An Efficient Algorithm for Computing k-Average-Regret Minimizing Sets in Databases

Xianhong Qiu[1] and Jiping Zheng[1,2](✉)

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
{qiuxianhong,jzh}@nuaa.edu.cn
[2] Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China

Abstract. Returning a small set of data points instead of the whole dataset to a user is a major task of a database system which has been studied extensively in recent years. In this paper, we study k-average-regret query, a recently proposed query, which uses "average regret ratio" as a metric to measure users' satisfaction to avoid the biases towards a few dissatisfied users that the best-known k-regret query suffers from. The main challenge of executing a k-average-regret query is the low efficiency of existing algorithms. Fortunately, as the average regret function exhibits the properties of *supermodularity* and *monotonictity*, the computational complexity of k-average-regret query can be significantly reduced exploiting lazy evaluations, thus leading to our accelerated algorithm which we called Lazy-Greedy. Experiments on both synthetic and real datasets confirm the efficiency and quality of output of our proposed algorithm.

Keywords: k-average-regret query · Representative skyline
Lazy evaluation

1 Introduction

Finding a small set of data points from a large dataset to support multi-criteria decision making is an important functionality in many application domains. A number of queries have been proposed in the literature to effectively support such functionality. Top-k [1] and skyline [2] are two representative queries. A top-k query returns k points that have the greatest scores under the utility/score functions specified by a user where k is a positive integer. A skyline query returns points that are not dominated by any other point in the database with no need to ask users to appoint utility functions. Instead, a concept called domination is applied in skyline queries. Specifically, a point p is said to dominate another point q if p is as good or better in all dimensions, and strictly better in at least one dimension. However, both queries suffer from some drawbacks. Top-k queries ask for users to specify their exact utility functions, which is difficult to most users. Skyline queries find all points that are not dominated by other points in the database, so the exact number of result set is uncontrollable and cannot be foreseen before the whole database is accessed. In addition, the output size of skyline queries will increase rapidly with the dimensionality.

Recently, the k-regret query [3] was proposed which integrates the merits of top-k and skyline queries. For k-regret queries, any utility function specified by a user is not required and the output size is controllable since only k points that minimize a criterion called maximum regret ratio are returned. However, k-regret queries only consider the regret ratio of the most unhappy user, in other words, there is only one point that satisfies a user among the selected points, so this query suffers from the drawback that it will be skewed towards the least satisfied users only and ignore the other users. Zeighami *et al.* [4] proposed "average regret ratio" as a metric to measure a user's satisfaction which gives a better impression of how a user in general feels towards the selected points and provided a result set of k points that the average regret ratio of the result set is minimized. Specifically, their method exploits N utility functions sampled from the probability distribution of all utility functions and picks out the point whose removal makes the average regret ratio minimized. Unfortunately, the calculation of the average regret ratio is time-consuming. Motivated by this, in this paper, an efficient algorithm called Lazy-Greedy is proposed with a (1-1/e) approximation guarantee to the optimum solution. Lazy-Greedy is extended from existing greedy algorithm by exploiting lazy evaluations and obtains significant speedups.

The main contributions of this paper are listed as follows.

1. Based on the supermodularity and monotonicity of the average regret ratio, we introduce an efficient approximation algorithm called Lazy-Greedy. The algorithm exploits some lazy evaluations to avoid some unnecessary calculations when picking out the point whose removal increases the average regret ratio the least.
2. Extensive experiments on both synthetic and real datasets are conducted to evaluate our method and the experimental results confirm that our proposed algorithm achieves the same minimum average regret ratio as WO-Greedy proposed in [4] but runs much faster.

The remainder of this paper is organized as follows. In Sect. 2, previous work related to this paper is discussed. Section 3 introduces the problem definition and background techniques which are applied in our algorithm. Followed by the accelerated greedy algorithm in Sect. 4. The performance of our algorithm compared with existing algorithms on synthetic and real datasets is presented in Sect. 5. Finally, Sect. 6 concludes this paper.

2 Related Work

Due to the drawbacks of top-k and skyline queries, a lot of alternatives have been proposed in recent years. First, efforts are put forward to improve top-k

queries. [5,6] asked users to specify some kinds of utilities/preferences to provide alternative ways to users to specify utility functions for top-k queries. Secondly, researchers attempt to reduce the output size of the skyline queries. The representative skyline [7,8] was proposed which returns k skyline points best representing the full skyline. Unfortunately, all these methods are not stable, scale-invariant or with deficiencies of top-k or skyline queries.

