Being Happy with the Least: Achieving α-happiness with Minimum Number of Tuples

Min Xie[1,2], Raymond Chi-Wing Wong[1], Peng Peng[3], Vassilis J. Tsotras[4]
[1]*The Hong Kong University of Science and Technology* [2]*Shenzhen Institute of Computing Sciences, Shenzhen University* [3]*inspir.ai* [4]*UC Riverside*
{mxieaa, raywong}@cse.ust.hk pp@inspirai.com tsotras@cs.ucr.edu

***Abstract*—When faced with a database containing millions of products, a user may be only interested in a (typically much) smaller representative subset. Various approaches were proposed to create a good representative subset that fits the user's needs which are expressed in the form of a *utility* function (e.g., the top-k and diversification query). Recently, a *regret minimization* query was proposed: it does not require users to provide their utility functions and returns a small set of tuples such that any user's favorite tuple in this subset is guaranteed to be not much worse than his/her favorite tuple in the whole database. In a sense, this query finds a small set of tuples that makes the user *happy* (i.e., not *regretful*) even if s/he gets the best tuple in the selected set but not the best tuple among all tuples in the database.**

In this paper, we study the min-size version of the regret minimization query; that is, we want to determine the least tuples needed to keep users happy at a given level. We term this problem as the α-*happiness query* where we quantify the user's happiness level by a criterion, called the *happiness ratio*, and guarantee that each user is at least α happy with the set returned (i.e., the happiness ratio is at least α) where α is a real number from 0 to 1. As this is an NP-hard problem, we derive an approximate solution with theoretical guarantee by considering the problem from a geometric perspective. Since in practical scenarios, users are interested in achieving higher happiness levels (i.e., α is closer to 1), we performed extensive experiments for these scenarios, using both real and synthetic datasets. Our evaluations show that our algorithm outperforms the best-known previous approaches in two ways: (i) it answers the α-happiness query by returning fewer tuples to users and, (ii) it answers much faster (up to two orders of magnitude times improvement for large α).

I. Introduction

A database system usually contains millions of tuples and an end user may be interested in only some of them. In order to assist a user's decision making, we need queries that obtain a small representative subset of tuples from a large database instead of asking the user to scan the whole database. Such queries can be considered as *multi-criteria decision making* problems [7], [20], [21]. An example is the traditional top-k query [20], [21], where a user provides her preference function, called the *utility function*, and an integer k (i.e., the output size). Based on the user's utility function, the *utility* of each tuple for this user can be computed. A high utility indicates that the corresponding tuple is preferred by the user. The output of a top-k query is the k tuples with the highest utilities. If the user's utility function is not known, the skyline query can be applied [7]; instead of asking for a utility function, it uses the "dominance" concept. A tuple p is said to dominate another tuple q if p is not worse than q on each attribute and p is better than q on at least one attribute. Intuitively, p will have a higher utility than q w.r.t. *all monotonic* utility functions. Tuples which are not dominated by any other tuples are returned in the skyline query. Unfortunately, the output size of a skyline query can be large (at worst the whole database). Motivated by this, a novel query called the *regret minimization query* [15] was proposed recently to overcome the deficiencies of both the top-k query (which requires the users to specify their utility functions) and the skyline query (which might have a large output size).

Informally, a regret minimization query computes a small set of tuples that makes the users *happy* (some papers use the term *not regretful*) without asking the users for their utility functions. The *happiness level* of each user is quantified as the *happiness ratio* of the user. Specifically, given a set of tuples, a user is $x\%$ happy with the set if the highest utility of tuples in the set is at least $x\%$ of the highest utility of all tuples in the whole database. In this case, the happiness ratio of the user is $x\%$. Clearly, the happiness ratio is a value from 0 to 1. The larger the happiness ratio, the happier the user. Two versions of regret minimization queries were studied in the literature: (1) the *min-error* version (also known as the k-*regret* query) [15]: it maximizes the happiness level (i.e., minimizes the regret level) of each user while guaranteeing the output size is at most k; (2) the *min-size* version [3], [13]: it minimizes the output size while guaranteeing the happiness ratio of each user is at least α. Depending on the search needs of users, different versions of regret minimization queries can be applied. In this paper, we focus on the min-size version of regret minimization, which we term as the α-*happiness* query.

Consider the following car database application. Assume that Alice attempts to buy a car from a car database where each car is represented by two attributes, namely horse power (HP) and miles per gallon (MPG). To assist Alice for making the decision, she should be provided with cars that she is potentially interested in (e.g., cars with high utilities if we know Alice's utility function). However, in practice, it is difficult for Alice to provide her utility function explicitly (hence, we cannot use a top-k query). Alternatively, we can approach this problem as an α-happiness query: Alice specifies an α value (which is easier for her to do), which represents the least happiness level she expects. In practice, Alice can set α to be at least 0.9 [13], which means that she wants a set of tuples whose highest utility is at least 90% of the highest

utility of all tuples in the database. Then, an α-happiness query returns a set of tuples from the database, with size as small as possible, so that Alice will be satisfied with the returned set (since the happiness ratio is at least α as specified by Alice).

In practical scenarios, users are interested in achieving high happiness in α-happiness queries. For example, when buying a car/house, which is one of the big purchases in our life, a user may want to find a car/house which is as close to his/her ideal one as possible by only examining a few options. Otherwise, the user might feel regretful (i.e., not happy) for not buying a better one for a long time. The best-known method for the α-happiness query is CORESETHS [13]. Unfortunately, when we experimentally evaluated CORESETHS for large α (e.g., α approaches 1), it experienced a long query time. This is because the core of CORESETHS relies on sampling a large number of utility functions to guarantee high happiness: the larger the happiness level a user requires, the more utility functions have to be sampled and the more time is needed to construct the solution. In particular, its running time is proportional to $\frac{1}{(1-\alpha)^{O(d)}}$, which becomes prohibitive when α is close 1 (even for a small number of attributes d). Different from CORESETHS, we propose a novel method, CONE-GREEDY, which removes the dependence on $\frac{1}{(1-\alpha)^{O(d)}}$. In particular, our experimental evaluation showcased that when users require high happiness, our method's running time decreases.

Note that a common characteristic of all approaches solving the α-happiness query is that the output size increases when α becomes larger. The output size represents the effort that a user needs to spend to make a decision (since a user has to examine the output to find the products s/he is interested in). A further advantage of our method is that it consistently returned solutions of smaller sizes empirically among all competitor algorithms, which effectively makes the user's decision easier. Our major contributions are summarized as follows:

- We provide a novel geometric interpretation of the α-happiness query, which has not been considered before.
- We propose a novel algorithm to answer the α-happiness query. Our algorithm enjoys a bounded output size and is not sensitive to large α in the way previous studies have.
- We present extensive experiments on both synthetic and real datasets, demonstrating our superiority. Under some practical settings, our method returns 30$\sim$80% fewer tuples than existing ones (e.g., existing ones can output more than 500 tuples, which is too large) while achieving up to two orders of improvement in running time (e.g., they might take half an hour while we finish in seconds).

Organization. The α-happiness query is formally defined in Section II. In Section III, we interpret the problem in a geometric way and provide an overview of our solution. The algorithm is described in Section IV and related works are discussed in Section V. Finally, the experimental evaluation is presented in Section VI while conclusions appear in Section VII.

II. PROBLEM DEFINITION

We are given a set D of n tuples (i.e., $|D| = n$) in a d-dimensional space (i.e., each tuple in D is described by d attributes). We assume that d is a fixed constant in this paper. The i-th dimensional value of a tuple p is denoted by $p[i]$ where $i \in [1, d]$. The norm of p (the L2-norm) is denoted by $\|p\|$. In the rest paper, we also call each tuple as a point in a d-dimensional space. Without loss of generality, we assume that each dimension is normalized to $(0, 1]$, such that there exists a point p in D and $p[i] = 1$ for each $i \in [1, d]$ and we assume that a larger value on each dimension is more preferable to all users. Recall that in the car database, each car is associated with 2 attributes, namely HP and MPG; in the example shown in Table I, the car database $D = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ consists of 6 car tuples with normalized attribute values.

TABLE I
CAR DATABASE AND UTILITIES

Car	HP	MPG	$f_{0.4,0.6}(p)$	$f_{0.2,0.8}(p)$	$f_{0.7,0.3}(p)$
p_1	0.2	1	0.68	0.84	0.44
p_2	0.6	0.9	0.78	0.84	0.69
p_3	0.9	0.6	0.72	0.66	0.81
p_4	1	0.2	0.52	0.36	0.76
p_5	0.35	0.2	0.26	0.23	0.305
p_6	0.3	0.6	0.48	0.54	0.39

Assume that user's happiness is measured by an unknown *utility function*, which is a mapping $f : \mathbb{R}_+^d \to \mathbb{R}_+$. Denote the *utility* of a point p w.r.t. f by $f(p)$. A user wants a point which maximizes the utility w.r.t. his/her utility function. Given a utility function f and $S \subseteq D$, we define the *maximum utility of S w.r.t. f*, denoted by $U_{max}(S, f)$, to be $\max_{p \in S} f(p)$. Clearly, for each $S \subseteq D$, $U_{max}(S, f) \leq U_{max}(D, f)$.

We define two important terms, namely the *function-wise ratio* (*happiness ratio*) and the *minimum happiness ratio*.

Definition 1 (Function-wise Ratio): Given a set $S \subseteq D$ and a utility function f, the function-wise ratio of S w.r.t. f, denoted by $\mathsf{fRatio}(S, f)$, is defined to be $\frac{U_{max}(S,f)}{U_{max}(D,f)}$.

The value of a function-wise ratio ranges from 0 to 1. Intuitively, when the maximum utility of S is closer to the maximum utility of D, the function-wise ratio of S w.r.t. the user's utility function becomes larger, which indicates that the user feels more satisfied with S. Thus, the function-wise ratio is also called the *happiness ratio*. Unfortunately, in reality, it is difficult to obtain the user's exact utility function. Thus, in this paper, we assume that all users' utility functions belong to a function class, denoted by $\mathcal{FC}$. A function class is defined to be a set of functions which share some common characteristics. An example is the class of *linear utility functions* [15] (to be defined shortly). The *minimum happiness ratio* of a set S is defined over a function class $\mathcal{FC}$, which can be regarded as the worst-case function-wise ratio w.r.t. a function in $\mathcal{FC}$.

