k-Hit Query: Top-k Query with Probabilistic Utility Function

Peng PENG
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
ppeng@cse.ust.hk

Raymond Chi-Wing WONG
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
raywong@cse.ust.hk

ABSTRACT

Multi-criteria decision making problem has been well studied for many years. One popular query for multi-criteria decision making is top-k queries which require each user to specify an *exact* utility function. In many cases, the utility function of a user is *probabilistic* and finding the distribution on the utility functions has been widely explored in the machine learning areas, such as user's recommender systems, Bayesian learning models and user's preference elicitation, for improving user's experience. Motivated by this, we propose a new type of queries called *k-hit queries*, which has not been studied before. Given a set D of tuples in the database, the distribution Θ on utility functions and a positive integer k, we would like to select a set of k tuples from D in order to maximize the probability that at least one of tuples in the selection set is the favorite of a user. All applications for top-k queries can naturally be used in k-hit queries. In this paper, we present various interesting properties of k-hit queries. Besides, based on these properties, we propose a novel algorithm called *k-hit_Alg* for k-hit queries. Finally, we conducted comprehensive experiments to show that the performance of our proposed method, *k-hit_Alg*, is superior compared with other existing algorithms which were originally used to answer other existing queries.

1. INTRODUCTION

Multi-criteria decision making problem has been well studied for many years. Over the last few decades, there were many different types of queries proposed in the literature for multi-criteria decision making. Two representative queries considered in the literature are *top-k queries* [27, 12, 17, 35, 10, 24] and *skyline queries* [3, 12, 17, 6, 18, 19, 29]. Top-k queries are used when the *exact* preference/utility function of a user is known. This is because when the exact utility function of a user is known, each top-k query returns a set of k tuples from the database which are the k *favorites* of the user. Here, the *favorite* of the user is the tuple in the database which has the highest utility computed based on the utility function of the user. However, in many cases, the exact utility function is unknown. Skyline queries, another kind of representative queries, are used when the utility function of a user is *unknown*. This is because when the utility function of a user is unknown, each skyline query returns a set of all possible candidate tuples which include the most *favorite* of the user. However, as found by many existing studies [18, 29, 35], there may be a vast number of all possible candidate tuples to be returned by a skyline query. Thus, it is difficult for the user to select his/her favorite among all the possible candidate tuples.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2723735.

1.1 Probabilistic Utility Function

In many cases, it is difficult for companies to obtain the exact utility function of a user because the user may not give his/her utility function to companies or the user may hardly give an exact utility function to companies. Instead, it is much easier for companies to obtain a distribution Θ on utility functions based on the histories of the user. How to obtain the distribution on utility functions has been widely explored in the machine learning area, such as user's recommender systems [4, 25], Bayesian learning models [8, 15] and user's preference elicitations [2, 7]. In these applications, people consider a direct way or an indirect way to analyze a utility function. Instead of asking or finding the utility function directly, a statistical model can be built for recovering the distribution of the utility function via observing data recording user's activities (e.g., search log). Since collecting statistics from the user is used to improve his/her experience on using the recommender system in the future, nowadays, many companies have a lot of statistics from a collection of users.

It is worth mentioning that the distribution on utility functions can be applied over not only a *single* user but also a *group* of users. When the distribution Θ on utility functions are applied over a group of users, companies can obtain the proportion of users with a certain utility function based on the distribution Θ. This can help companies a lot for decision-making, because companies can have an overall picture of utility functions from a lot of users, and can promote effective campaigns to a particular group of users.

1.2 k-Hit Query

In this paper, we study the following query called the *k-hit* query which has not been studied before. Given a set D of tuples in the database and the distribution Θ on utility functions, we would like to select a set S of k tuples from D in order to maximize the probability that at least one of

tuples in the selection set is the favorite of a user where k is a positive integer and a user parameter. In this paper, we show that the top-k query is a special case of our k-hit query when k is set to 1 and there is only one exact utility function considered in the distribution Θ. We also show that all tuples in the optimal selection set of our k-hit query come from the set of all tuples returned by the skyline query when the utility functions considered is *linear*, a common utility function form adopted in the literature, regardless of the distribution on utility functions.

Our k-hit query is very useful when the company does not know the *exact* utility function of a user. Firstly, when a company knows *some* information about the utility function of a user via some channels like the histories of the user, the company can learn the distribution on utility functions for this user and could use our k-hit query to return k tuples which are interesting to the user with the greatest probability. Secondly, when a company does *not* know any information about the utility function of a user, our k-hit query can also help the company increase the chance of returning tuples that the user is interested in. This is because with the distribution on utility functions over a group of users, the probability that this user is interested in one of the k tuples returned by our k-hit query is the greatest.

Furthermore, our k-regret query can be regarded as an "improved" version of many existing "k-representative" queries in the literature of database [29, 21, 20] with the help of additional information about the distribution on utility functions. Some existing k-representative queries [29] find k tuples such that the distance between any two tuples among these k tuples in the Cartesian space is maximized. Others [21, 20] select k tuples such that each of these k tuples has the greatest *score* computed based the dataset and the other selected tuples. However, these existing studies do not consider the distribution on utility functions, the important user information from real-life applications, which can help companies return k tuples which are interesting to a user with a higher chance. This is because the distribution on utility functions can "represent" a big picture of utility functions of users and thus the k tuples are "representative".

Unfortunately, to the best of our knowledge, there are no existing studies for our k-hit query. The closely related work is [36] considering the distribution on utility functions similar to us. But, the objective of [36] is different from ours. Specifically, [36] studied the problem called the *order-based representative skyline* problem. Given a set D of tuples in the database and the distribution Θ on utility functions, the order-based representative skyline problem is to select a set of k tuples from D such that (1) the probability that each tuple in the selection set is the favorite of a user is at least a given user parameter $\alpha \in [0, 1]$, and (2) the "similarity" between the selection set S and D is maximized. Here, the "similarity" introduced in [36] is defined based on a function $\mathcal{SIM}_m(\cdot, \cdot)$ which comes with another user parameter m which is a positive integer. Specifically, given a tuple t in S and another tuple t' in D, $\mathcal{SIM}_m(t, t')$ is defined to be the probability that t' is in the set of the top-m tuples based on the utility function which ranks t as the top-1 tuple. Roughly speaking, if t has the highest utility based on a utility function, then t' also has its high utility similar to t (though not as high as t). Given a selection set S and a tuple t' in D, the *representative value* of t' in S is defined to be $\max_{t \in S} \mathcal{SIM}_m(t, t')$. Given a selection set S and D, the "similarity" between S and D is defined to be the average representative value of a tuple in D.

It is easy to see that the order-based representative skyline problem is different from our k-hit query. Firstly, the objective function of the order-based representative skyline problem, which is the "similarity" between the selection set and D, is different from ours, which is the probability that at least one of tuples in the selection set is the favorite of a user. Secondly, the user parameter m introduced in the order-based representative skyline problem could not be found in our k-hit query, and it is difficult to know how to set this parameter properly.

1.3 Contributions & Organization

Our major contributions can be summarized as follows. Firstly, we are the first to study the k-hit query, which has a wide range of applications since all applications for top-k queries and skyline queries can naturally be used in our k-hit query. Secondly, since there are a lot of existing studies [36, 5, 7, 20, 3, 6, 27, 12, 17, 35] considering that the utility functions are in the linear form, we study our k-hit query when the functions are in this form and present its relevance to a geometry concept called "solid angle" [13, 26] and some existing database queries, namely top-k queries [27, 12, 17, 35, 10, 24], skyline queries [3, 12, 17, 6, 18, 19, 29], k-regret queries [20, 22] and k-representative skyline queries [29]. Besides, if the distribution Θ on (linear) utility functions is uniform, we find an interesting geometry property based on "solid angle" and develop a novel algorithm called *k-Hit_Alg* based on this property. If the distribution Θ is non-uniform, we also generalize *k-Hit_Alg* for our k-hit query. Thirdly, we extend *k-Hit_Alg* when the form of the utility functions are non-linear, regardless of whether the distribution Θ on utility functions is uniform or not. Lastly, we conducted comprehensive experiments for the k-hit query, empirically verifying the superior performance of our proposed algorithm, *k-Hit_Alg*, compared with other existing algorithms which were originally used to answer other queries.

The rest of the paper is organized as follows. Section 2 formally defines our k-hit query. Section 3 gives some interesting theoretical properties of our query when the utility functions considered are linear. Section 4 shows our proposed algorithm, *k-Hit_Alg*. Section 5 gives the related work. Section 6 presents our experimental results. Lastly, Section 7 presents the concluding remarks and the future work.

2. PROBLEM DEFINITION

We are given a set D of n tuples each represented in the form of a d-dimensional non-negative real vector. For each tuple $p \in D$, we denote the i-th dimensional value of p by $p[i]$ for $i \in [1, d]$. Each dimensional value ranges from 0 to 1. For each dimension, a larger value is more preferable. For the sake of illustration, we assume that for each $i \in [1, d]$, there exists a tuple $p \in D$ such that $p[i] = 1$. Since a tuple represented in the form of a d-dimensional vector can be regarded as a d-dimensional point, in the following, we use the terms "tuples" and "points" interchangeably since they correspond to the same concept.

Given a tuple p represented in the form of a d-dimensional non-negative real vector, a *utility function* $f(\cdot)$ is defined to be a mapping function which maps p to a non-negative real

number. In the following, for clarity, if the context is clear, we simply write $f(\cdot)$ as f.

Let $\mathcal{F}$ be the set of all possible utility functions. In this paper, following the existing studies about top-k queries [27, 12, 17, 35, 10, 24], we assume that $\mathcal{F}$ contains an infinite number of functions and is thus uncountable. In case that $\mathcal{F}$ is countable, the problem we are studying can be easily solved. For the sake of space, we include this discussion in Appendix A.

We denote Θ to be the distribution on these utility functions in $\mathcal{F}$. Specifically, for each function $f \in \mathcal{F}$, we define $\eta(f)$ to denote the "importance" of f based on the distribution Θ. $\eta(f)$ has different meanings in different scenarios. $\eta(f)$ can denote the proportion of users with the utility function as f for a whole *group* of users. $\eta(f)$ can also denote the probability that f is the utility function for a *particular user*. In the following, we assume that $\eta(f)$ has the former meaning for the ease of discussion.

Note that $\eta(f)$ is a probability density function and $\int_{f\in\mathcal{F}} \eta(f)df = 1$.

Given a tuple $p \in D$ and a function $f \in \mathcal{F}$, p is said to *achieve the highest utility* with respect to f if for each $q \in D$, $f(p) \geq f(q)$. If a user has this utility function f, then p will be selected by this user. Given a tuple $p \in D$, we define $F(p)$ to be the set of all functions in $\mathcal{F}$ such that p achieves the highest utility with respect to each of these functions. In some cases, there exists a utility function $f \in \mathcal{F}$ such that multiple distinct tuples in D achieve the highest utility with respect to f. For the sake of discussion, in the following, we simply break the ties by regarding that only one of these tuples achieves the highest utility. Under this strategy, we want to make sure that each user finally selects one tuple (instead of multiple tuples) in order to simplify our discussion.

Consider a user in the whole group of users. We do not know any information about the exact utility function of this user. We can determine the probability that a tuple $p \in D$ is selected by the user according to the distribution Θ, called the *standalone hit probability*. Given a tuple $p \in D$, the *standalone hit probability* of p (in short, standalone HP), denoted by $HP(p)$, is defined to be the probability that p is selected by a user. For clarity, in the paper, we simply write "standalone HP" as "HP' if the context is clear. Since in the distribution Θ, we know the proportion of users with each utility function in $\mathcal{F}$, we have the following.

$$HP(p) = \int_{f\in F(p)} \eta(f)df$$

We just defined the standalone hitting probability of a *tuple*. Next, we define the *collective hitting probability* of a *set* of tuples. Given a set S of tuples, the *collective hit probability* of S (in short, collective HP), denoted by $HP(S)$, is defined to be the probability that one of the tuples in S is selected by a user. Similarly, for clarity, in the paper, we simply write "collective HP" as "HP" if the context is clear. Since we know that each user selects only one tuple finally, we have the following.

$$HP(S) = \sum_{p\in S} HP(p) \quad (1)$$

In this paper, we study a new query called the *k-hit query*.

Problem 1 (k-Hit Query). *Given a positive integer k, we want to find a subset S of D containing k tuples such that $HP(S)$ is the greatest.*

The following lemma shows that the top-k query is a special case of our k-hit query where $k = 1$.

Lemma 1. *When $\mathcal{F}$ contains only one utility function, our k-hit query becomes the top-k query where $k = 1$.*

The following shows the lower bound of the optimal hit probability of a k-hit query.

Lemma 2. *Let M be the number of tuples in D which achieve the highest utility with respect to at least one utility function in $\mathcal{F}$. Let S be the optimal solution of a k-hit query. $HP(S) \geq \min\{\frac{k}{M}, 1\}$.*

According to this lemma, we know that if k is larger and M is smaller, the lower bound of the hit probability of the optimal solution of a k-hit query will be larger.

