Speed-Up Algorithms for Happiness-Maximizing Representative Databases

Xianhong Qiu[1], Jiping Zheng[1,2(✉)], Qi Dong[1], and Xingnan Huang[1]

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

Abstract. Helping user identify the ideal results of a manageable size k from a database, such that each user's ideal results will take a big picture of the whole database. This problem has been studied extensively in recent years under various models, resulting in a large number of interesting consequences. In this paper, we introduce the concept of minimum happiness ratio maximization and show that our objective function exhibits the property of *monotonictity*. Based on this property, two efficient polynomial-time approximation algorithms called Lazy NWF-Greedy and Lazy Stochastic-Greedy are developed. Both of them are extended to exploit lazy evaluations, yielding significant speedups as to basic RDP-Greedy algorithm. Extensive experiments on both synthetic and real datasets show that our Lazy NWF-Greedy achieves the same minimum happiness ratio as the best-known RDP-Greedy algorithm but can greatly reduce the number of function evaluations and our Lazy Stochastic-Greedy sacrifices a little happiness ratio but significantly decreases the number of function evaluations.

Keywords: Minimum happiness ratio · Representative skyline
Lazy evaluation

1 Introduction

When users query the entire database trying to find an item that they are interested in, they may not be willing to search through all of the results storing a multitude of items. Given this, multiple operators have been proposed to reduce output size of query results while still effectively representing the entire database. Among these, top-k [1–4] and skyline [5–9] are two well-studied operators that can effectively reduce output size of query results. Top-k operator takes as input a dataset, a utility function and a user-specified value k, then outputs k data points with the highest utility scores. In other words, top-k returns a customized

L. H. U and H. Xie (Eds.): APWeb-WAIM 2018, LNCS 11268, pp. 321–335, 2018.
https://doi.org/10.1007/978-3-030-01298-4_27

set of data points to users. Unlike top-k operator, skyline returns a set of interesting data points without the need to appoint a utility function. The concept of domination is crucial in skyline operator. In fact, skyline returns data points that are not dominated by any other point in the database. Specifically, a point p dominates another point q if p is as good or better in all dimensions, and strictly better in at least one dimension. However, both operators suffer from some drawbacks. Top-k operator asks for users to specify their utility functions and the size of the result set, but users may not provide their utility functions precisely. Skyline operator finds all points that are not dominated by other points in the database, so the exact number of results is uncontrollable, and cannot be foreseen before the whole database is accessed. In addition, the output size of skyline operator will increase rapidly with the dimensionality.

Fortunately, Nanongkai *et al.* [10] first introduced k-regret query and developed the best-known algorithm based on the framework of Ramer–Douglas–Peucker algorithm. We called the Greedy algorithm proposed in [10] RDP-Greedy. Given an integer k and a database D, RDP-Greedy algorithm returns a set S of k skyline points of D that minimizes the maximum regret ratio of S. However, as k-regret query exploits linear utility function space to simulate all possible utility functions that users may have, it has a side effect of performing too many function evaluations through Linear Programming (LP). In general, the number of function evaluations of RDP-Greedy is nk, where n is the size of whole database. Besides, for RDP-Greedy, LP for function evaluations is the overwhelming majority of the running time. Thus, reducing the number of function evaluations by LP should speed up the algorithm to a great extent.

Table 1(a) shows the classic skyline example of best hotels for a sample set of hotels where each hotel has two attributes, namely *Distance* and *Price*. Suppose that user's utility functions are the class of $U = \{u_{(0.4,0.6)}, u_{(0.5,0.5)}, u_{(0.6,0.4)}\}$ where $u_{(x,y)} = x \cdot Distance + y \cdot Price$. The utilities of each hotel are shown in Table 1(b). Given $k = 3$, the k-regret query will report a solution $S = \{p_8, p_1, p_5\}$ under the aforementioned class of utility functions U. The implementation of k-regret query is as follows, it picks the point that maximizes the first coordinate and then iteratively adds the *worst* point, *i.e.*, the point that is still outside the current solution and contributes the most to the maximum regret ratio of current solution. Unfortunately, even in this simple example, k-regret query performs 13 times of function evaluations and each function evaluation results in $O(k^2d)$ running time by LP [11]. However, the maximum regret ratio obtained by adding a point to a larger set isn't greater than adding the same point to a smaller set. The process of RDP-Greedy algorithm doesn't take this property into consideration thus resulting in an unnecessary function evaluation to p_7 which occurs to the selection of the 3rd point. For further explanation, please refer to Example 2 in Sect. 5.1 for details.