Definition 2 (Minimum Happiness Ratio): Given a set $S \subseteq D$ and a function class $\mathcal{FC}$, the minimum happiness ratio of S over $\mathcal{FC}$ is defined to be $\inf_{f \in \mathcal{FC}} \mathsf{fRatio}(S, f)$.

Example 1: Assume that $\mathcal{FC}$ consists of 3 utility functions, namely $f_{0.4,0.6}$, $f_{0.2,0.8}$ and $f_{0.7,0.3}$ where $f_{a,b}(p) = a \times p[1] + b \times p[2]$. Consider p_1 in Table I. Its utility w.r.t. $f_{0.4,0.6}$ is $f_{0.4,0.6}(p_1) = 0.4 \times 0.2 + 0.6 \times 1 = 0.68$. The utilities of other points in D are computed similarly in Table I. Given $S = \{p_1, p_4\}$, the maximum utility of S

w.r.t. $f_{0.4,0.6}$ *is* $U_{\max}(S, f_{0.4,0.6}) = \max_{p\in S} f_{0.4,0.6}(p) = f_{0.4,0.6}(p_1) = 0.68$. *Similarly,* $U_{\max}(D, f_{0.4,0.6})$ *is 0.78. Then,* $\mathsf{fRatio}(S, f_{0.4,0.6}) = \frac{U_{\max}(S,f_{0.4,0.6})}{U_{\max}(D,f_{0.4,0.6})} = \frac{0.68}{0.78} = 0.872$. *Similarly,* $\mathsf{fRatio}(S, f_{0.2,0.8}) = 1$ *and* $\mathsf{fRatio}(S, f_{0.7,0.3}) = 0.938$. *The minimum happiness ratio of* S *over* $\mathcal{FC}$ *is* $\inf_{f\in\mathcal{FC}} \mathsf{fRatio}(S, f) = \min\{0.872, 1, 0.938\} = 0.872$. □

In general, the utility functions in $\mathcal{FC}$ could have an arbitrary distribution. For the ease of presentation, we first assume that each utility function in $\mathcal{FC}$ is equally probable to be used by a user. However, we will later relax this assumption in Section IV-C2 and show that our techniques can be easily applied when arbitrary types of distributions are considered.

As in [3], [13]–[15], here, we assume that $\mathcal{FC}$ is the class of *linear utility functions* due to its popularity in modeling user preferences. Other classes of utility functions are considered in [11], [19] and are not the focus of this paper. Specifically, each linear utility function f in $\mathcal{FC}$ is associated with a *utility vector* u which is a d-dimensional non-negative real vector where $u[i]$ denotes the importance of the i-th dimension in user's preference and it can be expressed as: $f(p) = \sum_{i=1}^{d} u[i]p[i] = u\cdot p$. Without loss of generality, we assume that $\|u\| = 1$. Thus, we have $\mathcal{FC} = \{f|\ f(p) = u\cdot p$ where $u \in \mathbb{R}^d_+$ and $\|u\| = 1\}$. In the rest paper, we also refer each function f in $\mathcal{FC}$ by its utility vector u. Denote the minimum happiness ratio of S over the class of linear utility functions $\mathcal{FC}$ by $\mathsf{minHap}(S)$.

Since $\mathcal{FC}$ contains an *infinite* number of linear utility functions, it is not easy to compute $\mathsf{minHap}(S)$ for a given S directly according to Definition 2. Instead, [16] showed that the minimum happiness ratio $\mathsf{minHap}(S)$ can be computed in a tractable way using the *"point-wise ratio"*, pRatio, defined below. In Section III-A, we will use pRatio to interpret the α-happiness query from a novel geometric perspective.

For each point $p \in D$, we define the *orthotope set* of p, denoted by $\mathsf{Orth}(p)$, to be a set of 2^d d-dimensional points constructed by $\{0, p[1]\}\times\{0, p[2]\}\times ...\times\{0, p[d]\}$. That is, for each $i \in [1, d]$, the i-dimensional value of a point in $\mathsf{Orth}(p)$ is equal to either 0 or $p[i]$. Given a set $S \subseteq D$, we define the orthotope set of S, denoted by $\mathsf{Orth}(S)$, to be $\bigcup_{p\in S} \mathsf{Orth}(p)$. Given a set $S \subseteq D$, let $\mathsf{Conv}(S)$ be the *convex hull* (the smallest convex set) of the orthotope set of S [16]. Moreover, a point $p \in \mathsf{Conv}(S)$ is said to be a *vertex* of $\mathsf{Conv}(S)$ if p is not in the convex hull of the other points in $\mathsf{Orth}(S)$.

Example 2: Consider the example in Table I where $D = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. *For the ease of presentation, we visualize* D *in Figure 1 where the* X_1 *and* X_2 *coordinate represent HP and MPG, respectively. The points in* $\mathsf{Orth}(p_2)(= \{p_2, p_2', p_2'', O\})$ *are shown in Figure 1 where* $p_2' = (0, p_2[2])$, $p_2'' = (p_2[1], 0)$ *and* O *is the origin. Similarly,* $\mathsf{Orth}(p_3)$ *is shown in the same figure. Given* $S = \{p_2, p_3\}$, *we define* $\mathsf{Orth}(S)$ *to be* $\mathsf{Orth}(p_2) \cup \mathsf{Orth}(p_3)$. *Then, the convex hull* $\mathsf{Conv}(S)$ *is shown in Figure 2. Note that there are 5 vertices in* $\mathsf{Conv}(S)$, *namely* O, p_2', p_2, p_3 *and* p_3'', *each of which is not in the convex hull of the other points in* $\mathsf{Orth}(S)$. □

Definition 3 (Point-wise Ratio): Given a set S of points in D and a point $p \in D$, the point-wise ratio of p w.r.t. S, denoted by $\mathsf{pRatio}(S, p)$, is defined to be $\min\{\frac{\|p'\|}{\|p\|}, 1\}$, where p' is the intersection between the ray Op, which starts from the origin O and passes p, and the surface of $\mathsf{Conv}(S)$.

TABLE II
FREQUENTLY USED NOTATIONS

Notation	Meaning
D	The set of d-dimensional points with $\|D\| = n$
D'	The α-shrunk set of D
$U_{\max}(S, f)$	The maximum utility of S w.r.t. f
$\mathsf{fRatio}(S, f)$	The function-wise ratio of S w.r.t. f
$\mathsf{pRatio}(S, p)$	The point-wise ratio of p w.r.t. S
$\mathcal{FC}$	The class of linear utility functions
$\mathsf{minHap}(S)$	The minimum happiness ratio of S over $\mathcal{FC}$
$\mathsf{Conv}(S)$	The convex hull of the orthotope set of S
$\mathsf{Cone}(V, o)$	The conical hull of o w.r.t. V, i.e., $\mathsf{Cone}(V, o) = \{p \in \mathbb{R}^d \mid p = o + \sum_{v\in V} wv$ where $w \geq 0\}$
$\mathcal{F}_p$	A set of utility functions whose utility score is maximized by p over points in D'
V_p	$V_p = \{p' - p\|$ for each vertex p' of $\mathsf{Conv}(D')\}$
$\mathsf{Ext}(p)$	The set of extreme vectors of p
$\mathbb{S}$	The surface of a unit sphere in $\mathbb{R}^d_+$
C_p	A partial spherical surface of $\mathbb{S}$, $\mathsf{Cone}(\mathsf{Ext}(p), O) \cap \mathbb{S}$ (also called the conical hull of p if $\mathbb{S}$ is clear)
$\mathsf{Vol}(p)(\widetilde{\mathsf{Vol}}(p))$	The (estimated) volume of the conical hull C_p
$\varrho_p(S)(\tilde{\varrho}_p(S))$	The (estimated) marginal volume of p
S_t	The set of points selected after the t-th round
p_t	The point selected in the t-th round
θ_t $(\tilde{\theta}_t)$	$\min_{p\in D\setminus S_{t-1}} \frac{1}{\varrho_p(S_{t-1})}$ $(\min_{p\in D\setminus S_{t-1}} \frac{1}{\tilde{\varrho}_p(S_{t-1})})$
N	The sampling size
h_p	The hyperplane of p defined based on $\mathsf{Ext}(p)$

Lemma 1 (Computing Minimum Happiness [16]): Given a set $S \subseteq D$, we have $\mathsf{minHap}(S) = \min_{p\in D} \mathsf{pRatio}(S, p)$.

Example 3: Given p_1 *and* $S = \{p_2, p_3\}$ *in Figure 3, the intersection between* Op_1 *and the surface of* $\mathsf{Conv}(S)$ *is denoted by* p_1'. $\mathsf{pRatio}(S, p_1)$ *is computed to be* $\frac{\|p_1'\|}{\|p_1\|} = 0.9$. *Similarly, we can compute* $\mathsf{pRatio}(S, p_4)$ *to be 0.9 and the point-wise ratios of the remaining points in D are 1. According to Lemma 1,* $\mathsf{minHap}(S) = \min_{p\in D} \mathsf{pRatio}(S, p) = 0.9$. □

After knowing how to compute the minimum happiness ratio $\mathsf{minHap}(\cdot)$, we formally define the α-happiness query.

Problem 1 (The α-Happiness Query): Given a real number $\alpha \in [0, 1]$, the α-happiness query returns a set $S \subseteq D$ with $\mathsf{minHap}(S) \geq \alpha$ such that the size of S, i.e., $|S|$, is minimized.

If there are multiple sets with the minimum size, an α-happiness query returns one of them. Note that a user does not need to specify his/her utility function in the α-happiness query. Since $\mathsf{minHap}(S)$ is defined to be the worst-case happiness ratio w.r.t. *any* utility function in $\mathcal{FC}$, if $\mathsf{minHap}(S) \geq \alpha$ for a given set S, for *each* user, s/he will be at least α happy with S no matter which function s/he uses from $\mathcal{FC}$. Unfortunately, according to the existing results in [3], [8], [9], it is NP-hard to solve an α-happiness query optimally. Table II summarizes the frequently used notations in this paper.

