Efficient Processing of *k*-regret Queries via Skyline Priority

Sudong Han[1], Jiping Zheng[1,2](✉), and Qi Dong[1]

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

Abstract. Extracting interesting points from a large database is an important problem in multi-criteria decision making. The recent proposed k-regret query attracted people's attention because it does not require any complicated information from users and the output size is controlled within k for users easily to choose. However, most existing algorithms for k-regret query suffer from a heavy burden by taking the numerous skyline points as candidate set. In this paper, we define a subset of candidate points from skyline points, called *prior skyline points*, so that the k-regret algorithms can be applied efficiently on the smaller candidate set to improve their performance. A useful metric called skyline priority is proposed to help determine the candidate set and corresponding strategies are applied to accelerate the algorithm. Experiments on synthetic and real datasets show the efficiency and effectiveness of our proposed method.

Keywords: k-regret query · Skyline priority · Prior skyline points Candidate set determination

1 Introduction

Returning points that users may be interested in is one of the most important goals for multi-criteria decision making. Top-k [1] and skyline [2] queries are two well-studied tools used to return a representative subset of a large database. But these two types of queries suffer in either requiring a predefined utility function to model user's preference over the points, or returning an uncontrollable number of points. To avoid the deficiencies of top-k and skyline queries, Nanongkai *et al.* [3] first proposed regret-based query which returns k points that minimize a criterion called the maximum regret ratio. It quantifies how regretful a user is if s/he gets the best point among the selected k points but not the best point among all the tuples in the database.

Technically, the input of k-regret algorithms is the points in the whole dataset, but existing algorithms [3–5] usually take the skyline points as the candidate points. This is because the points dominated by skyline points have less possibility of being k representative points. By removing the non-skyline points, the running time of the algorithms can be largely reduced. However, taking skyline points as candidate points is of low efficiency for computing k representative points because the size of skyline points grows exponentially with dimensionality [6]. Extremely, the size of skyline points is even close to the whole dataset when the dimension is high.

Motivated by these, we devote to finding a small size of candidate set from the entire skyline points so that the k-regret algorithms can be applied efficiently on the smaller candidate set to improve its efficiency. In this paper, we define a set of candidate points called *prior skyline points* based on skyline priority. Skyline priority is a metric that indicates the least dimensionality of subspace in which a point p is a skyline point. Intuitively, a point with a high skyline priority is more interesting as it can be dominated on fewer combinations of dimensions. To avoid the expensive cost of calculating skyline priority given by the naive method, we further propose SP_{SKYCUBE} using sharing and promoting strategies to determine the candidate set. The main contributions of this paper are listed as follows:

– Skyline priority based on subspace skyline is firstly proposed to rank skyline points for candidate set determination of k-regret query.
– We present efficient algorithm based on sharing and promoting strategies to compute skyline priority.
– Results of extensive experiments on both synthetic and real datasets confirm the efficiency of our proposed algorithm.

The rest of this paper is organized as follows. We present related work in Sect. 2. Section 3 contains the required preliminaries and problem definition. In Sect. 4, we present our algorithm. We show our experimental results in Sect. 5. In Sect. 6, we conclude this paper and point out possible future work.

2 Related Work

Motivated by the deficiencies of top-k and skyline queries, various approaches [7, 8] were proposed to find the k skyline points that represented the skyline points. Unfortunately, the two methods are neither stable, nor scale-invariant. The most relevant study to our work is k-regret query was proposed by Nanongkai *et al.* [3]. A number of algorithms were proposed to extend the concept to some extent [4,5,9] However, these studies aim at minimizing the maximum regret ratio of a selected set, ignoring the importance of reducing the size of candidate set to improve the efficiency.

There are also some researches related to subspace skyline. Yuan *et al.* [10] and Pei *et al.* [11] proposed the concept of SKYCUBE, which computes the skylines of all possible subspaces. But the methods mentioned above focus on computing skylines in subspaces efficiently, ignoring to explore the properties, such as skyline priority, and they have not been involved in reducing the number of candidate points for k-regret query.

