k-Pleased Querying

Zitong Chen, Ada Wai-Chee Fu, Cheng Long, and Yang Wu

Abstract—k-Regret Querying is a well studied problem to query a dataset $D$ for a small subset $S$ of size $k$ with the minimal regret ratio for unknown utility functions. In this paper, we point out some issues in k-Regret Querying, including the assumption of non-negative dataset and the lack of shift invariance. Known algorithms for k-Regret Querying are limited in scope and result quality, and are based on the assumption of non-negative data. We introduce a new problem definition called k-pleased querying for dealing with the shift variance issue, and propose a strategy of random sampling of the utility functions. This strategy is based on a study of the theoretical guarantee of the sampling approach. We also introduce a dimensionality reduction strategy, an improved greedy algorithm, and a study of other utility function sampling methods. All of our solutions can handle negative data. Theoretically, we derive a guarantee on the approximation attained by our sampling algorithm. Experimental results on numerous real datasets show that our proposed method is effective even with a small number of samples and small values of $k$.

Index Terms—k-pleased querying, k-regret, top-k querying.

1 INTRODUCTION

Extracting a few tuples that users may be interested in is one of the most important goals for multi-criteria decision making. Consider an example of a car database in [27], shown in the table below:

<table>
<thead>
<tr>
<th>Car</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$p_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPG</td>
<td>51</td>
<td>40</td>
<td>41</td>
<td>34</td>
<td>30</td>
</tr>
<tr>
<td>HP</td>
<td>134</td>
<td>110</td>
<td>191</td>
<td>198</td>
<td>140</td>
</tr>
</tbody>
</table>

In this table, each column represents a brand of car, which has two attributes, miles per gallon (MPG) and horse power (HP). For a user who has a specific preference on each attribute, we can apply top-k querying to recommend cars to the user. For example, a user with preferences of 20% on MPG and 80% on HP is associated with a utility function of 0.2MPG + 0.8HP and will get the top-1 result $p_4$. This corresponds to a utility vector of $(0.2, 0.8)$ in which 0.2 and 0.8 are the weights.

It is often difficult for users to supply utility functions. For returning $k$ objects without user defined utility functions, k-regret querying has been studied in recent years [1], [6], [10], [20], [27], [28], [30], [38]. This querying is a combination of two existing well-known queries, the top-k queries and skyline queries, and finds a limited number of tuples that minimize the user regret based on a set of utility functions. We describe this querying problem in the following.

Definition 1 ($f(t)$ and $f(S)$). A utility function is a mapping $f : \mathbb{R}^d \rightarrow \mathbb{R}$. The utility of a tuple $t$ under a utility function $f$ is denoted by $f(t)$, and for a set of tuples $S$, we denote $\max_{t \in S} f(t)$ by $f(S)$.

As in most previous work we assume linear utility functions that are dot products of a utility vector and the data vector.

Definition 2 (Linear Utility Function). [27] A utility function $f$ is linear if there exist non-negative reals $a_1, \ldots, a_d$ such that $f(p) = \sum_{i=1}^d a_i p_i$ for any $d$-dimensional point $p = (p_1, \ldots, p_d)$. A linear utility function $f$ can be represented by a vector $v = (a_1, \ldots, a_d)$. $f(p)$ is the dot product $v \cdot p$.

Hence we would also use $f$ to refer to its vector $v$.

As a measure of the user’s regret, we define the $mrr$ regret ratio.

As in most previous work we assume linear utility functions that are dot products of a utility vector and the data vector.

Definition 2 (Linear Utility Function). [27] A utility function $f$ is linear if there exist non-negative reals $a_1, \ldots, a_d$ such that $f(p) = \sum_{i=1}^d a_i p_i$ for any $d$-dimensional point $p = (p_1, \ldots, p_d)$. A linear utility function $f$ can be represented by a vector $v = (a_1, \ldots, a_d)$. $f(p)$ is the dot product $v \cdot p$.

1.1 Motivations

A main motivation of our study is to improve on the status quo of the solutions for k-regret querying. The state-of-the-art mechanisms are the algorithm called “Discretize” in [2] and the “Sphere” algorithm in [38]. The idea of “Discretize” is to sample utility functions from the infinite set of possible...
functions. However, the proposed sampling is not evenly distributed in the sample space, which necessitates a large sample size to ensure the quality but such a large sample size often makes the computation infeasible. Our experimental study shows that Discretize returns poor quality results with realistic sample sizes. While Sphere can return results efficiently, it can only handle cases where \( k \geq d \), where \( d \) is the data dimensionality. Experimental results also show that the regret ratios returned by Sphere are not low for small \( k \) values. Since users typically require a small \( k \) for a concise solution, we would need a better solution that can return good quality results for small \( k \) values.

A second motivation for our work is that existing study on \( k \)-regret querying assumes that each tuple in the database is a non-negative real vector, i.e., all values in the database are non-negative. Let us call such a database non-negative, and a database containing some negative values is called negative. Major algorithms and guarantees are based on this assumption [1], [2], [6], [17], [30], [38]. This is not a reasonable assumption since negative values can exist in real datasets, e.g., negative gain (loss), negative temperatures. Even when there is no negative data in the original database, we may introduce negative data when smaller values are preferred to larger ones, because an obvious way to employ known solutions is to negate all values. When data is negative, existing algorithms may not work. Our experiments show that in some cases, such algorithms may even return results with regret-ratios greater than one.

Other than the above major motivations, to attain a more desirable solution, we also consider the problem of shift-variance in \( k \)-regret querying.

### 1.2 Shift Invariance

Though \( k \)-regret querying is known to be scale invariant, it is not shift invariant. If we add some constant to all values, the result may change. This is a problematic property. There exist attributes that naturally involve shifting when converting between different metric systems or different reference points, such as temperature, the coordinates of a location, and relative elevations. We illustrate the problems with the example below, where we assume a class of linear utility functions with a unit (1-norm) weight vector.

**Example 1.** Given four furnace models (C,F,K,N) as shown in Table 1 below. There are two attributes: the Eco-friendliness level of the furnaces; and the temperature at a standard energy level. We consider \( \mathcal{L} \) to be all possible 2-dimensional utility vectors with 1-norm equal to one. The utility functions are dot products of such utility vectors and the data vector.

Consider 1-regret querying for a single object that gives the smallest regret ratio. That is, we look for a solution \( S \) with size 1. First let the metric be Celsius. For model \( C \), the data vectors are \( t_C = (18.2, 19), t_F = (19, 18), t_K = (20, 16), \) and \( t_N = (17, 20) \). Let us consider the utility vectors \( f_0 = (0, 1), f_0(t_C) = 0 \times 18.2 + 1 \times 19 = 19, f_0(t_F) = 18, f_0(t_K) = 16, f_0(t_N) = 20 \). Thus, \( f_0(D) = 20 \). The regret ratio of \( t_C \) is \( r_{RD}(t_C, f_0) = 1 - 19/20 = 0.05 \), similarly, \( r_{RD}(t_F, f_0) = 1 - 18/20 = 0.1, r_{RD}(t_K, f_0) = 0.2, \) and \( r_{RD}(t_N, f_0) = 1 \). Next consider \( f_1 = (1, 0) \), \( f_1(t_C) = 18.2, f_1(t_F) = 19, f_1(t_K) = 20, f_1(t_N) = 17 \).

Thus, \( f_1(D) = 20 \). The regret ratios of \( t_C, t_F, t_K, t_N \) are 0.09, 0.05, 0, and 0.15, respectively. We can derive that \( mrr_D(t_C, \mathcal{L}) = 0.09, mrr_D(t_F, \mathcal{L}) = 0.1, mrr_D(t_K, \mathcal{L}) = 0.2, mrr_D(t_N, \mathcal{L}) = 0.15 \). To minimize the maximum regret ratio, \( t_C \) should be selected for \( S \). This is shown in the first row in Table 1 (b).

We can see that the 1-regret results are \( C, F \), and \( K \) when using Celsius, Fahrenheit, and Kelvin, respectively. However, the dataset has not changed, so results should be the same. The source of this problem is that \( k \)-regret is not shift invariant, and the changes in metrics in the above involve shifting of the data values.

A common practice of normalizing the data values to the range \([0, 1]\) does not help since it also involves shifting of data. E.g., the 1-regret result is \( N \) when using normalization in the above. Thus, shift variance can lead to inconsistency in results. Our solution aims also to mitigate this problem.

### 1.3 Our Contributions

1. We convert \( k \)-regret querying to a new problem called \( k \)-pleased querying, in which we explicitly define an anchor point to eliminate the effects of data shifting. Although the problem is fundamentally shift variant, this explicit definition helps us to avoid unexpected inconsistency: \( k \)-pleased querying returns a set \( S \) of up to \( k \) points that can best satisfy the utility functions. (2) We introduce a random utility function sampling technique for the solution of \( k \)-pleased querying, which is shown to outperform the state-of-the-art algorithms for the \( k \)-regret problem. Moreover, our solution can handle negative databases. (3) We prove that the gap between the exact pleased ratio and the pleased ratio attained by our algorithm is bounded and this bound is independent of the database size, and holds in spite of negative data. (4) For faster response time, we introduce an efficient greedy algorithm that also can handle negative data. (5) Our empirical studies show that our proposed method typically returns a pleased ratio that is significantly better than the state-of-the-art algorithms and also runs much faster than the best known utility sampling based algorithm.