Recently, k-regret queries were first proposed in [3] which do not heavily rely on top-k queries and skyline queries and have been studied in [9–11] using different approaches. However, the k-regret queries have the deficiencies that they will be skewed towards the least satisfied users only, ignoring the other users, as they only consider the regret ratio of the most unhappy user. [4] proposed a k-average-regret query which returns a result set that minimizes the average regret ratio and can avoid the drawback that the k-regret query suffers from. But, the efficiency of the algorithm proposed in [4] is very low as it will result in a total running time of $O(dNn^3)$ where d is the dimensionality of the database, n is the size of the database and N is the number of utility functions. Our research aims at answering the k-average-regret query with an efficient algorithm.

3 Problem Definition and Background Techniques

In this section, we first formulate our problem and then point out the background techniques used in our algorithm.

3.1 Problem Definition

Let D be a set of n d-dimensional points over positive real values. Each point in D can be regarded as a tuple in the database. For each point $p \in D$, the value on the *i-th* dimension is represented as $p[i]$. We assume that smaller values are better. If users prefer large values, we convert them to small values by subtraction with the maximum value. Before we define our problem, definitions of utility function, regret ratio and average regret ratio are given [3,4].

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

Definition 2 (Regret Ratio). *Given a dataset D, a subset $S \subseteq D$ and a utility function u. The regret ratio of S, represented as $rr_D(S,u)$, is defined to be*

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

Definition 3 (Average Regret Ratio). *Given a dataset D, a subset $S \subseteq D$, a set U containing all utility functions and the probability distribution function*

$\eta(u)(u \in U)$ of the utility functions for different users. The average regret ratio of S, represented as $arr_D(S, U)$, is defined to be

$$arr_D(S, U) = \int_{u \in U} rr_D(S, u)\eta(u)du$$

Especially, if the utility functions are defined on a discrete space, the average regret ratio of S can be rewritten as follows:

$$arr_D(S, U) = \sum_{u \in U} rr_D(S, u)\eta(u)$$

The calculation of the average regret ratio $arr_D(S, U)$ involves calculating the average of the regret ratios of all utility functions. If the distribution of the utility functions is continuous, the calculation of the average regret ratio requires the calculation of a d-dimensional integral. If the set of the utility functions is defined on a discrete space, we can calculate the average regret ratio by summing for all the utility functions their regret ratios multiplied by their probabilities. In this paper, our utility functions U are sampled from the linear utility space, since linear utility functions are widely used in modeling users' preferences. Specially, a utility function u is linear if there exist non-negative reals v_1, v_2, $\cdots$, v_d such that $u(p) = \sum_{i=1}^{d} v_i \cdot p[i]$ for any d-dimensional point p. Alternatively, a linear utility function can be represented by a vector $u = (v_1, \cdots, v_d)$, *i.e.*, $u(p)$ is the dot product $u \cdot p$.

Problem Definition: Given a dataset D of size n, a positive integer k, a set U of utility functions of size N and the probability distribution function $\eta(u)(u \in U)$ of the utility functions, the problem of average regret ratio minimizing is trying to find a subset $S \subseteq D$ containing at most k points such that the average regret ratio is minimized while simultaneously keeping the query time as short as possible.

$$S = \arg \min_{s' \subseteq D, |S'|=k} arr_D(S', U)$$

3.2 Background Techniques

We will introduce the concept of supermodular function and describe supermodularity and monotonicity of it as they are the properties used in our Lazy-Greedy.

Definition 4 (Supermodularity). *A set function $f : 2^D \to \mathbb{R}^+$ is supermodular if for every $S_1, S_2 \subseteq D$ it holds that*

$$f(S_1) + f(S_2) \leq f(S_1 \cup S_2) - f(S_1 \cap S_2)$$

Definition 5 (Monotonicity). *A set function $f : 2^D \to \mathbb{R}^+$ is monotone and non-increasing if for every $S_1 \subseteq S_2 \subseteq D$, it holds that $f(S_1) \geq f(S_2)$.*

Lemma 1. *[4] The average regret ratio $arr_D(S, U)$ is a monotone non-increasing supermodular set function, namely, it satisfies the properties of supermodularity and monotonicity.*

Based on the definitions of supermodularity and monotonicity, Lemma 1 can be proved directly. Due to space limitation, we omit proofs here.

