Supplementary Material for "Accelerated Algorithms for Maximizing Average Happiness Ratio in Databases"

Jiping ZHENG[1,2], Qi DONG[1], Xianhong QIU[1], Xingnan HUANG[1]
[1]Nanjing University of Aeronautics and Astronautics, Nanjing, China
[2]Nanjing University, Nanjing, China

I. Introduction

The following contents are provided in this supplementary material:

- Preliminaries about the sample size N of utility functions and properties of the k average happiness ratio function.
- The pseudo-code and examples of our proposed Lazy NWF-Greedy and Lazy Stochastic-Greedy algorithms as well as the description of heuristic strategy.
- Extensive experimental results on 1 synthetic and 4 real datasets.

II. Preliminaries

Table I summarizes the symbols frequently used throughout this supplementary material.

TABLE I
Summary of frequently used notations

Symbol	Description
D, S	dataset, skyline result set
U	a family of utility functions
$u(u \in U)$	utility function
$\eta(u)$	probability density function of u
k	number of representative skyline points
n	cardinality of a dataset
d	number of dimensions
N	number of utility functions sampled
α, β	utility sampling error, confidence level
ϵ	sampling error in stochastic algorithms
$hr_D(S,u)$	happiness ratio
$ahr_D(S,U)$	average happiness ratio

A. Sampling Utility Functions

We randomly sample N utility functions according to the probability distribution of the utility functions. Therefore, a suitable size of N must be chosen such that the calculated average happiness ratio maintains a high confidence level within an acceptable error under these utility functions.

Lemma 1: Given a confidence $\beta \in (0,1]$ and an error $\alpha \in (0,1]$, when the sample size N is at least $\frac{3\ln(\frac{1}{\beta})}{\alpha^2}$, it can guarantee a confidence of at least $1-\beta$ and the error between the estimated average happiness ratio and its true value is within α.

The proof of Lemma 1 is directly obtained by *Chernoff* inequality [9].

B. Properties of the k average happiness ratio function

We first revisit the definitions of average happiness ratio

$$ahr_D(S,U) = \int_{u\in U} hr_D(S,u)\eta(u)du$$

for continuous utility functions distribution, and

$$ahr_D(S,U) = \sum_{u\in U} hr_D(S,u)Pr(u)$$

for a discrete utility space.

The evaluation of the average happiness ratio $ahr_D(S,U)$ requires the evaluation of the happiness ratios of all utility functions on average, which depends on the distribution of the utility functions. Specifically, if the distribution of the utility functions is continuous, then the evaluation of the average happiness ratio needs to evaluate a d-dimensional integral as shown in the first definition of average happiness ratio. Otherwise, if the utility functions are sampled from the distribution, the average happiness ratio can be obtained by summing the products of the happiness ratios of utility functions and their probabilities for all the utility functions. In general, 5 kinds of utility functions are used to model users' preferences, they are linear [11], multiplicative [14], Convex [4], Concave [4] and CES [4] respectively. In this paper, we only consider linear utility functions as they are most popular in modeling the users' preferences [11], [10], [13], [2], [3], [6], [16], [15]. Specifically, a utility function u is linear if there exist some nonnegative reals v_1, v_2, $\cdots$, v_d, which indicates the user's preference for dimension $1, 2, \cdots, d$ respectively, then for a d-dimensional data point $p = (p[1], p[2], \cdots, p[d])$ the utility under u can be expressed by $u(p) = \sum_{i=1}^{d} v_i \cdot p[i]$. Equivalently, a linear utility function can be expressed by a

d-dimensional vector, *i.e.,* $u =< v_1, v_2, \cdots, v_d >$, then the utility $u(p)$ can be expressed as the dot product of the u and p, *i.e.,* $u(p) = u \cdot p$.

We present an example to illustrate the concepts above which is shown in Table II.

TABLE II
CAR EXAMPLE

(a) Car database

Car	MPG	HP
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) Car utilities

Car	$u_{(0.4,0.6)}$	$u_{(0.5,0.5)}$	$u_{(0.6,0.4)}$
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

Example 1: Consider the car database[1] containing 8 cars with two attributes MPG (Miles per Gallon) and HP (Horse Power) as shown in Table II(a). Suppose that the set U contains 3 utility functions, $U = \{u_{(0.4,0.6)}, u_{(0.5,0.5)}, u_{(0.6,0.4)}\}$ where $u_{(x,y)} = x \cdot MPG + y \cdot HP$ is defined on a discrete space and u satisfies a uniform distribution. Table II(b) shows the utilities of cars. Consider p_1 in Table II(b) whose utility for the utility function $u_{(0.4,0.6)}$ is $u_{(0.4,0.6)}(p_1) = 125 \times 0.4 + 1000 \times 0.6 = 650$. We can obtain the utilities of other points in a similar manner.

Assume that the set $S = \{p_2, p_3\}$ is presented to the user, then we can get the maximum utility of S under the utility function $u_{(0.4,0.6)}$ is 580 which is obtained by p_2, while the maximum utility of whole dataset is 650 which is achieved by p_1, thus the happiness ratio of S for the utility function $u_{(0.4,0.6)}$ is $hr_D(S, u_{(0.4,0.6)}) = 580/650 = 0.8923$. Similarly, we can get the happiness ratio of S for the utility function $u_{(0.5,0.5)}$ is $hr_D(S, u_{(0.5,0.5)}) = 562.5/593.75 = 0.9474$ and the happiness ratio of S for the utility function $u_{(0.6,0.4)}$ is $hr_D(S, u_{(0.6,0.4)}) = 487.5/630 = 0.7738$. Hence, the average happiness ratio of S is $ahr_D(S, U) = (0.8923 + 0.9474 + 0.7738)/3 = 0.8712$.

[1]In this paper, we use database and dataset interchangeably.

The average happiness ratio function $ahr_D(S, U)$ has several intuitive and important properties.

- When $S = \emptyset$, $ahr_D(S, U) = 0$, that is to say, if no representative points are recommended to the users, the users will not be satisfied at all.
- $ahr_D(S, U)$ is nondecreasing, *i.e.*, $ahr_D(S_1, U) \leq ahr_D(S_2, U)$ for all $S_1 \subseteq S_2 \subseteq D$. Therefore, adding some points to a subset does not lead to a decrease in the average happiness ratio.
- The last and most important property is if adding a point into a smaller subset S_1, the average happiness ratio increases at least as much as that of adding the same point into a larger subset S_2, where $S_1 \subseteq S_2$.

A set function with the aforementioned properties is called a *monotone nondecreasing submodular function*. Submodularity is a property of set functions with profound theoretical significance and far-reaching applications. It has been used widely in real applications [5] such as viral marketing, information gathering, image segmentation, document summarization and speeding up satisfiability solvers. Before we give the description that our average happiness ratio function satisfies above properties, definitions about *monotone submodular function* [5], [12] are given.