### TABLE 1

<table>
<thead>
<tr>
<th>Furnace Model</th>
<th>C</th>
<th>F</th>
<th>K</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>E (Eco Level)</td>
<td>18.2</td>
<td>19</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>T (temperature)(°C)</td>
<td>19</td>
<td>18</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>T (temperature)(°F)</td>
<td>66.2</td>
<td>64.4</td>
<td>60.8</td>
<td>68</td>
</tr>
<tr>
<td>T (temperature)(°K)</td>
<td>292.2</td>
<td>291.2</td>
<td>289.2</td>
<td>293.2</td>
</tr>
</tbody>
</table>

(a) Three Metrics

<table>
<thead>
<tr>
<th>Furnace Model</th>
<th>C</th>
<th>F</th>
<th>K</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celsius</td>
<td>9%</td>
<td>10%</td>
<td>20%</td>
<td>15%</td>
</tr>
<tr>
<td>Fahrenheit</td>
<td>9%</td>
<td>5.29%</td>
<td>10.59%</td>
<td>15%</td>
</tr>
<tr>
<td>Kelvin</td>
<td>9%</td>
<td>5%</td>
<td>1.36%</td>
<td>15%</td>
</tr>
<tr>
<td>Normalization</td>
<td>25%</td>
<td>50%</td>
<td>100%</td>
<td>15%</td>
</tr>
</tbody>
</table>

(b) Regret Ratio

1. Note that this result can be derived readily by the linear time algorithm for 1-pleased querying in Section 6.
This paper is organized as follows: We state the problem and some preliminaries in Sections 2 and 3. The random sampling technique is introduced in Section 4 with theoretical guarantee analysis in Section 5. In Section 6, we introduce our greedy algorithm and in Section 7, we present our experimental results. Section 8 is about related work. We conclude in Section 9.

2 k-PLEASED QUERYING

As with k-regret querying, we consider the setting where utility functions are unknown. We are given a set of data points in d-dimensional space, and we consider a class of non-negative linear utility functions $L$ (see Definition 2).

**Definition 3 (Pleased Ratio).** Given a database $D$ of tuples where each tuple is a data point in a d-dimensional space, and a reference point $t_0$ in the data space, we define the pleased ratio of a set of tuples $S \subseteq D$ under utility function $f$ to be:

$$ple_D(S, f) = \frac{\max_{t \in D} f(t-t_0)}{\max_{o \in D} f(o-t_0)}$$

(2)

where $t_0$ is treated as a data point and is shifted or scaled as other data points. The pleased ratio of $S$ under a utility function class $L$ is defined as

$$ple_D(S, L) = \inf_{f \in L} (ple_D(S, f))$$

(3)

Intuitively, $ple_D(S, f)$ is about the best relative utility we can get for $f$ given $S$ instead of $D$, and $ple_D(S, L)$ is the worst case pleased ratio among functions in $L$, it is about the most "hard-to-please" utility function in $L$ given $S$ instead of $D$. The pleased ratio is up to 100% and can be negative, since with negative data, the utility of a tuple can be negative. Our problem is to find a set $S$ of up to $k$ data points from a set $D$ of data points, so that the pleased ratio of $S$ is maximized.

**Problem 2 (k-pleased querying).** Given a dataset $D$ of tuples with $d$ numerical attributes, so that each tuple can be represented as a d-dimensional vector, an integer $k$, a linear utility function class $L$ with non-negative coefficients, k-pleased querying is to find a subset $S \subseteq D$ with at most $k$ tuples such that the pleased ratio of $S$ under $L$ is maximized. That is,

$$S = \arg \max_{T \subseteq D, |T| \leq k} \left( ple_D(T, L) \right)$$

$ple_D(S, L)$ is the k-pleased ratio of $D$.

Unlike k-regret querying, k-pleased querying is shift-invariant. The same result will be returned after any shifting in the attribute values.

**Lemma 1.** k-pleased querying is shift-invariant.

**Proof:** The definition of k-pleased querying is based on the reference point $t_0$, which will be shifted with all data points. Let us refer to the data set obtained when shifting all points in a data set $D$ by $t$ as $D+t$. That is, if $D = \{t_0, \ldots, t_n\}$, then $D+t = \{t_0+t, \ldots, t_n+t\}$. k-pleased querying is shift-invariant because the reference point $t_0$ is also shifting when points in $D$ are shifting. When $D$ becomes $D+t$, $t_0$ becomes $(t_0+t)$. Solving the k-pleased problem on $D+t$ with reference point $t_0+t$ is equivalent to solving the problem on database $D$ with reference point $t_0$:

$$ple_{D+t}(S+t, f) = \frac{\max_{t \in D} f((t+t)-(t_0+t))}{\max_{o \in D} f((o+t)-(t_0+t))}$$

$$= \frac{\max_{t \in D} f(t-t_0)}{\max_{o \in D} f(o-t_0)} = ple_D(S, f)$$

Similar to the k-regret operator, the k-pleased operator has the nice properties of being scale invariant and stable. We use the definition of stability in [27], that is, for any subset $S \subseteq D$, the pleased ratio of $S$ does not change after some junk tuples (tuples that are not optimal for any linear utility function) are added to $D$. The arguments for scale invariance and stability are similar to that for k-regret, and interested readers can refer to [27] for such proofs.

2.1 An Underlying Assumption

There is an underlying assumption about $D$ for both k-regret and k-pleased querying. It has been implicit in previous work.

**Assumption 1.** For any utility function $f \in L$ with non-negative weights, $f(D) > 0$.

This assumption is needed for the following reasons. Firstly, as a denominator in Equation (1), $f(D)$ should not be zero; Secondly, if $f(D) < 0$, clearly for any $S \subseteq D$, $f(S) \leq f(D) < 0$, then $\forall S \subseteq D$, $rr_D(S, f) = 1 - f(S)/f(D) < 0$. By definition, a smaller regret ratio of $S$ means better utility or a greater $f(S)$. However, given $f(D) < 0$, for two subset $S_1$ and $S_2$ with $f(S_1) < f(S_2)$, we have $rr_D(S_1, f) < rr_D(S_2, f)$, which is contradictory to our expectation. Similar arguments hold for the k-pleased problem.

2.2 k-regret as a Special Case

For the k-regret problem setting, when the reference point $t_0$ equals $(0,0,\ldots,0)$, $rr_D(S, f) = 1 - ple_D(S, f)$, the maximum requirement on the pleased ratio is equivalent to the minimum requirement on the regret ratio:

$$\arg \max_{S \subseteq D} \left( \inf_{f \in L} \frac{f(S) - f(t_0)}{f(D) - f(t_0)} \right) = \arg \max_{S \subseteq D} \left( \inf_{f \in L} \frac{f(S)}{f(D)} \right)$$

Thus, k-regret problem is a special case of the k-pleased problem. The k-regret problem is shown to be NP-hard in [10], thus the k-pleased problem is also NP-hard.

**Theorem 1.** k-pleased querying is NP-hard.

2.3 The Reference Point $t_0$

Given a dataset $D$, the reference point $t_0$ can be any reasonable point in the data space which satisfies Assumption 1. An example of an unreasonable reference can be $(\max_{i=1}^d, \max_{i=1}^d, \ldots, \max_{i=1}^d)$, where $\max_{i}$ is the maximum data value of $D$ in the $i$-th dimension, since Assumption 1 will be violated. Note that the system needs to keep track of the reference point $t_0$. If the data $D$ is shifted, $t_0$ will be shifted and the shifted $t_0$ is used as the reference point for the shifted $D$. Note that user needs to select $t_0$. In Example 1, the choice of $t_0$ would make one of C, F, K, N the answer, and this answer will not change.
Lemma 2. That is, we consider utility functions of the form \( \text{max}_{i} \), which we can safely discard some dimensions (Section 3.2). In the following, we show that instead of considering the utility function vectors with unit length in any norm (Section 3.2), we can replace \( \text{max}_{i} \) by definition, there exists some point \( q \) which dominates \( p \), we can replace \( p \) by \( q \) in \( S \) without reducing the pleased ratio value. Thus, in all algorithms, we first compute the skylines which will be used as our input data.

The third property helps us to limit our attention to utility function vectors with unit length in any norm (Section 3.1). The fourth property is about the conditions under which we can safely discard some dimensions (Section 3.2).

3.1 Unit Length Utility Functions

In the following, we show that instead of considering the set of utility function vectors of any length of any norm, we may limit our attention to those with a 1-norm equal to one. That is, we consider utility functions of the form \((x_1, x_2, \ldots, x_d)\) where \( \sum x_i = 1 \).

Lemma 2. Given any \( p \geq 1 \), let \( \mathcal{L}_p = \{ f \in \mathcal{L} \mid \|f\|_p = c \} \), we have \( p\text{le}(D, \mathcal{L}) = p\text{le}(D, \mathcal{L}_p^{c}) \) for any positive \( c \).

Proof: For a positive value \( c \), \( p\text{le}(D, S, cf) = \frac{\text{max}_{i \in \mathcal{L}(cf)}(s)}{\text{max}_{i \in \mathcal{L}(f)}(s)} = \frac{\text{max}_{i \in \mathcal{L}(cf)}(s)}{\text{max}_{i \in \mathcal{L}(f)}(s)} = p\text{le}(D, f) \), \( p\text{le}(D, S, \mathcal{L}_p) = \inf_{\|f\|_p = c} p\text{le}(D, S, f) = \inf_{\|f\|_p = 1} p\text{le}(D, S, cf) = \inf_{\|f\|_p = 1} p\text{le}(D, S, f) = p\text{le}(D, S, \mathcal{L}_p) \), \( p\text{le}(D, S, \mathcal{L}) = \inf_{f \in \mathcal{L}} p\text{le}(D, S, f) = \inf_{f \in \mathcal{L}} p\text{le}(D, S, cf) = \inf_{f \in \mathcal{L}} p\text{le}(D, S, f) = p\text{le}(D, S, \mathcal{L}_p) \), \( p\text{le}(D, S, \mathcal{L}^c) = \inf_{c > 0} p\text{le}(D, S, f) = \inf_{c > 0} p\text{le}(D, S, cf) = p\text{le}(D, S, \mathcal{L}_p^c) \).