III. GEOMETRIC PROPERTIES

We introduce a few important geometric properties used in our solution. In Section III-A, we introduce the spatial α-coverage problem, which is geometrically equivalent to the

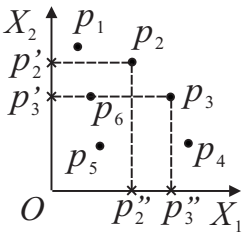


Fig. 1. Orthotope set

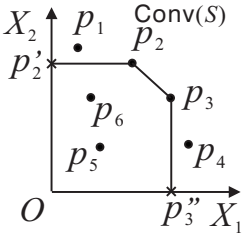


Fig. 2. Convex hull

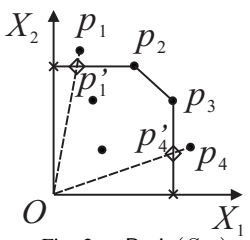


Fig. 3. pRatio(S, p)

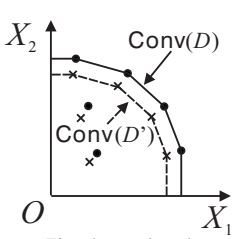


Fig. 4. α-shrunk set

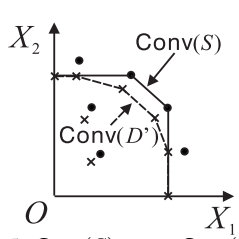


Fig. 5. Conv(S) covers Conv(D')

α-happiness query. Then, we present our solution overview for solving the spatial α-coverage problem in Section III-B.

A. Equivalence to Spatial Coverage

Given a real number $\alpha \in [0, 1]$, we define the *α-shrunk set* of D, denoted by D'_α , to be $\{p'_\alpha | p'_\alpha = \alpha p, \forall p \in D\}$ where p'_α is the *α-shrunk point* of p. When α is clear in the context, we denote D'_α by D' and denote a point in D' by p'. Intuitively, D' is a proportionally shrunk set of D. For example, let $\alpha = 0.9$ and D is shown in Table I. The α-shrunk set D' (shown in cross points) of D (shown in dot points) is drawn in Figure 4 where each point in D' is a proportionally scaled point in D. Similarly, Conv(D') can be regarded as a proportionally shrunk convex hull of Conv(D) as shown in Figure 4.

We formally define the *spatial α-coverage problem*, which provides us a novel interpretation of the α-happiness query.

Problem 2 (The Spatial α-Coverage Problem): Given $\alpha \in [0, 1]$, the spatial α-coverage problem finds a minimum size subset of D, denoted by S, such that for each point $p' \in D'$, p' is inside Conv(S) where D' is the α-shrunk set of D.

We say that Conv(S) *covers* Conv(D') if the above condition is satisfied since Conv(D') is "contained" inside Conv(S). For example, in Figure 5 where $S = \{p_2, p_3\}$, Conv(S) covers Conv(D'), i.e., for each p' in D', p' is inside Conv(S).

We show our first interesting result that the α-happiness query and the spatial α-coverage problem are equivalent in Theorem 1. For lack of space, the proofs of Theorems/Lemmas in this paper can be found in our technical report [27].

Theorem 1: Given $S \subseteq D$ and $\alpha \in [0, 1]$, S is a feasible solution of the spatial α-coverage problem on D if and only if S is a feasible solution of the α-happiness query on D.

Example 4: Given $\alpha = 0.9$ and $S = \{p_2, p_3\}$, S is a feasible solution of the spatial α-coverage problem on D since Conv(S) *covers* Conv(D') *as shown in Figure 5. According to Theorem 1, S is also a feasible solution of the α-happiness query on D, which conforms with our previous computation in Example 3 where* minHap$(S) = 0.9 \geq \alpha$. □

Theorem 1 provides a straightforward way for us to interpret the α-happiness query. The geometric explanation of the α-happiness query is obviously more intuitive than its original algebraic explanation. Since the α-happiness query is an NP-hard problem, it is also computationally expensive to find an optimal solution for the spatial α-coverage problem. In the following subsection, we show a different view of the spatial α-coverage problem, which helps us to determine a solution efficiently. Based on this result, we will develop our algorithm in Section IV for answering the α-happiness query.

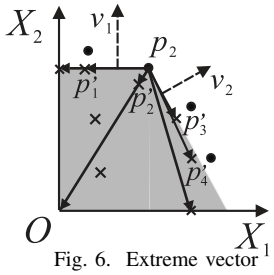


Fig. 6. Extreme vector

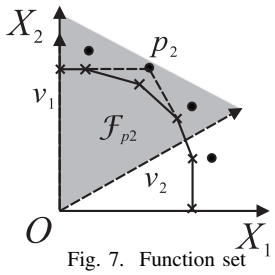


Fig. 7. Function set

B. Interesting Properties

Before we go through the details, we first introduce an overview of our solution to the spatial α-coverage problem.

Consider a point p in D. Let $\mathcal{F}_p$ be a set of utility functions such that for each $f \in \mathcal{F}_p$, $f(p) \geq \max_{p' \in D'} f(p')$ where D' is the α-shrunk set of D. Intuitively, $\mathcal{F}_p$ is a set of utility functions whose utilities are maximized by p over the points in D' (or simply, the utilities are maximized by p and p has the maximum utility if D' is clear in the context). Before showing the formal procedure for obtaining such $\mathcal{F}_p$, we first show how we use $\mathcal{F}_p$ to solve a spatial α-coverage problem effectively (proven in our technical report [27] due to the lack of space).

Lemma 2: Given $\alpha \in [0, 1]$, the function set $\mathcal{F}_p$ for each point $p \in D$ and a set $S \subseteq D$, if $\bigcup_{p \in S} \mathcal{F}_p = \mathcal{FC}$, Conv$(S)$ covers Conv(D') where D' is the α-shrunk set of D.

According to Theorem 1 and Lemma 2, a set S is a valid solution for the α-happiness query if $\bigcup_{p \in S} \mathcal{F}_p = \mathcal{FC}$.

Intuitively, given a point p in D, the (uncountable) number of utility functions in $\mathcal{F}_p$ can be regarded as the *importance* of p. A point with a higher importance indicates that this point has the maximum utility w.r.t. more utility functions. Based on this observation, our algorithm has two major steps. Firstly, we compute the function set $\mathcal{F}_p$ for each p in D and quantify its importance. Secondly, we leverage a greedy algorithm to select points to S according to the importance obtained in the first step until $\bigcup_{p \in S} \mathcal{F}_p = \mathcal{FC}$. In the following, we first present the formal definition of $\mathcal{F}_p$ and then provide the procedure for computing the desired $\mathcal{F}_p$. The quantification of its importance and the greedy strategy will be presented later in Section IV.

1) Preliminary Concepts: Let D be the given dataset and D' be its corresponding α-shrunk set. We first introduce a few useful notations and geometric concepts for defining $\mathcal{F}_p$.

Given a point p in D, let $V_p = \{p' - p |$ for each vertex p' of Conv$(D')\}$. For example, given a point p_2 in Figure 6, the vector set V_{p_2}, which is constructed by creating a vector for each vertex of Conv(D'), is shown in solid vectors.

Given a vector set V and a point o, we define the *conical hull of o w.r.t. V*, denoted by $\mathsf{Cone}(V, o)$, be the set of all vectors which are centered at o and are the *conical combination* [12] of vectors in V, i.e., $\mathsf{Cone}(V, o) = \{p \in \mathbb{R}^d |\ p = o + \sum_{v \in V} wv$ where $w \geq 0\}$. Intuitively, $\mathsf{Cone}(V, o)$ can be regarded as a *convex cone* with apex o. The boundaries of $\mathsf{Cone}(V, o)$ are some *unbounded facets*, each of which is enclosed by some vectors in V and is a flat surface that forms a part of the boundary of $\mathsf{Cone}(V, o)$. Given the previously defined vector set V_p, we can define a special conical hull $\mathsf{Cone}(V_p, p)$ for each p in D. A concrete example is as follows.

Example 5: Consider p_2 and the corresponding V_{p_2} in Figure 6. We draw $\mathsf{Cone}(V_{p_2}, p_2)$ in the shaded region in the figure, which is the set of all vectors with the form $p_2 + \sum_{v \in V_{p_2}} wv$ where $w \geq 0$. In this 2-dimensional example, the boundaries of $\mathsf{Cone}(V_{p_2}, p_2)$ are two unbounded facets, i.e., the rays shooting from p_2 to p_1' and from p_2 to p_3'. □

Intuitively, $\mathsf{Cone}(V_p, p)$ can be regarded as a conical hull that constrains the *maximum visible range* from p to $\mathsf{Conv}(D')$. For example, in Figure 6, $\mathsf{Cone}(V_{p_2}, p_2)$ constrains the maximum visible range from p_2 to $\mathsf{Conv}(D')$ since along any other direction (ray), p_2 cannot "see"(reach) any point in $\mathsf{Conv}(D')$ (also see the example in Figure 7 where $\mathsf{Conv}(D')$ is shown and we draw the boundary of the maximum visible range from p_2 to $\mathsf{Conv}(D')$ in two dashed lines passing through p_2).

Consider a boundary facet F of conical hull $\mathsf{Cone}(V_p, p)$ defined above. Facet F is said to be contained by a hyperplane if (1) for each point q on F, q is also on this hyperplane and (2) for each q in $\mathsf{Cone}(V_p, p)$ but not on F, q is below the hyperplane. In geometry, each boundary facet of a conical hull is contained by a *unique* hyperplane. Then, for each boundary facet F of $\mathsf{Cone}(V_p, p)$, we define an *extreme vector* of p to be the unit vector perpendicular to the hyperplane containing F. Denote the set of all *extreme vectors* of p by $\mathsf{Ext}(p)$.

Example 6: Consider conical hull $\mathsf{Cone}(V_{p_2}, p_2)$ in Example 5. The ray shooting from p_2 to p_1' is a boundary facet of $\mathsf{Cone}(V_{p_2}, p_2)$, which is contained by the hyperplane (i.e., a line in this 2-dimensional example) passing through p_1' and p_2. Since v_1 is a vector perpendicular to this hyperplane, v_1 is an extreme vector of p_2. Another extreme vector, namely v_2, is obtained similarly. Thus, we have $\mathsf{Ext}(p_2) = \{v_1, v_2\}$. □

2) Function Set: Based on the concepts introduced above, we are ready to formally define the function set $\mathcal{F}_p$, which is a set of utility functions whose utilities are maximized by p.

