Efficient Processing of k-regret Queries via Skyline Frequency

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. Helping end-users to find the most desired points in the database is an important task for database systems to support multi-criteria decision making. The recent proposed k-regret query doesn't ask for elaborate information and can output k points 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 aim at decreasing the candidate points from skyline points to a relative small subset of skyline points, called *frequent skyline points*, so that the k-regret algorithms can be applied efficiently on the smaller candidate set to improve their efficiency. A useful metric based on subspace skyline called skyline frequency is adopted to help determine the candidate set and corresponding algorithm is developed. Experiments on synthetic and real datasets show the efficiency and effectiveness of our proposed method.

Keywords: Regret minimization query
Candidate set determination · Skyline frequency
Frequent skyline points

1 Introduction

Extracting a few points to assist end-users to make multi-criteria decisions is an important functionality of database systems. Top-k [1] and skyline [2] queries are two well-studied tools that can effectively reduce the output size. But top-k query requires the utility functions from users and skyline query cannot control the output size. To solve these problems, Nanongkai *et al.* [3] first proposed regret-based query which gives the user a concise, summary result of the entire database and outputs k points that minimize the users' maximum regret ratio.

Technically, the input of k-regret algorithms is the points in the whole dataset, but existing algorithms [3–7] 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. However, taking skyline points

X. Meng et al. (Eds.): WISA 2018, LNCS 11242, pp. 434–441, 2018.
https://doi.org/10.1007/978-3-030-02934-0_40

as candidate points is of low efficiency for computing k representative points because the size of skyline points grows exponentially with dimensionality [8].

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 *frequent skyline points* based on skyline frequency. Skyline frequency is a metric that indicates how often the skyline points have been returned in the skyline when different numbers of dimensions are considered. Intuitively, a point with a high skyline frequency is more interesting as it can be dominated on fewer combinations of dimensions.

To avoid the expensive cost of calculating skyline frequency given by the naive method, we propose SF_{APPROX} to determine the candidate set. The main contributions of this paper are listed as follows:

- We define the concept of frequent skyline points based on skyline frequency which provide candidate points for k-regret query.
- We present efficient algorithm to determine the candidate points and corresponding strategy is provided to make a tradeoff between the maximum regret ratio and time complexity.
- Extensive experiments on both synthetic and real datasets are conducted to evaluate our proposed method and the experimental results confirm the efficiency and effectiveness of our proposed algorithm.

The rest of this paper is organized as follows. We present related work in Sect. 2. Section 3 contains required preliminaries and problem definition for our approach. In Sect. 4, we describe our algorithms to determine the candidate set. We show our experimental results in Sects. 5 and 6 concludes this paper and points out possible future work.

2 Related Work

Motivated by the deficiencies of top-k and skyline queries, the k-regret query was proposed by Nanongkai *et al.* [3]. A number of algorithms were proposed to extend the concept to some extent [4,9]. Recently, a number of algorithms [5–7] were proposed to obtain a solution with a small maximum regret ratio. Specially, the algorithm Sphere proposed by [7] is an elegant algorithm, which is considered as the state-of-art algorithm, because the maximum regret ratio of the result is no restriction bound and can be executed on datasets of any dimensionality. 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 can only compute the skylines in subspace, they have not been involved in reducing the number of candidate points for k-regret query.

We adopted the metric for ranking skyline points called skyline frequency proposed by [12]. However, the efficiency of skyline frequency is hard to verify. In this paper, we combine the concept of skyline frequency with our candidate set determination for k-regret query and provide a reasonable verification to the superiority of the points with high skyline frequency.

3 Preliminaries

In this section, the concepts of k-regret query and candidate set are introduced to bring out our problem definition.

Given a d-dimensional dataset D with n points and an output size k, where $d \leq k \leq n$, the k-regret query takes a dataset D as input and outputs a set R of k points from D such that the maximum regret ratio of R is minimized.