Definition 1 (Discrete derivative): For a set function $f : 2^D \to \mathbb{R}_+$, $S \subseteq D$, and a point $p \in D$, let $\triangle_f(p|S) := f(S \cup \{p\}) - f(S)$ be the discrete derivative of f at S with respect to p which is also regarded as the marginal gain. Usually, the function f is clear from the context. For convenience, we drop the subscript and simply rewrite it as $\triangle(p|S)$.

Definition 2 (Submodularity): A set function $f : 2^D \to \mathbb{R}_+$ is submodular if for every $S_1, S_2 \subseteq D$ it holds that

$$f(S_1) + f(S_2) \geq f(S_1 \cup S_2) + f(S_1 \cap S_2)$$

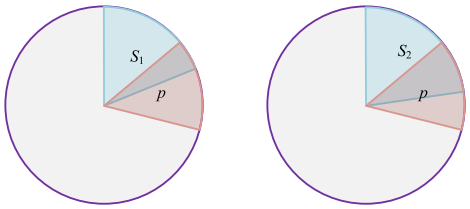
Equivalently, a set function $f : 2^D \to \mathbb{R}_+$ is submodular if for every $S_1 \subseteq S_2 \subseteq D$ and a point $p \in D \backslash S_2$ it holds that

$$f(S_1 \cup \{p\}) - f(S_1) \geq f(S_2 \cup \{p\}) - f(S_2)$$

Definition 3 (Monotonicity): A set function $f : 2^D \to \mathbb{R}_+$ is monotone nondecreasing if for every $S_1 \subseteq S_2 \subseteq D$, it holds that $f(S_1) \leq f(S_2)$.

For submodular maximization, the intuition provided by the second formula in Definition 2 is often helpful which exhibits a natural diminishing returns property. Figure 1 illustrates this effect in our average happiness ratio maximization problem. The blue parts indicate the average happiness ratio of the set S_1 and S_2, the red parts represent the average happiness ratio of the point p. From Figure 1, we can see that the marginal

gain of p added to set S_1 is no smaller than that of p added to set S_2.



(a) Adding p to S_1 (b) Adding p to S_2

Fig. 1. Illustration of the diminishing returns in context of the average happiness ratio maximizing

Lemma 2: The average happiness ratio function is a monotone nondecreasing submodular set function. Namely, it satisfies the properties of submodularity and monotonicity.

Proof sketch. First, we prove that our average happiness ratio function satisfies the property of submodularity. Suppose that there exist two arbitrary subsets S_1, S_2 where $S_1 \subseteq S_2 \subseteq D$ and a point $p \in D \backslash S_2$. According to Definition 1, we have

$$\begin{aligned}
&\triangle(p|S_1) = ahr_D(S_1 \cup \{p\}, U) - ahr_D(S_1, U) \\
&= \int_{u \in U} \frac{\max_{p' \in S_1 \cup \{p\}} u(p') - \max_{p' \in S_1} u(p')}{\max_{p' \in D} u(p')} \eta(u) du \\
&\triangle(p|S_2) = ahr_D(S_2 \cup \{p\}, U) - ahr_D(S_2, U) \\
&= \int_{u \in U} \frac{\max_{p' \in S_2 \cup \{p\}} u(p') - \max_{p' \in S_2} u(p')}{\max_{p' \in D} u(p')} \eta(u) du
\end{aligned}$$

Since $S_1 \subseteq S_2$, the average happiness ratio of S_1 is no larger than that of S_2, *i.e.*, $ahr_D(S_1, U) \leq ahr_D(S_2, U)$.

For the sake of convenience, we denote the maximum utility in $S_2 \cup \{p\}$ as $MaxU$ and then discuss it in three cases.

- Case 1: Suppose that $MaxU$ is achieved by the point in S_1, then $\triangle(p|S_1) = \triangle(p|S_2) = 0$.
- Case 2: Suppose that $MaxU$ is achieved by the point in $S_2 \backslash S_1$, then we consider the maximum utilities of S_1 and the utility of p respectively. When $\max_{p' \in S_1} u(p') \leq \max u(p)$, $\triangle(p|S_1) \geq 0$, $\triangle(p|S_2) = 0$, so $\triangle(p|S_1) \geq \triangle(p|S_2)$; otherwise, $\triangle(p|S_1) = \triangle(p|S_2) = 0$. Thus, in both situations, $\triangle(p|S_1) \geq \triangle(p|S_2)$.
- Case 3: Suppose that $MaxU$ is achieved by the point p, then we can also get $\triangle(p|S_1) \geq \triangle(p|S_2)$.

From all of above cases, $\triangle(p|S_1) \geq \triangle(p|S_2)$, and by the second form of submodularity definition in Definition 2, our average happiness ratio function is submodular.

The proof of monotonicity is as follows.

According to the proof of submodularity, it's easy to get $ahr_D(S_1, U) \leq ahr_D(S_2, U)$ for every $S_1 \subseteq S_2 \subseteq D$. Thus monotonicity is proved.

III. Lazy NWF-Greedy Algorithm

Lazy NWF-Greedy is an accelerated variant of the existing Greedy algorithm [7]. Compared to WO-Greedy [17] which deletes points from the whole dataset until there are only k points left as the result, the Greedy algorithm [7] adds points to an empty dataset until k data points are selected. In this section, we first introduce Lazy evaluations which are used in our algorithm to boost its performance, then present our Lazy NWF-Greedy algorithm along with an illustrated example.

We first provide an important property of function $ahr_{S \cup \{p\}}(S, U)$ by Lemma 3.

Lemma 3: Let S be the current solution and $p \in D \backslash S$, the function $ahr_{S \cup \{p\}}(S, U)$ is a monotone nondecreasing function of the set S.

Proof sketch. Given $ahr_{S \cup \{p\}}(S, U)$ and $ahr_{S \cup \{p, p'\}}(S \cup \{p'\}, U)$ where $p' \in D \backslash (S \cup \{p\})$, we only need to prove $ahr_{S \cup \{p, p'\}}(S \cup \{p'\}, U) - ahr_{S \cup \{p\}}(S, U) \geq 0$. By the definition of average happiness ratio function, we get

$$\begin{aligned}
&ahr_{S \cup \{p', p\}}(S \cup \{p'\}, U) - ahr_{S \cup \{p\}}(S, U) \\
&= \sum_{u \in U} \frac{\max_{q \in S \cup \{p'\}} u(q)}{\max_{q \in S \cup \{p', p\}} u(q)} Pr(u) - \sum_{u \in U} \frac{\max_{q \in S} u(q)}{\max_{q \in S \cup \{p\}} u(q)} Pr(u) \\
&= \sum_{u \in U} Pr(u) \left(\frac{\max_{q \in S \cup \{p'\}} u(q)}{\max_{q \in S \cup \{p, p'\}} u(q)} - \frac{\max_{q \in S} u(q)}{\max_{q \in S \cup \{p\}} u(q)} \right)
\end{aligned}$$

Then similar to the proof of Lemma 2, we discuss three cases for the maximum utility achieved in $S \cup \{p, p'\}$. We can see that in all the three cases, $\frac{\max_{q \in S \cup \{p'\}} u(q)}{\max_{q \in S \cup \{p, p'\}} u(q)} - \frac{\max_{q \in S} u(q)}{\max_{q \in S \cup \{p\}} u(q)} \geq 0$. Then Lemma 3 is proved.