4 Lazy-Greedy Algorithm

In the following, we first describe how to determine the appropriate size of utility functions that are required to sample from the linear utility space and then present our Lazy-Greedy whose performance is boosted with lazy evaluations.

4.1 Sampling *N* Utility Functions

Since we have no idea of user's utility functions, the number of utility functions is infinite in linear utility space. We can select N utility functions based on the probabilities distribution of the utility functions. For this, we need to choose the number of utility functions N that approximates the true value of the average regret ratio with a high confidence and within a reasonable error parameter.

Theorem 1. *Given a confidence parameter $\delta \in (0, 1]$ and an error parameter $\epsilon \in [0, 1]$, then when the confidence is at least 1-δ and the calculated average regret ratio is within ϵ of its true value, N is at least $\frac{3\ln(\frac{1}{\delta})}{\epsilon^2}$.*

Proof. We need to show that for $0 < \epsilon \leq 1$,

$$Pr(\frac{X-\mu}{\mu} \geq \epsilon) = Pr(X \geq (1+\epsilon)\mu) \leq \delta \tag{1}$$

According to Chernoff bounds and let $\mu = N > 0$, we can get

$$Pr(X \geq (1+\epsilon)\mu) \leq e^{-\mu\epsilon^2/3} \leq e^{-N\epsilon^2/3} \leq \delta$$

Take the logarithm of both sides and rearrange it, we obtain $N \geq \frac{3\ln(\frac{1}{\delta})}{\epsilon^2}$. □

4.2 Lazy Evaluation for Greedy Algorithm

At each iteration, WO-Greedy [4] must identify the point p whose removal makes the average regret ratio $arr_D(S_i \backslash \{p\}, U)$ minimized then remove it from S_i, where S_i is the result set of the $(i+1)$-*th* iteration. Unfortunately, a large number of calculations are needed when we run WO-Greedy algorithm [4] which is time-consuming. The key insight from the supermodularity of $arr_D(S, U)$, the average regret ratio obtained by any fixed point $p \in D$ is monotonically non-decreasing during the iterations of removing points, *i.e.*, $arr_D(S_i \backslash \{p\}, U) \geq arr_D(S_j \backslash \{p\}, U)$, whenever $i \leq j$. Instead of recomputing for each point $p \in S_i$, we can use lazy evaluations to maintain a list of lower bounds $\{\triangle(p)\}$ on the average regret ratio sorted in ascending order. Then in each iteration, Lazy-Greedy needs to extract the minimal point $p \in \arg\min_{p^{'}:S_{i-1}\backslash\{p\}}\{\triangle(p^{'})\}$ from the ordered list and then updates the bound $\triangle(p) \leftarrow arr_D(S_{i-1}\backslash\{p\}, U)$. After this update, if $\triangle(p) \leq \triangle(p^{'})$, then $arr_D(S_{i-1}\backslash\{p\}, U) \leq arr_D(S_{i-1}\backslash\{p^{'}\}, U)$ for all $p \neq p^{'}$, and therefore we have identified the point with the minimal average regret ratio, without having to compute $arr_D(S_{i-1}\backslash\{p^{'}\}, U)$ for a potentially large number of point $p^{'}$. We set $S_i \leftarrow S_{i-1}\backslash\{p\}$ and repeat until $i = n-k$. This idea of using lazy evaluations is useful to our algorithm and can lead to orders of magnitude performance speedups. The pseudocodes of lazy-Greedy are shown in Algorithm 1.

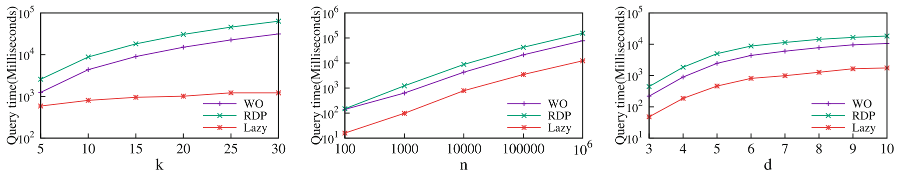
Algorithm 1. Lazy-Greedy(D, k, U)

```
Input: A set of $n$ $d$-dimensional points $D = \{p_1, p_2, \cdots, p_n\}$ and an integer $k$. $U$ denotes
       $d$-dimensional utility functions whose size is $N$.
Output: A result set $S$, $|S| = k$.
1 Initially, let $S = D, p^* = NULL$;
2 for ($i = 1$; $i \leq n-k$; $i++$) do
3     if $i = 1$ then
4         for each $p \in S$ do
5             calculate the value of $arr_S(S\backslash\{p\}, U)$;
6             $\triangle(p) = arr_S(S\backslash\{p\}, U)$;
7     $p^*$=Lazy-Evaluation$(D, S, i, \{\triangle(p)\}, U)$;
8     $S = S\backslash\{p^*\}$;
9 return $S$;
```