A user's preference to a point is represented by the utility function f. Thus the *utility* of a point p for the user with utility function f is $f(p)$. The maximum utility derived from a dataset D is the gain of D, denoted by $gain(D, f)$. If we consider $gain(D, f)$ as the user's satisfaction to the set D, then the *regret* the user seeing a representative set R instead of the whole database is $gain(D, f) - gain(R, f)$ and the regret ratio is $rr_D(R, f) = \frac{gain(D,f)-gain(R,f)}{gain(D,f)}$.

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 [3,4]. 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* $rr_D(R, \mathcal{F}) = sup_{f \in \mathcal{F}} \frac{max_{p \in D} f(p) - max_{p \in R} f(p)}{max_{p \in D} f(p)}$.

We take one of the classical k-regret query algorithms RDPGreedy [3] for example to illustrate how the size of candidate set affects the running time. According to the definition of maximum regret ratio, RDPGreedy needs to find the point that currently contributes to the 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. 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 frequency by a simple example.

Considering the following example, a Chinese school wants to recruit k English teachers. They measure the candidates from d aspects, such as the capability of *speaking*, *listening*, *reading*, *writing* and so on, where greater values are better. Table 1 shows a list of candidates which are also the skyline points of all the candidates (only consider the first three aspects). Intuitively, we always want to hire someone who is competent in more aspects among all the candidates. Table 2 lists all the subspace skylines. We observe that p_3 is the skyline of

Table 1. English teacher recruiting example

Candidate	S_1	S_2	S_3	*Frequency*
p_1	8.4	5.1	8.3	2
p_2	5.2	8.5	6.6	1
p_3	9.3	6.1	7.4	4
p_4	4.7	5.2	10	4
p_5	4.2	8.8	9.2	2
p_6	5.6	10	3.1	4
p_7	8.2	9.2	2.5	2
p_8	1.1	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	$\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$

subspace $\{S_1\}$, $\{S_1, S_2\}$, $\{S_1, S_3\}$ and $\{S_1, S_2, S_3\}$, which means p_3 is better than anyone else when considering these aspects mentioned above. So, if we choose only one person, p_3 has more probability of being selected. On the contrary, we observe that p_2 is not the skyline of any subspace, so the possibility of being selected of p_2 is small.

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

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

4 Candidate Set Determination via Skyline Frequency

In this section, we mainly concentrate on the computation of skyline frequency 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, 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.

Definition 2 (Skyline Frequency). *Given a skyline point $p \in D$, skyline frequency, denoted by $freq(p)$ is the number of subspaces in which p is a skyline point.*

For example, considering the example in Table 1, we can get all the subspace skyline points (Table 2) and count the $freq(p)$ of each point. The skyline frequencies of each skyline point are shown in Table 1.

The naive skycube-based approach, called SF_{SKYCUBE}, calculates the skyline frequency by dividing the full space into all $2^d - 1$ subspaces and counting the number of occurrences of each subspace skyline points. However, it requires the skylines to be computed for each of an exponential number of combinations of the dimensions, and the algorithm does not scale well with dimensionality. Thus, we propose an approximate algorithm named SF_{APPROX} to overcome these two drawbacks. From this point of view, to avoid the high complexity of precise counting, we present an effective approximate counting method that is based on extending a Monte-Carlo counting algorithm [13]. In this section, we illustrate how we adapt the method to combine with the computation of skyline frequency and show the optimization we do to make the algorithm more effective. The main procedure of our approach is shown in Algorithm 1.

Algorithm 1. SF_{APPROX} algorithm

Input: dataset D, n, $x\%$, k, δ, ϵ.
Output: the set of candidate points FP

```
1  Sort the points in non-descending order of the sum of dimension values;
2  initialize FP to be empty, θ = 0, m = MAX(n * x%, k), T = 2nln(2/δ)/ε²;
3  for i = 1 to T do
4      randomly choose a subspace B ⊆ S;
5      SKY_B(D, θ) = SkylineWithThreshold(D, B, θ, m, |FP|);
6      for each point p ∈ SKY_B(D) do
7          fre(p) + +;
8          if |FP| < m or fre(p) > θ then
9              if |FP| = m then
10                 remove the point with the smallest frequency in FP;
11             insert p into FP and update θ to be the smallest frequency in FP;
12 return FP;
```