As mentioned above, the deficiency of k-regret query is performing too many function evaluations. Motivated by this, in this paper, the concept of happiness ratio is first introduced. The utility provided by the best hotel in the result is user's happiness and the happiness ratio is by dividing happiness by the utility of her ideal hotel. The minimum happiness ratio is a measurement which

Table 1. Hotel example

(a) Hotel database

Hotel	Distance	Price
p_1	125	1000
p_2	250	800
p_3	312.5	750
p_4	375	650
p_5	562.5	625
p_6	625	450
p_7	750	250
p_8	1000	75

(b) Hotel utilities

Hotel	$u(0.4, 0.6)$	$u(0.5, 0.5)$	$u(0.6, 0.4)$
p_1	650	562.5	475
p_2	580	525	470
p_3	575	531.25	487.5
p_4	540	512.5	485
p_5	600	593.75	587.5
p_6	520	537.5	555
p_7	450	500	550
p_8	445	537.5	630

measures how happy the user will be after displaying k hotels instead of the whole database. Our purpose is to select k hotels that maximize the minimum happiness ratio of a user. Moreover, we demonstrate that our objective function for maximizing the minimum happiness ratio exhibits the property of *monotonicity*. Based on this property, two efficient algorithms by extending to exploit lazy evaluations, yielding significant speedups, called Lazy NWF-Greedy and Lazy Stochastic-Greedy are proposed. The former is an improvement of RDP-Greedy and the latter essentially follows the lazier idea of [12]. In our extensive experiments, Lazy NWF-Greedy achieves the same minimum happiness ratio as RDP-Greedy but can greatly reduce the number of function evaluations and Lazy Stochastic-Greedy sacrifices a little happiness ratio but significantly decreases the number of function evaluations.

The main contributions of this paper are listed as follows:

1. We propose the concept of minimum happiness ratio maximization[1] and show that our objective function for maximizing the minimum happiness ratio is a monotone non-decreasing function.
2. Based on the monotonicity of our objective function, we introduce two efficient greedy algorithms called Lazy NWF-Greedy and Lazy Stochastic-Greedy respectively. The former achieves the same minimum happiness ratio as RDP-Greedy but runs much faster, the latter sacrifices a little happiness ratio but offers a tradeoff between minimum happiness ratio and the number of function evaluations.
3. Extensive experiments on both synthetic and real datasets are conducted to evaluate our methods and the experimental results confirm that both of our proposed algorithms are superior to RDP-Greedy algorithm proposed by [10].

[1] Our minimum happiness ratio maximization is consistent with the k-regret proposed in [10]. However, k-regret denotes different things by Nanongkai *et al.* [10] and Chester *et al.* [13]. In the former, k-regret is the representative set of k objects, whereas in the latter, k-regret is used to denote the regret between the scores of top 1 and top k. To avoid confusion, we refer k-regret in [13] to $kRMS$.

The remainder of this paper is organized as follows. In Sect. 2, previous work related to this paper is described. The formal definitions of our problem are given in Sect. 3. In Sect. 4, important properties which are applied in our algorithms are discussed. Followed by two accelerated greedy algorithms in Sect. 5. The performance of our algorithms on synthetic and real datasets is presented in Sect. 6. Finally, Sect. 7 concludes this paper and points out our future work.

2 Related Work

Top-k [1–4] and skyline [5–9] operators have received considerable attentions during last two decades as they play an important role in multi-criteria decision making. However, top-k operator requires users to specify their utility functions and it may be too hard for users to specify their utility functions precisely while skyline operator has a potential large output problem which may make users feel overwhelmed. Motivated by the deficiencies of these two operators, a lot of alternatives have been proposed in recent years.

From top-k perspective, Mindolin *et al.* [8] asked users to specify a small number of possible weights each indicating the importance of a dimension. Lee *et al.* [4] asked users to specify some pair-wise comparisons between two dimensions to decide whether a dimension was more important than the other for a comparison. However, these studies ask users to specify their utility functions, which may be a heavy burden on users. In [7], skyline points were ranked according to the skyline frequency, which was a measure of how often points appear as skyline points in each particular subspace. The frequency ranking skyline query thus returned the k skyline points with the highest skyline frequency. However, this sort of quality measure is highly subjective and hard to verify.

Other works are in view of skyline operator. Researchers attempt to reduce the output size of the skyline operator. The k representative skyline proposed by Lin *et al.* [14] represented the whole skyline with only k skyline points which dominated the most non-skyline points in the database. Tao *et al.* [15] demonstrated the approach provided by [14] was not *stable*. Instead, they proposed distance-based representative skyline borrowing the idea of solving the k-center problem and provided a solution that the maximum distance from any skyline point to its nearest representative skyline was minimized. This method captures the contour of the full skyline well, but is not *scale-invariant*. Magnani *et al.* [16] introduced an approach, which was trying to make the diversity of the k representative skyline points returned as large as possible. A recent approach based on the diversity measure was proposed by Søholm *et al.* [17], which returned the k skyline points, such that the coverage was maximized. Also, [18] proposed a new criterion to choose k skyline points as the k representative skyline for data stream environments, termed the k largest dominance skyline. Unfortunately, all these methods are not stable, scale-invariant or with deficiencies of top-k or skyline operators.