Then we compute $ahr_{S_{i-1} \cup \{p\}}(S_{i-1}, U)$ for each point p and keep the point with the *minimum* value which is based on the framework of the Greedy algorithm in [11]. By Lemma 3, we have $ahr_{S_{i-1} \cup \{p\}}(S_{i-1}, U) \leq ahr_{S_i \cup \{p\}}(S_i, U)$. Let the accelerated greedy algorithm maintain a list l with increasing order to store the values of $ahr_{S \cup \{p\}}(S, U)$ for each point $p \in D \backslash S$. Then let $l(p, S_{i-1})$ be the value of $ahr_{S_{i-1} \cup \{p\}}(S_{i-1}, U)$ for p with current solution set S_{i-1} in the i-th iteration. In the $(i+1)$-th iteration, the accelerated algorithm only identifies the point $p \in \arg\min_{p' \in D \backslash S_i} l$ from the ordered list l. Let $l(p, S_i) \leftarrow ahr_{S_i \cup \{p\}}(S_i, U)$, then updates the value of p in l with $l(p, S_i)$. After this update, if $l(p, S_i) \leq l(p', S_{i-1})$ for all $p' \neq p$, we have

$$\begin{aligned}
ahr_{S_i \cup \{p\}}(S_i, U) &\leq ahr_{S_{i-1} \cup \{p'\}}(S_{i-1}, U) \\
&\leq ahr_{S_i \cup \{p'\}}(S_i, U)
\end{aligned}$$

Thus the algorithm pick out the point that makes the $ahr_{S_i \cup \{p\}}(S_i, U)$ minimized with no need to compute $ahr_{S_i \cup \{p'\}}(S_i, U)$. This is the lazy evaluation of the function

$ahr_{S\cup\{p\}}(S,U)$ and p will not be in result set at current iteration [7]. The accelerated algorithm will set $S_{i+1} \leftarrow S_i \cup \{p\}$ and repeat the process of selecting points until there are no points that meet the conditions. The performance of the greedy algorithm can be significantly improved by using the acceleration strategy of lazy evaluations whose pseudocode is shown in Algorithm 1. The aim of Algorithm 1 is to find the point p^* with the minimum average happiness ratio and add p^* to the result set. Assume p_{j_0} is the point with the minimum happiness ratio in $l(p_j)$ (line 1).Then add p_{j_0} to S_{i-1} to calculate the average happiness ratio (line 2). After resorting the list l (line 3), p_{j_0} is also the point with the minimum average happiness ratio and less than 1, then return p_{j_0} (lines 7-10). Otherwise re-execute Lazy-Evaluation function (lines 4-6). The main advantage of Lazy-Evaluation is to avoid the evaluations of the average happiness ratio function for each point in $D\backslash S_{i-1}$ and improve the efficiency. We will give an example to illustrate the process of lazy evaluation in Example 2.

Algorithm 1: Lazy-Evaluation$(D, S_{i-1}, i, l(p_j))$

Input: A set $D = \{p_1, p_2, \cdots, p_n\}$ that contains n d-dimensional points, a current result set S_{i-1}, an integer i (current number of iterations), and a sorted list $l(p_j)$ keeping the latest value of average happiness ratio of each point $p_j \in D\backslash S_{i-1}$.

Output: A point p^* that minimizes $l(p^*, S_{i-1})$.

1 Initially, let $p^* =$ NULL, $p_{j_0} = \arg\min_{p_j \in D\backslash S_{i-1}} l(p_j)$;
2 $h = ahr_{S_{i-1}\cup\{p_{j_0}\}}(S_{i-1}, U)$;
3 update the value of p_{j_0} in l with h;
4 **if** $h > \min_{p_j \in D\backslash S_{i-1}, p_j \neq p_{j_0}} l$ **then**
5 **return** Lazy-Evaluation$(D, S_{i-1}, i, l(p_j))$;
6 **end**
7 **else**
8 **if** $h < 1$ **then**
9 $p^* = p_{j_0}$;
10 **end**
11 **return** p^*.
12 **end**

After the above speedups, our Lazy NWF-Greedy is developed and its pseudocode is shown in Algorithm 2. Initially, the algorithm starts with an empty set, then adds a point that still outside the current result set and minimizes the average happiness ratio at each iteration. Note that the lazy evaluations are applied in Line 9 when $i > 1$, which means the calculations of average happiness ratio function for each point in D cannot be avoided at the first iteration.

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

Input: A set $D = \{p_1, p_2, \cdots, p_n\}$ that contains n d-dimensional points and an integer k.

Output: A result set S where $|S| \leq k$.

1 Initially, let $S_0 = \emptyset, p^* =$ NULL;
2 **for** *(*$i = 1$*;* $i \leq k$*;* $i++$*)* **do**
3 **if** $i = 1$ **then**
4 **for** *each* $p \in D\backslash S_{i-1}$ **do**
5 calculate $ahr_{S_{i-1}\cup\{p\}}(S_{i-1}, U)$;
6 $l(p, S_{i-1}) = ahr_{S_{i-1}\cup\{p\}}(S_{i-1}, U)$;
7 **end**
8 **end**
9 p^*=Lazy-Evaluation(D, S_{i-1}, i, l);
10 **if** $p^* = NULL$ **then**
11 return S_{i-1};
12 **end**
13 **else**
14 $S_i = S_{i-1} \cup \{p^*\}$;
15 **end**
16 **end**
17 **return** S_i.

By the steps of Algorithm 2, we can get following results.

Theorem 1: Let $\{S_i\}_{i\geq 0}$ be the current solution, then for all positive integers k, the approximation guarantee achieved by Lazy NWF-Greedy is $(1-\beta)(1-1/e)$.

Proof sketch. For the approximation guarantee, $1-1/e$ is obtained by the submodularity of our average happiness function [12] while $1-\beta$ is by the sampling confidence for the sample size is at least $\frac{3\ln(\frac{1}{\beta})}{\alpha^2}$ (Lemma 1).

An example is given to illustrate how to improve the efficiency of our Lazy NWF-Greedy by exploiting lazy evaluations.