Definition 4 (Function Set): Given p in D and its extreme vectors $\mathsf{Ext}(p)$, we define the function set of p, denoted by $\mathcal{F}_p$, to be $\{f \in \mathcal{FC} |\ f(p) = u \cdot p$ and $u \in \mathsf{Cone}(\mathsf{Ext}(p), O)\}$.

Lemma 3: If $f \in \mathcal{F}_p$, $f(p) \geq f(p')$ for each $p' \in D'$.

Example 7: In Example 6, $\mathsf{Ext}(p_2)$ is computed to be $\{v_1, v_2\}$. The conical hull $\mathsf{Cone}(\mathsf{Ext}(p_2), O)$ is shown in the shaded region in Figure 7, which is the set of all vectors with the form $w_1 v_1 + w_2 v_2$ where $w_1, w_2 \geq 0$. By Definition 4, we define $\mathcal{F}_{p_2}$ to be the functions whose utility vectors are in $\mathsf{Cone}(\mathsf{Ext}(p_2), O)$. Then, according to Lemma 3, if f is a function in $\mathcal{F}_{p_2}$, we have $f(p_2) \geq f(p')$ for each $p' \in D'$.

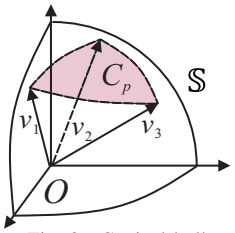


Fig. 8. Conical hull

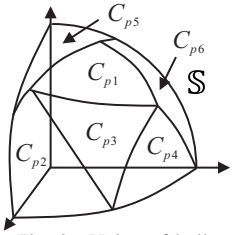


Fig. 9. Union of hulls

Figure 8 shows another example of $\mathsf{Cone}(\mathsf{Ext}(p), O)$ where $\mathsf{Ext}(p) = \{v_1, v_2, v_3\}$ for some p in a 3-dimensional space. □

Note that for a given p in D, its function set $\mathcal{F}_p$ is uniquely determined by the extreme vectors in $\mathsf{Ext}(p)$. Thus, we compute $\mathcal{F}_p$ by computing the corresponding $\mathsf{Ext}(p)$. Formally, the **procedure for computing** $\mathcal{F}_p$ is described as follows:

1) We compute the set of all vertices of $\mathsf{Conv}(D')$;
2) We define $V_p = \{p' - p|$ for each vertex p' of $\mathsf{Conv}(D')\}$ and construct the conical hull $\mathsf{Cone}(V_p, p)$;
3) We obtain the set $\mathsf{Ext}(p)$ and each unit extreme vector in it is perpendicular to a boundary facet of $\mathsf{Cone}(V_p, p)$.

Time complexity. Let m be the maximum extreme vectors of a point in D and B' be the set of all vertices of $\mathsf{Conv}(D')$. In practice, we can compute the vertex set B' in $O(n|B'|)$ time using linear programming (LP) [10]. For each p in D, we construct the vector set V_p in $O(|B'|)$ time and obtain $\mathsf{Ext}(p)$ in $O(m)$ time [5] since there are $O(m)$ boundary facets in $\mathsf{Cone}(V_p, p)$. Thus, the total time complexity of the above procedure is $O(n|B'| + nm)$. In low dimensional spaces, the Quickhull algorithm [5] can also be adapted to computing $\mathcal{F}_p$. We omit the details here and refer interested readers to [5].

IV. Algorithm

According to the solution overview described in Section III, our algorithm consists of two major steps. Firstly, we compute $\mathcal{F}_p$ (i.e., compute the extreme vectors $\mathsf{Ext}(p)$ that define $\mathcal{F}_p$) and quantify the importance of $\mathcal{F}_p$ for each p in D. Secondly, we employ a greedy algorithm based on the importance obtained in the first step and determine a set $S \subseteq D$ such that $\cup_{p \in S} \mathcal{F}_p = \mathcal{FC}$. In the following, we first show how we quantify the importance of each $\mathcal{F}_p$ in Section IV-A and then present the algorithms in Section IV-B and Section IV-C.

A. Importance of Function Set

Recall that each utility vector has its norm equal to 1. Then, we interpret that each utility vector lies on the surface of a sphere, denoted by $\mathbb{S}$, in the positive quadrant with radius 1 centered at the origin. That is, $\mathbb{S} = \{u \in \mathbb{R}^d_+ |\ \|u\| = 1\}$.

In Section III, $\mathcal{F}_p$ is defined to be a set of utility functions, whose utility vectors are in $\mathsf{Cone}(\mathsf{Ext}(p), O)$. In other words, $\mathcal{F}_p$ can be represented as a *partial spherical surface* of $\mathbb{S}$, i.e., $\mathsf{Cone}(\mathsf{Ext}(p), O) \cap \mathbb{S}$. With a slight abuse of terminology, when $\mathbb{S}$ is clear in the context, we also call the spherical surface $\mathsf{Cone}(\mathsf{Ext}(p), O) \cap \mathbb{S}$ as the *conical hull* of p, denoted by C_p. Clearly, C_p is also uniquely defined by $\mathsf{Ext}(p)$. Intuitively, if there are more vectors in C_p, p is more important since it has

the maximum utility w.r.t. more functions. However, since the number of vectors in C_p is uncountable, we use the "surface area" (i.e., the $(d-1)$-dimensional measure) to quantify the importance of a point p. Specifically, we define the *volume* of a conical hull of p (or simply, the volume of p), denoted by $\mathsf{Vol}(p)$, to be $\frac{\mathsf{Area}(C_p)}{\mathsf{Area}(\mathbb{S})}$, where $\mathsf{Area}(\cdot)$ denotes the surface area. The volume of a set S, denoted by $\mathsf{Vol}(S)$, is defined to be $\frac{\mathsf{Area}(\cup_{p\in S}C_p)}{\mathsf{Area}(\mathbb{S})}$. It is easy to see that $\mathsf{Vol}(S)\in[0,1]$. The following lemma shows that our goal of finding a set S with $\cup_{p\in S}\mathcal{F}_p=\mathcal{FC}$ is equivalent to finding a S with $\mathsf{Vol}(S)=1$.

Lemma 4: $\cup_{p\in S}\mathcal{F}_p=\mathcal{FC}$ if and only if $\mathsf{Vol}(S)=1$.

Example 8: Consider the 3-dimensional example in Figure 8. $\mathbb{S}$ *is drawn in solid lines, which is the set of all utility vectors with norm equal to 1. Assume that* C_p *is formed by 3 extreme vectors, i.e.,* $\mathsf{Ext}(p)=\{v_1,v_2,v_3\}$. C_p *can be regarded as a partial spherical surface of* $\mathbb{S}$*, shown in shaded. The volume* $\mathsf{Vol}(p)$ *is computed to be the surface area (i.e., the 2-dimensional measure) of* C_p *divided by the surface area of* $\mathbb{S}$*. Consider Figure 9 where* $\mathbb{S}$ *is covered by the union of 6 conical hulls. If* S *is the corresponding set of points,* $\mathsf{Vol}(S)=1$. □

To find the desired set S with $\mathsf{Vol}(S)$ equal to 1, a crucial operation is to compute $\mathsf{Vol}(S)$ for a given S. We first treat the computation of $\mathsf{Vol}(S)$ as a black box and present a conceptual algorithm in Section IV-B. Since the exact (actual) volume $\mathsf{Vol}(S)$ is time-consuming to compute especially in a high dimensional space, we present an algorithm which utilizes the approximate (estimated) volume $\widetilde{\mathsf{Vol}}(S)$ in Section IV-C.

B. The Conceptual Algorithm

In this section, we first treat the computation of $\mathsf{Vol}(S)$ for a given set S as a black box. A natural greedy algorithm works as follows. The algorithm adds points iteratively to the solution S, initialized to be an empty set. In each iteration, the point with the largest *marginal volume* is inserted into S until $\mathsf{Vol}(S)=1$. The marginal volume of a point p is defined to be $\mathsf{Vol}(S\cup\{p\})$ - $\mathsf{Vol}(S)$. Intuitively, a large marginal volume of p indicates that the probability that a utility function is maximized by p but not points already in S is large. Let T be the total number of iterations. For $t\in[1,T]$, S_t represents the set of points selected so far after the t-th round. Let p_t be the point selected in the t-th round. Denote the marginal volume of p by $\varrho_p(S)=\mathsf{Vol}(S\cup\{p\})-\mathsf{Vol}(S)$. In addition, let $\theta_t=\min_{p\in D\setminus S_{t-1}}\frac{1}{\varrho_p(S_{t-1})}$. The pseudocode is shown in Algorithm 1, whose upper bound is given in Theorem 2.

Theorem 2: Denote the set returned by Algorithm 1 by S_T and the optimal set with volume equal to 1 by S^*.

$$|S_T|\le(1+\ln\min\{k_1,k_2,k_3\})|S^*|$$

where $k_1=\max_{p\in D,r\in[1,T]}\left\{\frac{\varrho_p(S_0)}{\varrho_p(S_r)}:\varrho_p(S_r)>0\right\}$, $k_2=\frac{\theta_T}{\theta_1}$ and $k_3=1/(1-\mathsf{Vol}(S_{T-1}))$.

C. The Cone-Greedy Algorithm

Unfortunately, computing the exact value of $\mathsf{Vol}(S)$ (which is not necessarily convex) is very expensive especially in a high dimensional space. Thus, we utilize a sampling method to estimate the (marginal) volume of a conical hull. Specifically, given a conical hull, we generate N unit samples from $\mathbb{R}^d_+$ by a d-dimensional uniform distribution and determine the number of samples inside this conical hull (i.e., the samples which are the conical combination of the extreme vectors that defines the conical hull). Then, the volume of this conical hull is estimated to be the ratio of the number of samples locating inside the conical hull to the total number of samples, N.