To alleviate the burden of top-k for specifying accurate utility functions and skyline operator for outputting too many results, regret-based k representative

query was first proposed by Nanongkai *et al.* [10] to minimize user's maximum regret ratio. The stable, scale-invariant approach returned an approximation of the contour of the full skyline without asking users to input their utility functions. However, the approach proposed by them suffers from a heavy burden on the function evaluations by Linear Programming to seek the point with maximum regret ratio in linear utility space. Several works extended Nanongkai *et al.* [10] to some extent. Peng *et al.* [19] proposed the concept of *happy points* considered as candidate points for k-regret query and showed that *happy points* were better used as candidate points compared with skyline points due to their small size resulting in more efficient algorithms. However, the overall time complexity of finding all *happy points* is $O(d^2n^2)$ which is undesirable when n is very large. To reduce the bounds of regret ratio, [20] combined user's interactions into the process of selection. [13] introduced the relaxation to k-regret minimizing sets, [21] extended linear utility functions to non-linear utility functions for k-regret queries and [22] proposed the metric of *average regret ratio* to measure user's satisfaction. Recently, [23] developed a compact set to efficiently compute the k-regret minimizing set and [24] studied the $kRMS$, which returned r tuples from the database which minimized the k-Regratio. [25] and [26] developed efficient algorithms for $kRMS$. Rank-regret representative was proposed as a way of choosing a small subset of the database guaranteed to contain at least one good choice for every user in [27]. Xie *et al.* [28] proposed an elegant algorithm which has a restriction-free bound on the maximum regret ratio. Our research is from another respective: lazy evaluations by reducing LP calls thus improving the efficiency.

3 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 users prefer to the smaller values. Before we define our problem, some definitions of utility function, happiness ratio and minimum happiness ratio are given.

Definition 1 (Utility Function). *A utility function u is a mapping u: $\mathbb{R}_+^d \rightarrow \mathbb{R}_+$. 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 (Happiness Ratio). *Given a dataset D, a set of S with points in D and a utility function u. The happiness ratio of S, represented as $H_D(S, u)$, is defined to be*

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

The happiness ratio is in the range (0, 1]. According to the definition of happiness ratio, the larger the value of happiness ratio is, the happier the user feels.

Since we do not ask users for utility functions, we know nothing about users' preferences for the attributes. For each user, she may have arbitrary utility function. In this paper, we assume that user's utility functions are a class of functions, denoted by U. Usually, four kinds of utility functions [21] are considered, which are Linear, Convex, Concave and CES. In this paper, only the case of function class U consisting of all liner utility functions is considered since it is widely used in modeling user's preferences.

Definition 3 (Linear Utility Function [10]). Assume that existing some non-negative reals v_1, v_2, $\cdots$, v_d which denote the user's preferences for the i-th dimension, then a linear utility function can be represented as $u(p) = \sum_{i=1}^{d} v_i \cdot p[i]$ for any d-dimensional point with a liner utility function u. We can also say that a linear utility function can be expressed as a weight vector, *i.e.* $v = (v_1, v_2, ..., v_d)$, so the utility of any point p can be expressed as the dot product of v and p, namely, $u(p) = v \cdot p$.

The formal definition of minimum happiness ratio of a set S is as follows.

Definition 4 (Minimum Happiness Ratio). *Given a dataset D, a set of S with points in D and a class of utility functions U. The minimum happiness ratio of S, represented as $H_D(S, U)$, is defined to be*

$$H_D(S, U) = \inf_{u \in U} H_D(S, u) = \inf_{u \in U} \frac{\max_{p \in S} u(p)}{\max_{p \in D} u(p)}$$

Since U is allowed to be an infinite class of functions and the minimum value may not exist, so we use $\inf(S, U)$ to represent the minimum happiness ratio.

Example 1. In the following, we present an example for the illustration. Consider the hotel database containing 8 hotels as shown in Table 1(a) in the previous section. The utility function class U is $\{u_{(0.4,0.6)}, u_{(0.5,0.5)}, u_{(0.6,0.4)}\}$ where $u_{(x,y)} = x \cdot Distance + y \cdot Price$. The utilities of hotels for the utility functions in U are shown in Table 1(b). Consider p_1 in Table 1(b). Its utility for the utility function $u_{(0.4,0.6)}$ is $u_{(0.4,0.6)} = 125 \times 0.4 + 1000 \times 0.6 = 650$. The utilities of the remaining points are computed in a similar way. Consider a selection set $S = \{p_3, p_4\}$. Then, the maximum utility of S for the utility function $u_{(0.4,0.6)}$ is 575 which is achieved by p_3 while the maximum utility of whole database is 650 which is achieved by p_1. Then the happiness ratio of S for the utility function $u_{(0.4,0.6)}$ is $H_D(S, u_{(0.4,0.6)}) = 575/650 = 0.8846$. Similarly, we can get $H_D(S, u_{(0.5,0.5)}) = 531.25/593.75 = 0.8947$ and $H_D(S, u_{(0.6,0.4)}) = 487.5/630 = 0.7738$. Hence, the minimum happiness ratio of S is 0.7738.

Problem Definition: Given a dataset D, a positive integer k, our problem of minimum happiness ratio maximization is trying to find a subset S of D containing at most k points such that the minimum happiness ratio is maximized while simultaneously keeping the number of Linear Programming as small as possible.