The above lemma says that the pleased ratio of \( S \) considering \( \mathcal{L} \) is equivalent to that considering \( \mathcal{L}^c \) where \( \mathcal{L}^c = \{ f \in \mathcal{L} \mid \|f\|_1 = c \} \) for any \( p \geq 1 \) and any \( c > 0 \). Setting \( p = 1 \) and \( c = 1 \), we can see that adding a constraint of \( \|f\|_1 = 1 \) for any utility function \( f \) gives us the same result as the class of unrestricted linear utility functions.

3.2 Dimension Reduction

As we shall see, the dimensionality of the dataset has great impact on the computation complexity of our algorithms for \( k \)-pleased querying. Here we introduce a general condition for reducing the dimensionality of the data without affecting the overall result. This strategy will be incorporated in the proposed algorithms.

Denote \( D, S, \) and \( \mathcal{L} \) regardless of the dimensions in set \( I \) by \( D^I, S^I, \) and \( \mathcal{L}^I \), respectively. Denote the minimal and maximal values of the \( i \)-th dimension by \( \text{min}_{i} \) and \( \text{max}_{i} \), respectively. We have the following lemma, a proof of which is given in the Appendix.

Lemma 3. If \( \forall S \subset D \) and \( |S| = k, p\text{le}(D, \mathcal{L}) < c \), let \( I = \{ i \mid \max_{i} \geq c \} \), then \( p\text{le}(D, \mathcal{L}) = p\text{le}(D(S^I, \mathcal{L}^I)) \).

This lemma says that once we have an upper bound \( c \) on the maximal pleased ratio of \( k \) tuples, any dimension \( i \) with \( \min_{i} \geq c \) can be eliminated. In particular, we can ignore dimension \( i \) if all data points have the same value in this dimension, since \( \min_{i} = 1 \) and \( c \leq 1 \).

3.3 The Algorithm Framework

In this subsection, we introduce the framework of the main algorithm we propose for \( k \)-pleased querying. A major challenge in \( k \)-pleased querying is that we need to handle an infinite set of utility functions, we tackle this problem by selecting a finite subset of the utility functions for maximizing the pleased ratio. This idea of utility function sampling has been used in [1] and [2]. However, as a crucial factor which determines the performance, the utility function sampling strategy is not satisfactory in these existing studies. No theoretical guarantees in these works apply to a database that involves negative values. Our study introduces a much improved utility function sampling strategy with a theoretical guarantee that applies to negative data, which will be presented in Sections 4 and 5.

The sampling based algorithm is given in Algorithm 1. We keep a matrix \( M \), where \( M[f_i, f_j] \) is the pleased ratio of tuple \( t_j \) under only one utility function \( f_i \). A binary matrix \( M' \) is constructed from \( M \) to set binary values for indicating whether a given pleased ratio threshold is passed by adopting \( t_j \) under utility function \( f_i \). A minimum hitting set problem (MHS) is solved for finding a minimum set of columns (data tuples) in \( M' \) such that together they “hit” every row (function), where hitting means that at least one \( M' \) entry enters to one for each row when restricted to these tuple columns (which implies that the pleased ratio of the set of points corresponding to the columns in the set is at least the threshold). A binary search on the possible pleased ratio values will determine the greatest pleased ratio threshold for which the hitting set has size at most \( k \).

The above steps follow the framework of the MRST Oracle in [2]. A main difference here is that we also keep track of an upper bound of pleased ratio, \( ub \), at Line 16 so that we may apply Lemma 3 to reduce the dimensionality at subsequent MHS function calls. An approximate solution of the \( k \)-pleased problem is returned by Algorithm 1.

Example 2. Let \( D = \{ t_1 : (1, 0, 0, 0, 9), t_2 : (0, 1, 0, 9), t_3 : (0, 7, 0, 7, 0, 9), t_4 : (0, 0, 1) \} \) and \( k = 2 \) be the input of Algorithm 1. Let \( U \) be a randomly sampled utility function.
Algorithm 1 $k$-Pleased Querying Algorithm

Input: $D, k, U$

/* $U$ is a set of sampled utility functions */

Output: $S$

Initialization: $D$ ← the skyline of $D$
matrix $M$ with initial size $|U| 	imes |D|
vector $V$ to collect pleased ratios
upper bound of pleased ratio $ub = 1.0$
set of dimensions to be discarded $I \leftarrow \phi$.
1: for the $i^{th}$ utility function $u \in U$ do
2: for the $j^{th}$ tuple $t \in D$ do
3: $M[i, j] \leftarrow \text{ple}_D((t), u)$
4: end for
5: end for
6: insert distinct values in $M$ to $V$; sort $V$
7: binary search in $V$ to find the largest $\text{ple}$, s.t. $\text{MHS}(M, \text{ple}, k)$ returns true
8: Let $S$ be the set of tuples found in the last MHS call returning true.
9: return $S$

Function: $\text{MHS}(M, \text{ple}, k)$
10: initialize matrix $M'$ of the same size as $M$
11: set each entry $M'[i, j]$ to 1 if $M[i, j] \geq \text{ple}$, otherwise 0
12: Solve a minimum hitting set problem to find a smallest set of columns $S$ in $M'$ that "hits" each row of $M'$
13: if $|S| \leq k$ then
14: return true and $S$
15: else
16: update $ub \leftarrow \text{ple}$
17: apply dimensionality reduction to get $I$
18: if $I \neq \emptyset$ then
19: call Algorithm 1 on $D^{(I)}$
20: else
21: return false and $\emptyset$
22: end if
23: end if

end Function

set $\{f_1 : (1, 0, 0), f_2 : (0, 1, 0), f_3 : (0.64, 0.6, 0.48), f_4 : (0.48, 0.36, 0.8)\}$. Then $M$ can be represented by a table as follows:

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>1</td>
<td>0</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>$f_2$</td>
<td>0</td>
<td>1</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>$f_3$</td>
<td>0.825</td>
<td>0.794</td>
<td>1</td>
<td>0.369</td>
</tr>
<tr>
<td>$f_4$</td>
<td>0.917</td>
<td>0.826</td>
<td>1</td>
<td>0.612</td>
</tr>
</tbody>
</table>

$V$ will be $\{1, 0.917, 0.826, 0.825, 0.794, 0.7, 0.612, 0.369\}$ after being sorted. In Line 7, we begin a binary search in the middle of $V$, and we choose $\text{ple} = 0.794$ first, and for this $\text{ple}$ value we call Function MHS. Then, $M'$ will be the one shown in Table 2(a). It is easy to see that the first two columns (tuples) can "hit" all rows (functions), so that the values true and $S = \{t_1, t_2\}$ will be returned in Line 14. Next, we resume the binary search step in Line 7, and we choose $\text{ple} = 0.826$ in $V$, and call Function MHS again for this $\text{ple}$ value. Then $M'$ will be the one shown in Table 2(b). In $M'$, no two columns can "hit" all rows. Then an upper bound of pleased ratio, 0.826, is found in Line 16. Only for the third dimension, we have $\min_{\max} = 0.9 > 0.826$, thus, $I = \{3\}$ in Line 17. And in Line 19, we trigger Algorithm 1 on dataset $D'$, as we eliminate the third dimension, where $D' = \{t_1 : (1, 0), t_2 : (0, 1), t_3 : (0.7, 0.7), t_4 : (0, 0)\}$.

Table 2

<table>
<thead>
<tr>
<th>Matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
</tr>
<tr>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 0</td>
</tr>
</tbody>
</table>

Note that no assumption is made in this algorithm that the data is non-negative. Algorithm 1 works for both positive and negative data.

Time Complexity of Algorithm 1. Computing the matrix $M$ in Line 1 to 3 takes $O(|V|d)$ time (since we first compute $u(t)$'s for each $u \in U$ and $t \in D$ and based on that, we then compute $\text{ple}_D((t), u)$'s, where computing $u(t)$'s has the dominating cost of $O(|V||D|d) = O(|V|^2)$). Getting the distinct values in $V$ and sorting $V$ in Line 6 takes $O(|V| \log |V|)$ time. In Line 6, the number of iterations for binary search in $V$ is $O(\log |V|)$. Without dimension reduction, the function MHS takes $O(|V| + T)$ time, with $O(|V|)$ time for computing the matrix $M'$, and $T$ is the time complexity of the minimal hitting set solver in Line 12, which can be found in [26], hence the overall runtime is $O(|V|d + |V| \log |V| + T \log |V|)$. With dimension reduction, the worst case is that each time $I$ in Line 17 has size 1, thus, Algorithm 1 will be called at most $d$ times, however, $T$ will be smaller since it depends on the dimensionality.

4 Utility Function Sampling

In Algorithm 1, we assume that we are given a set $U$ of sampled utility function as input. Intuitively, the more sampled utility functions in $U$, the better the result quality but also the higher the computation cost. Thus it is important and challenging to design a good sampling method.

To guarantee a good result quality, we would like the sampled utility functions to be dense, that is for any utility function $f$, there exists a sampled utility function $u$ which is close to $f$. We define a concept called "$\delta$-cover" as follows for measuring how dense such a sample set is.

Definition 4 ($\delta$-cover). A utility function sample set $U$ from $L_1^d$ is a $\delta$-cover if $\forall f \in L_1^d, \exists u \in U$, s.t. $\|f - u\|_2 \leq \delta$.

A $\delta$-cover may contain a large number of sampled utility functions, which leads to a long running time. Thus, we propose a sampling method called Random (Section 4.1) to generate a sampled utility function set $U$ with some probability to be a $\delta$-cover (Corollary 1). In our empirical study, 1000 samples by Random are sufficient for all our datasets.