Next, we consider two types of the function class for our k-hit query. The first type of the function class is the *linear utility function class*. The reason why we study this function class is that there are a lot of existing studies [36, 5, 7, 20, 3, 6, 27, 12, 17, 35] considering that the utility functions are in the linear form. The second type is the *non-linear utility function class* which is more general.

Next, in Section 3, we give some theoretical properties when the linear utility function class is considered. In Section 4, we present our proposed algorithm called *k-hit_Alg* when the utility function class is either linear or not.

3. PROPERTIES OF LINEAR UTILITY FUNCTIONS

The *linear function class*, denoted by $\mathcal{L}$, is defined to be the set of all possible *linear* functions. Given a tuple $p \in D$, a linear function $f(p)$ is represented in the form of $f(p) = \sum_{i=1}^{d} \omega_i \cdot p[i]$ where ω_i is a non-negative real number denoting the importance of the i-th dimension in the function for each $i \in [1, d]$. For each $i \in [1, d]$, ω_i is called the *weight* of the i-th dimension in the function. Thus, each linear function is associated with a d-dimensional weight vector.

In general, there exists two distinct utility functions in $\mathcal{L}$ returning the same ranking result based on a set of tuples (or points) in the d-dimensional space. For example, when $d = 2$, the vector $\omega = (0.3, 0.7)$ and the vector $\omega' = (3, 7)$ can be regarded as the same utility function in terms of the ranking result. So, we would like not to include both ω and ω' simultaneously in $\mathcal{L}$ at the end.

Based on the above observation, we introduce one more constraint such that $\mathcal{L}$ is *concise* and *complete*. A function class $\mathcal{L}' \subseteq \mathcal{L}$ is said to be *concise* with respect to $\mathcal{L}$ if and only if any two different utility functions in $\mathcal{L}'$ do not return the same ranking result on any set of the d-dimensional points. $\mathcal{L}'$ is said to be *complete* with respect to $\mathcal{L}$ if and only if for each utility function $f \in \mathcal{L}$, there exists $f' \in \mathcal{L}'$ such that f and f' return the same ranking result on any set of the d-dimensional points. Interestingly, the following lemma shows that the function class containing all possible linear functions whose weight vectors have their norm equal to 1 is concise and complete with respect to $\mathcal{L}$.

Lemma 3. *Let $\mathcal{L}'$ the function class containing all possible linear functions whose weight vectors have their norm equal to 1. $\mathcal{L}'$ is concise and complete with respect to $\mathcal{L}$.*

In the following, for clarity, we just focus on the function class containing all possible linear functions whose weight vectors have their norm equal to 1 and we refer this function class simply as $\mathcal{L}$.

In the following, we first present the geometry properties of our problem in Section 3.1, describing how it is relevant to a concept called "*solid angle*" [13, 26] in the literature of geometry. In Section 3.2, we describe the relationship between the k-hit query and each of the existing database problems: (1) top-k queries [27, 12, 17, 35, 10, 24], (2) skyline queries [3, 12, 17, 6, 18, 19, 29], (3) k-regret queries [20] and (4) k-representative skyline queries [29].

3.1 Geometric Properties

In this section, we present some theoretical properties based on the concept of "solid angle" used in the literature of geometry. Then, we introduce a new concept called "hitting solid angle" based on "solid angle" for our k-hit query. This new concept will be used in our algorithm.

3.1.1 Fundamental Concepts & Properties

Before we present some theoretical properties, we first introduce some fundamental concepts.

For each $i \in [1, d]$, we define the *boundary point* for the i-th dimension, denoted by BP_i, to be a d-dimensional point where its i-th dimension is equal to 1 and each of its other dimensions is equal to 0. Let $\mathcal{B} = \cup_{i=1}^{d} BP_i$.

We denote $Conv(D)$ to be the convex hull of $D \cup \{(0, 0, ..., 0)\} \cup \mathcal{B}$ where $(0, 0, ..., 0)$ is the origin of the d-dimensional space. For example, Figure 1 shows a dataset D containing 7 tuples, namely $p_1, p_2, ..., p_7$, in a 2-dimensional space. The two dimensions are X_1 and X_2. In this dataset, the X_2 value of p_1 is 1 and the X_1 value of p_6 is 1 Besides, since p_1 has its X_1 value equal to 0 and its X_2 value equal to 1, p_1 is exactly the boundary point for the second dimension (i.e., BP_2). However, although p_6 has its X_1 value equal to 1, p_6 is not the boundary point of the first dimension because p_6 has its X_2 value equal to non-zero. In the figure, O denotes the origin. Figure 2 shows $Conv(D)$. $Conv(D)$ contains the face/line containing O and p_1, the line containing O and point BP_1, the line containing p_i and p_{i+1} for each $i \in [1, 5]$, and the line containing p_6 and BP_1. Note that the line containing p_6 and BP_1 is a vertical line since both p_6 and BP_1 have their X_1 value equal to 1. Given a face of a convex hull, the face is said to be *boundary* if it contains the origin. For example, the face/line containing O and p_1 is boundary but the line containing p_1 and p_2 is non-boundary. If a point is on one of the faces of $Conv(D)$, we say that this point is on the *surface* of $Conv(D)$.

A point p on the surface of $Conv(D)$ is said to be *redundant* if p is a non-boundary point and the surface of $Conv(D)$ (i.e., the set of all faces of $Conv(D)$) is equal to the surface of $Conv(D \setminus \{p\})$ (i.e., the set of all faces of $Conv(D \setminus \{p\})$. For example, in our running example, since p_1 is a boundary point, it is not redundant. Since p_2 is a not-boundary point and the surface of $Conv(D)$ is not equal to the surface of $Conv(D \setminus \{p_2\})$, it is not redundant. Given a convex hull CH, we denote $Surf(CH)$ to be the set of points on the surface of CH which are not redundant. For example, p_1 and p_2 are on the surface of $Conv(D)$ and thus $Surf(Conv(D))$ contains p_1 and p_2. In the following, for clarity, when we write that "a point on the surface of a convex hull CH", we mean that "a point on the surface of a convex hull CH which is not redundant".

We have the following properties based on $Conv(D)$.

LEMMA 4. *Let p be a tuple in D which is inside $Conv(D)$ but is not on the surface of $Conv(D)$. Then, $HP(p) = 0$.*

In Figure 2, p_7 is a tuple in D which is inside $Conv(D)$ but is not on the surface of $Conv(D)$. By Lemma 4, $HP(p_7) = 0$.

LEMMA 5. *Let $Y = Surf(Conv(D))$. Then, for any set $S \subseteq D$ such that $Y \subseteq S$, $HP(S) = 1$.*

In Figure 2, p_1, p_2, p_3, p_4, p_5 and p_6 are all the tuples which are on the surface of $Conv(D)$. If $S = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, then by Lemma 5, we have $HP(S) = 1$.

Note that both Lemma 4 and Lemma 5 are independent of Θ (i.e., the distribution for the utility function). These two lemmas suggest that it is sufficient to consider the tuples on the surface of $Conv(D)$ for the solution of a k-hit query. Let $Y = Surf(Conv(D))$. If Y contains at most k tuples, the final solution S of a k-hit query must contain all tuples in Y. In this case, $HP(S) = 1$ no matter what tuples other than the tuples in Y are included in S. If Y contains more than k tuples, the final solution S of a k-hit query must contain k tuples from Y.

Besides, based on Lemma 2, we derive the following corollary when the function class is the linear function class $\mathcal{L}$.

COROLLARY 1. *Let $M = |Surf(Conv(D))|$. Let S be the optimal solution of a k-hit query. Then, $HP(S) \geq \min\{\frac{k}{M}, 1\}$.*

In the following two subsections, we present concepts of "solid angle" and "hitting solid angle" which will be used in our algorithm when Θ is a uniform distribution on the linear utility function class $\mathcal{L}$. In the following, we assume that Θ is a uniform distribution on $\mathcal{L}$.

3.1.2 Solid Angle

Next, we introduce a concept called "cone" and then define "solid angle".

Let V be a set of d' vectors from the origin O in the d-dimensional space where $d' \geq d$. We denote $Cone(V)$ to be an unbounded d-dimensional *convex cone* such that its apex is centered at the origin O, the surface of this cone is a combination of multiple unbounded faces, each of which is enclosed by two vectors in V and the height of this cone is infinite. For example, Figure 4 shows two vectors v_1 and v_2 in a 2-dimensional space. Let $V = \{v_1, v_2\}$. $Cone(V)$ is a 2-dimensional object such that it contains two lines, namely the line starting from O with the direction of vector v_1 and the other line starting from O with the direction of vector v_2. Since this object is unbounded (which means that the convex cone has no base), the lengths of these 2 lines are infinity. The space enclosed by these 2 lines corresponds to $Cone(V)$.

Given a convex cone C, we denote the *cone vector set* of C, denoted by $ConeVector(C)$, to be the set V of all vectors used to construct C such that $Cone(V) = C$. For example, Figure 4 shows the shaded region denoting a convex cone C. $ConeVector(C) = \{v_1, v_2\}$.

Let $\mathbf{B}_d$ be a closed d-dimensional hypersphere/ball centered at the origin with radius equal to 1 in a d-dimensional

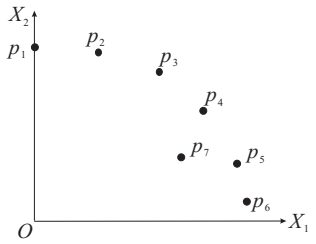


Figure 1: A running example

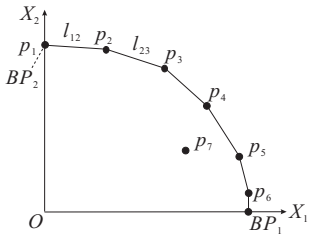


Figure 2: The convex hull of D ($Conv(D)$)

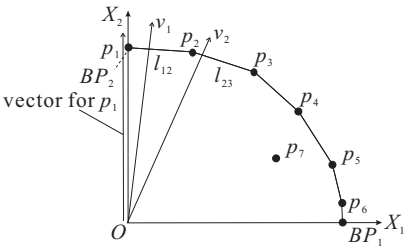


Figure 3: Vectors perpendicular to two faces of $Conv(D)$

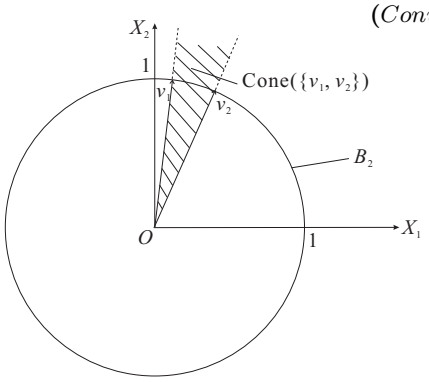


Figure 4: Unbounded Convex Cone in a 2-dimensional Space

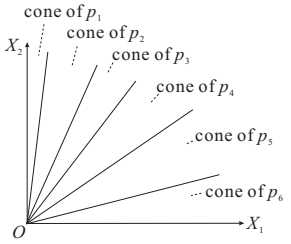


Figure 5: Partitions in a 2-dimensional Space

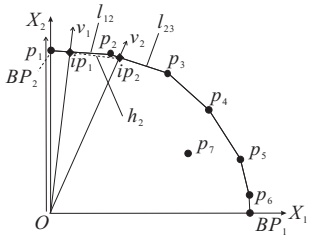


Figure 6: An Illustration of Indexing Method

space. Given an object Z in a d-dimensional space, the *volume* of Z is denoted by $Vol_d(Z)$. Given $\mathbf{B}_d$ and an unbounded d-dimensional convex cone C with its apex centered at the origin O, the *volume* of C with respect to $\mathbf{B}_d$ is defined to be $\frac{Vol_d(C \cap \mathbf{B}_d)}{Vol_d(\mathbf{B}_d)}$.

DEFINITION 1 (SOLID ANGLE). *Given a set V of d' vectors in a d-dimensional space where $d' \geq d$, the* solid angle *for V, denoted by $SA(V)$, is defined to be the volume of $Cone(V)$ with respect to $\mathbf{B}_d$.*

For example, Figure 4 shows $\mathbf{B}_2$ in a two-dimensional space. Let C be the unbounded convex cone enclosed by the two lines for v_1 and v_2 passing through the origin. We know that the volume of C with respect to $\mathbf{B}_2$ is the shaded area inside $\mathbf{B}_2$ divided by the area of $\mathbf{B}_2$.

We have just described the concept of "solid angle". Now, we are ready to define "hitting solid angle" for our k-hit query as follows.