Our aim is to maximize user's minimum happiness ratio while reducing the times of Linear Programming to return k representatives efficiently compared with RDP-Greedy in [10].

4 Properties of The Objective Function

The minimum happiness ratio function $H_D(S, U)$ has an intuitive and important property which can be exploited to improve the efficiency while solving our problem. First, when $S = \emptyset$, $H_D(S, U) = 0$, this means that we do not obtain any happiness if we do not select any point. Secondly, $H_D(S, U)$ is a non-decreasing function, *i.e.*, $H_D(S_1, U) \leq H_D(S_2, U)$ for all $S_1 \subseteq S_2 \subseteq D$. Hence, adding some points into a subset can increase the happiness ratio or at least keep it unchanged (not decreasing the minimum happiness ratio).

Definition 5 (Monotonicity). *A set function $f : 2^D \rightarrow \mathbb{R}_+$ is monotone if for every $S_1 \subseteq S_2 \subseteq D$, it holds that $f(S_1) \leq f(S_2)$. In addition, $f(S)$ is non-negative if $f(S) \geq 0$ for any set S.*

A set function satisfies the above properties, then we say it is a monotone non-decreasing function.

Lemma 1. *Our minimum happiness ratio maximization function is a monotone non-decreasing function, namely, it satisfies the property of monotonicity.*

The proof of monotonicity that our minimum happiness ratio maximization function meets is given below.

Proof. Suppose that there exist two arbitrary non-empty subsets S_1, S_2 where $S_1 \subseteq S_2 \subseteq D$. According to Definition 4, we have

$$H_D(S_1, U) = \inf_{u \in U} \frac{\max_{p_1 \in S_1} u(p_1)}{\max_{p_1 \in D} u(p_1)}$$

$$H_D(S_2, U) = \inf_{u \in U} \frac{\max_{p_2 \in S_2} u(p_2)}{\max_{p_2 \in D} u(p_2)}$$

Since $S_1 \subseteq S_2$, it's obvious that the minimum happiness ratio of S_1 is not greater than that of S_2, *i.e.*, $H_D(S_1, U) \leq H_D(S_2, U)$. □

The calculation of the maximum regret ratio based on the method introduced in [10] has to run LP at most nk times. In practice, an LP solver (such as variations of the Simplex method) will result in $O(k^2 d)$ running time per LP call and hence $O(nk^3 d)$ for the RDP-Greedy. This, however, can be improved using the property of monotonicity of our objective function. Detailed description is presented in the following section.

5 Proposed Algorithms

We are ready to present our algorithms, Lazy NWF-Greedy and Lazy Stochastic-Greedy, both of them are boosted in performance compared with RDP-Greedy.

5.1 Lazy NWF-Greedy Algorithm

The RDP-Greedy introduced in [10] picks the point that maximizes the first coordinate and adds the point that currently contributes the most to the maximum regret ratio in the subsequent iterations. Unfortunately, evaluating the minimum happiness ratio or the maximum regret ratio is time-consuming, in this section, we introduce Lazy NWF-Greedy which is an improvement of the RDP-Greedy, providing the same greedy solution since they actually share a same selection strategy. But Lazy NWF-Greedy is with fewer function evaluations of the minimum happiness ratio compared with RDP-Greedy as some lazy strategies are exploited.

Algorithm 1. Lazy-Evaluation$(D, S_{i-1}, i, \{\rho(p_j)\})$

Input: A set of n d-dimensional points $D = \{p_1, p_2, \cdots, p_n\}$, a current solution S_{i-1}, an integer i(the current number of points to find), and a list stored the minimum happiness ratio of each point $p_j \in D \backslash S_{i-1}$.
Output: A point p^*, where $\rho(p^*)$ is minimized.

let $h = 1$ and $p^* = NULL$;
$\rho(p_{j_0}) = \min_{p_j \in D \backslash S_{i-1}} \{\rho(p_j)\}$;
if p_{j_0} *has already been selected at step* i **then**
  $h = \rho(p_{j_0})$;
  if $h < 1$ **then**
    $p^* = p_{j_0}$;
  return p^*;
else
  calculate the value of $H_{S_{i-1} \cup \{p_{j_0}\}}(S_{i-1}, U)$ using Linear Programming;
  $h = H_{S_{i-1} \cup \{p_{j_0}\}}(S_{i-1}, U)$;
  $\rho(p_{j_0}) = h$;
  if $h > \min_{p_j \in D \backslash S_{i-1}, p_j \neq p_{j_0}} \{\rho(p_j)\}$ **then**
    Lazy-Evaluation$(D, S_{i-1}, i, \{\rho(p_j)\})$;
  else
    if $h < 1$ **then**
      $p^* = p_{j_0}$;
    return p^*;