In Section 4.3, we briefly review the state-of-the-art sampling algorithm called Discretize [2], which is based on $L_2$-norm space partitioning. We observe this algorithm could be improved if a $L_1$-norm space is used instead of a $L_2$-norm one. We call this improved version Discretize L1. Still, both Discretize and Discretize L1 use too many utility functions and are not as effective as Random, as shown in our experiments.

4.1 New Sampling Algorithm: Random

Our main method is a random selection of utility function vectors from $L_1^d$, which is very simple, but the analysis
about its properties and guarantees is more intricate. First we introduce the concepts of a range space for the utility function set, and define $\epsilon$-net. Next we derive a condition under which a utility sample set $U$ is an $\epsilon$-net. Range space and $\epsilon$-net are defined in [15] as follows:

**Definition 5 (Range Space).** A range space $S$ is a pair $(X, R)$ where $X$ is a set and $R$ is a set of subsets of $X$. Members of $X$ are called points of $S$ and members of $R$ are called ranges of $S$.

**Definition 6 ($\epsilon$-net).** Let $(X, R)$ be a range space, $A$ be a finite subset of $X$ and $\varepsilon \geq 0$ be a positive real number. Let $R_{A, \varepsilon}$ denote the set of all $r \in R$ that contain a fraction of the points in $A$ such that $\frac{|A \cap r|}{|A|} > \varepsilon$. A subset $N$ of $A$ is an $\varepsilon$-net of $A$ (for $R$) if $N$ contains a point in each $r \in R_{A, \varepsilon}$.

The study of $\varepsilon$-net is based on a finite set $X$. In our problem setting, $X$ is mapped to the utility function set, which is infinite. However, we can consider a finite set that can return the same results as for the infinite set by considering a number of points that is made arbitrarily large. In the following we make use of an integer $n$ that can be set as large as necessary to create a finite point set that is effectively equivalent to the infinite set.

For the utility sampling algorithm for $\delta$-pleased querying, we consider the following range space which we call the $kP$ range space:

**Definition 7 ($kP$ Range Space).** A $kP$ Range Space is a quintuple $(X, R, A, d, n)$ where $n$ is a positive integer, $X = A$ are points in the $d$-dimensional space with coordinates $(x_1, x_2, \ldots, x_d)$ where $x_i$ is a multiple of $(1/n)$, and $\sum_i x_i = 1$. That is, $x_i$ can take values from $0, 1/n, 2/n, \ldots$. Set $R$ is a set of subsets of $X$. For any $r \in R$, there exists a value $\Delta_r, 0 < \Delta_r < 1$, and a point $a \in X$, where $a = (a_1, \ldots, a_d)$, such that $r$ consists of the points $(y_1, y_2, \ldots, y_d)$ where $\sum_i |y_i - a_i| \leq \Delta_r$, and $\sum_i y_i = 1$. That is, each $r \in R$ is defined by some point $a \in X$, and a distance $\Delta_r$. We call $a$ the anchor point of $r$, and we call $\Delta_r$ the width of $r$.

We consider utility functions with non-negative weights, $A$ is the set of unit 1-norm utility functions ($A = L^1$). The following lemma allows us to link an $\varepsilon$-net for a $kP$ range space to a $\delta$-cover (definition 4) which is a critical factor for the guarantee derived in Section 5.

**Lemma 4.** Given a $kP$ range space $(X, R, A, d, n)$, if $U$ is an $\varepsilon$-net of $A$ in $R$, then it is a $\delta$-cover for $L^1$ with $\varepsilon = (\delta / \sqrt{2})d^{-1}$.

A proof of the above lemma is given in Section 4.2. Our goal is to attain a $\delta$-cover. The above lemma says that we can do so by sampling an $\varepsilon$-net where $\varepsilon$ depends on $\delta$. Next we can apply results in [15] and [3], which hold for range spaces with a finite VC dimension. For a $kP$ range space, it is easy to see that the VC dimension is finite, since $R$ cannot shatter a set of $d + 1$ points in $A$ where there are $d$ points with value 1 at one of the dimensions and one point with value 1/d in all dimensions. Then, from results in [15] and [3] we get the following lemma:

**Lemma 5.** For a $kP$ range space $(X, R, A, d, n)$, and $\epsilon, \sigma > 0$, if $N$ is a set of $m \geq O\left(\frac{\epsilon}{\delta} \log \frac{1}{\epsilon} \cdot \frac{\delta}{\log \frac{1}{\epsilon}}\right)$ random independent samples drawn from $A$, then $N$ is an $\epsilon$-net of $A$ for $R$ with probability at least $1 - \sigma$.

**Corollary 1.** For a $kP$ range space $(X, R, A, d, n)$, and $\epsilon > 0$, if $N$ is a set of $m \geq O\left(\frac{\epsilon}{\delta} \log \frac{1}{\epsilon}\right)$ samples random independent draws from $A$, then $N$ is an $\epsilon$-net of $A$ for $R$ with probability at least $1 - \epsilon$.

The above corollary says that by randomly and independently picking $O(\frac{\delta}{\epsilon} \log \frac{1}{\epsilon})$ points from $A$, we have a high probability to get an $\epsilon$-net. Lemma 4 says that given an $\epsilon$-net, we have a $\delta$-cover, where $\epsilon = (\delta / \sqrt{2})d^{-1}$. In Section 5 we will derive theoretical guarantees for the $kP$-solved algorithm of Section 1 given a $\delta$-cover. Hence, with sufficient sampling points, we have a high probability of forming a $\delta$-net which implies a $\delta$-cover and for which the theoretical guarantees would apply. Thus, we propose here a random sampling method, called Random, which effectively picks points randomly and independently from $A$ in the $kP$ range space, which corresponds to $L^1$.

To generate random 1-norm utility function, we need to generate $d$ coordinates which sum up to 1. It is equivalent to dividing a segment with length 1 into $d$ pieces. In order to have $d$ pieces, we need $d - 1$ cut points (empty pieces are allowed, thus, duplicated cut points are allowed). This is implemented by randomly picking $d - 1$ values in $[0,1]$, sorting the values, if the sorted values are $c_1, c_2, \ldots, c_{d-1}$, then the utility function is given by $(c_1, c_2 - c_1, c_3 - c_2, \ldots, 1 - c_{d-1})$. For example, given $d = 4$, the randomly generated values for the cut points are 0.8, 0.3, 0.7. Sort these values, 0.3, 0.7, 0.8. Then the 1-norm utility function will be $(0.3, 0.4, 0.1, 0.2)$. Notice that the probability of generating any point in $L^1$ is the same except at the boundary where the value at some dimension is 0, but we can ignore such points as this layer of points is almost infinitely thin and will not affect our result.

### 4.2 Proof of Lemma 4

Let us first introduce some lemmas.

**Lemma 6.** Given a $kP$ range space $(X, R, A, d, n)$, the total number of points in $A$ is $\binom{n+d-1}{d-1}$.

**Proof:** A point in $A$ can pick a value from a set of $n+1$ values, namely, $0, \frac{1}{n}, \frac{2}{n}, \ldots, 1$ in each of the $d$ dimensions. Thus, to form a point $a$ in $A$, we select one value from the $n+1$ values from each dimension, however, the total from all the $d$ selected values should add up to exactly 1. Note that 1 is made up of $n$ elements of value $1/n$ each. The way to form a point in $A$ can be seen as lining up $n$ points in a row and try to cut the row into $d$ portions, using $d - 1$ cut points. To allow a combinatorial analysis, we add the cut points to the set of $n$ points, so that we are lining up $n + d - 1$ points in a row. Next we select $d - 1$ points in this row as cut points to form $d$ segments. The length of the $i$-th segment is the multiple of $1/n$ for the $i$-th dimension of the point $a$. The number of ways to form $a$ is thus choosing $d - 1$ points from $n + d - 1$ points.

**Lemma 7.** Given a $kP$ range space $(X, R, A, d, n)$. Consider any element $r$ in $R$ with a width of $2\Delta$ and an anchor point $a = (a_1, \ldots, a_d)$. For any point $b = (b_1, \ldots, b_d)$ in $r$, $\|b - a\|_2 \leq 2\Delta$.

**Proof:** By the definitions of anchor point and width in Definition 7, $\sum_i |b_i - a_i| \leq 2\Delta$. Since both $b$ and $a$ are
in $A$, $\sum_i a_i = 1$, and $\sum_i b_i = 1$. Hence, $\sum_i (b_i - a_i) = 0$, then $b_i - a_i = -\sum_{j \neq i} (b_i - a_j)$, $2|b_i - a_i| = |b_i - a_i| + |\sum_{j \neq i} (b_i - a_j)| \leq \sum_i |a_i - b_i| \leq 2\Delta$, i.e. $|b_i - a_i| \leq \Delta$ for $1 \leq i \leq d$. Let $y_i = |b_i - a_i|$, then $|b - a|_2^2 = \sum_i y_i^2$. Let us consider the maximal value of $\sum_i y_i^2$ subject to $0 \leq y_i \leq \Delta$ and $\sum_i y_i \leq 2\Delta$. $\sum_i y_i^2$ gets its maximal value when at most one of $\{y_i\}$ is not equal to 0 and not equal to $\Delta$, otherwise, there exist $0 < y_i < \Delta$ and $0 < y_j < \Delta$, reset $y_i = y_i + y_j$, and $y_j = 0$ will also fit the subjection but increase $\sum_i y_i^2$. Thus $\sum_i y_i^2 \leq \Delta^2 + (\sum_i y_i - \Delta)^2 \leq 2\Delta^2$. We conclude that $\|b - a\|_2 \leq \sqrt{2}\Delta$.