The calculation of the average regret ratio based on WO-Greedy takes time $O(dnN)$ and there are $O(n^2)$ iterations in the greedy algorithm thus resulting in a total running time of $O(dNn^3)$. For our Lazy-Greedy, the algorithm keeps track of the points which have not been visited and their average regret ratios are the smallest in the list $\{\triangle(p)\}$ instead of calculating the average regret ratio of all points that still inside the current result set S in each iteration. Hence, Lazy-Greedy has some important features. First, it provides a greedy solution identical to the solution provided by WO-Greedy [4]. Secondly, Lazy-Greedy is more efficient compared with WO-Greedy as it can avoid some calculations to minimum average regret ratio by exploiting lazy evaluations. Unfortunately, these cannot be demonstrated theoretically and it is easy to build worst-case examples for which Lazy-Greedy requires the same number of calculations as WO-Greedy. However, subsequent experiments provide an experimental confirmation of the efficiency of Lazy-Greedy when applied to answer the k-average-regret minimizing set. In a sense, Lazy-Greedy is still optimal in terms of number of calculations.

5 Experimental Results

In this section, we show the performance of our proposed algorithm via experiments. The algorithms were implemented in C++ and run on a 64-bit 3.3 GHz Intel Core machine which was running Ubuntu 14.04 LTS operating system. We ran our experiments on both synthetic and real datasets. Unless stated explicitly, for synthetic datasets created adopting the dataset generator of [2], the number of points is set to 10,000 (*i.e.*, $n = 10,000$) and the dimensionality is set to 6 (*i.e.*, $d = 6$) and k is set to 10. The real-world datasets include a 5-dimensional *ElNino*[1] of 178,080 points, a 6-dimensional *Household*[2] of 127,391 points and an 8-dimensional *NBA*[3] of 17,265 points. All experiments are conducted with 964 utility functions sampled from a uniform distribution on the linear class of the utility functions where $\epsilon = 0.0707716$ and $\delta = 0.2$. Moreover, like studies in the literature [3,9], we computed the skyline first and our queries returned anywhere from 5 to 30 points on these datasets except the ElNino dataset (when $k > 10$, the average regret ratio on ElNino dataset is close to 0, so we only show the results when k is small). In our experiments, we consider Lazy-Greedy introduced in this paper. To verify the superiority of our proposed algorithm, we compare it with RDP-Greedy (they called Greedy in [3]) and WO-Greedy [4]. We measure the computational cost in terms of the running time of CPU and the quality of result set by means of the average regret ratio.

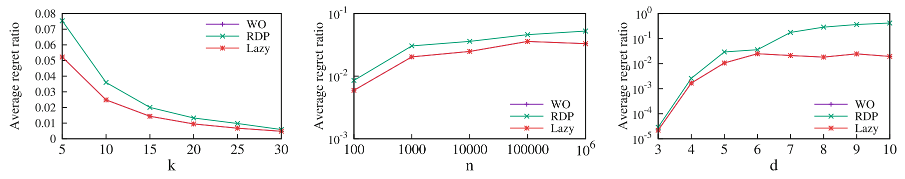


(a) Varying k (b) Varying n (c) Varying d

Fig. 1. Query time on the anti-correlated dataset

Results on Synthetic Datasets: The query time on anti-correlated datasets for different k, n and d are shown in log scale in Fig. 1. In all cases, our Lazy-Greedy has negligible query time as unnecessary calculations to average regret ratio are avoided while RDP-Greedy and WO-Greedy result in a much longer query time. The average regret ratio on anti-correlated datasets with k, d and n varied are presented in Fig. 2. Lazy-Greedy and WO-Greedy have the same and low average regret ratio which are much lower than RDP-Greedy as they share the same greedy skeleton. Besides, the average regret ratio of all algorithms degrade with the increase of k. But, the average regret ratio increases with d for all algorithms due to the curse of dimensionality, and increases with n.

