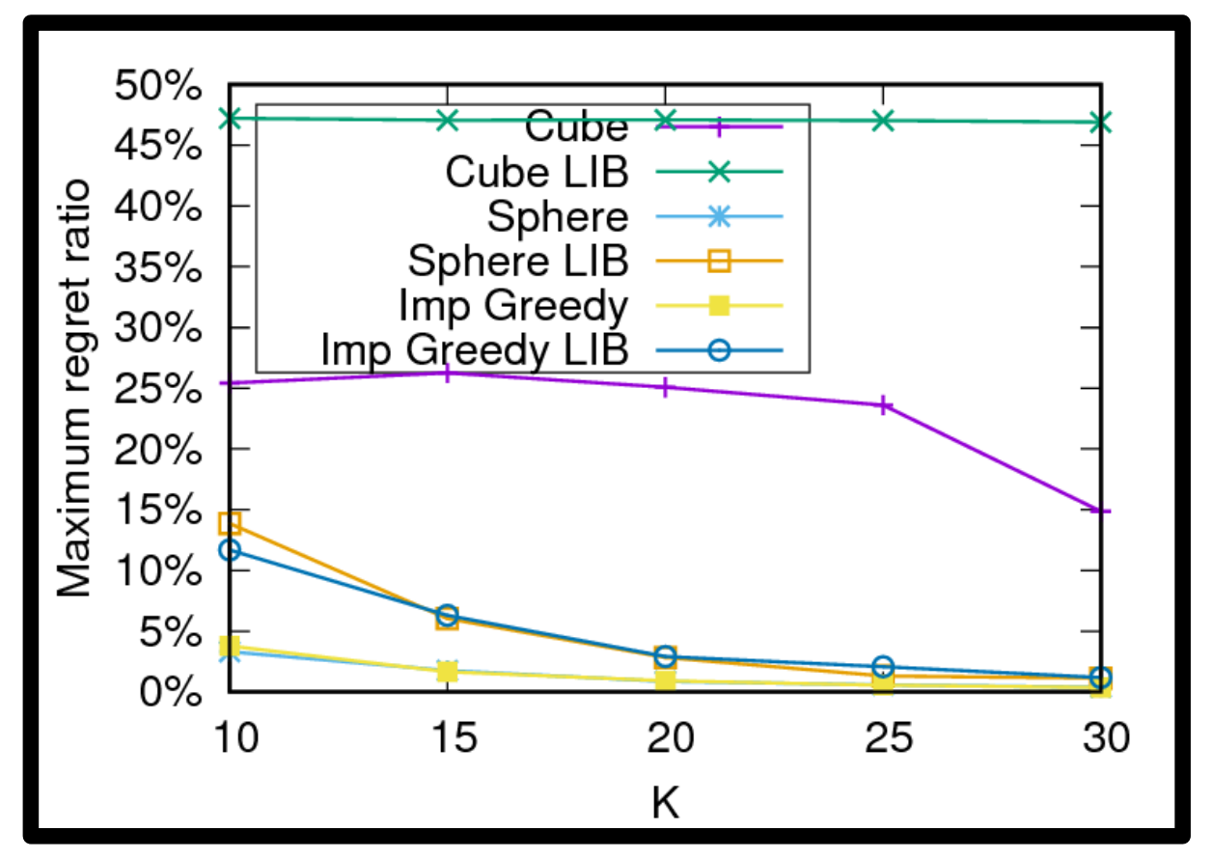

Fig. 2: Varying *k*

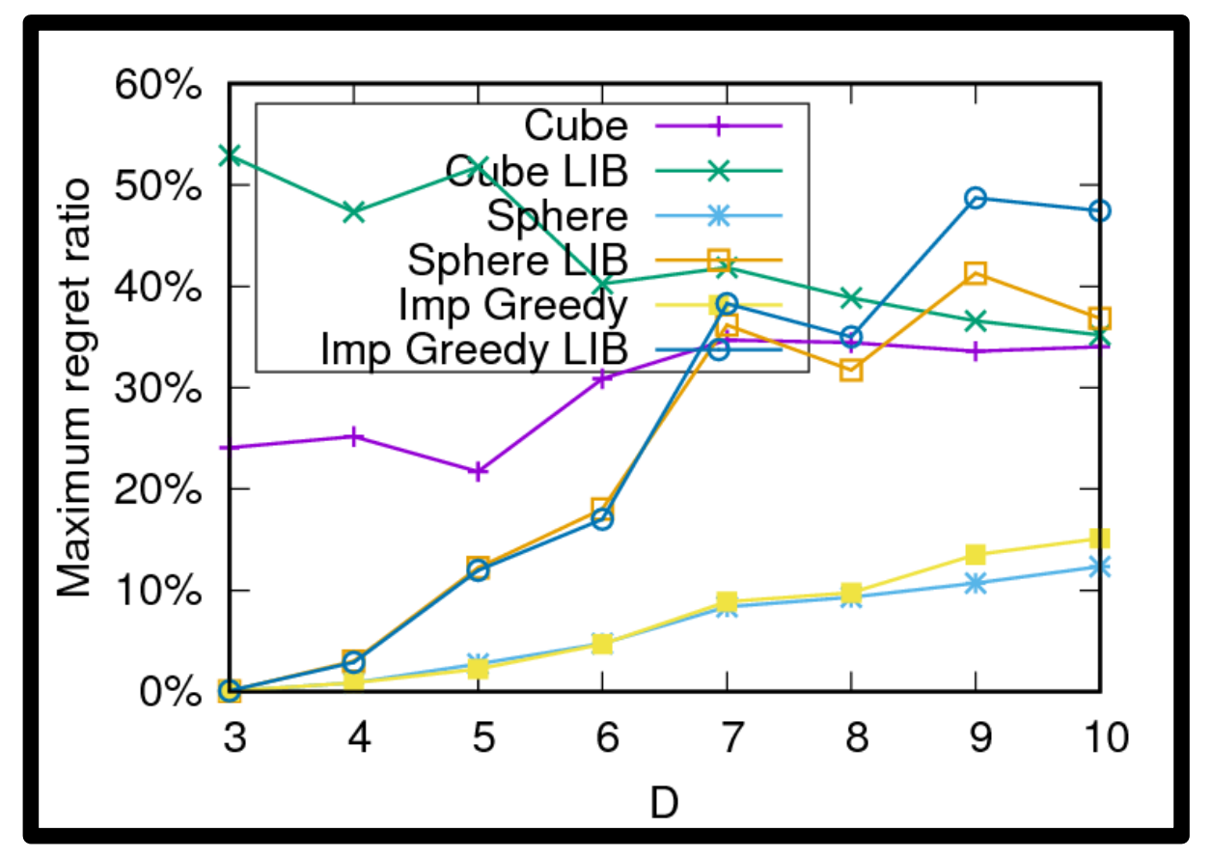

Fig. 3: Varying *d*

Future Work

In the future we believe that projects in the following areas are possible:

- Interactive Regret – using user interaction to lower regret ratios
- Expanding algorithms to include a larger set of utility functions to better model user preferences
- Lowering the time complexity of existing regret algorithms
- Tightening the bounds of existing regret algorithms

Selected References

S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In ICDE, 2001.
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.
D. Nanongkai, A. Das Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. Proceedings of the VLDB Endowment, 3(1-2), 2010.
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.

Acknowledgements

This project was funded by the Reid and Polly Anderson Summer Science Program at Denison University. Special thanks to Min Xie for support and guidance in our project.