Lemma 8. Given a kP range space $(X, R, A, d, n)$, if a utility function set $U \subseteq L^1_1$ is an $\epsilon$-net of $A$ for $R$, then $\forall f \in L^1_1$, $\exists u \in U$, s.t. $\|f - u\|_2 \leq \delta$, where $\epsilon = (\delta / \sqrt{2})^d - 1$.

Proof: $A$ is the set of utility function vectors with 1-norm $\leq 1$. We consider those sets in $R$ with a size of at least $\epsilon |A|$. Consider a set $B$ in $R$ with a width of $2\Delta$. The number of points that $B$ covers in $X$ is upper bounded by $(\Delta n + d - 1)$. To see this, consider the case where the anchor point of $B$ is a boundary point $a = (a_1, ..., a_d)$, meaning that only one dimension, say $a_j = 1$, and $a_i = 0$, for $i \neq j$. For example, $a$ can be $(1, 0, 0, ..., 0, B)$ is then bounded by points of the form $(b_1, ..., b_d)$, there $b_j = 1 - \Delta$ and $b_\delta = \Delta$, for some $j \neq \delta$, and $b_i = 0$ for $i \neq j$ and $i \neq \delta$, $\gamma = 2\Delta$. E.g., such a point may be $(1 - \Delta, \Delta, 0, ..., 0)$. The region of $X$ occupied by $B$ is similar in shape to that of $A$ but the hypercube containing $B$ has sides of length $\Delta$. Thus, the size of $B$ is given by $|B| = (\Delta n + d - 1)$.

The reasoning is similar to the proof of Lemma 6 since this is a distribution of $\Delta n$ points among $d$ dimensions. The regions for non-boundary points will have bigger sizes. (E.g., Fig. 1 and Fig. 2 compare a set with a boundary anchor point and a set with a non-boundary anchor point. The shaded areas show the sizes of the sets. Here the value of $\Delta$ is $1/3$.)

From Lemma 6, $|A| = \binom{n + d - 1}{d - 1}$. We note that $\epsilon$ can be expressed as $\frac{n + d - 1}{d - 1}$.

$|B| / |A| = \frac{(\Delta n + d - 1)}{(n + d - 1)!} \leq \frac{(\Delta n + d - 1)(\Delta n + d - 2) ... (\Delta n + 1)}{(n + d - 1)!} \leq \frac{\Delta n}{n(n + d - 2) ... (n + 1)} \leq \frac{\Delta n}{n^d} \leq \frac{\Delta}{n} \leq \frac{\Delta}{d - 1}$

Let $\epsilon = \frac{\Delta}{d - 1}$. If $U$ is an $\epsilon$-net, then it hits all elements in $R$ with a width of $2\Delta$ or above.

For any utility function vector $f$, if we consider points in $A$ with a 1-norm distance of $2\Delta$ from $f$, they form an element $r \in R$ with a size of at least $|B|$, and $r$ is the anchor point of $r$. If $U$ is an $\epsilon$-net, $r$ contains at least one element $u \in U$. From Lemma 7, $\|f - u\|_2 \leq \sqrt{2}\Delta$. Hence, there exists $u \in U$ such that $\|f - u\|_2 \leq \delta = \sqrt{2}\Delta$. With $\Delta = \frac{\delta}{\sqrt{2}}$, we derive that $\epsilon = (\delta / \sqrt{2})^d - 1$.

Then Lemma 4 follows by the definition of $\delta$-cover.

4.3 Existing Sampling Algorithm Discretize and an Enhanced Variation DiscretizeL1

Discretize is the name of Algorithm 3 of [2], with a control parameter $\gamma$ on the number of sampled utility functions.

The idea of sampling here is based on the partitioning of the space into hyper-cones with similar angles on each hyperplane. The number of equal-size partitions of the right angle is given by $\gamma$, thus, the angle of each partition is given by $\pi / (2\gamma)$. The set of utility functions generated by Discretize has a size of $\gamma + 1 - d$. However, the discretized utility functions are not evenly distributed, they are concentrated on the polar. This can be seen in Fig. 3 for discretization in a 3D space where $\gamma = 3$. Samples are taken at the intersection points of the latitude and longitude lines, which are dense near the $x$ pole but sparse near the $y$ pole. This uneven distribution is a major drawback.

We note that Discretize can be easily improved by considering $L_1$-norm space partitioning, we call this DiscretizeL1. Let $L^1_1$ be the set of utility functions with $L_1$-norm equal to 1, and let the number of equal-size partitions of the unit vector at each attribute be $\gamma$, then the DiscretizeL1 sampled utility function set is $U = \{f \in L^1_1 \mid \forall i, f_i \text{ is a multiple of } \frac{1}{\gamma}\}$.

The DiscretizeL1 sampled utility function set $U$ in 3D dataspace with $\gamma = 3$ is illustrated in Fig. 4. There are in total 10 points corresponding to 10 utility functions, the point in the middle is $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. To generate this $U$ is equivalent to enumerating $d$ non-negative integers whose sum equals $\gamma$, and then multiply by $\frac{1}{\gamma}$. Thus the size of $U$ is $(\gamma + 1) - d$. The algorithm for enumeration is trivial and is skipped here.

In Table 3, we compare the sample set sizes of Discretize and DiscretizeL1. We take some dimensionalities from the datasets used in our experiments, and $\gamma = 6$ is the default value used in [2]. DiscretizeL1 has much smaller sample set sizes when compared to Discretize. With Discretize, when $\gamma = 6$, dimensionalities of 7 or above would lead to a sample
size that is too big for computation.

5 THEORETICAL GUARANTEE

In this section, we derive a theoretical guarantee on the pleased ratio for Algorithm 1. The analysis makes no assumption of non-negative data.

We start by introducing three constants \((M_1, M_2, \text{and } Z)\) used in the analysis. From Lemma 2, Definition 3 for pleased ratio, and Equation 3, for any tuple set \(S\), there exists a utility function \(w\) with \(\|w\|_1 = 1\), such that \(ple_D(S, L_1^t) = ple_D(S, w)\). Let \(W\) be the set of such \(w\) for any \(S\).

**Definition 8** \((M_1, M_2, Z)\). Let \(M_1 = \max_{w \in E} \|t\|_2\), \(M_2 = \min_{w \in E} \|w\|_{D}(D), \text{and } Z = \frac{M_1}{M_2}\).

**Lemma 9.** \(M_2\) is lower bounded by the 1-pleased ratio of \(D\).

Proof: Since \(w(D) \geq 0\) according to Assumption 1 and \(|W|\) is finite, we have \(M_2 > 0\). For any non-negative \(w\), \(w(t) = \sum_{1 \leq i \leq d} w_i t_i \geq \sum_{1 \leq i \leq d} w_i \min_{1 \leq i \leq d} t_i = \|w\|_1 \min_{1 \leq i \leq d} t_i.

Thus, \(M_2 = \min_{w \in E} w(D) = \min_{w \in E} \max_{e \in E} w(t) \geq \min_{w \in E} \max_{e \in E} \|w\|_1 \min_{1 \leq i \leq d} t_i = \max_{e \in E} \min_{1 \leq i \leq d} t_i\). Then we get a lower bound of \(M_2\), i.e. \(\max_{e \in E} \min_{1 \leq i \leq d} t_i\), which is also the 1-pleased ratio of \(D\) (see Corollary 2). □

Note that a linear time algorithm for computing the 1-pleased ratio will be presented in Section 6.

As shown in Lemma 2, it is sufficient to only consider \(L_p^t\) (i.e. utility functions with \(L_p^t\)-norm equal to 1 for \(p \geq 1\)) instead of \(L\). Thus, we only consider possible utility functions with \(L_p^t\)-norm equal to 1 in this section, i.e. \(U \subset L_p^t\).

Based on the fact that the inner product of two vectors is bounded by the product of their \(L_2\)-norms, we can get this lemma:

**Lemma 10.** For any two utility functions \(f_1, f_2 \in L, \text{for any vector } v, \text{if } \|f_1 - f_2\|_2 \leq \delta, \text{then } |f_1(v) - f_2(v)| \leq \delta \|v\|_2\).

**Lemma 11.** Given a \(\delta\)-cover \(U, 0 < \delta < \frac{1}{2}, \forall S \subset D, ple_D(S, L_1^t) \geq \min\{1 - Z\delta, ple_D(S, U)\} - Z\delta \ 

Proof: Let \(w \in L_1^t\) and \(ple_D(S, L_1^t) = ple_D(S, w)\). There exists \(v \in U\) such that \(\|w - u\|_2 \leq \delta\). According to Lemma 10, \(v \in D\).

\[ u(t) \geq w(t) - \|t\|_2 \geq w(t) - M_1 \delta \]

Thus, \(u(D) = \max_{t \in E} u(t) \geq \max_{t \in E} w(t) - M_1 \delta = w(D) - M_1 \delta\). Similarly, \(u(D) \leq w(D) + M_1 \delta\).

\[
\frac{w(S)}{w(D)} \geq \frac{u(S) - M_1 \delta}{w(D)} = \frac{u(S)}{w(D)} - \frac{M_1 \delta}{w(D)}
\]

If \(u(S) \geq 0\),

\[
\frac{w(S)}{w(D)} \geq \frac{u(S) - M_1 \delta}{w(D)} = \frac{u(S)}{w(D)} - \frac{M_1 \delta}{w(D)}
\]

Thus, \(w(D) \geq (1 - M_1 \delta) \frac{u(S)}{w(D)} - M_1 \delta\). Since \(M_1 \delta < 1\),

\[
\frac{w(S)}{w(D)} \geq (1 - M_1 \delta) \frac{u(S)}{w(D)} - M_1 \delta
\]

\[
\frac{w(S)}{w(D)} \geq (1 - Z\delta) ple_D(S, U) - Z\delta
\]

\[
\frac{w(S)}{w(D)} \geq (1 + Z\delta) plea_D(S, U) - Z\delta
\]

\[
\frac{w(S)}{w(D)} \geq (1 + Z\delta) plea_D(S, U) - Z\delta
\]

Thus, for any \(u(S), plea_D(S, L) = \frac{w(S)}{w(D)} \geq \min\{1 - Z\delta, plea_D(S, U)\} - Z\delta \)

Let \(S_L\) be the \(k\)-pleased querying result considering all utility functions in \(L\). Let \(S_U\) be the \(k\)-pleased querying result considering all utility functions in \(U\).