Table 1. English teacher recruiting example

Candidate	S_1	S_2	S_3	*Priority*
p_1	8.4	5.1	8.3	2
p_2	5.2	6.2	6.6	3
p_3	9.3	6.1	7.4	1
p_4	4.7	5.2	10	1
p_5	4.2	8.8	9.2	2
p_6	5.6	10	3.1	1
p_7	8.2	9.2	2.5	2
p_8	5.1	9.3	5.3	2

Table 2. Skylines of all subspaces

Subspace	*Skyline*
S_1	$\{p_3\}$
S_2	$\{p_6\}$
S_3	$\{p_4\}$
S_1, S_2	$\{p_3, p_6, p_7\}$
S_2, S_3	$\{p_4, p_5, p_6, p_8\}$
S_1, S_3	$\{p_1, p_3, p_4\}$
S_1, S_2, S_3	$\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$

3 Preliminaries

To define the problem, we need to introduce the concepts of k-regret query and explain how the size of candidate set influences the performance of k-regret algorithms.

First, we assume that the user's preference to a point can be expressed using a utility function f, where $f(p)$ is the utility of a point p for the user with the utility function f. Then the regret ratio of a user with the utility function f after seeing a subset R instead of a database D is $rr_D(R, f) = \frac{max_{p \in D} f(p) - max_{p \in R} f(p)}{max_{p \in D} f(p)}$.

Since utility functions vary across users, any algorithm for a k-regret query must minimize the maximum regret ratio for a class of utility functions. In this paper, we only consider linear utility functions, denoted by $\mathcal{F}$, because they are very popular in modeling user preferences [3,12,13]. Thus the worst possible regret for any user with a utility function in $\mathcal{F}$ is defined as follows.

Definition 1 (Maximum Regret Ratio). *Define* $rr_D(R, \mathcal{F}) = sup_{f \in \mathcal{F}} \frac{max_{p \in D} f(p) - max_{p \in R} f(p)}{max_{p \in D} f(p)}$.

Now we explain how the size of candidate set influences the performance of k-regret algorithms. One of the classical algorithms for k-regret query is Greedy [3]. Based on the idea of "greedy", Greedy iteratively construct the solution by selecting the point that currently contributes to the greatest value of maximum regret ratio. To be specific, the algorithm needs to inspect each of the candidate points (except selected points) by computing the maximum regret ratio to decide whether the point will be included in the result set or not. So the running time of the algorithm is largely dependent on the size of the candidate set. By reducing the size of candidate set, the efficiency of the algorithm can be greatly improved. Now we explain the idea of reducing the size of candidate set via skyline priority by a simple example.

Consider the following example, a Chinese school wants to recruit k teachers. They measure the candidates from d aspects, such as the capability of *speaking*, *listening*, *reading*, *writing*, *educational background*, *computer skills* and so on. Table 1 shows a list of candidates which are also the skyline points of all the candidates (only consider three aspects: S_1: *reading*, S_2: *writing*, S_3: *speaking*). The values in Table 1 represent their ability in each corresponding attribute where greater values are better. Now our task is to select k best persons from the candidates. Intuitively, the school may want to hire someone who is the most popular in one aspect among all the candidates. For example, the school may prefer p_3, p_4 and p_6 with the highest priority because they are the best in S_1, S_3 and S_2 respectively. So, when choosing persons from the candidates, p_3, p_4 and p_6 have more probability of being selected than other candidates. When it comes to higher dimensions, the priority of higher dimensional skyline points are lower. For example, we observe that p_2 is only the skyline of the full space and p_2 has the lowest priority. So the possibility of being selected of p_2 is small for it is mediocre in all attributes.

The above example shows that if we consider a smaller subset of skyline points as the candidate set, the points with higher skyline priority are preferred. So we define the problem of candidate set determination based on skyline priority.

Problem Definition. Given a set D of n points in d dimensions, our problem of processing k-regret query via skyline priority is to determine the candidate set of k-regret query by selecting the points with high skyline priority, meanwhile keeping the maximum regret ratio as small as possible.

4 Algorithm

In this section, we first give the formal definition of skyline priority and we concentrate on the computation of skyline priority and develop efficient algorithm to solve our problem.