Example 2: Consider the car database in the previous section. Given $k = 2$, Lazy NWF-Greedy will report a solution $S = \{p_7, p_4\}$ whose average happiness ratio is 0.8557 in the worst case. Our algorithm starts with an empty set, then adds the point from $D\backslash S_i$ whose average happiness ratio increases the least at each iteration. In this example, the standard greedy performs 15 function evaluations (For first iteration 8 evaluations and second iteration 7 evaluations) to obtain the minimum average happiness ratio at each iteration. This, however, can be improved by the lazy evaluations. The implementation of improvement is shown in Table III where $l(p, S_i)$ denotes the value of $ahr_{S_i\cup\{p\}}(S_i, U)$ in iteration $i+1$ and $\times$ indicates that the calculation of $l(p, S_i)$ can be avoided by using lazy evaluations, *i.e.,* the calculations of function evaluations to p_6, p_8, p_1 and p_5 which occurs for the selection of the 2nd point can be avoided. When the data scale is large,

a lot of calculations of ahr function are avoided thus improve the efficiency to a great extent.

TABLE III
$l(p, S_i)$ FOR ITERATION i

(a) $i = 0$

Points	$l(p, S_0)$
p_1	0.9005
p_2	0.8408
p_3	0.8510
p_4	0.8213
p_5	0.9519
p_6	0.8621
p_7	0.8025
p_8	0.8633

(b) $i = 0(Resorting)$

Points	$l(p, S_0)$
$\boldsymbol{p_7}$	**0.8025**
p_4	0.8213
p_2	0.8408
p_3	0.8510
p_6	0.8621
p_8	0.8633
p_1	0.9005
p_5	0.9519

(c) $i = 1(Calculating)$

Points	$l(p, S_0)$	$l(p, S_1)$
$\boldsymbol{p_4}$	0.8213	**0.8557**
p_2	0.8408	0.8832
p_3	0.8510	0.8841
p_6	0.8621	×
p_8	0.8633	×
p_1	0.9005	×
p_5	0.9519	×

In practice, although there is no theoretical guarantee to the number of evaluations reduced by lazy evaluations. However, even in the worst case, Lazy NWF-Greedy outperforms WO-Greedy since Lazy NWF-Greedy requires much fewer iterations than WO-Greedy. Specifically, Lazy NWF-Greedy only needs k iterations to get the result set, while WO-Greedy needs to perform $n - k$ iterations, where n is the size of the whole dataset and k is the size of the result set. Since the time complexity for each evaluation of our average happiness ratio function is $O(dnN)$, the time complexity of our Lazy NWF-Greedy is $O(dkNn^2)$ while WO-Greedy is $O(dNn^3)$. Note that k is much smaller than n, so Lazy NWF-Greedy is more efficient than WO-Greedy.

IV. Heuristic Strategy

In our preprocessing step, we calculate the skyline set and the algorithms are carried out on the skyline set to obtain final k size results. Instead, as a heuristic we may change the skyline set with the set of best point for each sampled utility function $u \in U$ which can avoid the calculation of the average happiness functions for some sampled utility functions. According to the definition of average happiness ratio, we have

$$ahr_{S\cup\{q\}}(S, U) = \sum_{u\in U} \frac{\max_{p\in S} u(p)}{\max_{p\in S\cup\{q\}} u(p)} Pr(u)$$

If we uniformly sample utility functions from utility function space, then the previous equation can be rewritten as

$$ahr_{S\cup\{q\}}(S, U) = \frac{1}{|U|} \sum_{u\in U} \frac{\max_{p\in S} u(p)}{\max_{p\in S\cup\{q\}} u(p)}$$

Hence, we can pre-store the maximal utility of each utility function $u \in U$ over the current solution set S, and it is efficient since the size of S is very small. Then the calculation of $ahr_{S\cup\{q\}}(S, U)$ can be simplified. The details are shown as follows.

- Case 1: If the utilities of utility functions are maximized by the points $p \in S$, then we can know that $\max_{p\in S\cup\{q\}} u(p) = \max_{p\in S} u(p)$, and $\frac{\max_{p\in S} u(p)}{\max_{p\in S\cup\{q\}} u(p)} = 1$, because the candidate set consists of the best points. Therefore, these utility functions do not need to be calculated.
- Case 2: If the utilities of utility functions are maximized by the point q, then we only calculate these utility functions which can make the utility of point q be larger than $\max_{p\in S} u(p)$.
- Case 3: If the utilities of utility functions are neither maximized by the point q nor $p \in S$, then we need to calculate the utilities of these utility functions of q, and compare with $\max_{p\in S} u(p)$, then get the value of $\frac{\max_{p\in S} u(p)}{\max_{p\in S\cup\{q\}} u(p)}$.

For further improving the efficiency of calculating $ahr_{S\cup\{q\}}(S, U)$, we provide a heuristic strategy, in which we only calculate the utility functions which are maximized by the point q in Case 2, and omit the calculations of the utility functions in Case 3. As we think that if the point q makes more utility functions maximized, the value of $ahr_{S\cup\{q\}}(S, U)$ will be smaller. Therefore, in the experiments, we use this method to improve the efficiency of our algorithms, and the experimental results show that the algorithms adopting heuristic strategy perform well.

V. Lazy Stochastic-Greedy Algorithm

We provide an algorithm which is called Lazy Stochastic-Greedy with a random sampling phase and can accelerate the processing but sacrifice a little happiness ratio. The pseudocode of Lazy Stochastic-Greedy is shown in Algorithm 3. The algorithm repeats the operation of picking out points until k data points are selected and at each round we select a point from a sampled subset R instead of the whole dataset D which accelerates the query processing. In addition, the process of selecting representative points from R can also be accelerated by lazy evaluations. Hence, in Line 11 of Algorithm 3, we can apply lazy evaluations to accelerate our processing. We

omit the details here for the process is similar to Lazy NWF-Greedy. By the steps of Algorithm 3 and the results of [8], we can get Lemma 4 and Theorem 2.

Lemma 4: Given a current solution S, the expected average happiness ratio of Lazy Stochastic-Greedy in one step is at least $\frac{1-\epsilon}{k}\sum_{p\in S^*\setminus S}\triangle(p|S)$.

Lemma 4 gives us one step expectation of average happiness ratio for our Lazy Stochastic-Greedy algorithm. Further, we can obtain following theoretical guarantee of Lazy Stochastic-Greedy.

Theorem 2: Suppose that $s = (n/k)log(1/\epsilon)$, then Lazy Stochastic-Greedy can obtain a $(1-\beta)(1-1/e-\epsilon)$ approximation guarantee in expectation to the optimal solution of the k-average-happiness query with at most $O(nlog(1/\epsilon))$ function evaluations of ahr_D.