Algorithm 1 The Conceptual Algorithm

Input: D, α
Output: A set $S_T\subseteq D$
1: $S_0\leftarrow\emptyset$, $t\leftarrow0$
2: **for** each point $p\in D$ **do**
3: $\quad C_p\leftarrow$ the conical hull of p, $\mathsf{Vol}(p)\leftarrow\frac{\mathsf{Area}(C_p)}{\mathsf{Area}(\mathbb{S})}$
4: **end for**
5: **while** $\mathsf{Vol}(S_t)\neq1$ **do**
6: $\quad t\leftarrow t+1$
7: $\quad p_t\leftarrow\arg\max_{p\in D\setminus S_{t-1}}\varrho_p(S_{t-1})$, $S_t\leftarrow S_{t-1}\cup\{p_t\}$
8: **end while**
9: $T\leftarrow t$
10: **return** S_T

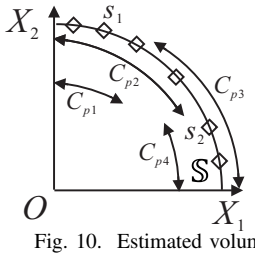


Fig. 10. Estimated volume

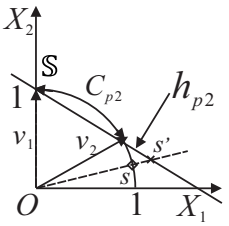


Fig. 11. s is not in C_{p_2}

We propose an algorithm, called CONE-GREEDY, for answering the α-happiness query, which works in a similar manner as Algorithm 1. The only difference is that we consider the estimated volumes instead of the exact volumes. With a slight abuse of notations, we let T be the total number of iterations. For each t in $[1,T]$, S_t denotes the set of points selected so far after the t-th round and let p_t be the point selected in the t-th round. Note that T, S_t and p_t in CONE-GREEDY might be different from those in Algorithm 1. We denote the estimated (marginal) volume p by $\widetilde{\mathsf{Vol}}(p)$ ($\tilde{\varrho}_p(S)=\widetilde{\mathsf{Vol}}(S\cup\{p\})-\widetilde{\mathsf{Vol}}(S)$) and the actual (marginal) volume of p by $\mathsf{Vol}(p)$ ($\varrho_p(S)=\mathsf{Vol}(S\cup\{p\})-\mathsf{Vol}(S)$). Let $\tilde{\theta}_t=\min_{p\in D\setminus S_{t-1}}\frac{1}{\tilde{\varrho}_p(S_{t-1})}$ and $\theta_t=\min_{p\in D\setminus S_{t-1}}\frac{1}{\varrho_p(S_{t-1})}$. The pseudocode of CONE-GREEDY is shown in Algorithm 2.

CONE-GREEDY can also be interpreted in a set-cover manner. We say that a sample s is *covered* by a conical hull C_p if s lies inside C_p. The greedy algorithm starts with an empty solution set S, and it adds points to S iteratively until all samples are covered by some conical hulls corresponding to the points in S. In each iteration, it adds the point whose conical hull covers the largest number of uncovered samples (i.e., the point with the largest estimated marginal volume).

Example 9: Consider the example in Figure 10, which are the conical hulls of points in Table I with $\alpha=0.9$. *In this 2-dimensional example, each* C_p *is an "arc" on* $\mathbb{S}$. *We generate 6 samples (i.e.,* $N=6$*), drawn in the diamonds in the figure. The sample* s_1 *is inside* C_{p_2} *while* s_2 *is not inside* C_{p_2}. *Since*

there are 4 out of 6 samples lying inside C_{p_2}, the estimated volume of p_2, $\widetilde{\mathsf{Vol}}(p_2)$, is $\frac{4}{6} = 0.666$. For comparison, the actual volume of p_2, $\mathsf{Vol}(p_2)$, can be computed to be 0.664.

CONE-GREEDY *on this example works as follows. In each iteration, it selects the point which has the largest estimated marginal volume, or equivalently, the point whose conical hull covers the largest number of uncovered samples. Specifically,* CONE-GREEDY *first selects p_2, whose conical hull covers 4 samples. The remaining 2 samples are covered by C_{p_3}, which is inserted to the solution S next. Finally, S is $\{p_2, p_3\}$.* □

A basic operation of CONE-GREEDY is that given a sample s and a conical hull C_p, we determine whether s is in C_p, which we call the *conical hull location problem*. The conical hull location problem needs to be solved for *each* sample and *each* conical hull combination. Thus, it is important to solve it efficiently, which we will discuss in Section IV-C1. The theoretical and complexity analysis of CONE-GREEDY are presented in Section IV-C2 and Section IV-C3, respectively.

1) The Conical Hull Location Problem: We focus on the conical hull location problem in this section: given a sample s and a conical hull C_p, is s inside C_p, denoted as "$s \in C_p$"?

According to the definition of conical hulls, $s \in C_p$ if and only if s is a conical combination of the extreme vectors that defines C_p. A naive solution of determining if $s \in C_p$ is to formulate it as a linear programming (LP) problem. However, if the number of extreme vectors is large, it is time-consuming to solve such an LP for each sample and each conical hull combination. In the following, we present a necessary condition for the conical hull location problem.

Given a conical hull C_p specified by m unit extreme vectors, namely $v_1, \ldots, v_m$ ($\|v_i\| = 1$), we define a hyperplane for p, denoted by h_p. In geometry, a hyperplane is uniquely defined by its *normal* and *offset*. Specifically, the normal of h_p, denoted by n_p, is defined to be the unit vector in the same direction as $\frac{1}{m}\sum_{i=1}^{m} v_i$ and the *offset* of h_p, denoted by c_p, is defined to be $\min_{j\in[1,m]} n_p \cdot v_j$. If a point is on h_p, its dot product with n_p is equal to c_p. Assume that the ray shooting from O in the direction of v_i intersects h_p at v_i'. When the context is clear, we simply say that v_i intersects h_p at v_i'. Similarly, we say that a sample s intersects h_p at s' if the ray shooting from O to s intersects h_p at s'. The necessary condition for determining if $s \in C_p$ is summarized as follows.

Lemma 5: If $s \in C_p$, $\|s'\| \leq 1$.

According to Lemma 5, if $\|s'\| > 1$, we can directly conclude that $s \notin C_p$ without solving the expensive LP.

Example 10: Consider Figure 11 where $\mathbb{S}$ (the set of all utility vectors with norms equal to 1) is shown. The conical hull C_{p_2}, defined by extreme vectors v_1 and v_2, is drawn in the figure. According to the construction above, we can define the hyperplane h_{p_2}. The intersections between h_{p_2} and both v_1 and v_2 (which are in C_{p_2}) have norms at most 1. Consider another intersection, denoted by s', between h_{p_2} and a sample s. Since $\|s'\| > 1$, we conclude $s \notin C_{p_2}$ by Lemma 5. □

In short, we solve the conical hull problems in three steps:

1) Transform each C_p to its corresponding hyperplane h_p;
2) For each sample s, we determine the set of hyperplanes whose intersections with s has norm at most 1 and they correspond to the candidate conical hulls containing s;
3) For each candidate hyperplane h_p, we check whether the corresponding C_p contains s by solving it as an LP.

Algorithm 2 The CONE-GREEDY Algorithm

Input: D, α, confidence parameter δ, error parameter ϵ
Output: A set $S_T \subseteq D$

```
Create N samples from a d-dimensional uniform distribution
S_0 ← ∅, t ← 0
for each point p ∈ D do
    C_p ← conical hull of p, Vol~(p) ← the estimated volume of p
end for
while Vol~(S_t) ≠ 1 do
    t ← t + 1
    p_t ← arg max_{p∈D\S_{t-1}} ϱ̃_p(S_{t-1}), S_t ← S_{t-1} ∪ {p_t}
end while
T ← t
return S_T
```

Time complexity: Assume that C_p is defined by m extreme vectors. Determining whether $s \in C_p$ can be formulated as an LP with m variables and d constraints, which can be solved in $O(m)$ time in practice [6]. If the necessary condition in Lemma 5 is satisfied, it simply takes $O(1)$ to conclude $s \notin C_p$ by checking the intersection between s and h_p. Note that the hyperplanes can be indexed using the techniques designed for the *ray shooting query* [2]. Then, for each sample s, the candidate conical hulls can be determined more efficiently.

2) Theoretical Analysis: Denote the set returned by CONE-GREEDY by S_T. We analyze the performance of CONE-GREEDY in this section. Specifically, we prove a lower bound on $\mathsf{Vol}(S_T)$ and prove upper bounds on the output size $|S_T|$.

Since we approximate $\mathsf{Vol}(S_T)$ by $\widetilde{\mathsf{Vol}}(S_T)$ using sampling, $\mathsf{Vol}(S_T)$ can be less than 1 even if $\widetilde{\mathsf{Vol}}(S_T) = 1$. Recall that each function in $\mathcal{FC}$ is equally probable to be used by a user. Given a set S_T with $\mathsf{Vol}(S_T) = \beta$ where $\beta \in [0, 1]$ for the α-happiness query, for each user, the probability that s/he will be at least α happy with S_T is at least β. The following lemma provides a lower bound on the actual volume $\mathsf{Vol}(S_T)$.

Lemma 6: Given a confidence parameter δ, a sufficiently small error parameter ϵ and a sampling size $N = O(\frac{d+\ln(1/\delta)}{\epsilon^2})$, with probability at least $1 - \delta$,

$$\mathsf{Vol}(S_T) \geq 1 - |S_T|\epsilon.$$

Proof Sketch. We prove it with the well-known Chernoff-Hoeffding Inequality [18] on the estimated volumes. For lack of space, the proof appears in the technical report [27]. □

According to Lemma 6, the probability that a user will be at least α happy with S_T is at least $1 - |S_T|\epsilon$. Note that ϵ is a parameter that we can make arbitrarily small and the size $|S_T|$ is small in practice. Then, under practical settings, we have $|S_T|\epsilon < 1$ and the bound is valid (i.e., $\mathsf{Vol}(S_T) \geq 1 - |S_T|\epsilon > 0$). To verify this assumption and show the usefulness of our algorithm, we conducted experiments in Section VI by setting a large ϵ (i.e., we set a small sampling size N) and show that

even this assumption is not true empirically, the output size is still small while guaranteeing the happiness ratio.

The following lemma is a variation of Theorem 2, which takes sampling into consideration and gives bounds on $|S_T|$.