Given a tuple $p \in Surf(Conv(D))$, we denote $Face(p)$ to be the set of all non-boundary faces of $Conv(D)$ containing p. In Figure 2, since $p_2 \in Surf(Conv(D))$, $Face(p_2)$ contains two faces/lines only. The first one is the line l_{12} with two end-points equal to p_1 and p_2, and the second one is the line l_{23} with two end-points equal to p_2 and p_3. That is, $Face(p_1) = \{l_{12}, l_{23}\}$. Similarly, $Face(p_1) = \{l_{12}\}$.

Given a tuple $p \in Surf(Conv(D))$, we denote $FaceVector(p)$ to be the set of the unit vectors perpendicular to all faces in $Face(p)$. That is, $FaceVector(p) = \{v | v \text{ is a unit vector perpendicular to } f \text{ where } f \in Face(p)\}$. For example, in Figure 3, $FaceVector(p_2) = \{v_1, v_2\}$ and $FaceVector(p_1) = \{v_1\}$.

Given a tuple $p \in Surf(Conv(D))$, we define the *hit vector set* of p, denoted by $HitVector(p)$, to be $FaceVector(p)$ if p is a non-boundary point, and $FaceVector(p) \cup \{p\}$ otherwise. Note that the point p itself can be regarded as a vector from the origin to point p. We simply write p to denote the vector from the origin to point p if the context is in terms of vectors instead of points. For example, consider p_2. Since p_2 is a non-boundary point, $HitVector(p_2) = FaceVector(p_2) = \{v_1, v_2\}$. Consider p_1. Since p_1 is a boundary point, $HitVector(p_1) = FaceVector(p_1) \cup \{p_1\} = \{v_1, p_1\}$. Note that p_1 in $HitVector(p_1)$ is a vector from the origin to point p_1 (denoted as the "vector for p_1" in Figure 3).

LEMMA 6. *We are given a tuple $p \in Surf(Conv(D))$. Then, $HP(p) = 2^d \cdot SA(HitVector(p))$.*

In Figure 3, consider p_2. $Face(p_2) = \{l_{12}, l_{23}\}$ and $FaceVector(p_2) = \{v_1, v_2\}$. By Lemma 6, $HP(p_2) = 2^2 \cdot SA(FaceVector(p_2))$.

3.1.3 Hitting Solid Angle

Next, we define "hitting solid angle" and re-define "cone" in our context of the k-hit query.

Given a tuple $p \in Surf(Conv(D))$, the *hitting solid angle for p* is defined to be $SA(HitVector(p))$. Besides, given a tuple $p \in Surf(Conv(D))$, we define the *cone* of p to be $Cone(HitVector(p))$. For example, consider our running example as shown in Figure 3. in Figure 4, the shaded region is the cone of p_2 (i.e., $Cone(HitVector(p_2))(= Cone(\{v_1, v_2\}))$). Figure 5 shows the cone of p_i for each $i \in [1, 6]$ in our running example.

Let $\mathcal{S}$ be the d-dimensional non-negative real space.

OBSERVATION 1. *Let $R = |Surf(Conv(D))|$. $\mathcal{S}$ is divided into R partitions where each partition is the cone of a tuple in $Surf(Conv(D))$.*

Consider our running example. Figure 5 shows an example that the space $\mathcal{S}$ is divided into 6 partitions where each partition is the cone of tuple p_i for $i \in [1, 6]$.

3.2 Relevance to Existing Database Queries

We know that a traditional top-k query is a special case of our k-hit query where $k = 1$ as described in Lemma 1. The following lemma shows that the hit probability of the set of skyline tuples is equal to 1 and the optimal set of a k-hit query is a subset of the set of skyline tuples.

LEMMA 7. *Let S be the set of skyline tuples and S_o be the optimal set of a k-hit query. Then, $HP(S) = 1$. Besides, if $|Surf(Conv(D))| \geq k$, then $S_o \subseteq S$.*

The following two lemmas suggest that a k-regret query [20] and a k-representative skyline query [29], two existing queries returning k tuples recently proposed in the literature, do not give accurate solutions for our k-hit query.

LEMMA 8. *There exists a dataset such that S is the solution of a k-regret query and $HP(S) = 0$.*

LEMMA 9. *There exists a dataset such that S is the solution of a k-representative skyline query and $HP(S) = 0$.*

4. ALGORITHM

In this section, firstly, we give a general framework of our proposed algorithm *k-hit_Alg* for the k-hit query in Section 4.1. Secondly, due to some interesting properties of the linear utility function class described in Section 3, in Section 4.2, we present our algorithm *k-hit_Alg* using these properties when the utility functions used are linear. Thirdly, in Section 4.3, we give *k-hit_Alg* when the utility functions considered are non-linear.

4.1 Overview

A k-hit query is to find a set S containing k tuples in D such that $HP(S)$ is the greatest. According to Equation (1), we know that $HP(S)$ is the sum of the standalone hit probabilities of all tuples in S. Thus, we present an overview framework of our proposed algorithm *k-hit_Alg* for the k-hit query as follows.

- **Step 1 (Standalone HP Computation):** For each tuple p in D, we compute the standalone hit probability of p (i.e., $HP(p)$) based on Θ.
- **Step 2 (Output):** We find a set S of k tuples in D with the greatest standalone hit probabilities

It is easy to verify the correctness of *k-hit_Alg*.

THEOREM 1. *The general framework of* k-hit_Alg *returns the optimal solution of a k-hit query.*

This overview framework looks straightforward. However, there are two major issues. The first issue is how to compute the standalone hit probability of each tuple (used in Step 1). The second issue is how to use the interesting properties described in Section 3 when the utility functions used are linear. In the following two sub-sections, we present the detailed description of this overview framework.

4.2 Linear Utility Function Class

In this section, we present our detailed algorithm *k-hit_Alg* for the k-hit query when the utility functions considered are linear.

Algorithm 1 Our Detailed Algorithm *k-Hit_Alg* (for Uniform Distribution Θ on Linear Utility Function Class)

Input: A set of n d-dimensional points $D = \{p_1, p_2, ..., p_n\}$, a positive integer k
Output: A subset S of D of size k
1: // Step 1 (Convex Hull Construction)
2: Compute $Conv(D)$
3: Find $Surf(Conv(D))$
4: // Step 2 (Standalone HP Computation)
5: **for** each point $p \in Surf(Conv(D))$ **do**
6: Compute the hitting solid angle for p (i.e., $SA(HitVector(p))$)
7: Compute $HP(p)$ which is equal to $2^d \cdot SA(HitVector(p))$
8: // Step 3 (Output)
9: $S \leftarrow$ the set of the k tuples in $Surf(Conv(D))$ with the greatest standalone hit probabilities
10: **return** S

When the utility functions considered are linear, due to the properties described in Section 3, the detailed algorithm *k-hit_Alg* to be described in this section has the following two differences. The first difference is that in the detailed algorithm, at the beginning, we introduce an additional step of finding a small set of candidate tuples for the optimal solution of a k-hit query. In Section 3, based on Lemma 4 and Lemma 5, we know that all tuples in $Surf(Conv(D))$ are candidate tuples for a k-hit query. Thus, in our detailed algorithm, we consider all tuples in $Surf(Conv(D))$ only instead of the whole dataset D considered in the overview framework.

Specifically, our detailed algorithm *k-Hit_Alg* with the first difference has the following three steps.

- **Step 1 (Convex Hull Construction):** We find $Surf(Conv(D))$ from D (because the tuples in $Surf(Conv(D))$ are candidates for the optimal solution for a k-hit query as described before).
- **Step 2 (Standalone HP Computation):** For each $p \in Surf(Conv(D))$, we compute the standalone hit probabilities of p based on Θ.
- **Step 3 (Output):** We find a set S of the k tuples in $Surf(Conv(D))$ with the greatest standalone hit probabilities

The second difference is related to how to compute the standalone hit probabilities of tuples based on the concept of "hitting solid angle". We consider two cases.

- Case 1 (Θ is a Uniform Distribution on $\mathcal{L}$)
- Case 2 (Θ is not a Uniform Distribution on $\mathcal{L}$)

The concept of "hitting solid angle" can be used in Case 1 (due to the simplest uniform distribution) but it cannot be used in Case 2.

Case 1 (Θ is a Uniform Distribution on $\mathcal{L}$): In this case, based on Lemma 6, we determine the standalone hit probability of a tuple by computing the hitting solid angle for this tuple. Thus, the implementation of Step 2 is described as follows. For each $p \in Surf(Conv(D))$, we compute the hitting solid angle for p, $SA(HitVector(p))$, (i.e., the solid angle for the hit vector set $HitVector(p)$) and then determine the standalone hit probability of p (i.e., $HP(p)$) which is equal to $2^d \cdot SA(HitVector(p))$ (by Lemma 6).

Algorithm 1 shows the pseudo-code of our detailed algorithm *k-Hit_Alg* in this case.

Based on Lemma 6 and Equation (1), it is easy to verify the correctness of the algorithm.

THEOREM 2. *Algorithm 1 returns the optimal solution of a k-hit query.*

In Algorithm 1, we know that we have to compute the hitting solid angle for each tuple p in $Surf(Conv(D))$, which involves the computation of the solid angle for a set of vectors in $HitVector(p)$. Since it is known that computing the solid angle for a set of d-dimensional vectors is costly [13, 26] and many existing techniques finding the solid angle are sampling-based methods [31, 33, 13], in this paper, we also adopt a sampling-based method for computing the solid angle for a set of vectors (i.e., the hitting solid angle for a tuple).

Before we present this sampling-based method, we give the following lemma which will be used in the method.

LEMMA 10. *Let V be a set of N randomly sampled d-dimensional non-negative real vectors of unit length. Let $V(p)$ be the set of sampled vectors in V which are inside the cone of p. Then, when N approaches ∞, for each tuple p in D, the hitting solid angle for p is $\frac{|V(p)|}{N} \cdot \frac{1}{2^d}$ and $HP(p) = \frac{|V(p)|}{N}$.*

Lemma 10 suggests that we can compute the standalone hit probability of each tuple p in D by finding the size of $V(p)$ and obtaining the hit probability with a division of this size by N.

Specifically, based on Lemma 10, we propose the following steps for computing the hitting solid angles for all tuples in $Surf(Conv(D))$ (used in Step 2).

- **Step (a) (Sampling):** We randomly pick N d-dimensional non-negative real vectors of unit length in $\mathcal{S}$ where N is a parameter for the sampling size and can be determined from some error parameters (which will be described later in Section 4.2.1).
- **Step (b) (Cone Location):** For each vector v from the N vectors generated, we determine which tuple p in $Surf(Conv(D))$ has its cone containing v and then increment variable $count(p)$, initialized to 0, by 1
- **Step (c) (Standalone HP Estimation):** For each $p \in Surf(Conv(D))$, the hitting solid angle for p is estimated to be $\frac{count(p)}{N} \cdot \frac{1}{2^d}$. The estimated $HP(p)$ is equal to $\frac{count(p)}{N}$.

Algorithm 2 shows the pseudo-code of the implementation of our detailed algorithm *k-Hit_Alg*.

Case 2 (Θ is Not a Uniform Distribution on $\mathcal{L}$): In this case, we do not determine the standalone hit probability of each tuple by computing the hitting solid angle of this tuple (because Θ is not a uniform distribution on $\mathcal{L}$ and thus the concept of "hitting solid angle" cannot be used). Instead, we compute the standalone hit probability of each tuple based on the distribution Θ in the following way.

- **Step (a) (Sampling):** We randomly pick N d-dimensional non-negative real vectors of unit length in $\mathcal{S}$ based on distribution Θ where N is a parameter for the sampling size and can be determined from some error parameters (which will be described later in Section 4.2.1)

Algorithm 2 Implementation of Our Detailed Algorithm *k-Hit_Alg* (for Linear Utility Function Class)

Input: A set of n d-dimensional points $D = \{p_1, p_2, ..., p_n\}$, a sampling size N, a positive integer k.
Output: A subset S of D of size k
1: // Step 1 (Convex Hull Construction)
2: Compute $Conv(D)$
3: Find $Surf(Conv(D))$
4: // Step 2 (Standalone HP Estimation)
5: Sample N d-dimensional non-negative real vectors of unit length
6: **for** each vector v from the N vectors generated **do**
7: we determine which tuple p in $Surf(Conv(D))$ has its cone containing v and then increment variable $count(p)$, initialized to 0, by 1
8: // Step 3 (Output)
9: $S \leftarrow$ the set of the k tuples in $Surf(Conv(D))$ with the greatest $\frac{count(\cdot)}{N}$ values
10: **return** S

- **Step (b) (Cone Location):** For each vector v from the N vectors generated, we determine which tuple p in $Surf(Conv(D))$ has its cone containing v and then increment variable $count(p)$, initialized to 0, by 1
- **Step (c) (Standalone HP Estimation):** For each $p \in Surf(Conv(D))$, $HP(p)$ is estimated to be $\frac{count(p)}{N}$.

The pseudo-code of this algorithm is exactly the same as Algorithm 2 except that in this case, sampling N vectors is based on distribution Θ instead of a uniform distribution. For the ease of naming, we also call this algorithm as *k-Hit_Alg*.