Proof sketch. For the approximation guarantee, $1-1/e-\epsilon$ is obtained by the stochastic strategy of our average happiness function [8] while $1-\beta$ is confidence level by sampling (Lemma 1).

Similar to the Lazy NWF-Greedy algorithm, we also provide an example (Example 3) to illustrate our Lazy Stochastic-Greedy algorithm.

Example 3: Recall the car example in the previous section. Given k=2, Lazy NWF-Greedy will report a solution $S = \{p_7, p_4\}$. For our Lazy Stochastic-Greedy, it also starts with an empty set. Then it iterates from $i = 1$ to k and at each iteration samples a subset from $D\setminus S_i$ of size $\frac{n}{k}\log(\frac{1}{\epsilon})$ to obtain the approximation guarantee in Theorem 2. When $k = 2$, suppose that we have set $\epsilon = 0.1$, then the number of random sampling is 4. As described before, WO-Greedy needs 15 function evaluations to obtain the final solution. Instead, it can be reduced to 11 times by exploiting lazy evaluations for Lazy NWF-Greedy as shown in Example 2. For our Lazy Stochastic-Greedy, the total number of function evaluations will be no more than 8 when the size of R is 4. That is, the first iteration results in 4 function evaluations and the second iteration only needs 4 function evaluations. Note that this is the worst case. Suppose that at the first iteration, the random sampled subset $R = \{p_7, p_8, p_1, p_5\}$, then the algorithm will set the list of upper bound $l(p, S_0) = \{0.8025, 0.8633, 0.9005, 0.9519\}$ and result in 4 function evaluations at the first iteration. Suppose that at the second iteration, the random sampled subset is $R = \{p_4, p_8, p_1, p_5\}$, and the average happiness ratio obtained by the algorithm is 0.8557 when adding the point p_4 from the current result set and updates the values of p in l as $l(p, S_1) = \{0.8557, 0.8633, 0.9005, 0.9519\}$. At this moment, the algorithm finds the average happiness ratio achieved by p_4 is still the smallest in the list, then it just needs to pick out p_4 directly without calculating the average happiness ratios of p_8, p_1 and p_5. In this case, the algorithm only results in 5 function evaluations in total.

In Example 3, even in the worst case, the number of function evaluations resulted in by our Lazy Stochastic-Greedy algorithm doesn't exceed the numbers of function evaluations resulted in by WO-Greedy or Lazy NWF-Greedy.

By sampling a suitable set R, Lazy Stochastic-Greedy can provide a solution which is almost identical to Lazy NWF-Greedy in terms of the average happiness ratio. In addition, Lazy NWF-Greedy will result in a computational complexity of $O(nk)$ in terms of function evaluations, while Lazy Stochastic-Greedy only requires a computational complexity of $O(nlog(1/\epsilon))$ which is independent of any user specified k. Therefore, Lazy Stochastic-Greedy is more efficient than Lazy NWF-Greedy, let alone the WO-Greedy algorithm. Also, the heuristic strategy mentioned in Section IV can be applied to Lazy Stochastic-Greedy.

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

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

```
Initially, S_0 = ∅;
for (i = 1; i ≤ k; i++) do
    let p* = NULL;
    randomly sample a subset R of size s from D\S_{i-1};
    for each p_j ∈ R do
        if p_j has not been sampled in previous sampling procedure then
            calculate ahr_{S_{i-1}∪{p_j}}(S_{i-1}, U);
            l(p_j, S_{i-1}) = ahr_{S_{i-1}∪{p_j}}(S_{i-1}, U);
        end
    end
    p* =Lazy-Evaluation(R, S_{i-1}, i, l);
    if p* = NULL then
        return S_{i-1};
    end
    else
        S_i = S_{i-1} ∪ {p*};
    end
end
return S_i.
```

VI. Experimental Results

For synthetic datasets, we adopt the dataset generator of [1] to generate our synthetic datasets. For independent and correlated datasets, the skyline results are much smaller than anti-correlated datasets. Thus, we only consider anti-correlated

datasets in our experiments. For real datasets, we use Household, NBA, Color and Weather datasets for our experiments. The details about the datasets are listed in Table IV.

TABLE IV
DATASETS LIST

dataset	dimension	number
anti-correlated	6	10,000
Household	6	127,391
NBA	8	17,265
Color	9	68,000
Weather	15	566,268

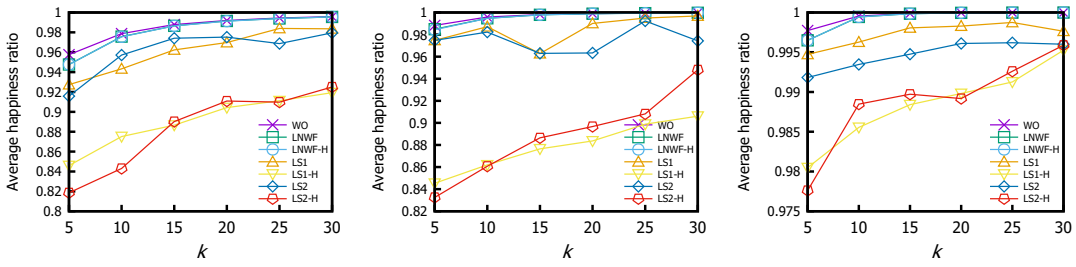
In the experiments, we assume that the users' utility functions are linear and follow a uniform distribution. We sampled 69,077 utility functions where $\alpha = 0.01$ and $\beta = 0.1$. Moreover, similar to the researches in the literature [11], [10], [13], [4], we computed the skyline set first. For the algorithms adopting the heuristic strategy, we further pre-computed the set containing the points maximizing each utility function over the skyline set. Unless stated explicitly, our queries returned 5 to 30 points on these datasets. In addition, the algorithms tested in our experiments include Lazy NWF-Greedy, Lazy Stochastic-Greedy and WO-Greedy, where Lazy NWF-Greedy and Lazy Stochastic-Greedy are introduced in this paper, and WO-Greedy is proposed in [17] (they called WORST-OUT-Greedy). Meanwhile, for our Lazy NWF-Greedy and Lazy Stochastic-Greedy algorithms, we also accelerate them with our heuristic strategy and compare with the accelerated algorithms. We consider the computational cost in terms of the running time of CPU and the quality of the result set in terms of the average happiness ratio. In addition, we calculated the standard deviation and distribution of the happiness ratio of the result set for all pre-sampled utility functions.

The average happiness ratios and query time of all datasets for different k are shown in Figure 2 and Figure 3 respectively. We omit the description about them since they are the same with previous experiments.