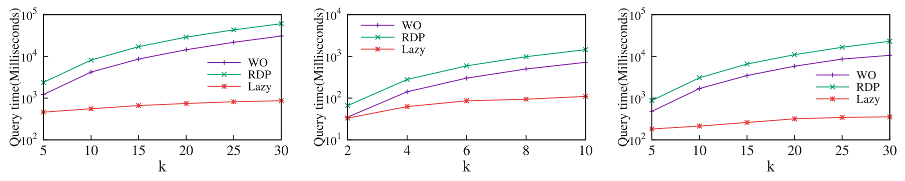
[1] http://archive.ics.uci.edu/ml/datasets/El+Nino.
[2] http://www.ipums.org/.
[3] https://www.basketball-reference.com/.



(a) Varying k (b) Varying n (c) Varying d

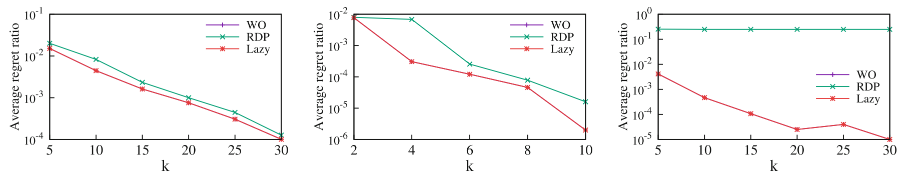
Fig. 2. Average regret ratio on the anti-correlated dataset

Results on Real Datasets: Figure 3 shows the query time on real datasets. The query time of all algorithms increase with k, but our Lazy-Greedy keeps much less query time and maintains a stable level as we analyze in the previous section. The average regret ratio of all algorithms for different k are shown in Fig. 4. We observe similar trends as the experiments on synthetic datasets presented in Fig. 2(a). Besides, similar to the experiments on synthetic datasets, our Lazy-Greedy achieves near-minimal average regret ratio with substantially shorter query time compared with RDP-Greedy and WO-Greedy as it's optimal in terms of numbers of calculations.



(a) Household (b) ElNino (c) NBA

Fig. 3. Query time on the real datasets



(a) Household (b) ElNino (c) NBA

Fig. 4. Average regret ratio on the real datasets

6 Conclusions

This paper studies a k-average-regret query and proposes an accelerated algorithm exploiting some lazy evaluations. Experiments on synthetic and real datasets confirm the efficiency and effectiveness to answer k-average-regret queries. Future work aims at extending a single user into a multi-user.

Acknowledgment. This work is partially supported by the National Natural Science Foundation of China under grants U1733112,61702260, Funding of Graduate Innovation Center in NUAA under grant KFJJ20171605.

References

1. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4), 11:1–11:58 (2008)
2. Börzsöny, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
3. Nanongkai, D., Sarma, A.D., Lall, A., Lipton, R.J., Xu, J.: Regret-minimizing representative databases. In: VLDB, pp. 1114–1124 (2010)
4. Zeighami, S., Wong, R.C.W.: Minimizing average regret ratio in database. In: SIGMOD, pp. 2265–2266 (2016)
5. Mindolin, D., Chomicki, J.: Discovering relative importance of skyline attributes. Proc. VLDB Endow. **2**, 610–621 (2009)
6. Lee, J., You, G.W., Hwang, S.W.: Personalized top-k skyline queries in high-dimensional space. Inf. Syst. **34**(1), 45–61 (2009)
7. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representative skyline operator. In: ICDE, pp. 86–95 (2007)
8. Tao, Y., Ding, L., Lin, X., Pei, J.: Distance-based representative skyline. In: ICDE, pp. 892–903 (2009)
9. Peng, P., Wong, R.C.W.: Geometry approach for k-regret query. In: ICDE, pp. 772–783 (2014)
10. Xie, M., Wong, R.C.W., Li, J., Long, C., Lall, A.: Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In: SIGMOD (2018)
11. Qi, J., Zuo, F., Samet, H., Yao, J.: K-regret queries using multiplicative utility functions. ACM Trans. Database Syst. **43**, 10 (2018)