In this section, we give some details of *k-hit_Alg*. All steps in Algorithm 2 are straightforward but there are two issues. The first issue is how to determine the sampling size N used in Line 5 of Algorithm 2 (Section 4.2.1). The second issue is how to determine which tuple p has its cone containing a sampled vector v efficiently (Section 4.2.2).

4.2.1 Issue 1: Determining Parameter N

In this section, we give a way to determine the sampling size N.

Let S be the solution set returned by our algorithm (no matter what distribution are considered). Let S_o be the optimal solution of a k-hit query. Let s_i be the i-th largest estimated standalone HP value of our algorithm for each $i \in [1, k]$.

THEOREM 3. *Given a confidence parameter $\delta \in [0, 1]$ and an error parameter $\epsilon \in [0, 1]$, if the sampling size N is at least $\frac{8 s_1 k^2 \ln 2/(\delta/2)^{\frac{1}{k}}}{\epsilon^2}$, then with confidence at least $1 - \delta$,*

$$|HP(S) - HP(S_o)| \leq \epsilon$$

Based on this theorem, we know that given two user-given parameters δ and ϵ, we can determine the minimum sampling size N as $\frac{8 s_1 k^2 \ln 2/(\delta/2)^{\frac{1}{k}}}{\epsilon^2}$ (since s_1 can be found from the dataset and k is given).

4.2.2 Issue 2: Determining Which Tuple Has Its Cone Containing a Sampled Vector Efficiently

In our proposed algorithm, *k-Hit_Alg*, we have to execute Step (b) (Cone Location). In this section, we present an in-

dexing technique which executes Step (b) efficiently. Specifically, given a sampled vector v, we want to determine which cone contains v.

A straightforward method is to check whether v is inside each cone one-by-one and then to return the cone containing v. Consider a cone C. Determining whether v is inside C is equivalent to determining whether v is a *convex combination*[1] [9] of all vectors in $ConeVector(C)$. It is known [9] that determining whether a given vector is a convex combination of a set X of r vectors in a d-dimensional space can be done in $O(r^2)$ time. Let $|H|$ be the total number of cones and r be the greatest number of vectors in the cone vector set of a cone (i.e., $ConeVector(\cdot)$). Thus, the time complexity of this straightforward method is $O(|H|r^2)$ time.

In the following, we propose to use an index built on all cones in the dataset and thus we can determine which cone contains a given sample vector v in $O(\log |H|)$ time.

Before we introduce our method, we first describe a *ray shooting query* [1], which is used in our method.

DEFINITION 2 ([1]). *Let H be a set of M hyperplanes and $P(H)$ be its polytope. Given a ray ρ from a point x towards a direction where $x \in P(H)$, a* ray shooting query *is to find the first hyperplane $h \in H$ hit by ρ.*

For example, in Figure 3, the polytope is referred to the convex hull $Conv(D)$. Except the line containing O and p_1 and the line containing O and BP_1, the hyperplanes on the surface of $Conv(D)$ are the line containing p_i and p_{i+1} for each $i \in [1, 5]$ and the line containing p_6 and BP_1. When a ray ρ from the origin O with a direction v_1 is the input of a ray shooting query, the answer for the query is the line containing p_1 and p_2.

It is known [1] that given a set H of hyperplanes in a d-dimensional space, a ray shooting query for the polytope $P(H)$ can be answered in $O(\log |H|)$ time [1].

In our problem setting, we can adopt an existing technique of the ray shooting query for the step of cone location. Specifically, we have the following steps.

- **Step 1 (Hyperplane Set Construction):** for each tuple $p \in Surf(Conv(D))$, we find $HitVector(p)$ and then, for each $u \in HitVector(p)$, we find the intersection point ip between u and the surface of $Conv(D)$. Thus, each tuple $p \in Surf(Conv(D))$ is associated with a set of intersection points. For example, in Figure 6, since $HitVector(p_2) = \{v_1, v_2\}$, the 2 intersection points are ip_1 and ip_2. p_1 is associated with ip_1 and ip_2. We know that there are at least d intersection points. If there are exactly d intersection points, then we construct a hyperplane based on these d intersection points and insert it into a variable H, initialized to $\emptyset$. If there are more than d intersection points, then we randomly pick d intersection points which can construct a hyperplane. In this case, we construct a hyperplane based on these d points and insert it into H. For instance, in Figure 6, tuple p_2 is associated with two intersection points, namely ip_1 and ip_2. Since there are two intersection points, the hyperplane/line h_2 is constructed based on these two points. Note that finally, the size of H is equal to $|Surf(Conv(D))|$. In conclusion, each tuple $p \in Surf(Conv(D))$ has a corresponding hyperplane in H.
- **Step 2 (Index Construction):** We build an index according to [1].
- **Step 3 (Ray Shooting Query):** Given a sampled vector v, we issue a ray shooting query with the input as v and determine which hyperplane h in H hit by the ray. The corresponding tuple p of the hyperplane h is returned as an output. This tuple is the tuple whose cone contains v.

It is easy to verify the correctness of the above algorithm.

4.3 Non-Linear Utility Function Class

In Section 3, we described our detailed algorithm *k-Hit_Alg* when the utility function class is the linear utility function class. In this section, we describe our detailed algorithm *k-Hit_Alg* when the utility function class is the non-linear utility function class.

4.3.1 Theoretical Properties

Before we present the algorithm, we first give the following two lemmas which will be used in the algorithm in the context of the non-linear utility function class.

LEMMA 11. *Let $Y = Surf(Conv(D))$. Let p be a tuple in D but not in $Surf(Conv(D))$. It is possible that $HP(p) \neq 0$ and the optimal solution of a k-hit query contains p.*

Lemma 11 suggests that some tuples not in $Surf(Conv(D))$ can be candidates under the non-linear function class case.

LEMMA 12. *Let V be a set of N randomly sampled d-dimensional non-negative real vectors of unit length. Let $V(p)$ be the set of sampled vectors in V such that for each vector v in this set, tuple p has the greatest value of $p \cdot v$. Then, when N approaches ∞, for each tuple p in D, $HP(p) = \frac{|V(p)|}{N}$.*

Lemma 12 suggests that we can compute the standalone hit probability of each tuple p in D by finding the size of $V(p)$ and obtaining the hit probability with a division of this size by N.

4.3.2 Algorithm

Based on Lemma 11 and Lemma 12, we modify Algorithm 2 (our detailed algorithm *k-Hit_Alg* for the linear utility function class) with the following three changes for the non-linear function class case. The first two changes are due to Lemma 11 and the last change is due to Lemma 12. The first change is to remove Lines 1-3 of Algorithm 2 (which correspond to the additional step of finding the candidates from $Surf(Conv(D))$). The second change is to replace each occurrence of $Surf(Conv(D))$ by D. The third change is to replace Line 7 of Algorithm 2 (which corresponds to determining which tuple p in $Surf(Conv(D))$ has its contains a sampled vector v and incrementing variable $count(p)$ by 1) with a similar operation of determining which tuple p in D has the greatest value of $p \cdot v$ (where v is a sampled vector)

[1]Given a vector v and a set X of vectors, namely $x_1, x_2, ..., x_r$, v is said to be a *convex combination* of X if there exists a non-negative real number α_i for each $i \in [1, r]$ such that $v = \sum_{i=1}^{r} \alpha_i x_i$.

Algorithm 3 Implementation of Our Detailed Algorithm *k-Hit_Alg* (for Non-Linear Utility Function Class)

Input: A set of n d-dimensional points $D = \{p_1, p_2, ..., p_n\}$, a sampling size N, a positive integer k.
Output: A subset S of D of size k
// Step 1 (Standalone HP Estimation)
Sample N d-dimensional non-negative real vectors of unit length
for each vector v from the N vectors generated **do**
    we determine which tuple p in D has the greatest value of $p \cdot v$ and then increment variable $count(p)$, initialized to 0, by 1
// Step 2 (Output)
$S \leftarrow$ the set of the k tuples in D with the greatest $\frac{count(\cdot)}{N}$ values
return S

and incrementing variable $count(p)$ by 1. The pseudo-code can be found in Algorithm 3.

Similar to the linear utility function class, for the non-linear utility function class, we can also determine the parameter N as described in Section 4.2.1.

5. RELATED WORK

We categorize the related work into two parts. The first part is to describe how our work is related to existing queries (Section 5.1). The second part is to describe the related work about utility function distribution mining (Section 5.2).

5.1 Existing Queries

Top-k queries [30, 14, 27, 34, 24, 32] are one kind of popular queries which were studied by many researchers. There are two branches of top-k queries. The first branch is top-k queries on *certain* databases. In this branch, as described in Section 1, a concrete utility function f is given. Each top-k query returns k tuples with the greatest utility values computed based on this concrete utility function f. Some representative papers are [30, 14]. As described before, these studies are different from us because they require a concrete utility function given by a user which is one of the utility functions in $\mathcal{F}$ studied by us.

The second branch is top-k queries on *uncertain* databases. Some representative papers are [27, 34, 24, 32]. To the best of our knowledge, most (if not all) existing studies in this branch focused on the uncertainty of the dataset D but did not study the uncertainty of the utility function space $\mathcal{F}$. Besides, they focused on the *possible world model* which is based on a *discrete* uncertain space but the uncertainty space we are considering (i.e., $\mathcal{F}$) is the utility function space and is continuous. Specifically, under this branch of these studies, each dataset D is uncertain. That is, each tuple in D has a probability that it is (was) present or not. Based on the tuple probabilities in D, these studies construct a possible world model and calculate some probabilities related to the top-k queries based on this possible world model. There are different criteria for outputting the output set of the top-k queries for different studies. For example, one query called *U-Topk* [27] is to return a set S of tuples such that the sum of the probabilities that S is contained in all possible worlds is maximized. Another query called *U-kRanks* [27] is to return a set S of tuples, namely $t_1, t_2, .., t_k$, such that for each $i \in [1, k]$, the sum of the probabilities that t_i is ranked at the i-th position in all possible words is maximized.

Although there are some studies [28, 11] about top-k queries studying the uncertainty of the utility values (called scores) of tuples, they are different from us. [28] considers that each tuple is associated with a score range in the form of (a, b) meaning that a is the smallest possible score value and b is the greatest possible score. However, the uncertainty considered in [28] comes from each individual tuple but the uncertainty considered in our work comes from the whole utility function space. Besides, the uncertainty in [28] is represented in a particular range form but the uncertainty in our work is very general. [11] enumerates all possible words and compute the sum of the scores of all tuples in the top-k query computed based on a fixed utility function. Finally, [11] constructs the distribution on the (score) sums and then returns a set of tuples based on this distribution. Obviously, [11] is different from us since [11] considers a single fixed utility function but we focus on a whole function utility function space $\mathcal{F}$.

Many existing studies [3] also investigated skyline queries. A tuple t is said to be *dominated* by another tuple t' if t is better than t' in at least one dimension and t is no worse than t' in each dimension. A tuple is said to be a *skyline tuple* in D if this tuple is not dominated by any other tuples in D. A skyline query is to return a set of all skyline tuples in D. Intuitively, a tuple in the output of the skyline query is a candidate tuple which is the favorite of a user with any monotonic utility function. However, it is found that the skyline query has its undesirable drawback of a large output size. In the worst case, all tuples in the dataset can form the output of a skyline query.

There are some variants of skyline queries which try to get rid of this drawback and introduce a parameter k, a positive integer, denoting the output size of a skyline query. Some examples are a *representative skyline query* (*RepSky*) [29] and a *dominating skyline query* (*DomSky*) [21]. Given a parameter k, *RepSky* is to find a set S of k tuples such that each tuple in S is a skyline tuple in D and the maximum Euclidean distance between each tuple in D and its closest tuple in S is minimized. Given a parameter k, *DomSky* is to find a set S of k tuples such that the *scores* of these k-tuples are the greatest where the score of a tuple is defined to be the number of tuples dominating by this tuple in D.

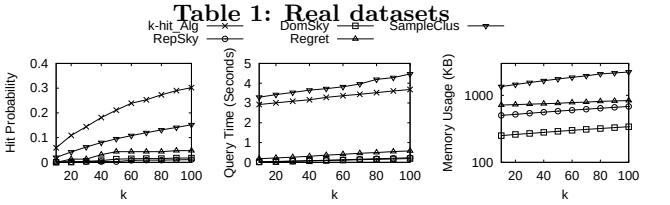
However, the above variants do not consider the distribution of utility functions even if the distribution is available. There are some other variants of skyline queries such as [18] which is to find a set of tuples such that the number of tuples dominated by at least one tuple in the selection set is maximized. But, they also do not consider the distribution.