Speeding up Lazy NWF-Greedy with lazy evaluations. Since we have no idea of user's utility functions, the number of utility functions is infinite in the linear utility space. In order to estimate the maximum regret ratio or the minimum happiness ratio of a subset when we add a point into, a large number of function evaluations by Linear Programming need to be performed when we run RDP-Greedy algorithm. Fortunately, monotonicity of our minimum happiness ratio maximization function can be exploited algorithmically to implement an accelerated variant of RDP-Greedy to reduce the number of function evaluations. In each iteration i, RDP-Greedy must identify the point p whose $H_{S_{i-1} \cup \{p\}}(S_{i-1}, U)$ is minimized(equivalent to maximizing the maximum regret ratio), where S_{i-1} is the set of points selected in the previous iterations. The key insight from the monotonicity of H, the minimum happiness ratio obtained by

any fixed point $p \in D$ is monotonically non-decreasing during the iterations of adding points, *i.e.*,$H_D(S_i \cup \{p\}, U) \leq H_D(S_j \cup \{p\}, U)$, whenever $i \leq j$. Instead of recomputing for each point $p \in D$, we can use lazy evaluations to maintain a list of lower bounds $\{\rho(p)\}$ on the minimum happiness ratio sorted in ascending order. Then in each iteration, the accelerated algorithm needs to extract the minimal point $p \in \arg\min_{p':S_{i-1}\cup\{p\}}\{\rho(p')\}$ from the ordered list and then updates the bound $\rho(p) \leftarrow H_{S_{i-1}\cup\{p\}}(S_{i-1}, U)$. After this update, if $\rho(p) \leq \rho(p')$, then $H_{S_{i-1}\cup\{p\}}(S_{i-1}, U) \leq H_{S_{i-1}\cup\{p'\}}(S_{i-1}, U)$ for all $p \neq p'$, and therefore we have identified the point that contributes the least to the minimum happiness ratio, without having to compute $H_{S_{i-1}\cup\{p'\}}(S_{i-1}, U)$ for a potentially large number of point p'. We set $S_i \leftarrow S_{i-1} \cup \{p\}$ and repeat until there is no further feasible point which can be added. This idea of using lazy evaluations is useful to our algorithm and can lead to orders of magnitude performance speedups. The pseudocodes of lazy evaluation and Lazy NWF-Greedy are shown in Algorithms 1 and 2 respectively. Example 2 is given to show how to combine the procedure of calculating minimum happiness ratio with lazy evaluations to boost the performance of our Lazy NWF-Greedy.

Algorithm 2. Lazy NWF-Greedy(D, k)

Input: A set of n d-dimensional points $D = \{p_1, p_2, \cdots, p_n\}$ and an integer k, which is the desired output size.
Output: A result set S, $|S| = k$.

```
1  Initially, let S_1 = {p_1^*}, where p_1^* = arg max_{p∈D} p[1];
2  for (i = 2; i ≤ k; i++) do
3      let p^* = NULL;
4      if i = 2 then
5          for each p_j ∈ D\S_{i-1} do
6              calculate the value of H_{S_{i-1}∪{p_j}}(S_{i-1}, U) using Linear Programming;
7              ρ(p_j) = H_{S_{i-1}∪{p_j}}(S_{i-1}, U);
8      p^* =Lazy-Evaluation(D, S_{i-1}, i, {ρ(p_j)});
9      if p^* = NULL then
10         return S_{i-1};
11     else
12         S_i = S_{i-1} ∪ {p^*};
13 return S_k;
```

Example 2. Consider the example in Table 1, we want to select 3 points among 8 points and first pick p_8 into the current solution S_1, namely $S_1 = \{p_8\}$. In next iteration, we have a list on the minimum happiness ratio sorted in ascending order as shown in Table 2. The second point Lazy NWF-Greedy selects is p_1 as $H_{S_1\cup\{p_1\}}(S_1, U)$ is the minimum, then $H_{S_1\cup\{p_1\}}(S_1, U)$ will remove from the list and $S_2 = \{p_8, p_1\}$. After this operation, the algorithm begins to select the third point. As $H_{S_1\cup\{p_5\}}(S_1, U)$ is the minimum in the current list, so Lazy NWF-Greedy computes the minimum happiness ratio p_5 will obtain if it is added to S_2 and gets that $H_{S_2\cup\{p_5\}}(S_2, U)$ is 0.947 which is not the minimum in the list as

$H_{S_1\cup\{p_2\}}(S_1, U)$ to $H_{S_1\cup\{p_6\}}(S_1, U)$ are all less than $H_{S_2\cup\{p_5\}}(S_2, U)$, hence the minimum happiness ratio obtained by adding them to S_2 should be calculated. In our example, they are all equal to 1. At this moment, this algorithm can directly get the smallest minimum happiness ratio is 0.947 which is achieved by p_5 with no need to compute $H_{S_2\cup\{p_7\}}(S_2, U)$ since $H_{S_2\cup\{p_7\}}(S_2, U) \geq H_{S_1\cup\{p_7\}}(S_1, U) = 0.989$. In Example 2, Lazy NWF-Greedy reduces the function evaluations for 1 time. Obviously, in the settings of large datasets, our Lazy NWF-Greedy can reduce overall times of function evaluations dramatically compared with RDP-Greedy.