Given a d-dimensional dataset D, $\mathcal{S}$ is the dimension set consisting of all the d dimensions and S_i represents each dimension. Let p and q be two data points in D, we denote the value of p and q on dimension S_i as $p.S_i$ and $q.S_i$. For any dimension set $\mathcal{B}$, where $\mathcal{B} \subseteq \mathcal{S}$, p dominates q if $\forall S_i \in \mathcal{B}$, $p.S_i \leq q.S_i$ and $\exists S_j \in \mathcal{B}, p.S_j > q.S_j (i \geq 1, j \leq d)$. The skyline query on $\mathcal{B}$ returns all data points that are not dominated by any other points on $\mathcal{B}$. The result is called subspace skyline points, denoted by $SKY_{\mathcal{B}}(D)$. See the running example in Table 1, $SKY_{\mathcal{B}_1}(D) = \{p_3\}$, $SKY_{\mathcal{B}_2}(D) = \{p_3, p_6, p_7\}$, where $\mathcal{B}_1 = \{S_1\}$ and $\mathcal{B}_2 = \{S_1, S_2\}$.

Given D on $\mathcal{S}$, a Skycube consists of a set of subspace skyline points in $2^d - 1$ non-empty subspaces. The Skycube is shown in Table 2. In Skycube, each $SKY_{\mathcal{B}}(D)$ is called cuboid $\mathcal{B}$. For two cuboids $\mathcal{B}_1$ and $\mathcal{B}_2$ in the Skycube, if $\mathcal{B}_1 \subseteq \mathcal{B}_2$, we call $\mathcal{B}_2$ ($\mathcal{B}_1$) ancestor (descendant) cuboid. If their levels differ by one, we also call $\mathcal{B}_2$ ($\mathcal{B}_1$) parent (child) cuboid.

Definition 2 (Skyline Priority). *Given a skyline point p, skyline priority, denoted by $prt(p)$ is the least dimensionality of subspace in which p is a skyline point. The smaller the value of $prt(p)$ is, the higher priority p has.*

For example, Table 2 shows all the subspace skyline points. p_3 is the skyline point of subspace $\{S_1\}$, $\{S_1, S_2\}$, $\{S_1, S_3\}$ and $\{S_1, S_2, S_3\}$, and the least dimensionality of the subspaces is 1, so the skyline priority of p_3 is 1. Table 1 shows the priority of 8 candidates.

Now we focus on how to compute skyline priority efficiently. The naive method to compute prior skyline points is to enumerate all the $2^d - 1$ subspace skyline points and tag priority of the point with the least dimensionality. The low efficiency of the naive method lies in that the skyline points of each subspace are computed separately, resulting in a great number of redundant computation.

Algorithm 1. SP_{SKYCUBE} algorithm

```
Input: dataset $D$, $x\%$, $n$, $k$.
Output: the set of prior skyline points $SP$
1  sort $D$ on every dimension to form $d$ sorted list $L_{S_i}$;
2  initialize $SP$ to be empty;
3  $m = MAX(n * x\%, k)$;
4  for each cuboid $\mathcal{B}$ from bottom to top of the skycube do
5      $SKY$ = the union of all the child cuboids;
6      choose a sorted list $L_{S_i}$;
7      for $q$ in $D \setminus SKY$ do
8          evaluate($q, SKY_{\mathcal{B}}(D)$);
9      if $|SP| \leq m$ then
10         remove the point with the largest priority in $SP$ when $|SP| = k$;
11         insert $q$ into $SP$;
12 return $SP$;
```

To cover the problem of repeated computation, we proposed a sharing strategy of subspace skyline points based on the relationship between a parent and child cuboid, as stated in the following Lemma 1.