Recently, there is a newly proposed query called a k-regret query [20, 22] which consider the utility function space $\mathcal{F}$. Given a parameter k, a k-regret query is to return a set S of k tuples such that the "difference" called the *maximum regret ratio* between the selection set S and the whole dataset D is minimized. Given a utility function f and a selection set S, the *regret ratio* of S with respect to f is defined to $1 - \frac{\max_{p \in S} f(p)}{\max_{p' \in D} f(p')}$. The *maximum regret ratio* between S and D is defined to be the maximum of the regret ratios of S with respect to all possible utility functions in $\mathcal{F}$. Though the k-regret query considers the utility function space $\mathcal{F}$ in the problem/query formulation, it does not consider the distribution on utility functions.

The closely related work is [36] called the *order-based representative skyline* problem considering the distribution of utility functions. As described in Section 1, the objective of the order-based representative skyline problem is differ-

Dataset	No. of Dim.	Size	$\lvert D_{conv}\rvert$	$\lvert H\rvert$
household-10d	10	1,103,241	753	14235
household-6d	6	903,077	927	10394
nba	5	21,962	65	2774
color	9	68,040	124	4813
stocks	5	122,574	396	5892

Table 1: Real datasets



(a) Hit Probability (b) Query Time (c) Memory Usage

Figure 7: Effect of k based on Synthetic Datasets

ent from ours though the distribution on utility functions is considered.

5.2 Utility Function Distribution Mining

The distribution on utility functions has been widely explored in the machine learning area, such as user's recommender systems [4, 25], Bayesian learning models [8, 15] and user's preference elicitations [2, 7].

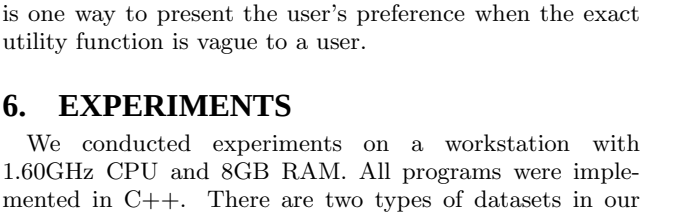
User's recommender systems [4, 25] give a way to predict the ratings that a user could give to different items (or tuples in our context). In general, two common approaches are adopted for recommendation. One is *content-based filtering* which models the description of the items and users' profiles to estimate the user's preference (or utility functions in our context) to each item, and the other is *collaborative filtering* which leverages the similarities between a user and other users to predict the user's ratings on each item without knowing the content of the items.

Bayesian learning models [8, 15] use the Bayes' rule to update the probability estimates when new information is obtained. By defining a prior probability on utility functions properly, the posterior probability on utility functions will also follow the same distribution with the updated parameters defined in the model so that it can be considered as the prior probability afterwards and the distribution on the utility function can be updated iteratively.

User's preference elicitations [2, 7] contain a lot of ways of representing a user's preference and design queries to obtain the users' preferences. The distribution on utility functions is one way to present the user's preference when the exact utility function is vague to a user.

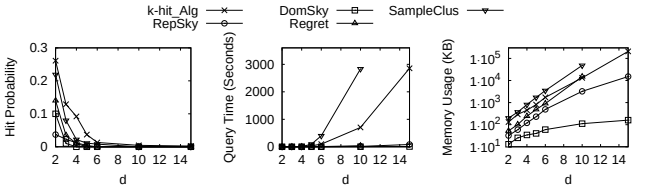
6. EXPERIMENTS

We conducted experiments on a workstation with 1.60GHz CPU and 8GB RAM. All programs were implemented in C++. There are two types of datasets in our experiments, namely *synthetic datasets* and *real datasets*.



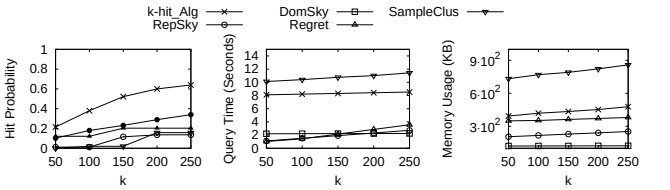
(a) Hit Probability (b) Query Time (c) Memory Usage

Figure 8: Effect of n based on Synthetic Datasets



(a) Hit Probability (b) Query Time (c) Memory Usage

Figure 9: Effect of d based on Synthetic Datasets



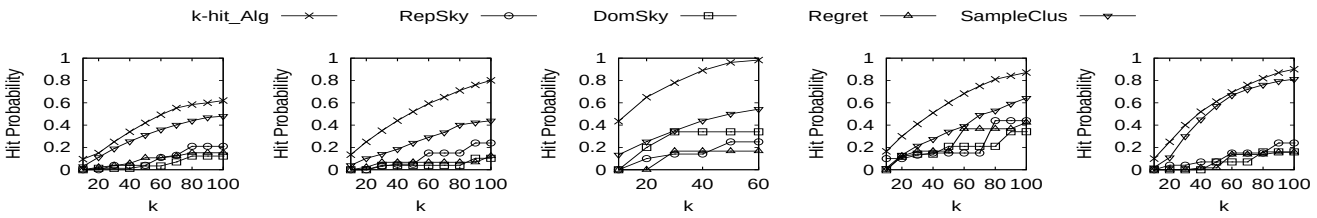
(a) Hit Probability (b) Query Time (c) Memory Usage

Figure 13: Effect of k on *Yahoo!Mustic* Datasets

The synthetic datasets were generated by the synthetic dataset generator for skyline operators developed by [3]. Unless stated explicitly, following [36], for synthetic datasets, we fix the number of tuples to be 1,000,000 (i.e., n = 1,000,000), the dimensionality to be 3 (i.e., $d = 3$), and k to be 10. For synthetic datasets, the distribution Θ we adopted is the uniform distribution on the linear utility function class.

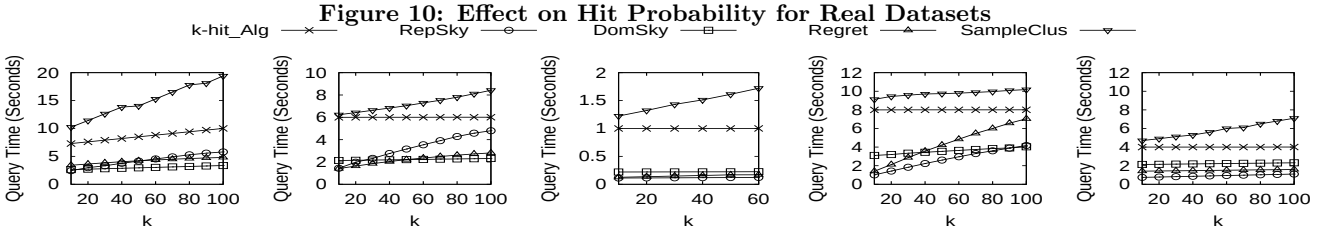
There are two categories of real datasets for experiments. The first category of real datasets contains the five datasets commonly used in the existing studies for skyline queries and top-k queries, namely *household-6d*, *household-10d* (http://www.ipums.org), *nba* (http://www.basketballreference.com), *color* (http://kdd.ics.uci.edu) and *stocks* (http://pages.swcp.com/stocks). The distribution Θ we adopted is also the uniform distribution on the linear utility function class. The number of dimensions and the data size of each of these real datasets can be found in Table 1. The second category of real datasets contains a large real dataset called the *Yahoo!music* dataset which was used for KDD-Cup 2011. From this dataset, we can obtain the distribution Θ on non-linear utility functions by existing machine learning techniques [16]. Details of how to obtain Θ will be described later in Section 6.2.2. Since the experiment for the second category considers the most general case that the utility functions considered are non-linear and the distribution is non-uniform, we do not conduct experiments on other distributions (e.g., non-uniform distribution on linear utility functions).

Since our k-hit query considers a utility function space $\mathcal{F}$ containing many utility functions and a traditional top-k query considers an exact utility function, we do not compare our proposed method (k-*hit_Alg*) with the algorithms originally designed for the top-k query. Instead, we compared the performance of our proposed algorithm (k-*hit_Alg*) with the state-of-the-art algorithms originally proposed for solving traditional queries which do not rely on an exact concrete utility function. They are (1) the algorithm *RepSky* [29] for a k-representative skyline query, (2) the algorithm *DomSky* [21] for a dominating skyline query, (3) the algorithm *Regret* [20] for a k-regret query and (4) the algorithm *SampleClus* [36] for an order-based k-representative skyline query/problem.



(a) household-10d (b) household-6d (c) nba (d) color (e) stocks

Figure 10: Effect on Hit Probability for Real Datasets



(a) household-10d (b) household-6d (c) nba (d) color (e) stocks

Figure 11: Effect on Query Time for Real Datasets

There are some parameter settings for these algorithms. The parameter k for all these algorithms is set to the same value in all of our experiments. There are no other parameters to be set in *RepSky*, *DomSky* and *Regret*. *k-hit_Alg* has two additional parameters related to sampling, namely ϵ and δ, to determine the sampling size N. By default, in *k-hit_Alg*, we set $\epsilon = 0.0001$ and $\delta = 0.1$. In *SampleClus*, in addition to parameter k, there are 5 parameters. The first two parameters are related to sampling similar to our algorithm *k-hit_Alg*. They are parameter ϵ and parameter δ (which are written as δ and ρ in the paper [36] with the same meaning, respectively). We set these parameters used in *SampleClus* exactly the same as the ones used in *k-hit_Alg*. The other three parameters are related to [36] only. The first two are parameters m and α which have been described in Section 1, and the third parameter is parameter β, a parameter introduced in the algorithm *SampleClus*, where m is a positive integer and α and β is a non-negative real number. In [36], parameter m is set to $|S|/k$ where $|S|$ is the number of skyline tuples in D. Parameter β is a new parameter denoting the minimum $\mathcal{SIM}_m(\cdot,\cdot)$ value between any two points in the selection set. Following [36], we set the default values of the two parameters α and β to 0.01 and 0.1, respectively.

For each algorithm, we measured with three measurements, namely (1) the hit probability of the solution returned, (2) the query time of the algorithm and (3) the memory consumption of the algorithm. The *query time* of an algorithm corresponds to the execution time of the algorithm without the *pre-processing* step. Specifically, the query time of each of *RepSky*, *DomSky*, *Regret* and *SampleClus* is its execution time excluding the pre-processing step of finding skyline tuples. The query time of *k-hit_Alg* is its execution time excluding the pre-processing step of finding the convex hull and building the index. The *memory consumption* of an algorithm is the maximum memory consumed by the algorithm when the algorithm is executed.

In the following, we conducted two groups of experiments. The first experiment (Section 6.1) is to study the performance of each algorithm on synthetic datasets. The second experiment (Section 6.2) is to study the performance of each algorithm on real datasets.

6.1 Results on Synthetic Datasets

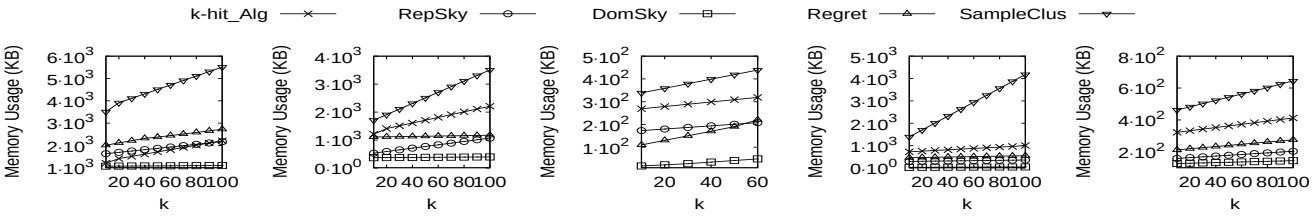
We conducted experiments on synthetic datasets. In particular, we study k (parameter k in our k-hit query), n (dataset size) and the effect of d (dimensionality) on our synthetic datasets.

Hit Probability: Figure 7(a), Figure 8(a) and Figure 9(a) show the hit probability returned by each algorithm when we vary k, n and d, respectively. In these figures, *k-Hit_Alg* has the greatest hit probability among all algorithms. *SampleClus* gives the second greatest one and the other algorithms gives very low probabilities. The reason why *k-hit_Alg* has the greatest hit probability among all algorithms is that the goal of *k-hit_Alg* is to maximize the hit probability but the goal of each of the other algorithms is not. *SampleClus* gives the second greatest hit probability. This is because *SampleClus* needs to compute the probability term and uses this term for calculating the similarity between the selection set and the dataset. Since the probability term is correlated to the hit probability we want to maximize and each of the other algorithms (i.e., *RepSky*, *DomSky* and *Regret*) do not compute the probability term, *SampleClus* returns a greater hit probability than each of the other algorithms. Note that *SampleClus* still returns a low hit probability compared with *k-hit_Alg* since the goal of *SampleClus* is not to maximize the hit probability. For example, in Figure 7(a), when $k = 10$, the hit probability returned by *SampleClus* is 5.6 times smaller than the hit probability returned by *k-hit_Alg*.

In Figure 7(a), the hit probability of the set returned by each algorithm increases with k because the output set contains more tuples and it is more likely that a user can find his/her favorite in the output set.