Table 2. $H_{S_i\cup\{p\}}(S_i, U)$ for iteration i

Points	p_1	p_5	p_2	p_3	p_4	p_6	p_7
$H_{S_1\{p\}}(S_1, U)$	0.685	0.742	0.767	0.774	0.824	0.856	0.989
$H_{S_2\{p\}}(S_2, U)$	×	0.947	1	1	1	1	×

5.2 Lazy Stochastic-Greedy Algorithm

As described in Sect. 5.1, Lazy NWF-Greedy identifies an ideal point from $D\backslash S_{i-1}$ which is almost the whole dataset. When k points need to be picked out, the algorithm has to go through the whole dataset for k times. In this section, we show how the performance of Lazy NWF-Greedy can be boosted by a random sampling phase thus leading to a randomized greedy algorithm called Lazy Stochastic-Greedy. Lazy Stochastic-Greedy essentially follows the framework of STOCHASTIC-GREEDY algorithm proposed in [12]. The algorithm offers a tradeoff between minimum happiness ratio and the number of function evaluations. It means that this algorithm sacrifices a little happiness ratio while reducing the number of LPs to improve its efficiency.

We present our Lazy Stochastic-Greedy algorithm in Algorithm 3. Initially, similar to our Lazy NWF-Greedy algorithm, the algorithm starts with a set containing the point that maximizes the first coordinate, then adds a point to our solution whose minimum happiness ratio is minimized. Note that the difference between Lazy Stochastic-Greedy and Lazy NWF-Greedy is that Lazy NWF-Greedy finds a point from $p \in D\backslash S_{i-1}$ directly, but Stochastic-Greedy samples a subset R of size $(n/k)log(1/\epsilon)(\epsilon > 0$ is an arbitrarily small constant) from $D\backslash S_{i-1}$ randomly and then finds the point in R whose minimum happiness ratio is minimized. We can combine the the random sampling procedure with lazy evaluations to speed up the implementation of the algorithm as the randomly sampled sets can overlap and we can exploit the previously evaluated minimum happiness ratio. Hence in line 9 of Algorithm 3 we can apply lazy evaluations as described in Sect. 5.1.

Lazy Stochastic-Greedy has two important features. The first feature is that it is almost identical to Lazy NWF-Greedy in terms of minimum happiness ratio.

Algorithm 3. Lazy Stochastic-Greedy(D, k)

Input: A set of n d-dimensional points $D = \{p_1, p_2, \cdots, p_n\}$ and an integer k, which is the desired output size.
Output: A result set S, $|S| = k$.

```
1  Initially, let's S_1 = {p_1^*}, where p_1^* = arg max_{p∈D} p[1];
2  for (i = 2; i ≤ k; i++) do
3      let p^* = NULL;
4      obtain a random subset R by sampling s random points from D\S_{i-1};
5      for each p_j ∈ R do
6          if p_j has not been sampled then
7              calculate the value of H_{S_{i-1}∪{p_j}}(S_{i-1}, U) using Linear Programming;
8              ρ(p_j) = H_{S_{i-1}∪{p_j}}(S_{i-1}, U);
9      p^* =Lazy-Evaluation(R, S_{i-1}, i, {ρ(p_j)});
10     if p^* = NULL then
11         return S_{i-1};
12     else
13         S_i = S_{i-1} ∪ {p^*};
14 return S_k;
```

The second feature is that it is more efficient than Lazy NWF-Greedy, let alone the best-known algorithm, RDP-Greedy.

Theorem 1. *Suppose that the size of random sampling set R is $s = (n/k)log(1/\epsilon)$, then* Lazy Stochastic-Greedy *provides a greedy solution to our minimum happiness ratio maximization problem with at most $O(nlog(1/\epsilon))$ function evaluations of the minimum happiness ratio H.*

Since there are $k - 1$ iterations in total and at each iteration we have $(n/k)log(1/\epsilon)$ points, the total number of function evaluations cannot be more than $(k-1) \times (n/k)log(1/\epsilon) \leq k \times (n/k)log(1/\epsilon) = nlog(1/\epsilon)$.

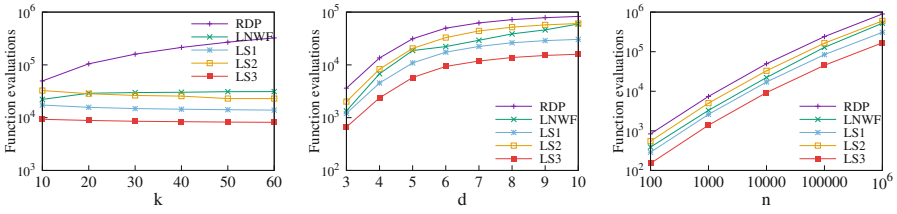
6 Experimental Results

In this section we show the performance of the proposed algorithms via experiments. All algorithms were implemented in C++ and the experiments were all conducted 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. The synthetic datasets were created using the dataset generator of [5]. Unless otherwise stated, our synthetic dataset consists of a 6-dimensional anti-correlated dataset of 10,000 points. The real-world dataset we have used are a 6-dimensional *Household* of 127,391 points, an 8-dimensional *NBA* of 17,265 points and a 9-dimensional *Color* of 68,040 points. Moreover, like studies in the literature [10,19–21], we computed the skyline first and our queries on these datasets returned anywhere from 10 to 60 points and evaluated the minimum happiness ratio using Linear Programming implemented in the GUN Linear Programming Kit[2].