**Lemma 12.** Given \(0 < \delta < \frac{1}{2}\), and a \(\delta\)-cover \(U\), we have: \(ple(S_U, L) \geq plea(S_U, L) - (1 + |ple(S_U, L)|)Z\delta\)

Proof: \(ple(S_U, L) \geq \min\{1 - Z\delta, plea_D(S_U, U), (1 + Z\delta) plea_D(S_U, U)\} - Z\delta\)

The first two steps are in accordance with Lemma 11, and the fact that \(ple_D(S_U, U)\) is maximal. The third step is based on the fact that \(ple_D(S_U, U) \geq plea_D(S, L)\).

Lemma 12 shows that the result \(S_U\) has a pleased ratio \(ple(S_U, L)\) with a lower bound in terms of \(\delta\) and the optimal pleased ratio \(ple_D(S_U, L)\). Our main guarantee is as follows.

**Theorem 2.** Given a non-negative database \(D\), and \(\varepsilon > 0\), we can find an approximate solution \(S_U\) of \(k\)-pleased querying with \(ple_D(S_U, L) - plea(S_U, L) \leq \varepsilon\), by Algorithm 1, using a \(\delta\)-cover \(U\) as the utility function sample set, where \(\delta = \frac{\varepsilon}{2d\sqrt{d}}\).

Proof: Note that \(ple(S_U, L) \leq 1\), and \(ple(S_U, L) \geq 0\) due to a non-negative \(D\), then from Lemma 12,

\[
ple(S_U, L) - plea_D(S_U, L) \leq 2Z\delta.
\]

Moreover, from Property 1, as \(M_1\) is the maximal \(L_2\)-norm of a tuple, \(M_1 \leq \sqrt{d}\). For each attribute, pick one tuple with value equal to 1 in the attribute, then we get a group of \(d\) tuples, say \(B\). Let \(\bar{B}\) be \(\frac{1}{2} \sum_{t \in B} t\), then \(\bar{B}\) has value at least \(\frac{1}{2}\) in each dimension. For any utility function \(f, f(t) \geq \frac{1}{2} \|f\|_1\), and \(f(t) = \frac{1}{2} \sum_{t \in B} f(t) = \frac{1}{2} \sum_{t \in B} f(B) = f(B)\) thus, \(f(B) \geq f(t) \geq \frac{1}{4} \|f\|_1\). Thus \(M_2 \geq \inf_{f \in L_1} f(B) \geq \inf_{f \in L_1} \frac{1}{4} \|f\|_1 \geq \frac{1}{4} \|f\|_p = \frac{1}{d}\). Hence \(Z \leq \frac{M_1}{M_2} \leq d \sqrt{d}\). Substituting this and \(\delta = \frac{\varepsilon}{2d\sqrt{d}}\) into inequality (5) above completes the proof. □
Comparison with Previous Work. The idea of sampling utility functions is proposed in the MRST Oracle in [2]. They derived a theoretical guarantee on the regret ratio, but the proof (Theorem 4) assumes that $D$ is non-negative: the tuple $t'$ (see Fig. 33 in [2]) for minimizing $f'(t')$ is no longer on the $y$-axis if we have tuples with negative $x$ values. [1] adopts utility function sampling for the dual problem of regret minimizing. The theoretical analysis and the algorithm based on Lemma 6 in the paper rely on non-negative datasets, since with negative data, the lower bound of $w(u, P')$ is not $\frac{1}{\sqrt{d}}||u||$ as stated at the end of the proof for the lemma, when there can be negative values in the vector $P'$. Thus, we present the first theoretical analysis that accommodates negative data.

6 Greedy Algorithm

Greedy algorithms have been proposed for the $k$-regret problem, and similar ideas can be used for the $k$-pleased problem. Such algorithms make use of boundary tuples for the initial choices, which are tuples with the maximal value in one of the dimensions. In [27], only one boundary point is picked, while in [30], for each dimension, one boundary tuple is chosen. However, such an algorithm may return negative pleased ratios when given a negative dataset (see Table 5). We illustrate this issue with an example.

Example 3. Suppose $D$ is 4 dimensional, and consists of the four data points: $(1, -1, -1, -1)$, $(-1, 1, -1, -1)$, $(-1, -1, 1, -1)$, and $(0.9, 0.9, 0.9, 0.9)$. Let $k = 3$. Let $t_0$ be the origin. With the existing methods, where boundary points are selected, the first three points will be selected, and this will give a negative pleased ratio, with the worst case utility function being $(0, 0, 0, 1)$. The best solution should at least include the fourth point $(0.9, 0.9, 0.9, 0.9)$ which gives a positive pleased ratio.

Here we propose a different greedy selection step. In particular, the first point we select for our solution is the

\begin{algorithm}
\caption{Greedy}
\textbf{Input:} $D, k$
\textbf{Output:} $S$
\begin{algorithmic}[1]
\STATE $S \leftarrow$ solution of 1-pleased querying
\WHILE {$|S| \leq k$ and $ple_D(S) < 1.0$}
\STATE $g \leftarrow PickNext(D, S)$
\STATE $S \leftarrow S \cup \{g\}$
\ENDWHILE
\STATE return $S$
\end{algorithmic}
\end{algorithm}

such value among all tuples gives the 1-pleased ratio and the corresponding tuple is the 1-pleased solution. Hence, the 1-pleased solution can be found in $O(|D| \times d)$ time.

It is easy to see that as long as a dataset contains a data point $(p_1, ..., p_d)$ where $p_i \geq 0$ for $1 \leq i \leq d$, then the 1-pleased ratio will not be negative, since the 1-pleased solution should be at least as good as $p$, and clearly, $ple_D \{(p_i, L) \geq 0$. For example, for the dataset in Example 3, the point $(0.9, 0.9, 0.9, 0.9)$ will be returned as the 1-pleased solution and the pleased ratio will be positive.

The greedy algorithm is outlined in Algorithm 2. For the initial point of the algorithm, we compute the 1-pleased solution $S$ by Corollary 2. This solution contains only one tuple. Next we add more tuples to $S$ by the function $PickNext$ greedily, each time we pick the tuple whose absence leads to the current minimum pleased ratio. With any current set $S$, $PickNext(D, S)$ selects the next tuple as follows: for each $t \notin S$, compute the pleased ratio of $S$ when the database is $S \cup \{t\}$, pick the tuple $t$ giving the smallest such pleased ratio to be returned as $g$. Similar to [27], for a given tuple set $S$ and a given tuple $t$, we can compute $ple_{S \cup \{t\}}(S, L) = \inf f(C, L) \leq \inf f(C, D) = sup \frac{c_1}{f(C, D)} = c_1$.

Consider $f'(0, 1, 0, 0, 0, ..., 0)$, by Property 1, $f'(D) = 1$. $ple_D(C, L) = \inf f(C, L) \leq \inf f(C, D) = c_1$. Thus, $ple_D(C, L) = c_1$.

Corollary 2. The 1-pleased ratio of $D$ is given by

\[ \max_{i \in \mathbb{E}} \min_{1 \leq i \leq d} t_i \]

Linear Time Algorithm for 1-pleased Querying. From the above discussion, in order to compute a 1-pleased solution, we need only examine each tuple $t$ in $D$, and get the smallest attribute in $t$, and this is the value of $ple_D(t, L)$. The greatest

7 Experiments

We have conducted experiments to compare different utility function sampling strategies and evaluate the performance of our proposed algorithms. Our experimental setup is as follows.

Hardware and Platform: We run experiments on a machine with a 3.4GHz 8 Intel Core i7-4770 CPU and 16 GB RAM, running Ubuntu 12.04 LTS Linux OS. All algorithms were implemented in C++ and compiled with GNU C++ 4.6.3 compiler.

Real-world Datasets: We used seven real-world datasets: Airline (AIR), which is used in both [2] and [38], Household (House), and Movie which are all used in [38], Department of Transportation (DOT), Color dataset (COL), and Weather dataset (WE), which are used in [2], [30] and [10], respectively. The characteristics of the datasets are shown in Table 4.

Synthetic Data: Following existing papers on skyline querying and $k$-regret querying, we experiment with synthetic datasets. In particular, anti-correlated data is widely known
to be challenging and serves as our stress test data. We adopt the dataset generator developed by [4] to generate anti-correlated datasets. Unless stated explicitly, for synthetic datasets, the number of tuples is set to 10,000, the dimensionality varies from 3 to 10, with a default value of 6. The value of $k$ varies from 3 to 20, and the default value of $k$ is 10. Such settings are adopted from [27] and [2].

**Settings and Objectives:** We evaluated each of the algorithms in terms of both its pleased ratio and its computation time (including time for computing skylines, sampling utility function set $U$ and querying). In all experiments, the skyline of the dataset is computed at the beginning.

For the $k$-pleased problem, we set the reference point $t_0$ to be the origin. The choice of $t_0$ has no impact on the methods, but choosing the origin means that for non-negative data, our $k$-pleased solutions are also solutions for the $k$-regret problem. In particular, if the solution is $S$, then $ple_D(S, L) = 1 - mrr_D(S, L)$. Thus, we can compare our results with those of previous algorithms for $k$-regret.