Lemma 7: Given a confidence parameter δ, a sufficiently small error parameter ϵ and a sampling size $N = O(\frac{d+\ln(1/\delta)}{\epsilon^2})$ such that $\tilde{\varrho}_{p_T}(S_{T-1}) > \epsilon$, with probability at least $1-\delta$,

$$|S_T| \leq [\tilde{c}(1 + \ln \min\{\tilde{k}_1, \tilde{k}_2\})]|S^*|$$

where $\tilde{c} = \frac{\tilde{\varrho}_{p_T}(S_{T-1})+\epsilon}{\tilde{\varrho}_{p_T}(S_{T-1})-\epsilon}$, $\tilde{k}_1 = \frac{\tilde{\theta}_T}{\tilde{\theta}_1}$, $\tilde{k}_2 = \frac{1}{\tilde{\varrho}_{p_T}(S_{T-1})-\epsilon}$, and S^* is the optimal set with $\mathsf{Vol}(S^*) = 1$.

Proof Sketch. The proof follows a similar framework as [23] where actual volumes are considered. However, since actual volumes are unknown (and expensive to obtain), we prove the bounds using the estimated volumes in the sampling strategy. The complete proof appears in our technical report [27]. □

The following lemma gives the upper bound on the output size of CONE-GREEDY from the set-cover perspective.

Lemma 8: Given a confidence parameter δ, a sufficiently small error parameter ϵ and a sampling size $N = O(\frac{d+\ln(1/\delta)}{\epsilon^2})$, with probability at least $1-\delta$,

$$|S_T| \leq (1 + \log N)|S^*|.$$

where S^* is the optimal solution with $\mathsf{Vol}(S^*) = 1$.

Proof Sketch. It is proven by the well-known approximate ratio of the greedy algorithm for the set-cover problem. □

We summarize our results in the following theorem.

Theorem 3: Given a confidence parameter δ, a sufficiently small parameter ϵ and a sampling size $N = O(\frac{d+\ln(1/\delta)}{\epsilon^2})$ such that $\tilde{\varrho}_{p_T}(S_{T-1}) > \epsilon$, with probability at least $1-\delta$, CONE-GREEDY returns a set S_T for the α-happiness query such that

1) $|S_T| \leq \min\{[\tilde{c}(1 + \ln \min\{\tilde{k}_1, \tilde{k}_2\})], 1 + \log N\}|S^*|$ where $\tilde{c} = \frac{\tilde{\varrho}_{p_T}(S_{T-1})+\epsilon}{\tilde{\varrho}_{p_T}(S_{T-1})-\epsilon}$, $\tilde{k}_1 = \frac{\tilde{\theta}_T}{\tilde{\theta}_1}$, $\tilde{k}_2 = \frac{1}{\tilde{\varrho}_{p_T}(S_{T-1})-\epsilon}$, and S^* is the optimal set with $\mathsf{Vol}(S^*) = 1$;
2) $\mathsf{Vol}(S_T) \geq 1 - |S_T|\epsilon$, i.e., for each user, the probability that s/he will be at least α happy is at least $1 - |S_T|\epsilon$.

Proof. Directly from Lemma 6, Lemma 7 and Lemma 8. □

Other distributions of $\mathcal{FC}$. While we have assumed that all functions in the class $\mathcal{FC}$ are equally probable to be used by a user, the methods presented in this paper can be generalized to other distributions of $\mathcal{FC}$ with the following two modifications. Firstly, $\mathsf{Vol}(S)$ is defined based on the distribution of $\mathcal{FC}$ (e.g., by taking the integral over $\mathcal{FC}$) instead of simply the surface area. Secondly, when approximating $\widetilde{\mathsf{Vol}}(S)$ by sampling, N unit samples are generated based on the distribution of $\mathcal{FC}$ instead of the d-dimensional uniform distribution.

3) Time Complexity Analysis: In Section III-B, we compute C_p for all p in D in $O(n|B'| + nm)$ time where B' is the set of all vertices of $\mathsf{Conv}(D')$ and m is the maximum extreme vectors of a point in D. N samples can be generated in $O(N)$ time. For each sample s and each C_p, we check whether $s \in C_p$, resulting in $O(Nnm)$ time in total. The greedy set-cover algorithm takes $O(|S_T|Nn)$ time. Thus, the total time complexity of CONE-GREEDY is $O(n|B'|+Nnm+|S_T|Nn)$.

Comparison. Compared with the existing algorithms, CONE-GREEDY has some attractive differences. Firstly, the execution time of CONE-GREEDY is less sensitive to large α, while existing algorithms degrade rapidly when users require a high happiness level (e.g., the time complexity of [13] is proportional to $\frac{1}{(1-\alpha)^{O(d)}}$). Secondly, CONE-GREEDY provides a log-factor bound on the output size by utilizing sampling information, which has not been considered before. Finally, different from previous approaches, the larger the happiness level a user requires, the less time CONE-GREEDY needs empirically. Intuitively, this is because that when α is larger, the number of extreme vectors of a point in D tends to become smaller, resulting in less time to solve the α-happiness query. We will experimentally verify the last advantage in Section VI.

V. RELATED WORK

The regret minimization query was first introduced by Nanongkai et. al in [15]. In particular, they focus only on the k-regret query (i.e., the min-error version). Given an integer k, a k-regret query returns a set S of at most k tuples such that the "difference" between the maximum utility of tuples in S and the maximum utility of tuples in the whole dataset D is minimized. Equivalently, we can also say that a k-regret query maximizes the happiness level of each user (measured by how close the maximum utility in S is to the maximum utility in D) which we quantify as maximizing the *minimum happiness ratio* of users (the worst-case happiness ratio over all users). Finding an optimal solution for a k-regret query was proven in [3], [8], [9] to be an NP-hard problem. Nanongkai et. al [15] proposed a space-partition based algorithm which returns a set of tuples whose minimum happiness ratio is lower bounded. This bound was improved in [8], [26] which is derived based on the well-known ϵ-*kernel* problem [1]. Greedy-based algorithms which construct the solutions iteratively were proposed in [9], [15], [16], [26]. Besides, the k-regret query can be solved in a set-cover manner [4] while user interactions were considered in [14], [24], [25].

The min-size version of regret minimization, namely the α-happiness query, was first considered by Agarwal et. al in [3]. Specifically, given a real number $\alpha \in [0, 1]$, an α-happiness query minimizes the output size while keeping the users happy (i.e., the minimum happiness ratio is at least α). Various algorithms were proposed for the α-happiness query and they can be categorized into the following approaches. The first approach formulated the α-happiness query as an ϵ-*kernel* problem [1] and approximate algorithms [3], [8] were proposed to obtain a desired solution. However, due to the large size of an ϵ-kernel, ϵ-kernel based approaches might have large output size. The second approach [3], [4] discretized the function space and formulated the α-happiness query as a hitting set problem (or a set-cover problem), which provides user-controlled approximations on both the minimum happiness ratio and output size. More recently, Kumar et. al [13] proposed CORESETHS (which is included in our experiments) which combines ϵ-kernel and hitting set approaches and achieves better efficiency than previous algorithms.

Some k-regret query algorithms can be modified to answer an α-happiness query. One could set a proper k in the theoretically bounded algorithms [8], [15], [26] so that the lower bound on the minimum happiness ratio is at least α. Unfortunately, the value of k set in this approach can be extremely large. For example, to guarantee 0.95 happy on a 3-dimensional dataset, the space-partition based algorithm in [15] has to return 1400 tuples, which is unacceptable in real scenarios. One could also modify the greedy-based algorithms [9], [15], [16], [26] so as to include tuples greedily until the minimum happiness level of users is satisfied (we consider this approach in our experiments). Similarly, solutions for the α-happiness query can also be extended to answer the k-regret query: using an approach discussed in [3], [13], one can perform a binary search on different happiness ratios to obtain a solution set whose output size is at most k.

Compared with the existing studies [3], [4], [8], [13], we interpret the α-happiness query from a geometric perspective and our algorithm enjoys novel theoretical bounds on the output size. In particular, the execution times of existing approaches are large when the users require high happiness levels (since they have to consider a large number of utility functions when α is large), while our algorithm, which is less sensitive to α, can guarantee high happiness efficiently. According to our experimental evaluations, our algorithm performs more efficiently and effectively than previous approaches. Under practical settings, we achieve up to two orders of improvements in running time and return the least tuples among all methods while guaranteeing the required happiness.

There is also recent literature on multi-criteria decision making that uses geometric concepts. Peng et. al [16] defined a *convex hull* for a fixed size set of tuples in k-regret queries, but we focus on minimizing the solution set size in α-happiness queries. Different from them, we *shrink/scale* (a concept that does not appear in [16]) each tuple in the database according to the required happiness level so that all shrunk tuples are *inside* the convex hull of the solution set. This idea cannot be applied in [16] for k-regret queries where the happiness level is not known in advance. Soma and Yoshida [22] also considered a shrunk convex hull but they focused on the multi-objective function maximization, which takes an approximation algorithm for each single objective function as an input. Our problem has a single objective function (which is treated as a black box in [22]) and thus, their analysis could not be applied to our problem. Peng and Wong [17] also determined a set of *non-overlapping* conical hulls so that the total space "covered" by those conical hulls are maximized, while the conical hulls in our problem are *overlapping*. This makes the problem more challenging: in their case, the total coverage of conical hulls is simply the sum of the individual coverage of each conical hull (since their conical hulls are non-overlapping), which, however, is not true in our case.

VI. Experimental Evaluation

We conducted experiments on a machine with 3.20GHz CPU and 8GB RAM. All programs were implemented in C/C++. Most experimental settings follow those in [3], [13]. Both *synthetic* and *real datasets* were used in our experiments.

We generated anti-correlated datasets (the most interesting synthetic dataset where the skyline size is large) by a dataset generator originally developed for the skyline query in [7]. Unless stated explicitly, for each synthetic dataset, the number of tuples is set to be 100,000 (i.e., n = 100,000), the dimensionality is set to be 3 (i.e., d = 3), the sampling size is set to be 10,000 (i.e., $N = 10,000$), and α is set to be 0.99. The real datasets contain three datasets commonly used in the existing studies. They are the *Island* dataset [15], the *Airline* dataset [4] and the *El Nino* dataset [3], [9]. Island is 2-dimensional, containing 63,383 points, which characterize geographic positions. Airline contains 5,810,462 records with 3 characterizing attributes, namely the actual elapsed time, the distance and the arrival delay. El Nino contains 178,080 tuples with four oceanographic attributes taken at the Pacific Ocean. For all datasets, each attribute is normalized to (0, 1]. The characteristics of real datasets are summarized in Table III.