Lemma 1. *Given a set D of data points on dimension set $\mathcal{S}$. For any two data points p and q, $p.S_i \neq q.S_i (\forall S_i \in \mathcal{S})$, for two sub dimension sets $\mathcal{B}_1, \mathcal{B}_2 (\mathcal{B}_1, \mathcal{B}_2 \subseteq \mathcal{S})$, where $\mathcal{B}_1 \subset \mathcal{B}_2$, $SKY_{\mathcal{B}_1}(D) \subseteq SKY_{\mathcal{B}_2}(D)$.*

The conclusion that the union of child cuboids belongs to the parent cuboid can be easily obtained. Therefore, if we compute the priority from bottom to top, once the priority of a point p in the child cuboid is settled, it is unnecessary to compute the priority in parent cuboids. For example, in Table 2, since p_3 is the skyline of $\{S_1\}$, $prt(p_3) = 1$. So there is no need to compute the priority of p_3 in the skyline of subspace $\{S_1, S_2\}$. A large number of skyline computation are saved due to this strategy.

Besides, a pre-sorting of the data is executed for optimization. This is based on the observation that the efficiency of skyline computation is dependent on the order in which the data points are processed. Furthermore, we also define a value called *Entropy* to promote skyline points without dominating test where $E_{\mathcal{B}}(p) = \sum_{\forall a_i \in \mathcal{B}} p(a_i)$. The procedure using the promoting strategy is shown in Algorithm 2. This is based on the property that for two data points p and q, if

$E_{\mathcal{B}}(p) \leq E_{\mathcal{B}}(q)$ and p is already a skyline point, then q is definitely a skyline point (Line 2 to 3), avoiding extra dominating test between p and q. Otherwise, we do a dominating test to decide whether q is a skyline point (Line 4 to 6).

Based on the above techniques, we develop the algorithm SP_{Skycube}, which computes skyline priority and determines the candidate set. The main procedure of our approach is shown in Algorithm 1. At the beginning of the algorithm, the pre-sorting operation of the data is invoked for optimization (Line 1). m, the maximum of the size of candidate points and k, ensures at least k points in candidate set. The tunable parameter x provides a flexible tradeoff between CPU time and maximum regret ratio for users. If a data is a skyline point of child cuboid, there is no necessary to compute its priority. Otherwise it is compared to the current skyline to determine its priority by calling the function *Evaluate* (Line 7 to 8). To get the set of at least k candidate points with high skyline priority, a point p is inserted into SP when the set SP has fewer than m points. Specially, the point with largest priority is replaced by p if p has smaller priority and the size of SP is m.

Algorithm 2. evaluate(p, $SKY_{\mathcal{B}}(D)$)

```
Input: a data point $q$, the computed cuboid $\mathcal{B}$
Output: rank the $q$ with skyline priority if $p$ is a skyline of $\mathcal{B}$
1 for each $p$ in $SKY_{\mathcal{B}}(D)$ do
2     if $f_{\mathcal{B}}(q) > f_{\mathcal{B}}(p)$ then
3         insert $q$ into $SKY_{\mathcal{B}}(D)$; $prt(p) = |\mathcal{B}|$; return;
4     else if $p$ dominates $q$ on $\mathcal{B}$ then
5         discard $q$; return;
6     insert $q$ into $SKY_{\mathcal{B}}(D)$; $prt(p) = |\mathcal{B}|$;
```

5 Experiments

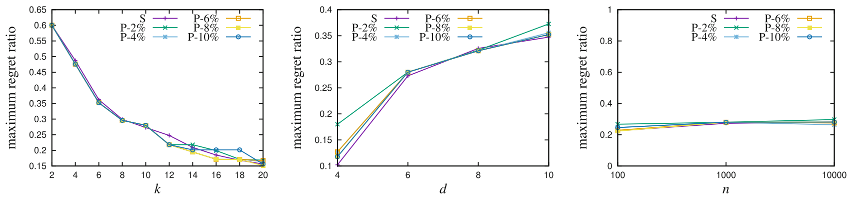
In this section, we show the prior skyline points are efficient to select the k-representative skyline points by comparing them with the skyline points, running on the k-regret algorithm Greedy [3]. All the experiments were implemented in C++ and the experiments were performed on a 3.3 GHz CPU and 4 G RAM machine running Ubuntu 14.04 LTS.

We ran our experiments on both synthetic and real datasets. For synthetic datasets, we use anti-correlated synthetic data, generated by the generator in [2]. Unless specially stated, the cardinality is ten thousand ($n = 10{,}000$), the dimensionality is six ($d = 6$), and $k = 10$. The real-world datasets we adopted are three datasets called *Household* (www.ipums.org), *Color* (kdd.ics.uci.edu), and *NBA* (www.basketballreference.com), which are widely used in this area.