The values of $k$ are 3, 5, 7, 10, 13, 15, 20, and its default setting is 10. Such settings are adopted from [27] and [2]. For all experiments which involve randomness, the average performance of 30 test trials is reported.

For the hitting set Function MHS in Algorithm 1, we use the source code from the Vera-Licona Research Group \footnote{2. https://github.com/VeraLiconaResearchGroup/Minimal-Hitting-Set-Algorithms.git} for which the corresponding algorithm is proposed in [26]. To our knowledge, this is the best known implementation for the hitting set problem. We modified the code to stop searching for minimal hitting sets if 10 of them are found, since typically there are many solutions for a given problem instance, and we only need one such set. We pick the one with the highest pleased ratio among these 10 minimal hitting sets.

### 7.1 Comparison on Sampling Methods

We first compare the performance of different utility function sampling methods for $U$ in Algorithm 1. Three methods to generate $U$, namely, Random, Discretize and DiscretizeL1, are considered (see Section 4). The value of $k$ is 10 in these experiments. Different sizes of $|U|$ are studied, the maximum size is $10^5$.

The results on the datasets DOT and COL are shown in Fig. 5 and Fig. 6, respectively (those results on other datasets show similar trends and they are not shown. According to these results, Discretize does not perform well. For example, in Fig. 5, on the dataset DOT, when $|U| = \gamma^{d-1} = 3^6, 4^6, 5^6$, the pleased ratios are $0.094, 0.094$, and $0.424$, respectively. In the empirical study in [2], it is found that setting $\gamma$ between 4 and 6 seems appropriate. However, even when $|U| = 6^6 = 46656$, i.e. $\gamma = 5$, the pleased ratio is only 0.504. In contrast, with $|U| \approx 500$, Random already gives a pleased ratio of 0.927. Since Discretize needs a large number of sampled utility functions, quite a number of points are missing in the figures because the computation becomes infeasible. In terms of the pleased ratio, Random performs the best in most cases. For the running time, it increases as the number of samples increases for all sampling methods, and the trends for all methods are similar since they follow the same algorithm framework.

We also find that increasing the number of sampled utility functions helps only a little in improving the pleased ratio, while the running time increases significantly.

### 7.2 Comparison on Querying Algorithms

We compare the performance of different algorithms, namely Discretize, Sphere, Greedy, and Random. Discretize is the algorithm from [2], which is Algorithm 1 in Section 3.3 using the Discretize sampling method in Section 4. Sphere is from the most recent work of [38], Greedy is our algorithm described in Section 6. Random is Algorithm 1 described in Section 3.3 using the random utility sampling method in Section 4.1. Note that Random is shown to be the best sampling approach according to the first experiment. We compare with Discretize and Sphere because they are shown to outperform previous methods and are considered the state-of-the-art solutions for the $k$-regret problem.

The control parameter $\gamma$ for Discretize is set to be 6 (AIR), 3 (House), 2 (Movie), 6 (DOT), 4 (COL) and 2 (WE). The reason for this setting is as follows: $\gamma = 6$ is the default setting for both AIR and DOT datasets in [2]; for the other datasets, a larger $\gamma$ would make it run out of memory or take too long in the runtime. For Random, we fix the number of sampled utility functions to be 1000, since we found that 1000 is a good tradeoff in the first set of experiments.

The comparison on different datasets is shown in Fig. 7 - 12. We observe that even when we increase the value...
of \( k \), the pleased ratio by Discretize does not change much (please see Fig. 7 for an example). The reason is that the sampling based algorithm may reach a small set of tuples which already achieves 100% pleased ratio with respect to \( U \). In such a case, no more tuples are inserted even when \( k \) increases. For example, in Fig. 9, when \( k = 13 \), a tuple set \( S \) with 100% pleased ratio on \( U \) is returned, so the same tuple set \( S \) with 13 tuples is returned when \( k \) = 15, 20, thus, the pleased ratio remains unchanged.

We also observe that the pleased ratio remains unchanged when \( k \) increases in Fig. 8. This is due to the high dimensionality of the Movie dataset, which is 19. Adding a few more tuples helps little in improving the pleased ratio.

A major weakness of Sphere is that it can only handle the cases where \( k \geq d \), thus, we can see quite many missing results in the figures. Another weakness we can see is that it generates poor results for \( k \) below 20. This shows that their theoretical guarantee is quite loose, the bound is loose because it is asymptotically optimal with respect to the worst case database and not the given database. Random not only returns results for all values of \( k \), it also returns very high pleased ratios for the small values of \( k \), and these are the more difficult cases since it is difficult to satisfy all utility functions with very few data points, but which is a reasonable user requirement.

We have also tested with the improved greedy algorithm in [38] and it performs almost the same as Sphere in all of our tests, since they are so similar, its results are not shown. The similarity may be explained by the fact that both of them first select \( d \) boundary points, and when \( k \) close to \( d \), the large proportion of boundary points dominates the effects, and it makes little difference what additional points are included.

Overall, Random performs better than all the other methods in terms of the pleased ratio results and with a comparable running time in most cases.

### 7.3 Performance of the Greedy Algorithm

This experiment is trying to show the effect of different initialization methods of the Greedy algorithm. In [27], the initialization is choosing the tuple with the highest value in the first attribute. Since the order of attributes does not affect the \( k \)-regret problem or the \( k \)-pleased problem, we may choose the tuple with the highest value in the second or other attribute. It means that we have \( d \) ways of choosing the first tuple, and the average pleased ratio is used for comparison. Let us call this original greedy algorithm OGreedy.

The algorithm that we propose in Section 6 is called Greedy. The comparison of OGreedy versus Greedy on the real datasets is shown in Table 5. In Section 6, we pointed out the issue about negative pleased ratio which may be returned by OGreedy with negative data. This happens with the dataset COL when \( k = 3 \). We remark that Sphere suffers from the same problem since the first step of Sphere is to select \( d \) boundary points, although this is not observed in
TABLE 5
Pleased Ratio Results

<table>
<thead>
<tr>
<th></th>
<th>k = 4</th>
<th>k = 6</th>
<th>k = 10</th>
<th>k = 15</th>
<th>k = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVIE</td>
<td>0.694</td>
<td>0.79</td>
<td>0.826</td>
<td>0.845</td>
<td>0.858</td>
</tr>
<tr>
<td>DOT</td>
<td>0.828</td>
<td>0.837</td>
<td>0.849</td>
<td>0.861</td>
<td>0.867</td>
</tr>
<tr>
<td>COL</td>
<td>0.341</td>
<td>0.464</td>
<td>0.624</td>
<td>0.78</td>
<td>0.791</td>
</tr>
<tr>
<td>AIR</td>
<td>-0.01</td>
<td>0.14</td>
<td>0.382</td>
<td>0.531</td>
<td>0.592</td>
</tr>
<tr>
<td>WE</td>
<td>0.334</td>
<td>0.356</td>
<td>0.483</td>
<td>0.539</td>
<td>0.6</td>
</tr>
<tr>
<td>HOUSE</td>
<td>1.0</td>
<td>0.887</td>
<td>0.986</td>
<td>0.987</td>
<td>0.987</td>
</tr>
</tbody>
</table>

*1 in each table entry, the top value is from OGreedy, the bottom value is from Greedy.*

Fig. 13. Impact of k on Anti-Correlated Data

Fig. 14. Impact of d on Anti-Correlated Data

our experiments. Overall, Greedy not only avoids negative results, but also gives better pleased ratios in general. And from the perspective of running time, OGreedy and Greedy are almost the same.

7.4 Anti-Correlated Data

We study the effect of all algorithms mentioned above, on anti-correlated data. We set the number of sampled utility functions |U| to be 50.

Varying k: We generated an anti-correlated dataset with 10,000 6-dimensional points as in [27], [30], [38]. The results are shown in Fig. 13.

Varying dimensionality d: We generated an anti-correlated data set with 10,000 d-dimensional points, and we fixed k to be 10. The results are shown in Fig. 14.

Random has the best pleased ratio in all cases, however, it also takes the longest running time, since it sometimes gets stuck at the calculation of minimal hitting set.

7.5 Summary

In summary, we conducted various experiments on both real and synthetic datasets, showing that our k-Pleased Querying Algorithm (Random) enjoys the following strong points:

(1) performs well in terms of pleased ratio on both real and synthetic datasets, when compared to state-of-the-art algorithms Sphere and Discrete; (2) can handle all k values, which is not possible for Sphere; (3) can handle negative datasets, and will not return negative pleased ratios; (4) has comparable running time on real datasets.

8 Related Work

Returning k objects or records that are deemed the best according to some scoring function has been found very useful and a lot of study has been made on related topics. A survey of top-k query processing techniques can be found in [19]. The top-k querying assumes the input of utility functions by the users. NRA [13] assumes sorted records and TA [12] also considers random access. Other well known techniques include CA [12], upper/Pick [5], [24], PREFER [16], ONION [9], and ranked join indices [37]. Top-k querying in uncertain databases has been studied in [21] and [34]. Diversity in the objective is considered in [14].

Skyline querying also does not require user defined utility functions but there is no control over the result size [4]. There have been studies to return k representative skyline tuples, however, as pointed out in [27], such approaches are problematic. [22] proposes to select k skyline points so that the number of points dominated by at least one of these k skyline points, is maximized. This approach is scale invariant but not stable. [36] considers distance-based representative skyline, which corresponds to a k-center problem. It is not scale invariant since the definition is based on distances. [29] develops an I/O optimal algorithm named branch-and-bound skyline (BBS) based on nearest-neighbor search. [25] studies the problem of discovering attribute importance in skylines. The skyline definition has been extended with different variants. Chan et al. study skylines on high dimensional space, propose k-dominant skylines [7] base on a relax dominance condition, and top-k skylines [8] ranked by the frequency of being skylines in subspace. [11] proposes restricted skyline queries, where the dominance is defined base on a set of scoring functions of interest. The G-skyline is proposed in [23] based on group dominance.