The main difference between SF_{APPROX} and SF_{SKYCUBE} is that SF_{APPROX} returns an approximate rank of skyline frequency within an error of ϵ and a confidence level of at least $1 - \delta$. To obtain the desired error bound, we select a sample of $T = 2nln(2/\delta)/\epsilon^2$ number of subspaces out of $2^d - 1$ subspaces randomly. The proof of the error bound follows from [13].

At the beginning of the algorithm, a preprocessing operation of the data is invoked for optimization (Line 1). This is based on the observation that the efficiency of the algorithm is dependent on the order in which the data points are processed. For sake of simplicity, we pre-sort the points in non-descending order of the sum of dimension values. Intuitively, a smaller sum is likely to have smaller values on more dimensions and is therefore likely to have a higher skyline frequency. Initialization of parameters are done in Line 2. To speed up the algorithm, we maintain a frequency threshold θ to avoid dominating tests of a point p if p is determined not to be among the $x\%$ frequent skylines. For

each sample (Line 3), we randomly choose a subspace $\mathcal{B}$ (Line 4 to 11), we compute the skyline of $\mathcal{B}$ and for each point p in the skyline of $\mathcal{B}$, we increase the frequency of p by 1. A point p is inserted into FP when the set FP has fewer than m points or the threshold is larger than θ. Meanwhile, we update θ to be the smallest frequency in FP. Specially, the point with smallest frequency is replaced by p if p has larger frequency and the size of FP is m. Finally, the set of candidate points FP is returned (Line 12).

The *SkylineWithThreshold* procedure computes the skyline points of D in subspace $\mathcal{B}$ where skyline frequency exceeds θ when the size of candidate points $|FP|$ equals to m. The main idea of computing skyline in subspace is to inspect all pairs of points and returns an object if it is not dominated by any other object (Line 4 to 10). Specially, if there are already m intermediate frequent skylines exceeds θ (Line 3), then the point clearly cannot be among the top $x\%$ frequent skylines. So it is unnecessary to do the dominating test. Therefore, the algorithm can be accelerated by only considering the points with potentiality to be frequent skyline points.

Algorithm 2. SkylineWithThreshold(D, $\mathcal{B}$, θ, m, r)

```
   Input: dataset D, subspace B, threshold θ, size m, size r.
   Output: SKY_B(D, θ)
1  initialize SKY_B(D, θ) with the first point in D, isSkyline = false;
2  for each point q ∈ D do
3      if r = m and fre(q) > θ then
4          for each point p ∈ SKY_B(D, θ) do
5              if q dominates p in subspace B then
6                  replace p with q;
7              else if p dominates q in subspace B then
8                  isSkyline = false; break;
9              else
10                 isSkyline = true; insert q to SKY_B(D, θ);
11 return SKY_B(D, θ);
```

5 Experiments

In this section, we show the frequent skyline points are efficient to select the k-representative points by comparing them with the skyline points and the points in the whole dataset, and taking them as input of the state-of-art k-regret algorithm Sphere [7]. All the experiments were implemented in C++ and the experiments were performed on a 3.3GHz CPU and 4G RAM machine running Ubuntu 14.04 LTS.

We ran our experiments on both synthetic and real datasets. For synthetic datasets, we use correlated and anti-correlated synthetic data, generated by the generator in [2]. Unless specially stated, the cardinality is ten thousand (*i.e.* $n = 10,000$), $k = 10$, $d = 6$, $\epsilon = 0.2$, $\delta = 0.05$, $x\% = 0.02$. The real-world datasets we adopted are three datasets, a 6-dimensional *Household* of 127,931 points