In Figure 8(a), when n increases, the hit probability of the set returned by an algorithm decreases. This is because there are more tuples and it is less likely that a user can find his/her favorite.

Figure 9(a) shows the effect of d where the total number of tuples is fixed to 100,000. When d increases, the hit probability of the set returned by each algorithm decreases. This is because it is less likely that a tuple is found to be a user's favorite when the dimensionality is larger.



(a) household-10d (b) household-6d (c) nba (d) color (e) stocks

Figure 12: Effect on Memory Usage for Real Datasets

Query Time: Figure 7(b), Figure 8(b) and Figure 9(b) show the query time of each algorithm when we vary k, n and d, respectively.

SampleClus, the closely related work, gives the greatest query time since it computes a probability term which is costly to be computed. The query time of *k-Hit_Alg* is the second greatest among all algorithms since *k-Hit_Alg* needs to compute the probability term but all other algorithms except *SampleClus* do not need.

Figure 7(b) shows that the query time of each algorithm increases with k because it needs more time for processing the data. Similarly, Figure 8(b) and Figure 9(b) shows that the query time of each algorithm increases with n and d, respectively.

Memory Usage: Figure 7(c), Figure 8(c) and Figure 9(c) show the memory usage of each algorithm when we vary k, n and d, respectively. Generally, *SampleClus* needs higher memory space than the other three algorithms.

6.2 Results on Real Datasets

We study the effect of k on each real dataset in terms of the query time and the hit probability of the set returned by each algorithm. In the experiments, we vary k from 10 to 100.

The experimental results based on the first-type real datasets can be found in Section 6.2.1 and the experimental results based on the second-type real datasets can be found in Section 6.2.2.

6.2.1 First-Type Real Datasets

Hit Probability: Figures 10(a), (b), (c), (d) and (e) shows the hit probability of each algorithm on five different first-type real datasets. Similar to the results on synthetic datasets, *k-hit_Alg* has the greatest hit probability but the others do not. Besides, similarly, in Figures 10(a), (b), (c), (d) and (e), when k increases, the hit probability of the set returned by each algorithm increases. In Figures 10(b) (for the household-6d dataset), when $k = 10$, the hit probability returned by *k-Hit_Alg* is 3.17 times larger than the hit probability returned by *SampleClus*.

Query Time: Figures 11(a), (b), (c), (d) and (e) show the query time of each algorithm on five first-type real datasets. Similar to before, the query time of *k-Hit_Alg* is small (within 10 seconds in the figures). Besides, in general, the query time of each algorithm increases with k. In Figure 11(c) (for the nba dataset), when $k = 60$, the hit probability returned by *k-Hit_Alg* is nearly equal to 1, which means that nearly all the users (or utility functions) are covered by the solution set returned by *k-Hit_Alg*. But, the hit probability returned by *SampleClus* is equal to 0.543, which means that nearly half of the users (or utility functions) are covered by the solution returned by *SampleClus*. Thus, *k-Hit_Alg* can cover nearly more than half of the users compared with *SampleClus*, which is a good result.

Memory Usage: Figures 12(a), (b), (c), (d) and (e) show the memory usage of each algorithm on five first-type real datasets. Similar to before, the memory usage of *k-Hit_Alg* is relatively small compared with its competitor *SampleClus*. Besides, the memory usage of each algorithm increases with k generally.

6.2.2 Second-Type Real Datasets

We first describe the details of the second-type dataset and the method of obtaining the distribution Θ. Then, we present the experimental results on this dataset.

In this dataset, there are 12,690 users and 37,813 musical songs. A user can give a numerical score to each of the musical songs s/he would like to rate. There are 100,000 user-song pairs for scoring. We construct a 12,690×37,813 matrix $\mathcal{M}$ where the entry at the i-th row and at the j-th column is equal to the score given by the i-th user for the j-th song if there is a score, and is equal to a missing value otherwise for each $i \in [1, 12, 690]$ and each $j \in [1, 37, 813]$. Then, we adopt a matrix factorization technique [16] to obtain two matrices, namely P and Q, by introducing 6 latent variables. Matrix P is of order 12,690×6, and matrix Q is of order 6×37,813. Each latent variable in this results corresponds to an attribute in our problem setting. In P, each row vector denotes a 6-dimensional weight vector of the utility function for a user. Based on these 12,690 row vectors, we can learn a distribution Θ by using the Multivariate Gaussian Mixture model with 5 mixture models. In Q, each column vector denotes a 6-dimensional tuple in our problem setting. So, all 37,813 vectors form our dataset D.

Figure 13(a), 13(b) and 13(c) shows the hit probability, the query time and the memory usage based on the *Yahoo!Music* dataset. We also obtain similar results.

7. CONCLUSION

In this paper, we study a k-hit query, a newly proposed query, which returns a set of k tuples such that the probability that one of the k tuples achieves the highest utility is maximized. We proposed an algorithm called *k-hit_Alg* for finding these k points efficiently. The experimental results showed the efficiency and the effectiveness of our algorithm.

There are a lot of future studies. The first one is to study the top-k queries when both the database and the utility functions are uncertain. In this paper, we study the case when the database is certain and the utility functions are uncertain while most traditional uncertain top-k queries study the case when the database is uncertain and the utility functions are certain. The second one is to study how to answer a k-hit query when the database changes and the distribution on utility functions changes.

Acknowledgement: We are grateful to the anonymous reviewers for their constructive comments on this paper. This research is supported by grant FSGRF13EG27.

8. REFERENCES

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[2] A. Blum, J. Jackson, T. Sandholm, and M. Zinkevich. Preference elicitation and query learning. *The Journal of Machine Learning Research*, 5:649–667, 2004.

[3] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[4] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 2002.

[5] U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[6] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.

[7] L. Chen and P. Pu. Survey of preference elicitation methods. *Rapport technique*, 2004.

[8] W. Chu and Z. Ghahramani. Preference learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 137–144. ACM, 2005.

[9] M. De Berg. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[10] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD*, 2012.

[11] T. Ge, S. Zdonik, and S. Madden. Top-k queries on uncertain data: on score distribution and typical answers. In *SIGMOD*, 2009.

[12] M. Goncalves and M. Vidal. Top-k skyline: A unified approach. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 790–799. Springer, 2005.

[13] D. Gourion and A. Seeger. Deterministic and stochastic methods for computing volumetric moduli of convex cones. *Computational & Applied Mathematics*, 29(2):215–246, 2010.

[14] Z. He and E. Lo. Answering why-not questions on top-k queries. In *ICDE*, 2012.

[15] N. Houlsby, J. M. Hernandez-Lobato, F. Huszar, and Z. Ghahramani. Collaborative gaussian processes for preference learning. In *Advances in Neural Information Processing Systems 25*, pages 2105–2113, 2012.

[16] S. Lai, Y. Liu, H. Gu, L. Xu, K. Liu, S. Xiang, J. Zhao, R. Diao, L. Xiang, H. Li, et al. Hybrid recommendation models for binary user preference prediction problem. *Journal of Machine Learning Research*, 2012.

[17] J. Lee, G. You, and S. Hwang. Personalized top-k skyline queries in high-dimensional space. *Information Systems*, 2009.

[18] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.

[19] D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *VLDB*, 2009.

[20] D. Nanongkai, A. Sarma, A. Lall, R. Lipton, and J. Xu. Regret-minimizing representative databases. *VLDB*, 2010.

[21] A. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos. Domination mining and querying. *Data Warehousing and Knowledge Discovery*, pages 145–156, 2007.

[22] P. Peng and R. C.-W. Wong. Geometry approach for k-regret query. In *ICDE*, 2014.

[23] P. Peng and R. C.-W. Wong. k-hit query: Top-k query with probabilistic utility function (technical report). In *http://www.cse.ust.hk/~raywong/paper/kHitQuery-technicalReport.pdf*, 2015.

[24] L. Qin, J. Yu, and L. Chang. Diversifying top-k results. *VLDB*, 2012.

[25] A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *ACM SIGKDD Explorations Newsletter*, 2008.

[26] J. M. Ribando. Measuring solid angles beyond dimension three. *Discrete & Computational Geometry*, 36(3):479–487, 2006.

[27] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

[28] M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *ICDE*, 2009.

Algorithm 4 Implementation of Our Detailed Algorithm *k-Hit_Alg* (for Countable Utility Function Class)

Input: A set of n d-dimensional points $D = \{p_1, p_2, ..., p_n\}$, a sampling size N, a positive integer k.
Output: A subset S of D of size k
1: // Step 1 (Standalone HP Estimation)
2: **for** each weight vector ω in $\mathcal{W}$ **do**
3: we determine which tuple p in D has the greatest value of $p \cdot \omega$ and then increment variable $weight(p)$, initialized to 0, by $Prob_\omega$
4: // Step 2 (Output)
5: $S \leftarrow$ the set of the k tuples in D with the greatest $weight(\cdot)$ values
6: **return** S

[29] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.

[30] Y. Tao, X. Xiao, and J. Pei. Efficient skyline and top-k retrieval in subspaces. *TKDE*, 2007.

[31] Y.-Y. Tsai, C.-M. Wang, C.-H. Chang, and Y.-M. Cheng. Tunable bounding volumes for monte carlo applications. In *Computational Science and Its Applications-ICCSA*. 2006.

[32] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. On efficient top-k query processing in highly distributed environments. In *SIGMOD*, 2008.

[33] R. Whitcher. A monte carlo method to calculate the average solid angle subtended by a right cylinder to a source that is circular or rectangular, plane or thick, at any position and orientation. *Radiation protection dosimetry*, 2006.

[34] D. Xin, J. Han, H. Cheng, and X. Li. Answering top-k queries with multi-dimensional selections: The ranking cube approach. In *VLDB*, 2006.

[35] M. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. *The VLDB Journal*, 18(3):695–718, 2009.

[36] F. Zhao, G. Das, K.-L. Tan, and A. K. Tung. Call to order: a hierarchical browsing approach to eliciting users' preference. In *SIGMOD*, 2010.

APPENDIX

A. DISCUSSION ON COUNTABLE UTILITY FUNCTION

When a set $\mathcal{F}$ of utility functions are countable, we are given a limited number of utility functions. As we know, each utility function is associated with a weight vector. Let $\mathcal{W}$ be the set of the weight vectors of all utility functions in $\mathcal{F}$. Each utility function with a weight vector ω in $\mathcal{F}$ is associated with a probability $Prob_\omega$. Note that $\sum_{\omega \in \mathcal{W}} Prob_\omega = 1$.

Given a tuple p in D, we denote $G(p)$ to be a set of weight vectors in $\mathcal{W}$ such that for each ω in this set, p achieves the highest utility with respect to the utility function with the weight vector ω. It is easy to verify that $HP(p) = \sum_{\omega \in G(p)} Prob_\omega$. Based on this observation, we present the pseudo-code of our algorithm on how to handle the case when the utility functions are countable in Algorithm 4.

B. COMPLEXITY ANALYSIS

B.1 Linear Utility Function Class

Time Complexity: In the context of the linear utility function class, *k-Hit_Alg* involves three steps. Step 1 is to construct the convex hull, whose cost is denoted by T_{conv}, and to build an index described in Section 4.2.2, whose cost is denoted by T_{index}. Step 2 is to compute the standalone hit probabilities of all tuples in $Surf(Conv(D))$, whose cost is denoted by T_{prob}. Step 3 is to output the k tuples with the greatest standalone probabilities, whose

cost is denoted by T_{output}. Thus, the time complexity of *k-Hit_Alg* is equal to $O(T_{conv} + T_{index} + T_{prob} + T_{output})$. Consider T_{conv} for Step 1. The time complexity of building a convex hull [9] is $O(n^{d/2} \log n)$. Thus, $T_{conv} = O(n^{d/2} \log n)$. Consider T_{index} for Step 2. The time complexity of building the index [1] is $O(|H|^{\lfloor d/2 \rfloor}(\log |H|)^{O(1)})$. Thus, $T_{index} = |H|^{\lfloor d/2 \rfloor}(\log |H|)^{O(1)}$. Consider T_{prob} for Step 2. In Step 2, there are three sub-steps, Step (a), Step (b) and Step (c). Step (a) takes $O(N)$ time. Step (b) takes $O(N \log |H|)$ time. Step (c) can be done in $O(|H|)$. Thus, $T_{prob} = O(N + N \log |H| + |H|) = O(N \log |H| + |H|)$. Consider T_{output} for Step 3 which can be done in $O(|H|)$ time. Thus, the time complexity of *k-Hit_Alg* is $O(n^{d/2} \log n + |H|^{\lfloor d/2 \rfloor}(\log |H|)^{O(1)} + N \log |H| + |H| + |H|)$. Since $|H| = O(n)$, the time complexity becomes $O(n^{d/2}(\log n)^{O(1)} + N \log |H| + |H|)$.