We conduct our experiments on two kinds of candidate sets, namely skyline points (S) and prior skyline points ($P - x\%$), where $x\%$ means the candidate set is composed of the top $x\%$ prior skyline points. Then, we show the prior skyline points can be used for the algorithm Greedy for k-regret query in terms of maximum regret ratio and running time. Different values of x (x = 2, 4, 6, 8, 10) have been chosen.

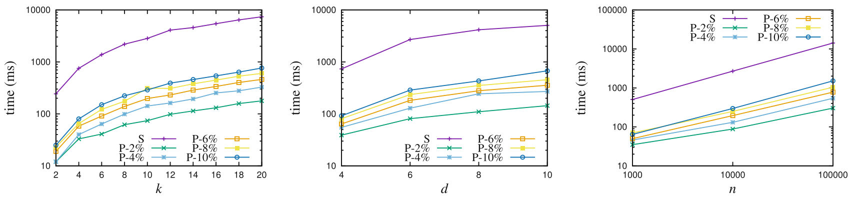
The effects of P on maximum regret ratio and time on anti-correlated datasets for different k are shown in Figs. 1(a) and 2(a) respectively. When k increases, the maximum regret ratio of all candidate sets decreases which is in accordance with the results in [3]. We observe that the maximum regret ratio of P is close to S or even smaller in some situation while the running time is only $\frac{1}{10}$ of S. This is because the candidate points P with high skyline priority have more possibility to be the representative skyline points. From the above, we have the conclusion that prior skyline points can efficiently speed up the original algorithm with a close maximum regret ratio.

Besides, we evaluate the effect of different cardinalities and dimensionalities on our algorithm. Figure 1(b) shows that when d increases, the maximum regret ratio of different candidate sets increases. Figure 1(c) shows that the maximum regret ratio is independent of n. The same conclusion can be reached that the maximum regret ratio of prior skyline points is close to S and the running time in Fig. 2(b) and (c) is much less than that of S.



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

Fig. 1. Effect on maximum regret ratio of different candidate sets



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

Fig. 2. Effect on time of different candidate sets.eps

We can observe the similar trends on real datasets. Similar to the experiments on synthetic datasets, with prior skyline points, the algorithm GREEDY achieves a close maximum regret ratio with less running time shown compared with the skyline points. The figures are omitted due to space limitation.

6 Conclusions and Future Work

We studied the problem of reducing the size of candidate set for k-regret query basing on skyline priority. Skyline priority is proposed to rank skyline points and efficient algorithm is provided to determine the candidate set for k-regret query. Extensive experiments verify the efficiency and accuracy of our method. Since our candidate set are the points of "the best - *higher skyline priority* of the best - *skyline points of the whole dataset*", we do think there are certain theoretical results to be found to guarantee our proposed method to be efficient and effective. We leave this as an open problem and also as our future work.

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 KFJJ20171601.

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), 1–58 (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. VLDB **3**, 1114–1124 (2010)
4. Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Efficient algorithms for k-regret minimizing sets. CoRR, abs/1702.01446 (2017)
5. 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)
6. Godfrey, P.: Skyline cardinality for relational processing. In: Seipel, D., Turull-Torres, J.M. (eds.) FoIKS 2004. LNCS, vol. 2942, pp. 78–97. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24627-5_7
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. Qi, J., Zuo, F., Samet, H., Yao, J.C.: From additive to multiplicative utilities, K-regret queries (2016)
10. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: VLDB, pp. 241–252 (2005)
11. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: a semantic approach based on decisive subspaces. In: VLDB, pp. 253–264 (2005)
12. Nanongkai, D., Lall, A., Das Sarma, A., Makino, K.: Interactive regret minimization. In: SIGMOD, pp. 109–120 (2012)
13. Chester, S., Thomo, A., Venkatesh, S., Whitesides, S.: Computing k-regret minimizing sets. In: VLDB, pp. 389–400 (2014)