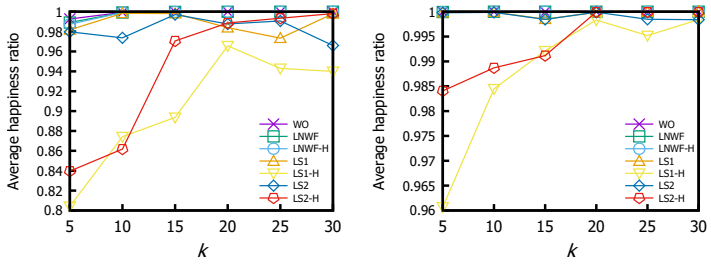
Standard deviations are shown in Figure 4, WO-Greedy and Lazy NWF-Greedy have low standard deviations while Lazy Stochastic-Greedy has higher values, although the standard deviation decreases as more points are selected except the Lazy Stochastic-Greedy. For the Lazy Stochastic-Greedy, the standard deviation is still not higher than 0.1, which is a very low value. Moreover, Figure 5 shows that the happiness ratios of more than 85% of users for the solution sets selected by WO-Greedy and Lazy NWF-Greedy are concentrated on the average happiness ratios, and combining Figure 2 and 4, it can be seen that the happiness ratios are very high. In particular, for the real-world datasets, more than 90% of users have high happiness ratios. For Lazy Stochastic-Greedy, the happiness ratio is not with monotone trends because of the randomness. But there are still more than 80% of users who have a high happiness ratio.

Effect of d and n. In Figure 6, we vary the value of d on the synthetic datasets where n is set to 10,000 and k is set to 10. When d increases, the average happiness ratios of WO-Greedy and Lazy NWF-Greedy decrease slightly while that of Lazy Stochastic-Greedy is in decline. Figure 6(b) shows that the corresponding query time of each algorithm increases as d increases since the number of skyline points increases rapidly due to the curse of dimensionality. It also can be seen that the query times of the stochastic algorithms are always less than that of the non-stochastic algorithms. Moreover, the algorithms with the heuristic strategy run faster than the other algorithms. We present the result of standard deviation in order to show that our stochastic points select strategy in LS is good enough since all standard deviations are comparable with other non-random algorithms. The standard deviation is used to evaluate how significant our stochastic points selection strategy influences the average regret ratio. We see that the standard deviations of all algorithms are not larger than the very small value 0.08. In Figure 6(c), it seems that there exists an abnormal result at $d = 4$. This is because LS is a random algorithm. We also vary the values of n and use the default setting on the synthetic datasets. The results are shown in Figure 7. The average happiness ratios of WO-Greedy and Lazy NWF-Greedy decrease when the cardinality of the dataset n increases, but Lazy NWF-Greedy is more efficient than WO-Greedy, especially when the heuristic strategy is exploited. The Lazy Stochastic-Greedy algorithm maintains a high average happiness ratio all the time as n increases, and run significantly faster than the other algorithms. All algorithms are with low standard deviations. As Figure 6 and 7 show, we can see that our algorithms are more scalable and capable of handling large values of d and n compared with WO-Greedy.

Comparison with the brute-force method. We conduct the experiments for the brute-force method, which can return the optimal solution but is very computationally expensive. To make sure the brute-force method can be completed in acceptable running time, we only sample 100 points from the anti-correlated dataset, and vary the values of k. As Figure 8 shows, the average happiness ratio of WO-Greedy is very close to that of optimal solution. But our Lazy NWF-Greedy algorithm performs not much worse than the optimal solution, even the gap decreases rapidly as k increases. The Lazy Stochastic-Greedy algorithm without the heuristic strategy can

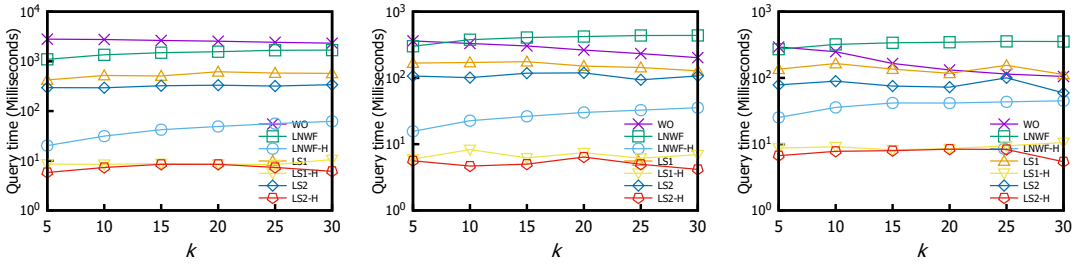


(a) Anti-correlated (b) Household (c) NBA

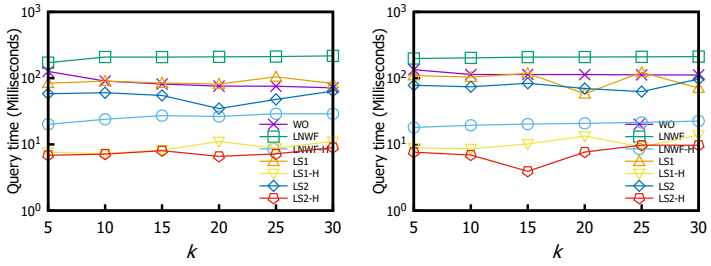


(d) Color (e) Weather

Fig. 2. Average happiness ratios of all the algorithms on anti-correlated and real-world datasets



(a) Anti-correlated (b) Household (c) NBA

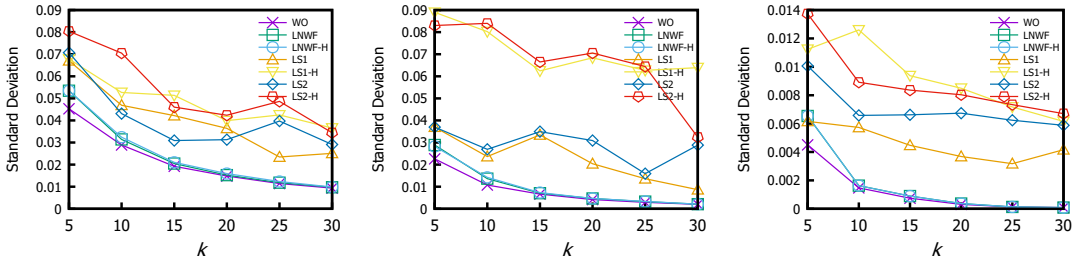


(d) Color (e) Weather

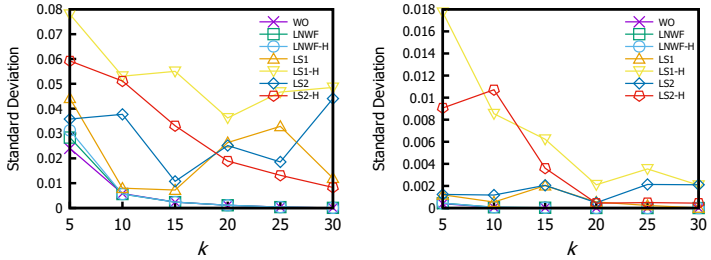
Fig. 3. Query time of all the algorithms on anti-correlated and real-world datasets

return similar results of Lazy NWF-Greedy. When exploiting the heuristic strategy, the results of Lazy Stochastic-Greedy appear not as well as the other algorithms, but it has the least query time, while the Brute-Force, WO-Greedy, Lazy NWF-Greedy and Lazy Stochastic-Greedy algorithms without the heuristic strategy all have 10 times less running time than that of Lazy Stochastic-Greedy with the heuristic strategy. By the way, the Lazy NWF-Greedy algorithm with the heuristic strategy runs as fast as Lazy Stochastic-Greedy with the heuristic strategy but has a better approximation to the optimal solution.