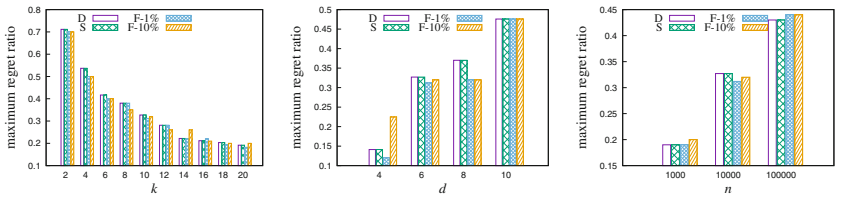
(www.ipums.org), a 9-dimensional *Color* of 68,040 points (kdd.ics.uci.edu), and an 8-dimensional *NBA* of 17,265 points (www.basketballreference.com), which are widely used in this area.

For the simplicity of describing, we first denote the three candidate sets, the whole dataset points as D, skyline points as S, frequent skyline points $F - x\%$, where $x\%$ means the candidate set is composed of the top $x\%$ frequent skyline points. Different values of x ($x = 1, 10$) have been chosen.

Then, we show the frequent skyline points can be used for the state-of-art algorithm Sphere [7] for k-regret query in terms of maximum regret ratio and running time. The effects of F 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 F is close to D and S or even smaller in some situation while the running time is only $\frac{1}{10}$ of D and S. This is because the candidate points F with high skyline frequency have more possibility to be the representative skyline points. From the above, we have the conclusion that frequent skyline points can efficiently speed up the original algorithm with a close maximum regret ratio.

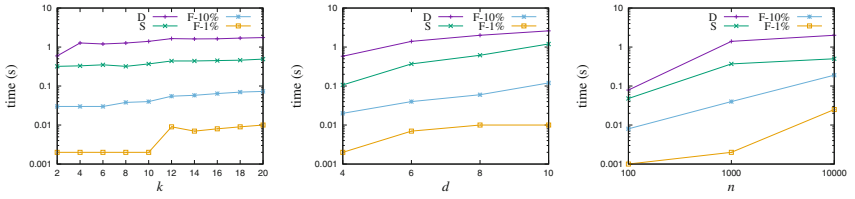
Besides, we evaluate the effect of different cardinalities and dimensionalities on our algorithm. Figure 1(b) shows that when d increases, the maximum regret ratio of different candidate sets increases. Figure 1(c) shows that the maximum regret ratio is independent of n. The same conclusion can be reached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

We can observe the similar trends on real datasets. Similar to the experiments on synthetic datasets, with frequent skyline points, the algorithm achieves a close



(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

maximum regret ratio with less running time compared with experiments on the whole dataset and the skyline points. The figures are omitted for the same reason.

6 Conclusions and Future Work

We studied the problem of reducing the size of candidate set for k-regret query basing on skyline frequency. Skyline frequency was adopted to rank skyline points and an algorithm, SF_{APPROX} was provided to determine the candidate set for k-regret query. Our experimental study demonstrated the effectiveness and efficiency of the proposed algorithm. Since our candidate set are the points of "the best - *higher skyline frequency* of the best - *skyline points of the whole dataset*", we do think there are certain theoretical results to be found to guarantee our validity. 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. Chester, S., Thomo, A., Venkatesh, S., Whitesides, S.: Computing k-regret minimizing sets. VLDB **7**, 389–400 (2014)
5. Asudeh, A., Nazi, A., Zhang, N., Das, G.: Efficient computation of regret-ratio minimizing set: a compact maxima representative. In: SIGMOD, pp. 821–834 (2017)
6. Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Efficient algorithms for k-regret minimizing sets. CoRR, abs/1702.01446 (2017)
7. 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)
8. 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
9. Qi, J., Zuo, F., Samet, H., Yao, J.C.: K-regret queries: from additive to multiplicative utilities (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. Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: On high dimensional skylines. In: Ioannidis, Y., et al. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 478–495. Springer, Heidelberg (2006). https://doi.org/10.1007/11687238_30
13. Karp, R.M., Luby, M., Madras, N.: Monte Carlo approximation algorithms for enumeration problems. J. Algorithms **10**(3), 429–448 (1989)