Space Complexity: *k-Hit_Alg* maintains the following components during its execution: (1) all points in $Surf(Conv(p))$, whose space cost is denoted by X_{conv}, (2) the index described in Section 4.2.2, whose space cost is denoted by X_{index}, and (3) N sampled vectors, whose space cost is denoted by X_{sample}. The space complexity is equal to $O(X_{conv} + X_{index} + X_{sample})$. Note that $X_{conv} = O(n)$ and $X_{sample} = O(N)$. The space complexity of the index proposed in [1] (i.e., X_{index}) is $O(|H|^{\lfloor d/2 \rfloor}(\log |H|)^{O(1)})$. In our experiments, in practice, X_{index} is very small and is at most 0.205GB in all experiments. The overall space complexity is equal to $O(n + N + |H|^{\lfloor d/2 \rfloor}(\log |H|)^{O(1)})$.

B.2 Non-Linear Utility Function Class

Time Complexity: In the context of the non-linear utility function class, *k-Hit_Alg* involves two steps. Step 1 is to compute the standalone hit probabilities of all tuples in D, whose cost is denoted by T_{prob}. Step 2 is to output the k tuples with the greatest standalone hit probabilities, whose cost is denoted by T_{output}. The time complexity of *k-Hit_Alg* is $O(T_{prob} + T_{output})$. Consider T_{prob} for Step 1. Similar to the linear utility function class, in the context of the non-linear utility function class, Step 1 includes three corresponding sub-steps, Step (a), Step (b) and Step (c). Step (a) takes $O(N)$ time. Step (b) takes $O(Nn)$ time. Step (c) of estimating the standalone hit probabilities of all tuples in D takes $O(n)$ time. Thus, $T_{prob} = O(N + Nn + n) = O(Nn)$. Consider T_{output} for Step 2 which can be done in $O(n)$ time. Thus, the time complexity of *k-Hit_Alg*is $O(Nn + n) = O(Nn)$.

Space Complexity: The space complexity of *k-Hit_Alg* maintaining D and N sampled vectors is $O(n + N)$.

C. PROOF OF LEMMAS/THEOREMS

Proof of Lemma 1: We denote the only one utility function in $\mathcal{F}$ by f. Let p be the tuple in D which achieves the highest utility with respect to f. Let $S = \{p\}$. We derive that the probability that one of the tuples in S is the favorite of a user is equal to 1, the greatest possible value. Thus, S is the optimal selection set of a 1-hit query. Note that S is the selection set of the top-1 query where the utility function used is f. Thus, the k-hit query becomes the top-k query where $k = 1$. □

Proof of Lemma 2: Let D' be the set of candidate tuples in D which achieves the highest utility with respect to at least one utility function in $\mathcal{F}$. Note that $|D'| = M$. We know that $HP(D') = 1$.

Consider two cases. *Case 1:* $k \geq M$. Since $|D'| = M$, we have $|D'| \leq k$. Since $HP(D') = 1$, for any subset X of D, $HP(D' \cup X) = 1$. Thus, there exists a set $S \subseteq D$ such that $D' \subseteq S$ and $HP(S) = 1$. Since $\min\{\frac{k}{M}, 1\} = 1$, we conclude that $HP(S) \geq \min\{\frac{k}{M}, 1\}$. *Case 2:* $k < M$. Thus, $\min\{\frac{k}{M}, 1\} = k/M$. We want to show that $HP(S) \geq \frac{k}{M}$. We prove by contradiction. Suppose that $HP(S) < \frac{k}{M}$. Since the size of the optimal set S is k, there are $M - k$ tuples which are not in S. Since $HP(D' \setminus S) + HP(S) = 1$ (by Equation (1)) and $HP(S) < \frac{k}{M}$, we derive that $HP(D' \setminus S) \geq \frac{M-k}{M}$. Let $t_1, t_2, ..., t_{k+1}$ be the $k+1$ tuples in D' such that they have the greatest standalone HP values and $HP(t_i) \geq HP(t_{i+1})$ for each $i \in [1, k]$. Since $t_1, t_2, ..., t_k$ are the k tuples in D' with the greatest standalone HP values, by Equation (1), we know that the optimal set S is equal to $\{t_1, t_2, ..., t_k\}$. Note that t_{k+1} has the greatest HP value in $D' \setminus S$. Since $HP(D' \setminus S) \geq \frac{M-k}{M}$ and there are $M - k$ tuples in $D' \setminus S$, by the pigeonhole's principle and Equation (1), we deduce that $HP(t_{k+1}) \geq \frac{M-k}{M} \cdot \frac{1}{M-k} = \frac{1}{M}$. Since $HP(t_k) \geq HP(t_{k+1})$, we have $HP(t_k) \geq \frac{1}{M}$. Since $HP(S) = \sum_{i=1}^{k} HP(t_i)$ (by Equation (1)) and $HP(t_i) \geq HP(t_k)$ for each $i \in [1, k]$, we deduce that $HP(S) \geq k \cdot HP(t_k) \geq \frac{k}{M}$ which leads to a contradiction. □

Proof of Lemma 3: Firstly, we want to show that $\mathcal{L}'$ is concise with respect to $\mathcal{L}$. That is, we want to show that any two different utility functions in $\mathcal{L}'$ do not return the same ranking result on any set D of the d-dimensional points. Consider any two different utility functions in $\mathcal{L}'$, namely f and f'. Let the weight vectors of functions f and f' be ω and ω', respectively. Note that ω and ω' are different. Note that the norm of ω (i.e., $\|\omega\|$) is equal to 1 and the norm of ω' (i.e., $\|\omega'\|$) is equal to 1. Thus, $\|\omega\| = \|\omega'\|$. We construct a set D containing two d-dimensional points, namely p and q, where p is set to ω and q is set to ω'. Consider two cases. *Case 1:* p is ranked higher than q with respect to the utility function with the weight vector ω. That is, $\omega \cdot p > \omega \cdot q$. We derive that $\omega \cdot \omega > \omega \cdot \omega'$. Thus, $\|\omega\|^2 > \omega \cdot \omega'$. Since $\|\omega\| = \|\omega'\|$, we have $\|\omega'\|^2 > \omega \cdot \omega'$. Thus, $\omega' \cdot \omega' > \omega' \cdot \omega$. We derive that $\omega' \cdot q > \omega' \cdot p$. In other words, p is ranked lower than q with respect to the utility function with the weight vector ω'. In conclusion, the two different utility functions f and f' (with their weight vectors ω and ω', respectively) do not return the same ranking result on a set D. *Case 2:* p is ranked lower than q with respect to the utility function with the weight vector ω. We also obtain a similar conclusion obtained in Case 1 since the proof is similar. *Case 3:* p has the same rank as q with respect to the utility function with the weight vector ω. Similar to the proof of Case 1, we obtain that $\omega' \cdot \omega' = \omega' \cdot \omega$. Since ω and ω' are two d-dimensional non-negative real vectors, we derive that $\omega' = \omega$, which leads to a contradiction that ω and ω' are different. In other words, Case 3 is not possible.

Secondly, we want to show that $\mathcal{L}'$ is complete with respect to $\mathcal{L}$. That is, we want to show that for each utility function $f \in \mathcal{L}$, there exists $f' \in \mathcal{L}'$ such that f and f' return the same ranking result on any set D of the d-dimensional points. Consider an arbitrary utility function f in $\mathcal{L}$. Let ω be the weight vector of f. Let ω' be a d-dimensional non-negative real vector such that for each $i \in [1, d]$, $\omega'_i = \frac{\omega_i}{\|\omega\|}$. Note that $\omega' = \frac{\omega}{\|\omega\|}$. Consider $\|\omega'\|^2 =$

$\sum_{i=1}^{d}(\omega_i')^2 = \sum_{i=1}^{d}(\frac{\omega_i}{\|\omega\|})^2 = \sum_{i=1}^{d}\frac{\omega_i^2}{\|\omega\|^2} = \frac{1}{\|\omega\|^2}\cdot\sum_{i=1}^{d}\omega_i^2 = \frac{1}{\|\omega\|^2}\cdot\|\omega\|^2 = 1$. Thus, we derive that $\|\omega'\| = 1$. Let f' be the utility function with the weight vector f'. Since $\|\omega'\| = 1$, we know that f' is in $\mathcal{L}'$. Next, we want to show that f and f' return the same ranking result on any set D of the d-dimensional points. Let D be a set of n tuples, namely $p_1, p_2, ..., p_n$. Without loss of generality, we assume that for each $i \in [1, n]$, p_i has the i-th highest utility with respect to f. In other words, $f(p_1) \geq f(p_2) \geq ... \geq f(p_n)$. We derive that $\omega \cdot p_1 \geq \omega \cdot p_2 ... \geq \omega \cdot p_n$. Dividing the expression by $\|\omega\|$, we obtain that $\frac{\omega}{\|\omega\|}\cdot p_1 \geq \frac{\omega}{\|\omega\|}\cdot p_2 ... \geq \frac{\omega}{\|\omega\|}\cdot p_n$. Since $\omega' = \frac{\omega}{\|\omega\|}$, we have $\omega' \cdot p_1 \geq \omega' \cdot p_2 ... \geq \omega' \cdot p_n$. Thus, $f'(p_1) \geq f'(p_2) \geq ... \geq f'(p_n)$. In other words, for each $i \in [1, n]$, p_i has the i-th highest utility with respect to f'. So, f and f' return the same raking result. □

Proof of Lemma 4: Firstly, we show that for any point $p' \in D$ and any (linear) function $f \in \mathcal{L}$ associated with its weight vector ω, $f(p')$ is equal to the length of the projection of the vector from the origin to point p' on vector ω. This can be shown as follows. We denote the vector from the origin to point p' by $\overrightarrow{op'}$. Let θ be the angle between vector ω and vector $\overrightarrow{op'}$. $f(p') = \omega \cdot p' = ||\omega|| \cdot ||\overrightarrow{op'}|| \cos\theta$. Since the norm of the weight vector ω is 1 (by definition) (i.e., $||\omega|| = 1$), we have $f(p') = ||\overrightarrow{op'}|| \cos\theta$. Note that $||\overrightarrow{op'}|| \cos\theta$ is equal to the length of the projection of vector $\overrightarrow{op'}$ on vector ω, which completes the proof.

Secondly, we show that $HP(p) = 0$. For any weight vector ω where its corresponding utility function f is in $\mathcal{L}$, there exists a point p' on the surface of $Conv(D)$ such that the length of the projection of the vector from the origin to point p' on vector ω is greater than the length of the projection of the vector from the origin to point p on vector ω. This means that $f(p') > f(p)$. In other words, for any utility function f in $\mathcal{L}$, there exists a point p' on the surface of $Conv(D)$ such that $f(p') > f(p)$. This implies that $HP(p) = 0$. □

Proof of Lemma 5: Since $HP(D) = 1$ and S is a subset of D, we have $HP(S) \leq 1$. Next, we want to consider two cases. *Case 1:* S is equal to Y. We prove by contradiction. Suppose that $HP(S) < 1$. Since $S = Y$, S is the set of points on the surface of the convex hull of D. Since $HP(S) < 1$ and $HP(D) = 1$, by Equation (1), we deduce that there exists a point p in $D \setminus S$ such that $HP(p) > 0$. Note that p is a point in D which is inside $Conv(D)$ but is not on the surface of $Conv(D)$. By Lemma 4, we know that $HP(p) = 0$, which leads to a contradiction. *Case 2:* S is a proper superset of Y. From Case 1, we know that $HP(Y) = 1$. Since $Y \subset S$ and $HP(S) \leq 1$, by Equation (1), $HP(S) = 1$. □

Proof of Corollary 1: It is easy to verify that the set of the points on the surface of $Conv(D)$ is the set of the points (or tuples) in D which achieve the highest utility with respect to at least one utility function in $\mathcal{F}$ (by Lemma 4). Since M is the number of the points on the surface of $Conv(D)$ (i.e., $|Surf(Conv(D))|$), according to Lemma 2, we deduce that $HP(S) \geq \min\{\frac{k}{M}, 1\}$. □

Proof of Lemma 6: Suppose that $HitVector(p) = \{v_1, v_2, ..., v_l\}$. For each $i \in [1, l]$, vector v_i from the origin can be written as a point v_i' in the form of $(v_i[1], v_i[2], ..., v_i[d])$. Let $V' = \{v_1', v_2', ..., v_l'\}$. Let $\mathcal{X}$ be the set of all non-boundary faces of $Conv(V')$ each of which contains at least d points in V'.

Firstly, we show that for each $i \in [1, l]$ and each $p' \in D$, $v_i' \cdot p' \leq v_i' \cdot p$. In the first part of the proof of of Lemma 4, we know that for any point $p' \in D$ and any (linear) function $f \in \mathcal{L}$ associated with its weight vector ω, $f(p')(= \omega \cdot p')$ is equal to the length of the projection of the vector from the origin to point p' on vector ω. For each $i \in [1, l]$, there exists a face in $Face(p)$ such that v_i is perpendicular to the face. Since p is in $Surf(Conv(D))$, due to the convex hull property, we deduce that for each $i \in [1, l]$ and each point $p' \in D$, the length of the projection of the vector from the origin to point p' on vector v_i is at most the length of the projection of the vector from the origin to point p on vector v_i. In other words, $v_i \cdot p' \leq v_i \cdot p$. That is, $v_i' \cdot p' \leq v_i' \cdot p$.