Effect of α and β. Note that a larger α means smaller number of sampled utility functions used in our sampling. Figure 9 shows that the average happiness ratios of all algorithms does not change a lot with α. But the query time of each algorithm decreases rapidly when α increases, because all algorithms calculate the average happiness ratio relying on the number of sampled utility functions. But the query time of WO-Greedy is

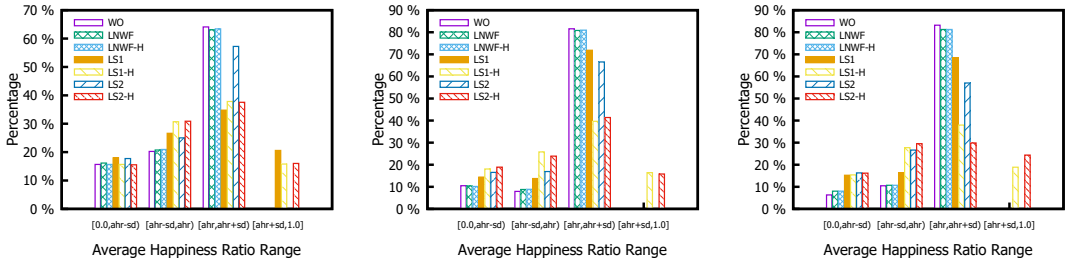


(a) Anti-correlated (b) Household (c) NBA

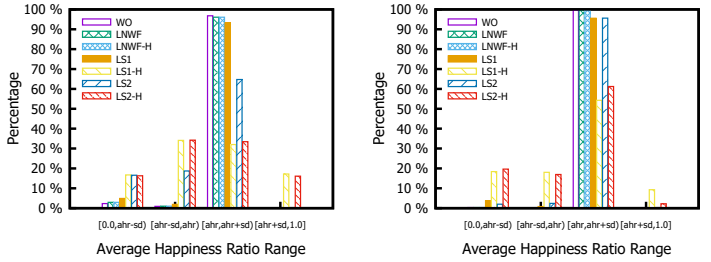


(d) Color (e) Weather

Fig. 4. Standard deviations of all the algorithms on anti-correlated and real-world datasets

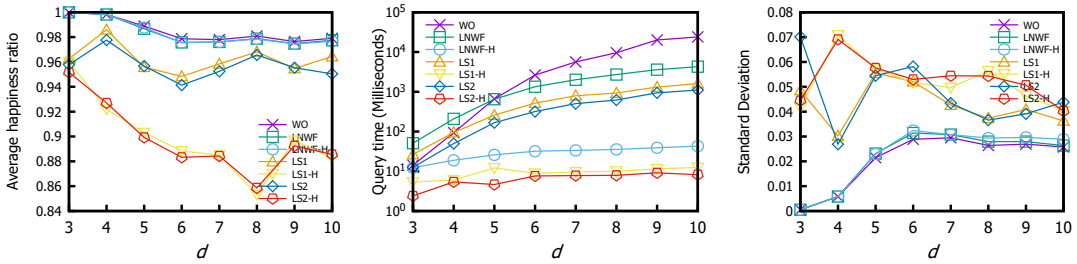


(a) Anti-correlated (b) Household (c) NBA



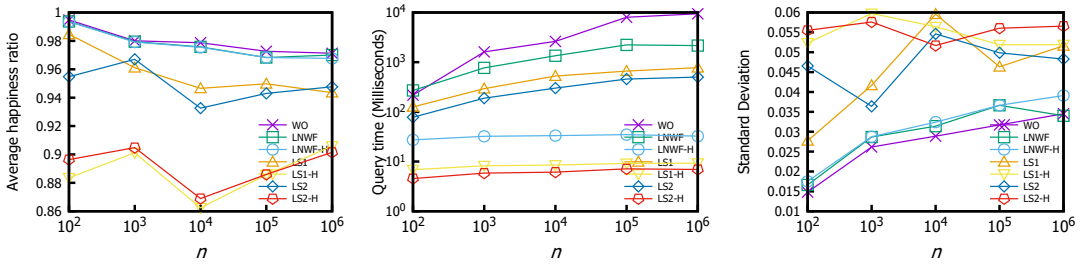
(d) Color (e) Weather

Fig. 5. Distributions of happiness ratio of all the algorithms on anti-correlated and real-world datasets



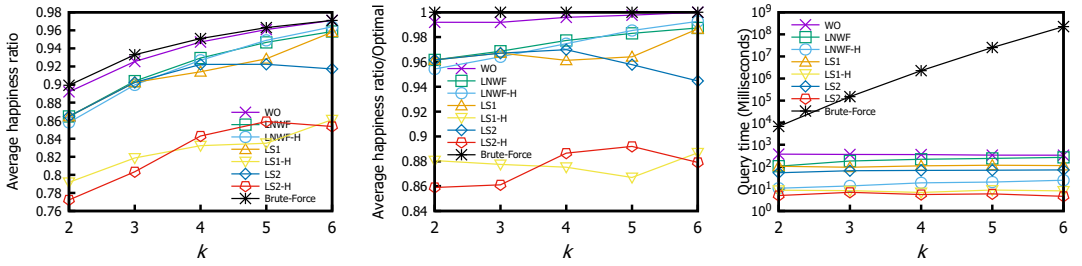
(a) Average happiness ratio (b) Query time (c) Standard deviation

Fig. 6. Performance comparisons of all the algorithms for different values of d on anti-correlated dataset



(a) Average happiness ratio (b) Query time (c) Standard deviation

Fig. 7. Performance comparisons of all the algorithms for different values of n on anti-correlated dataset



(a) Average happiness ratio (b) Average happiness ratio/Optimal (c) Query time

Fig. 8. Performance comparisons of all the algorithms for different values of k on small sampled anti-correlated dataset

always higher than our proposed algorithms. In addition, the standard deviations of all algorithms are with small values. Larger β also means few sampled utility functions, and the results are shown in Figure 10. Similar trends can be observed among the algorithms.