In the following, we assume that we are given a database D of data points in a d-dimensional space. k-regret querying is introduced in [27] and shown to be NP-complete in [1]. For two dimensional databases, an O(|D|log|D|) time exact algorithm is given in [6]. The cube algorithm in [27] first selects the point with maximum value in each attribute, except for the last attribute. Then, it divides every dimension, except for the last dimension, into t = [(k − d − 1)! + 1] equal-sized intervals. The data space will be partitioned into buckets, and the tuple with the highest value in the last attribute in each bucket is selected. The result returned has is an approximation algorithm that guarantees a maximum regret ratio of at most d−1 d+1. The Cube algorithm is also used in [20] for nonlinear utility functions. [32] considers multiplicative utility functions, and improves Cube by adaptive partitioning and a heuristic to lower the maximum regret ratio.

The greedy algorithm in [27] first selects the tuple with the highest value on the first attribute, then iteratively
selects a remaining tuple that contributes most to the regret ratio. It performs very well when \( k \) is large in experiments, but there is no theoretical guarantee on the approximation. The greedy idea is extended to solve the \( r \)-Regret Minimizing Sets (rRMS) in [10], which changes the definition of regret ratio to be the loss over the top-\( r \) algorithm that we use, with is a bound relative to the exact top-1 utility. They show that IRMS is NP-hard.

A GeoGreedy algorithm is proposed in [30], which introduces a concept called critical ratio. This technique does not apply to negative data, since critical ratio depends on the origin and boundary points on the axes.

[1] pointed out that an \( \varepsilon \)-kernel coreset of \( D \) is a set with regret ratio at most \( \varepsilon \), where an \( \varepsilon \)-kernel coreset is a subset of \( D \) which intuitively preserves the “width” of \( D \) in each dimension. The proof is based on the assumption that \( D \) is non-negative. An \( \varepsilon \)-kernel based algorithm is proposed in [6]. However, this algorithm cannot support negative data.

A related problem is studied in [1], which is to return a tuple set with minimal size such that the regret ratio is smaller than a given value \( \varepsilon \). They propose to sample the utility functions to get a set \( U \) so that for each \( u \) in the function class \( F \), there exists \( u' \in U \) where the angle between \( u \) and \( u' \) is smaller than \( \delta \). Finally, they apply a hitting set algorithm to obtain the solution. Their theoretical analysis assumes a non-negative database.

The idea of utility function sampling in [2] is based on the partitioning of the space into hyper-cones with similar angles on each hyperplane. Their theoretical guarantee also depends on a non-negative database.

The most recent work on \( k \)-regret querying is reported in [38]. They propose an efficient algorithm called Sphere, which can return good quality results for large \( k \) values. Sphere is asymptotically optimal in the maximum regret ratio. However, the optimality is with respect to the worst possible database instance but not the given database instance, so the bound is quite loose. A major drawback with Sphere is that it can only handle cases of \( k \geq d \). Moreover, it cannot handle negative datasets. Another problem is that this work assumes a non-negative database.

An equivalent problem of regret minimizing representational databases, named Happiness-Maximizing representational databases, is studied in [33]. It proposes mechanisms to speed up the greedy algorithm in [27]. However, it does not improve the result quality which is quite poor compared to other algorithms. In [18], the target function takes into account both diversity and regret ratio. [35] considers multi-objective submodular functions; and [31] proposes \( k \)-Hit Query based on the knowledge of the distribution of utility functions.

**9 Conclusion**

We propose a new problem of \( k \)-pleased querying, which serves similar purposes as \( k \)-regret querying, but without the limitation of assuming non-negative data, while avoiding anomalies of shift-invariance. We propose a new random utility function sampling technique which can return much better results than the best known algorithms. We derive a bound on the result quality for the sampling based algorithm that we use, with is a bound relative to the exact solution of the given dataset. For future work, a possible extension of our work is to handle dynamic updates on the dataset.

**APPENDIX A**

**Proof of Lemma 3 for Dimension Reduction**

Denote \( D, f, S, \) and \( L \) regardless of the \( i \)-th dimension by \( D(i), f(i), S(i), \) and \( L(i) \), respectively. Denote \( D, f, S, \) and \( L \) regardless of the dimensions in set \( I \) by \( D(I), f(I), S(I), \) and \( L(I) \), respectively. Denote the minimal and maximal values of the \( i \)-th dimension by \( \min_i \) and \( \max_i \), respectively. Denote \( D(S) \) replaced by the maximal (minimal) value in the \( i \)-th dimension by \( D^{(\max)}(S^{(\min)}) \) respectively.

**Lemma 14.** For any \( S \subset D \), we have \( \text{ple}_D(S, L) \geq \text{ple}_D(S, L(I)) \geq \min \{ \text{ple}_D(S, L(I)), \min \frac{\min_i}{\max_i} \} \).

**Proof:**

\[
\text{ple}_D(S, L) = \inf_{f \in L} \frac{\sum_{i \in S} f(i)}{\sum_{i \in D} f(i)} = \inf_{f \in L} \frac{\sum_{i \in S(I)} f(i)}{\sum_{i \in D(I)} f(i)} \geq \inf_{f \in L} \frac{\sum_{i \in S(I)} f(i)}{\sum_{i \in D(I)} f(i)} = \text{ple}_D(S, L(I)) \\
\text{ple}_D(S, L) \geq \inf_{f \in L} \frac{\sum_{i \in S(I)} f(i)}{\sum_{i \in D(I)} f(i)} = \text{ple}_D(S(I), L(I)) \geq \min \{ \text{ple}_D(S(I), L(I)), \min \frac{\min_i}{\max_i} \} \]

The deduction from step (1) to (2) in based on the fact that \( \frac{\sum_{i \in S(I)} f(i)}{\sum_{i \in D(I)} f(i)} \geq \min \frac{\min_i}{\max_i} \) always holds if \( A > 0 \) and \( B > 0 \). \( f(i)(D(i)) \) and \( \max_i \) are also positive because from Assumption 1 : Let \( g = f \) except \( g[i] = 0 \), then \( f(i)(D(i)) = g(D) > 0 \); Let \( g = 0 \) except \( g[i] = 1 \), then \( \max_i = g(D) > 0 \).

By recursively applying Lemma 14 we can get the following lemma for reducing a set of dimensions \( I \).

**Corollary 3.** For any \( S \subset D \), and a set of dimensions \( I \) to be eliminated, \( \text{ple}_D(S(I), L(I)) \geq \text{ple}_D(S, L) \geq \min \{ \text{ple}_D(S(I), L(I)), \min \frac{\min_i}{\max_i} \} \).

Thus, if \( \min \frac{\min_i}{\max_i} \geq \text{ple}_D(S(I), L(I)) \), we can see that \( \text{ple}_D(S(I), L(I)) = \text{ple}_D(S, L) \), which means that the optimal solution after reducing the dimensions in \( I \) is also the optimal solution in the original dataset.

An immediate deduction is Lemma 3:

**Lemma 3:** If \( \forall S \subset D, |S| = k, \text{ple}_D(S, L) < c \) then \( \text{ple}_D(S, L) = \text{ple}_D(S(I), L(I)) \).

**Proof:** From Corollary 3, \( \text{ple}_D(S, L) \leq \text{ple}_D(S(I), L(I)) \). We need to prove \( \text{ple}_D(S, L) \geq \text{ple}_D(S(I), L(I)) \). We have \( \min \frac{\min_i}{\max_i} \geq c \), according to the definition of \( I \). If \( \min \{ \text{ple}_D(S(I), L(I)), c \} = c \), then by Corollary 3,

\[
\text{ple}_D(S, L) \geq \min \{ \text{ple}_D(S(I), L(I)), \min \frac{\min_i}{\max_i} \} \geq \min \{ \text{ple}_D(S(I), L(I)), c \} = c
\]

which is contradictory to \( \text{ple}_D(S, L) < c \).

Thus, \( \min \{ \text{ple}_D(S(I), L(I)), c \} = \text{ple}_D(S(I), L(I)) \), and \( \text{ple}_D(S, L) \geq \text{ple}_D(S(I), L(I)) \). \( \square \)
ACKNOWLEDGMENTS
This work is supported by the RGC-funded project CUHK412313 and a Direct Grant 2017-18 of the Engineering Faculty of CUHK.

REFERENCES

Zitong Chen Zitong Chen is currently an Assistant Professor at the School of Artificial Intelligence, Sun Yat-Sen University, China. He received the Ph.D degree from the Department of Computer Science and Engineering, the Chinese University of Hong Kong in 2021. His research interests include database, data mining, and machine learning.

Ada Wai-Chee Fu Ada Wai-Chee Fu received her B.Sc degree in computer science in the Chinese University of Hong Kong in 1983, and both M.Sc and Ph.D degrees in Computer Science in Simon Fraser University of Canada in 1986, 1990, respectively; worked at Bell Northern Research in Ottawa, Canada from 1989 to 1993; joined the Chinese University of Hong Kong in 1993.

Cheng Long Cheng Long is currently an Assistant Professor at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU). From 2016 to 2018, he has worked as a lecturer (Assistant Professor) at Queen’s University Belfast, UK. He received the Ph.D degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST) in 2015. His research interests are broadly in data management, data mining and big data analytics. He is a member of IEEE.

Yang Wu Yang Wu is currently working towards the Ph.D degree in the Department of Computer Science and Engineering, the Chinese University of Hong Kong. His research interests include data mining and graph database.