According to the results reported in [15], the naive algorithms adapted from top-k and skyline queries (or even constructing the solution by randomly selecting points) can achieve a minimum happiness ratio of around 0.9 with a small number of points (e.g., $<$ 10 points); thus it becomes less interesting to minimize the output size in these cases (since the output size is already very small). This observation conforms with our argument in Section I that users are interested in small sets achieving high happiness levels. Motivated by this, in our experiments, we concentrate on α close to 1. Similar setting was also adopted in the existing studies [3], [13].

We implemented our algorithm, Cone-Greedy. The competitor algorithms are (1) the ϵ-kernel based algorithm [3], [8], denoted by Coreset; (2) the best performing hitting-set based algorithm [13], denoted by CoresetHS; and (3) a variation of the greedy algorithms, denoted by LP-Greedy [15] (see Section V), which is originally designed for the k-regret query. The implementations of Coreset and CoresetHS are provided by the authors in [13]. Note that theoretically, both Coreset and CoresetHS have to sample $O(\frac{1}{(1-\alpha)^{O(d)}})$ utility functions to construct the solutions. However, in their practical implementations, the number of utility functions they sampled is determined empirically (i.e., a number much smaller than $O(\frac{1}{(1-\alpha)^{O(d)}})$). We follow this setting and use the reported parameters in [13] in our experiments.

Unless specified explicitly, the performance of each algorithm is measured in terms of *query time* (i.e., efficiency) and *output size* (i.e., quality). The query time of an algorithm is the execution time of the algorithm. The output size of an algorithm is the number of points returned by the algorithm. Some results are plotted in log-scale for better visualization.

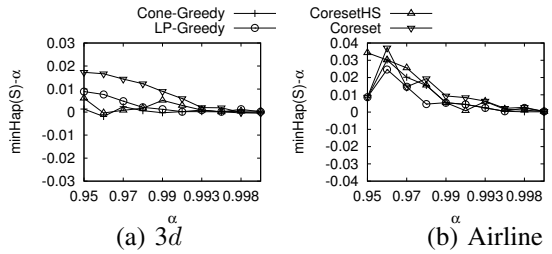
We first evaluate the sampling strategy in our algorithm in Section VI-A. Then, we proceed with the experiments on the synthetic and real datasets in Section VI-B and Section VI-C. Finally, we summarize our findings in Section VI-D.

TABLE III
REAL DATASETS

Dataset	d	$\|D\|$
Island	2	63,383
Airline	3	5,810,462
El Nino	4	178,080

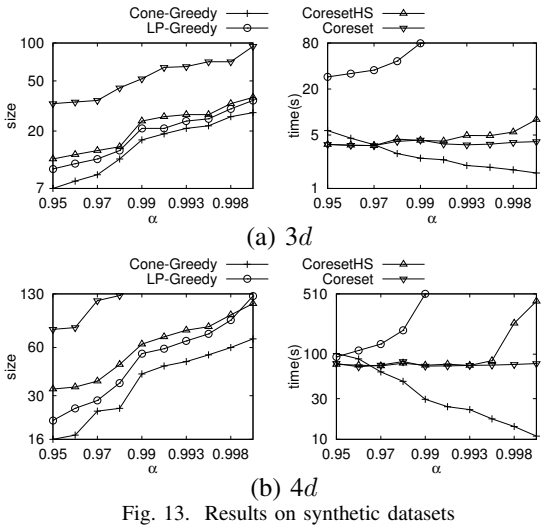
TABLE IV
VARY N IN CONE-GREEDY

	Cone-Greedy							LP-Greedy	CoresetHS	Coreset
N	100	500	1k	5k	10k	50k	100k	-	-	-
$\mathsf{minHap}(S)$	0.980	0.981	0.987	0.990	0.990	0.990	0.990	0.990	0.991	0.991
$\|S\|$	11	14	14	15	15	15	15	19	18	45



(a) $3d$ (b) Airline

Fig. 12. Results on the minimum happiness ratio



(a) $3d$

(b) $4d$

Fig. 13. Results on synthetic datasets

A. Effectiveness of the Sampling Strategy

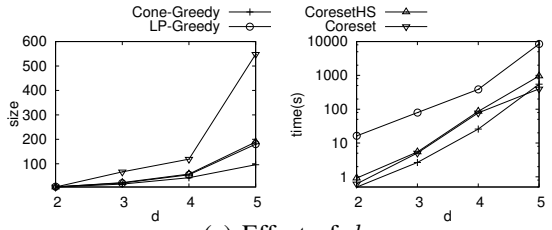
In this section, we demonstrate the effectiveness of the sampling strategy in our CONE-GREEDY algorithm.

Similar to the existing algorithms for both k-regret queries and α-happiness queries (e.g., CORESET and CORESETHS), which relax both the happiness ratio and the output size simultaneously, the happiness ratio in CONE-GREEDY is guaranteed with a certain probability. Thus, in this section, we first verify whether the happiness ratio is guaranteed empirically in CONE-GREEDY. Specifically, we conducted experiments by varying α (and fixing the sampling size to the default value) on both the 3-dimensional synthetic dataset and the Airline dataset in Figure 12 (other datasets are similar). We reported the difference between the minimum happiness ratio of the returned set S, namely $\mathsf{minHap}(S)$, and the required happiness level α for each algorithm. Recall that $\mathsf{minHap}(S)$ is not optimized in an α-happiness query. Thus, a larger difference between $\mathsf{minHap}(S)$ and α does *not* mean that the solution is of better quality. Instead, as long as the difference is non-negative (i.e., $\mathsf{minHap}(S) \geq \alpha$), S is a valid solution for the α-happiness query and its quality is measured by the output size, which will be studied shortly in later subsections. In particular, if the happiness difference of CONE-GREEDY is non-negative, it means that the probability that a user will be at least α happy with S is 100%. According to the results shown in Figure 12, CONE-GREEDY achieves comparable performance as the existing methods and its happiness difference is non-negative in most of the cases. In other words, the happiness ratio can be empirically guaranteed in CONE-GREEDY.

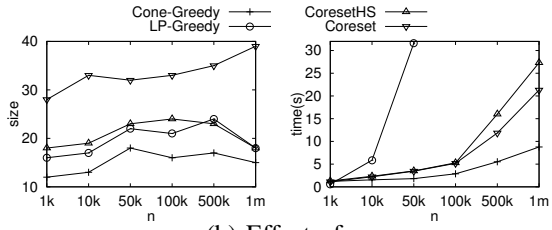
We also studied the effect of our sampling strategy by varying the sampling size N (and set α to be 0.99, the default value). Equivalently, it also means that we are varying the error parameter ϵ in the sampling strategy since they are closely related (i.e., the larger the sampling size, the smaller the error). The results on a 3-dimensional dataset are shown in Table IV. When N is very small, $\mathsf{minHap}(S)$ can be less than (but close to) α; for example, we can achieve a 0.98 happiness ratio with only 100 samples. When N becomes larger (i.e., ϵ becomes smaller), the estimated volume is closer to the actual volume and thus, $\mathsf{minHap}(S)$ becomes closer to α. In particular, when $N \geq 5,000$, the returned set S achieves the required happiness level 0.99 and its size becomes stable, i.e., the size does not increase further with more sampling points. Compared with the competitors which achieve the same happiness level, CONE-GREEDY outputs the *smallest* number of points. Note that a large sampling size N is useful in achieving the required happiness ratio, but it can introduce additional time in solving the conical hull location problems. On the other hand, a small N results in a faster algorithm but it can also give a large error in the estimated volume. Following the practical way of determining the sampling size in CORESET and CORESETHS, we set the default sampling size to be $10,000$ in our experiments since it achieves a reasonable trade-off between the efficiency and the effectiveness according to the results in Table IV.

B. Results on Synthetic Datasets

We evaluated our algorithm, CONE-GREEDY, on both 3-dimensional and 4-dimensional anti-correlated datasets. The results are presented in Figure 13(a) and (b), respectively. When considering the running times, LP-GREEDY has the largest execution time (e.g., 30 minutes on the 4-dimensional dataset when $\alpha = 0.999$, which is too large to be plotted in the figure), which is also observed in [4], [13]. Besides, we observe that CONE-GREEDY enjoys a different trend compared to other algorithms: when α increases, most of the other algorithms consume more time to construct the solution (since they have to sample more utility functions to guarantee higher happiness) while the times needed by CONE-GREEDY decreases. Intuitively, this is because when α is large, the convex hull $\mathsf{Conv}(D')$ is "close" to $\mathsf{Conv}(D)$ and thus, the



(a) Effect of d



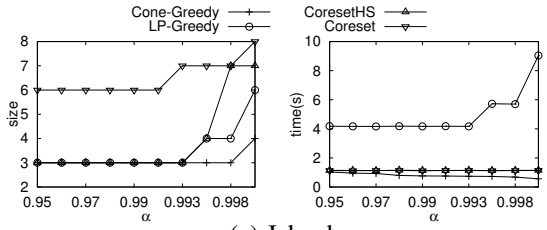
(b) Effect of n

Fig. 14. Scalability test

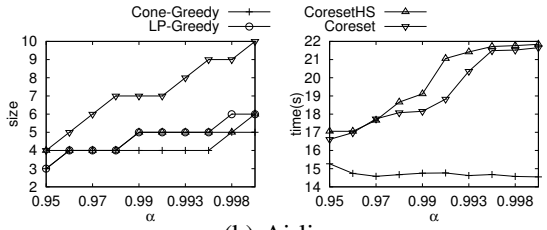
portion of $\mathsf{Conv}(D')$, which is visible from each p, becomes small, resulting in a small set of extreme vectors in CONE-GREEDY and a short time to solve the conical hull location problems. In comparison, the execution time of the best-known algorithm CORESETHS is sensitive to large α since its running time is proportional to $\frac{1}{(1-\alpha)^{O(d)}}$ (which, however, is removed in our algorithm). For example, when $\alpha = 0.999$ on the 4-dimensional dataset, it takes CORESETHS around 500 seconds to execute, but CONE-GREEDY only spends around 10 seconds (an order of magnitude improvement) to return the solution. Despite the significant speedup, CONE-GREEDY also produces the *smallest* solution set in *all settings*. For example, CONE-GREEDY achieves 27.82% and 39.97% improvement in terms of the output size over the best-known previous algorithm, CORESETHS, on $3d$ and $4d$ datasets, respectively. CORESET has the largest output size in most cases due to the large size of an ϵ-kernel in CORESET under practical settings.