Effect of ϵ. Figure 11 shows that the performance of our Lazy Stochastic-Greedy algorithm with different values of sampling parameter ϵ. The WO-Greedy and Lazy NWF-Greedy algorithms do not use stochastic strategy, so their results remain unchanged. For our Lazy Stochastic-Greedy algorithm, increasing the value of ϵ can decrease the average happiness ratio of the result set, but the query time will also decrease because the number of sampled points decreases. The standard deviation seems to increase, but not more than 0.065, which means that the happiness ratios of most users are around the average happiness ratio which is a high value. Therefore, our Lazy Stochastic-Greedy algorithm can achieve good trade-offs between the average happiness ratio and the query time compared to the other benchmarks.

In summary, different kinds of experiments were carried out on both synthetic and real datasets. The experimental results show the superiority of our proposed algorithms over the existing algorithms. Specifically, Lazy NWF-Greedy has the same average happiness ratio as WO-Greedy. Lazy Stochastic-Greedy sacrifices a little average happiness ratio but the running time is much less than WO-Greedy. In addition, the heuristic strategy can improve our algorithms to a great extent. The Lazy NWF-Greedy algorithm with the heuristic strategy can run at least 10 times faster than WO-Greedy without sacrificing the average happiness ratio. If adding the heuristic strategy to Lazy Stochastic-Greedy, the algorithm will run faster than other algorithms, and just lose acceptable average happiness ratio. If the user cares about the query efficiency, she/he totally can choose this algorithm.

REFERENCES

[1] S. Börzsöny, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

[2] Y. Chang, L. D. Bergman, V. Castelli, C. Li, M. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 391–402, 2000.

[3] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. In *41st International Conference on Very Large Data Bases (VLDB)*, pages 389–400, 2014.

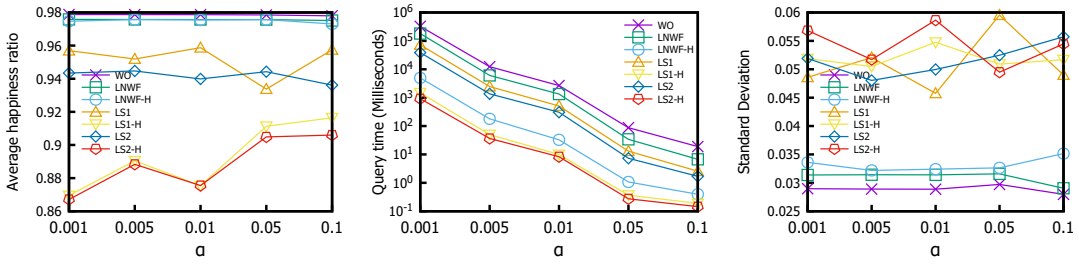
[4] T. K. Faulkner, W. Brackenbury, and A. Lall. k-regret queries with nonlinear utilities. In *42nd International Conference on Very Large Data Bases (VLDB)*, pages 2098–2109, 2015.

[5] A. Krause and D. Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:71–104, 2014.

[6] X. Lian and L. Chen. Top-k dominating queries in uncertain databases. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT)*, pages 660–671, 2009.

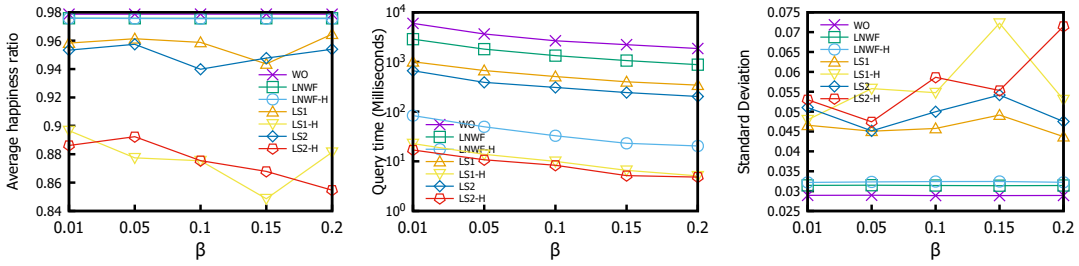
[7] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Proceedings of the 8th IFIP Conference on Optimization Techniques*, pages 234–243, 1978.

[8] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrak, and A. Krause. Lazier than lazy greedy. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)*, pages 1812–1818, 2015.



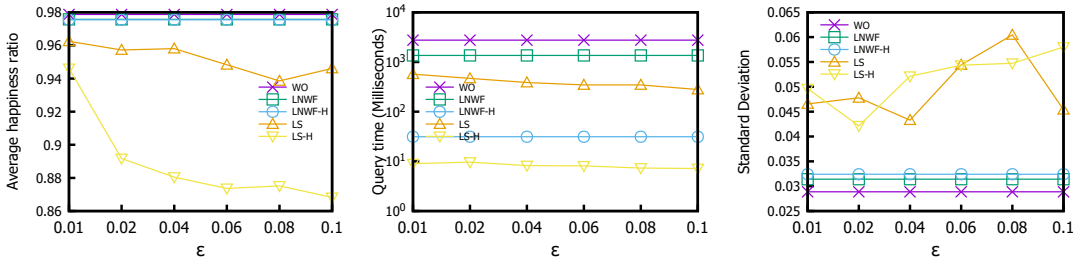
(a) Average happiness ratio (b) Query time (c) Standard deviation

Fig. 9. Performance comparisons of all the algorithms for different values of α on anti-correlated dataset



(a) Average happiness ratio (b) Query time (c) Standard deviation

Fig. 10. Performance comparisons of all the algorithms for different values of β on anti-correlated dataset



(a) Average happiness ratio (b) Query time (c) Standard deviation

Fig. 11. Performances of Lazy Stochastic-Greedy algorithm for different values of ϵ on anti-correlated dataset

[9] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[10] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino. Interactive regret minimization. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 109–120, 2012.

[11] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. In *36th International Conference on Very Large Data Bases (VLDB)*, pages 1114–1124, 2010.

[12] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[13] P. Peng and R. C. W. Wong. Geometry approach for k-regret query. In *IEEE 30th International Conference on Data Engineering (ICDE)*, pages 772–783, 2014.

[14] J. Qi, F. Zuo, H. Samet, and J. C. Yao. K-regret queries using multiplicative utility functions. *ACM Transactions on Database Systems (TODS)*, 43(2):10:1–10:41, 2018.

[15] M. Xie, R. C.-W. Wong, , and A. Lall. Strongly truthful interactive regret minimization. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, pages 281–298, 2019.

[16] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 959–974, 2018.

[17] S. Zeighami and R. C.-W. Wong. Minimizing average regret ratio in database. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, pages 2265–2266, 2016.