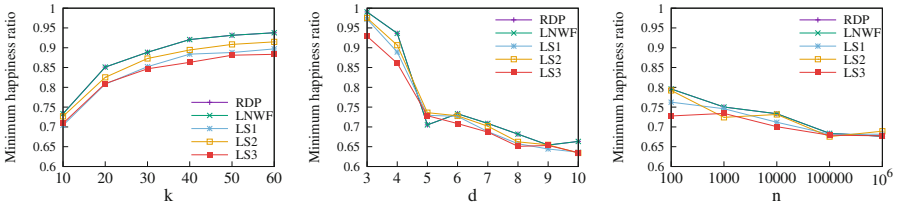
[2] https://www.gnu.org/software/glpk/.

In our experiments, we consider both Lazy NWF-Greedy and Lazy Stochastic-Greedy algorithms introduced in this paper. To verify the superiority of our proposed algorithms, we compared them with RDP-Greedy [10]. Due to the number of function evaluations by Linear Programming accounting for the majority of the total time of RDP-Greedy, we measure the computational cost in terms of the number of function evaluations by Linear Programming performed instead of the running time of CPU. In addition, for Lazy Stochastic-Greedy, different values of $\epsilon(\epsilon = 0.01, 0.1, 0.3)$ have been chosen. In the following experimental result figures, we abbreviate RDP-Greedy as RDP, Lazy NWF-Greedy as LNWF, Lazy Stochastic-Greedy as LS1 when $\epsilon = 0.1$, Lazy Stochastic-Greedy as LS2 when $\epsilon = 0.01$, Lazy Stochastic-Greedy as LS3 when $\epsilon = 0.3$.



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

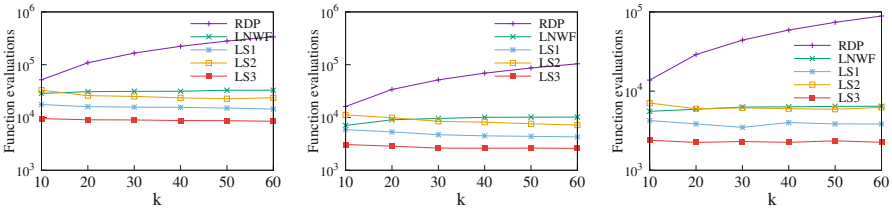
Fig. 1. Function evaluations for anti-correlated data



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

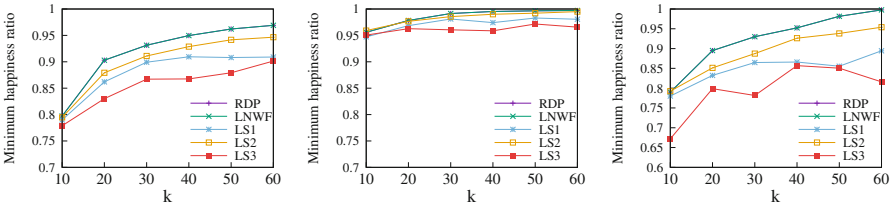
Fig. 2. Minimum happiness ratio for anti-correlated data

Results on Synthetic Datasets: The effects on function evaluations and minimum happiness ratio on anti-correlated datasets for different k are presented in log scale in Figs. 1(a) and 2(a) respectively. Lazy NWF-Greedy and Lazy Stochastic-Greedy have negligible number of function evaluations which do not increase with k and keep a relative small stable number of evaluations even for very large inputs. However, RDP-Greedy performs linear function evaluations as k increases, that's because it performs all possible evaluations of user's utilities, namely, for a total of nk times. The minimum happiness ratio of Lazy NWF-Greedy is the same as that of RDP-Greedy as they share the same greedy



(a) Household (b) NBA (c) Color

Fig. 3. Function evaluations on real datasets



(a) Household (b) NBA (c) Color

Fig. 4. Minimum happiness ratio on real datasets

skeleton and increases when k increases, appearing to much above the theoretical bound proposed by [10]. For Lazy Stochastic-Greedy, different values of ϵ result in a performance close to that of Lazy NWF-Greedy. It means that Lazy Stochastic-Greedy provides very compelling tradeoffs between the number of function evaluations and minimum happiness ratio compared with RDP-Greedy and our Lazy NWF-Greedy.

We varied the number of dimensions on anti-correlated data when the number of points was fixed to $n = 10,000$ and $k = 10$. As illustrated in Fig. 1(b) and in Fig. 2(b), the number of function evaluations increases with the increase of dimensions, but our proposed algorithms still have less function evaluations than RDP-Greedy since they are extended with lazy evaluations. Due to the curse of dimensionality, the minimum happiness ratio resulted in by all algorithms decreases with the increase of d. In Figs. 1(c) and 2(c), the effects of varying n are presented. The effects are similar to those of varying dimensions.