Secondly, we show that for any unit vector u from the origin, there exists at least a face X in $\mathcal{X}$ such that u intersects with X if and only if for each $p' \in D$, $u \cdot p \geq u \cdot p'$. Consider the "only if" part. Consider any unit vector u from the origin. Let X be the face in $\mathcal{X}$ such that u intersects with X. Without loss of generality, we assume that face X contains the first d points in V', i.e., $v_1', v_2', ..., v_d'$. Since u intersects with X, there exists an intersection point u' between u and face X. By the convex combination, we know that there exist d non-negative real numbers, namely $a_1, a_2, ..., a_d$, such that $u' = \sum_{i=1}^{d} a_i \cdot v_i'$. Consider $u' \cdot p = (\sum_{i=1}^{d} a_i \cdot v_i') \cdot p = [\sum_{i=1}^{d} a_i \cdot (v_i' \cdot p)]$. Since for each $p' \in D$, $v_i' \cdot p \geq v_i' \cdot p'$, we derive that for each $p' \in D$, $u' \cdot p \geq [\sum_{i=1}^{d} a_i \cdot (v_i' \cdot p')] = (\sum_{i=1}^{d} a_i \cdot v_i') \cdot p' = u' \cdot p'$. Thus, we conclude that for each $p' \in D$, $u' \cdot p \geq u' \cdot p'$. Since the vector from the origin to point u' has the same direction of vector u, there exists a non-negative real number h such that $||u'|| = h \cdot ||u||$. We derive that for each $p' \in D$, $u \cdot p \geq u \cdot p'$.

Consider the "if" part. We can use the same technique used in the "only if" part for this part. We do not show the proof since it is straightforward.

Thirdly, we show that $HP(p) = 2^d \cdot SA(HitVector(p))$. Let P be the partition containing all possible d-dimensional points in the d-dimensional space such that each point of these points has a non-negative real value for any dimension. Let $\mathcal{W}$ be a set of all possible weight vectors of utility functions in $\mathcal{L}$. Note that each vector in $\mathcal{W}$ has its norm/length equal to 1. We can regard that each point in P on the surface of $\mathbf{B}_d$ is a weight vector in $\mathcal{W}$. Let $C = Cone(HitVector(p))$.

Let $\mathcal{L}''$ be the set of utility functions such that p achieves the highest utility with respect to each of these utility functions. Let $\mathcal{W}''$ be the set of all weight vectors of utility functions in $\mathcal{L}''$. Since the distribution on the utility functions is uniform, we have $HP(p) = \frac{|\mathcal{L}''|}{|\mathcal{L}|}$. Since $\mathcal{L}$ and $\mathcal{W}$ have a one-to-one correspondence relationship, we have $HP(p) = \frac{|\mathcal{W}''|}{|\mathcal{W}|}$.

Note that in the second part of the proof, we know that for any unit vector $u \in \mathcal{W}$ from the origin, there exists a face X in $\mathcal{X}$ such that u intersects with X if and only if for each $p' \in D$, $u \cdot p \geq u \cdot p'$. In other words, for any unit vector $u \in \mathcal{W}$ from the origin, there exists a face X in $\mathcal{X}$ such that u intersects with X if and only if p achieves the highest utility with respect to the utility function with its weight vector equal to u. We derive that $\mathcal{W}''$ is the set of all weight vectors intersecting with at least a face in $\mathcal{X}$. Since (1) $\mathcal{X}$ is the set of all non-boundary faces of $Conv(V')$ each of which contains at least d points in V', (2) $V' = HitVector(p)$ and (3) $\mathcal{W}''$ is the set of all weight vectors intersecting with at least a face in $\mathcal{X}$, we derive that all unit vectors in $\mathcal{W}''$

form the region denoted by the intersection between $C(= Cone(HitVector(p)))$ and the surface of $\mathbf{B}_d$. Thus, $\frac{|\mathcal{W}''|}{|\mathcal{W}|}$ is equal to the ratio of the area of the intersection between C and the surface of $\mathbf{B}_d$ to the area of the surface of $\mathbf{B}_d$ in P. Note that the ratio of the area of the intersection between C and the surface of $\mathbf{B}_d$ to the area of the surface of $\mathbf{B}_d$ in P is equal to the ratio of the volume of C in P to the volume of $\mathbf{B}_d$ in P. Since the volume of C in P is equal to $Vol_d(Cone(HitVectors(p)) \cap \mathbf{B}_d)$ and the volume of $\mathbf{B}_d$ in P is $(\frac{1}{2})^d Vol_d(\mathbf{B}_d)$, we derive that $HP(p) = \frac{Vol_d(C) \cap \mathbf{B}_d)}{(\frac{1}{2})^d Vol_d(\mathbf{B}_d)}$. Since $C = Cone(HitVector(p))$, we have

$$HP(p) = \frac{Vol_d(Cone(HitVector(p))) \cap \mathbf{B}_d)}{(\frac{1}{2})^d Vol_d(\mathbf{B}_d)} \quad (2)$$

Since $SA(HitVector(p))$ is equal to the volume of $Conv(HitVector(p))$ and the volume of $Conv(HitVector(p))$ is equal to $\frac{Vol_d(Cone(HitVectors(p)) \cap \mathbf{B}_d)}{Vol_d(\mathbf{B}_d)}$, we have $SA(HitVector(p)) = \frac{Vol_d(Cone(HitVectors(p)) \cap \mathbf{B}_d)}{Vol_d(\mathbf{B}_d)}$. From (2), we derive that $HP(p) = 2^d \cdot SA(HitVector(p))$. □

Proof of Lemma 7: Firstly, we show that $HP(S) = 1$. Let $Y = Surf(Conv(D))$. Since S is a superset of the set of all points on the surface of the convex hull of D (i.e., $Surf(Conv(D))$), by Lemma 5, $HP(S) = 1$.

Secondly, we show that $S_o \subseteq S$. We prove by contradiction. Suppose that there exists a point $p \in S_o$ such that $p \in D \setminus S$. Since $p \in D \setminus S$, p is not a skyline tuple. Thus, there exists a tuple q in D such that p is dominated by a tuple q. This means that no user chooses p (since q is better than p). Thus, $HP(p) = 0$. Besides, since p is a non-skyline point, we know that p is not in $Surf(Conv(D))$. Since $k \geq |Surf(Conv(D))|$ and S_o contains p which is not in $Surf(Conv(D))$, we deduce that there exists a point $q' \in Surf(Conv(D))$ such that $q' \notin S_o$. Since $q' \in Surf(Conv(D))$, $HitVector(q)$ contains at least d distinct vectors. Thus, $Cone(HitVector(q))$ is non-empty and thus $SA(HitVector(p)) > 0$. Thus, by Lemma 6, $HP(q') > 0$. Let $S' = S_o \setminus \{p\} \cup \{q'\}$. Note that $|S'| = k$. By Equation (1), we know that $HP(S') = HP(S_o \setminus \{p\} \cup \{q'\}) = HP(S_o \setminus \{p\}) + HP(q')$. Since $HP(p) = 0$, we have $HP(S') = HP(S_o) + HP(q')$. Since $HP(q') > 0$, we have $HP(S') > HP(S_o)$, which leads to a contradiction that S_o is the optimal selection set (with the greatest HP value). □

Proof of Lemma 8: According to [22], the solution of a k-regret query can be a set which includes no points in $Surf(Conv(D))$. According to Equation (1) and Lemma 4, $HP(S) = \sum_{p \in S} HP(p)$, which is equal to 0.

We give a concrete counter example as follows. Let $D = \{p_1, p_2, p_3, p_4, p_5\}$, where $p_1 = (0, 1)$, $p_2 = (0.7, 0.81)$, $p_3 = (0.8, 0.8)$, $p_4 = (0.81, 0.7)$ and $p_5 = (1, 0)$. A 2-hit query returns the point p_1 and p_3 (note that $HP(p_1) = HP(p_5)$ in our example). An optimal solution of a 2-regret query in terms of D is $\{p_2, p_4\}$. The points on the surface of $Conv(D)$ are p_1, p_3, p_5 (i.e., $Surf(Conv(D)) = \{p_1, p_3, p_5\}$). Note that both p_2 and p_4 are not the extreme points of $Conv(D)$, which implies that $HP(p_2) = 0$ and $HP(p_4) = 0$. Therefore, we know that $HP(\{p_2, p_4\}) = 0$. □

Proof of Lemma 9: Since the solution of a k-representative skyline query can be a set which includes no points in $Surf(Conv(D))$, according to Equation (1) and Lemma 4, $HP(S) = \sum_{p \in S} HP(p)$, which is equal to 0.

We give a concrete counter example as follows. Let $D = \{p_1, p_2, p_3, p_4, p_5\}$, where $p_1 = (0, 1)$, $p_2 = (0.5, 0.9)$, $p_3 = (0.8, 0.8)$, $p_4 = (0.9, 0.5)$ and $p_5 = (1, 0)$. It is obvious that the set of skylines $S = D$ in our example. A 2-hit query returns the point p_1 and p_3 (note that $HP(p_1) = HP(p_5)$ in our example). Let K be the optimal solution of a 2-representative query. Then, we have $K = \{p_2, p_4\}$. The points on the surface of $Conv(D)$ are p_1, p_3, p_5 (i.e., $Surf(Conv(D)) = \{p_1, p_3, p_5\}$). Note that both p_2 and p_4 are not the extreme points of $Conv(D)$, which implies that $HP(p_2) = 0$ and $HP(p_4) = 0$. Therefore, we know that $HP(K) = 0$. □

Proof of Lemma 10: According to the definition of $HP(p)$, $HP(p) = \int_{f \in F(p)} \eta(f) df$. Note that $\eta(f)$ is the probability density function of distribution Θ.

Firstly, we want to use the claim of the first part of the proof of Lemma 6 to show that if a sampled vector v in V is inside the cone of p, then p achieves the highest utility with respect to the utility function with v. In this part, we adopt the notations from the proof of Lemma 6. From this proof, we know that for each $i \in [1, l]$ and each $p' \in D$, $v'_i \cdot p' \leq v'_i \cdot p$. Since a sampled vector v in V is inside the cone of p, v is a convex combination of V' and thus $v = \sum_{i=1}^{l} A_i \times v'_i$ where $A_i \geq 0$ for each $i \in [1, l]$. For each $p' \in D$, we derive that $v \cdot p' = \sum_{i=1}^{l} A_i \times (v'_i \cdot p') \leq \sum_{i=1}^{l} A_i \times (v'_i \cdot p) \leq v \cdot p$, which implies that p achieves the highest utility with respect to the utility function with the weight vector v.

Secondly, we show the main result of our lemma based on the first part. $V(p)$ denotes the set of all sampled vectors in V such that p achieves the highest utility with respect to the utility function with the weight vector equal to each of these sampled vectors. When N approaches ∞, $F(p)$ is equal to the set of all utility functions with the weight vectors equal to each of $V(p)$. Thus, $F(p)$ and $V(p)$ have a one-to-one correspondence relationship. Thus, $HP(p) = \frac{|V(p)|}{N}$. By Lemma 6, the hitting solid angle for p is $\frac{|V(p)|}{N} \cdot \frac{1}{2^d}$. □

Proof Sketch of Theorem 3: For the sake of space, we omit the details of the proof. The major idea is to use Chernoff's bound to link the three parameters, δ, ϵ and N. The complete proof can be found in [23]. □

Proof of Lemma 11: To prove this lemma, it is equivalent to showing that there exists a tuple p in D but not in $Surf(Conv(D))$ such that p achieves the highest utility with respect to a non-linear utility function. We give a concrete counter example as follows. Let $D = \{p_1, p_2, p_3\}$, where $p_1 = (0.25, 1)$, $p_2 = (1, 0.25)$ and $p_3 = (0.1, 0.1)$. We know that $Surf(Conv(D)) = \{p_1, p_2\}$. Note that p_3 is a tuple in D but not in $Surf(Conv(D))$. We construct a function f such that $f(p_1) = 0$, $f(p_2) = 0$ and $f(p_3) = 1$. Note that f is non-linear. Consider a function class $\mathcal{F} = \{f\}$ and a distribution Θ on $\mathcal{F}$ where $\eta(f) = 1$. We know that $HP(p_3) \neq 0$ and the optimal solution of a 1-hit query contains p. □

Proof of Lemma 12: The proof is similar to the proof of Lemma 10 which is based on any distribution Θ by replacing the first part of the proof with the following observation: If a sampled vector v in V has the greatest value of $p \cdot v$, then p achieves the highest utility with respect to the utility function with the weight vector v. □