Note that given $\alpha_1, \alpha_2 \in [0, 1]$ with $\alpha_1 \geq \alpha_2$, a solution for the α_1-happiness query is also a solution for the α_2-happiness query (but it may not be minimal). Although solving the α_1-happiness query (and using its solution as the solution for the α_2-happiness query) might take less time in CONE-GREEDY, we argue that it is still necessary to spend slightly more time to solve the α_2-happiness query since its solution could have a much smaller size; e.g., the output size of the 0.95-happiness query is smaller than half of the output size of the 0.99-happiness query in Figure 13(a). Besides, the solution of an α_1-happiness query can be large and it may not be practical to return them as the output. For example, the 1-happiness query is the special case of our problem whose solution is the set of vertices of $\mathsf{Conv}(D)$. Although the solution for a 1-happiness query is a solution for any α-happiness query with an arbitrary α, there are 569 points in the solution set on a 5-dimensional dataset, which is too large to return. To make the user's decision easier, it is desirable to have a smaller output size by setting a smaller α. This is a typical trade-off between user happiness and the output size in the α-happiness query.

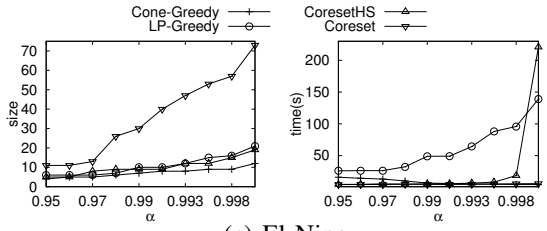
We evaluated the scalability of CONE-GREEDY by varying



(a) Island



(b) Airline



(c) El Nino

Fig. 15. Results on real datasets

the dimensionality d and the dataset size n in Figure 14 where other parameters are fixed to the default setting stated at the beginning of this section. According to the results, our algorithm scales well w.r.t. both d and n compared with the existing algorithms in most cases and its output size is consistently smaller than the output sizes of *all* other algorithms in *all* cases. For example, on a 5-dimensional dataset, CONE-GREEDY returns 96 points as the solution, while the output sizes of LP-GREEDY, CORESETHS and CORESET are 180, 189 and 548, respectively. When $n = 1,000,000$, CONE-GREEDY takes around 10 seconds to return a solution with 15 points while CORESETHS spends around 30 seconds to find a solution set with 18 points. In other words, CONE-GREEDY does not only return a solution with better quality (i.e., a smaller size), it does so faster than previous approaches.

C. Results on Real Datasets

In this section, we conducted experiments on three commonly used real datasets. The results are shown in Figure 15.

On the 2-dimensional Island dataset (Figure 15(a)), the output sizes of all algorithms (except for CORESET which has the worst output size) are small when α is small. However, when α is large, the output sizes of the competitor algorithms become much larger than our CONE-GREEDY algorithm, which conforms with our argument at the beginning that large α is the case of practical interest since a small number of points can satisfy a small happiness ratio, making it useless to minimize the output size in this case. On the other hand, the running time of CONE-GREEDY decreases slightly when α increases and it runs the fastest under all values of α. In

particular, when $\alpha = 0.999$, CONE-GREEDY only spends 0.56 second to execute, which is 16 times faster than LP-GREEDY.

The results on Airline are similar and they are shown in Figure 15(b). Note that due to the large dataset size on Airline, it takes an unnecessary long time for LP-GREEDY to return the solution. Thus, the running time of LP-GREEDY is omitted in the figure. According to Figure 15(b), CONE-GREEDY outperforms CORESETHS and CORESET in both output size and running time under all settings. For example, the output size of CONE-GREEDY is 12.17% and 40.69% (on average) smaller than the output sizes of CORESETHS and CORESET, respectively. Meanwhile, CONE-GREEDY is consistently faster than CORESETHS and CORESET under all values of α.

Finally, consider the experiments on the El Nino dataset in Figure 15(c). Similar to the previous experiments, CORESET performs poorly since it outputs much more points to guarantee the same happiness level compared with the other algorithms. In contrast, the output size of CONE-GREEDY is the smallest. Moreover, although CONE-GREEDY is slightly slower than some competitors for small α, it becomes faster when the user requires a higher happiness ratio. In comparison, the running times of CORESETHS and LP-GREEDY are very sensitive to α and they become much slower than CONE-GREEDY for large α. In particular, when $\alpha = 0.999$, CONE-GREEDY achieves a 30 times improvement in running time compared with both CORESETHS and LP-GREEDY.

D. Summary

The experiments on both real and synthetic datasets demonstrated our superiority over previous approaches. Specifically, we achieve up to two orders of improvement in running time compared with CORESETHS and LP-GREEDY under typical settings; e.g., in Figure 13(b), CORESETHS and LP-GREEDY took 8 and 30 minutes to return the solutions for $\alpha = 0.999$, which is too long, while we only needed 10 seconds to execute (i.e., 50 times and 180 times improvements over CORESETHS and LP-GREEDY, respectively). Meanwhile, our solution has the smallest size in all cases (e.g., half of CORESET on Island for all α). The scalability of our solution and the effectiveness of our sampling strategy are also demonstrated. For example, on a 5-dimensional dataset, our CONE-GREEDY algorithm returned 96 points for $\alpha = 0.999$, while the output size of CORESET is 548 which is too large for a user to examine.

VII. CONCLUSIONS

This paper studies the α-happiness query, which returns a minimum number of tuples from the database such that the minimum happiness ratio is at least α. We interpret the problem from a geometric perspective and give a solution with novel theoretical guarantees and superior empirical performance. We conducted comprehensive experiments to verify the efficiency and effectiveness of the proposed solution. As for future research, we plan to devise a model that estimates time/size cost given an α value. Other directions include handling larger datasets (which could not fit in main memory) and with higher dimensionalities as well as exploring parallelism.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Min Xie and Raymond Chi-Wing Wong was supported by HKRGC GRF 16214017, while Vassilis J. Tsotras was supported by NSF grants: IIS-1838222, IIS-1527984 and IIS-1447826.

REFERENCES

[1] P. Agarwal, S. Har-Peled, and K. Varadarajan. Approximating extent measures of points. In *JACM*, volume 51, pages 606–635. ACM, 2004.

[2] P. Agarwal and J. Matousek. Ray shooting and parametric search. In *SIAM Journal on Computing, vol. 22, no. 4*. IEEE, 1993.

[3] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. Efficient algorithms for k-regret minimizing sets. In *SEA*, 2017.

[4] A. Asudeh, A. Nazi, N. Zhang, and G. Das. Efficient computation of regret-ratio minimizing set: A compact maxima representative. In *Proc. of 2017 ACM International Conference on Management of Data*, 2017.

[5] C. Barber, D. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. In *TOMS*, volume 22, pages 469–483. Acm, 1996.

[6] D. Bertsimas and J. Tsitsiklis. Introduction to linear optimization. volume 6. Athena Scientific Belmont, MA, 1997.

[7] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of 17th International Conference on Data Engineering*, 2001.

[8] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R. Wong, and W. Zhan. k-regret minimizing: Efficient algorithms and hardness. In *ICDT*, 2017.

[9] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. In *Proc. of the VLDB Endowment*, 2014.

[10] J. Dulá, R. Helgason, and N. Venugopal. An algorithm for identifying the frame of a pointed finite conical hull. INFORMS, 1998.

[11] T. K. Faulkner, W. Brackenbury, and A. Lall. k-regret queries with nonlinear utilities. In *Proc. of the VLDB Endowment*, 2015.

[12] J. Hiriart-Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms I: Fundamentals*. 2013.

[13] N. Kumar and S. Sintos. Faster approximation algorithm for the k-regret minimizing set and related problems. In *ALENEX*, 2018.

[14] D. Nanongkai, A. Lall, A. Sarma, and K. Makino. Interactive regret minimization. In *Proc. of the 2012 International Conference on Management of Data*, 2012.

[15] D. Nanongkai, A. Sarma, A. Lall, R. Lipton, and J. Xu. Regret-minimizing representative databases. In *Proc. of the VLDB Endowment*, volume 3, pages 1114–1124. VLDB Endowment, 2010.

[16] P. Peng and R. Wong. Geometry approach for k-regret query. In *Proc. of the 30th International Conference on Data Engineering*, 2014.

[17] P. Peng and R. Wong. k-hit query: Top-k query with probabilistic utility function. In *Proc. of the 2015 International Conference on Management of Data*, 2015.

[18] J. M. Phillips. Chernoff-hoeffding inequality and applications. In *arXiv preprint arXiv:1209.6396*, 2012.

[19] J. Qi, F. Zuo, and J. Yao. K-regret queries: From additive to multiplicative utilities. In *CoRR*, 2016.

[20] L. Qin, J. Yu, and L. Chang. Diversifying top-k results. In *Proc. of the VLDB Endowment*, volume 5, pages 1124–1135, 2012.

[21] M. Soliman, I. Ilyas, and C. Chang. Top-k query processing in uncertain databases. In *Proc. of the 23rd International Conference on Data Engineering*, pages 896–905. IEEE, 2007.

[22] T. Soma and Y. Yoshida. Regret ratio minimization in multi-objective submodular function maximization. In *AAAI*, pages 905–911, 2017.

[23] L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. In *Combinatorica*, pages 385–393. Springer, 1982.

[24] M. Xie, T. Chen, and R. Wong. FindYourFavorite: An interactive system for finding the user's favorite tuple in the database. In *Proc. of the 2019 ACM International Conference on Management of Data*. ACM, 2019.

[25] M. Xie, R. Wong, and A. Lall. Strongly truthful interactive regret minimization. In *Proc. of the 2019 ACM International Conference on Management of Data*. ACM, 2019.

[26] M. Xie, R. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proc. of the 2018 International Conference on Management of Data*, 2018.

[27] M. Xie, R. Wong, P. Peng, and V. Tsotras. Being happy with the least: achieving α-happiness with minimum number of tuples (technical report), http://home.cse.ust.hk/~raywong/paper/icde20-happy-technical.pdf. 2020.