Results on Real Datasets: Figs. 3(a), (b) and (c) show that the number of function evaluations of RDP-Greedy increases dramatically with k, but our proposed algorithms keep much less function evaluations and maintain a stable level. The minimum happiness ratio of all algorithms are shown in Figs. 4(a), (b) and (c) respectively. We observe similar trends. Besides, similar to the experiments on synthetic datasets, Lazy Stochastic-Greedy achieves near-maximal minimum happiness ratio with substantially less function evaluations compared with the other algorithms.

7 Conclusions

In this paper, we introduce minimum happiness ratio and propose two algorithms called Lazy NWF-Greedy and Lazy Stochastic-Greedy to speed up RDP-Greedy, both of them are extended from basic greedy algorithms by exploiting lazy evaluations. Experiments on real and synthetic datasets verify that our Lazy NWF-Greedy achieves the same minimum happiness ratio as the best-known RDP-Greedy algorithm, but can lead to orders of magnitude speedups and our Lazy Stochastic-Greedy sacrifices a little happiness ratio but significantly decreases the number of function evaluations compared with RDP-Greedy or Lazy NWF-Greedy. Our future work considers CONVEX, CONCAVE and CES utility functions with the framework of our algorithms.

Acknowledgment. This work is partially supported by the National Natural Science Foundation of China under grant Nos. U1733112,61702260, the Natural Science Foundation of Jiangsu Province of China under grant No. BK20140826, the Fundamental Research Funds for the Central Universities under grant No. NS2015095, Funding of Graduate Innovation Center in NUAA under grant No. 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. Lian, X., Chen, L.: Top-k dominating queries in uncertain databases. In: EDBT, pp. 660–671 (2009)
3. Soliman, M.A., Ilyas, I.F., Chang, K.C.-C.: Top-k query processing in uncertain databases. In: ICDE, pp. 896–905 (2007)
4. Lee, J., You, G.W., Hwang, S.W.: Personalized top-k skyline queries in high-dimensional space. Inf. Syst. **34**(1), 45–61 (2009)
5. Börzsöny, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
6. Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: SIGMOD, pp. 503–514. ACM (2006)
7. 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
8. Mindolin, D., Chomicki, J.: Discovering relative importance of skyline attributes. In: VLDB, pp. 610–621 (2009)
9. Papadias, D., Tao, Y., Greg, F., Seeger, B.: Progressive skyline computation in database systems. TODS **30**(1), 41–82 (2005)
10. Nanongkai, D., Sarma, A.D., Lall, A., Lipton, R.J., Xu, J.: Regret-minimizing representative databases. In: VLDB, pp. 1114–1124 (2010)
11. Bertsimas, D., Tsitsiklis, J.: Introduction to Linear Optimization. Athena Scientific, Belmont (1997)
12. Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrak, J., Krause, A.: Lazier than lazy greedy. In AAAI, pp. 1812–1818 (2015)

13. Chester, S., Thomo, A., Venkatesh, S., Whitesides, S.: Computing k-regret minimizing sets. In: VLDB, pp. 389–400 (2014)
14. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representative skyline operator. In: ICDE, pp. 86–95 (2007)
15. Tao, Y., Ding, L., Lin, X., Pei, J.: Distance-based representative skyline. In: ICDE, pp. 892–903, 2009
16. Magnani, M., Assent, I., Mortensen, M.L.: Taking the big picture: representative skylines based on significance and diversity. VLDB J. **23**(5), 795–815 (2014). October
17. Søholm, M., Chester, S., Assent, I.: Maximum coverage representative skyline. In: EDBT, pp. 702–703 (2016)
18. Bai, M., et al.: Discovering the k representative skyline over a sliding window. TKDE **28**(8), 2041–2056 (2016)
19. Peng, P., Wong, R.C.W.: Geometry approach for k-regret query. In: ICDE, pp. 772–783 (2014)
20. Nanongkai, D., Lall, A., Sarma, A.D., Makino, K.: Interactive regret minimization. In: SIGMOD, pp. 109–120 (2012)
21. Faulkner, T.K., Brackenbury, W., Lall, A.: k-regret queries with nonlinear utilities. In: VLDB, pp. 2098–2109 (2015)
22. Zeighami, S., Wong, R.C-W.: Minimizing average regret ratio in database. In: SIGMOD, pp. 2265–2266 (2016)
23. 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)
24. Cao, W. et al.: k-regret minimizing set: efficient algorithms and hardness. In: ICDT, pp. 11:1–19 (2017)
25. Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Efficient algorithms for k-regret minimizing sets. In: International Symposium on Experimental Algorithms, pp. 7:1–7:23 (2017)
26. Kumar, N., Sintos, S.: Faster approximation algorithm for the k-regret minimizing set and related problems. In: Proceedings of the 20th Workshop on Algorithm Engineering and Experiments, pp. 62–74 (2018)
27. Asudeh, A., Nazi, A., Zhang, N., Das, G., Jagadish, H.V.: RRR: rank-regret representative. CoRR (2018)
28. 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, pp. 959–974 (2018)