Geometry Approach for k-Regret Query

Peng Peng, Raymond Chi-Wing Wong

The Hong Kong University of Science and Technology
{ppeng,raywong}@cse.ust.hk

***Abstract*—Returning tuples that users may be interested in is one of the most important goals for multi-criteria decision making. Top-k queries and skyline queries are two representative queries. A top-k query has its merit of returning a limited number of tuples to users but requires users to give their exact utility functions. A skyline query has its merit that users do not need to give their exact utility functions but has no control over the number of tuples to be returned. In this paper, we study a k-regret query, a recently proposed query, which integrates the merits of the two representative queries. We first identify some interesting geometry properties for the k-regret query. Based on these properties, we define a set of candidate points called *happy points* for the k-regret query, which has not been studied in the literature. This result is very fundamental and beneficial to not only all existing algorithms but also all new algorithms to be developed for the k-regret query. Since it is found that the number of happy points is very small, the efficiency of all existing algorithms can be improved significantly. Furthermore, based on other geometry properties, we propose two efficient algorithms each of which performs more efficiently than the best-known fastest algorithm. Our experimental results show that our proposed algorithms run faster than the best-known method on both synthetic and real datasets. In particular, in our experiments on real datasets, the best-known method took more than 3 hours to answer a k-regret query but one of our proposed methods took about a few minutes and the other took within a second.**

***Index Terms*—Query Processing, Skyline, k-Regret Query**

I. Introduction

A database system normally provides various operators that allow users to acquire their answers conveniently. Returning tuples that users may be interested in is one of the most important goals for multi-criteria decision making. Top-k queries [1], [2], [3], [4], [5] and skyline queries [6], [7], [8], [9], [10], [11] are two representative queries.

Consider the following example. Assume that we have a database storing a multitude of cars, each of which is associated with two *attributes*, namely, horse power (HP) and miles per gallon (MPG). Suppose that Alice wants to buy a car with high HP and high MPG. Since the size of a car database is usually very large (e.g., over 350,000 cars in the database of www.autotrader.co.uk), the database system selects a small number of cars from the database based on a certain criterion and displays them to Alice.

In order to complete such a car selection process, the database system must follow a criterion, which is determined by Alice, to calculate the *utility* of each car such that it can differentiate the cars in the database. It is implicitly assumed that there exists such a criterion, which is also called a *utility function*, in Alice's mind. Obviously, the car with higher utility in terms of Alice's utility function is more preferred by Alice. Depending on whether Alice knows her utility function or not, different queries are considered.

In the setting of top-k queries, the user's utility function is given, and the k tuples with the highest values on this utility function are returned. For example, let Alice's utility function be linear with weights on HP and on MPG being equal to 80% and 20%, respectively. Then, we can calculate all cars' utilities by substituting their HP and MPG into the utility function. Only k cars with the highest values on the utility function are chosen. The deficiency of top-k queries is that the user may not provide his or her utility function precisely.

In the setting of skyline queries, the utility function is not provided by the user but the output size is not specified beforehand. In the worst-case scenario, all tuples in the database may be returned. The concept of *domination* is crucial in skyline queries. We say that point q dominates point p if and only if for each dimension, the value of q is not smaller than the value of p, and the value of q is greater than the value of p on at least one dimension. For example, if we have a car A with HP 400 and MPG 25 and a car B with HP 414 and MPG 31 in the database, we say that car A is dominated by car B. Every point which is dominated by another point in the database will not be returned. The reason is that whichever utility function[1] Alice provides, B always has higher utility than A. Since the output size of a skyline query cannot be foreseen before the whole database is accessed, it becomes useless when too many tuples are displayed to the user.

In this paper, we study a k-regret query, which is simultaneously superior to top-k queries and skyline queries. In the setting of a k-regret query, the user's utility function is unknown and only k tuples are returned. Therefore, the deficiency in either the top-k query or the skyline query can be overcome by the k-regret query. By restricting to display only k tuples to the user, we expect that the greatest difference between the maximum utility of the point over these k tuples and the maximum utility of the point over all tuples in the database is negligible in terms of all (an infinite number of) linear utility functions. The *maximum regret ratio*, a measurement which quantifies such a difference, intuitively measures how *unhappy* the user will be after observing k tuples instead of the whole database. Our goal is to select k tuples such that the maximum regret ratio could be as small as possible.

The k-regret query was first proposed in [12]. Unfortunately, the fastest existing algorithm for the k-regret query called

[1] We implicitly assume that the utility function is linear.

Greedy is very time-consuming and in our experiment, it took more than 3 hours to answer a k-regret query.

Motivated by this, in this paper, we observe some geometry properties for this query. Based on one of the geometry properties, we define a set of candidate points called *happy points* for this query, which has not been studied in the literature. This result is very fundamental and beneficial to not only all existing algorithms but also all new algorithms to be developed for the k-regret query. Since it is found that the number of happy points is very small (at most 0.34% of each of the real datasets used in our experiment), the efficiency of all existing algorithms can be improved significantly. Note that all existing algorithms for the k-regret query are performed based on the set of skyline points which is regarded as the set of candidate points in the literature. However, we find that the set of happy points is a subset of the set of skyline points. Furthermore, all happy points are just a small portion of skyline points. In our experiments, in all real datasets, only at most 16% of skyline points are happy points, which suggests that happy points are better used as candidate points compared with skyline points due to its small size resulting in more efficient algorithms.

Besides, based on other geometry properties, we propose two efficient algorithms, namely *GeoGreedy* and *StoredList*. The *GeoGreedy* algorithm not only can return the solution whose maximum regret ratio is the same as *Greedy* but also can be executed very quickly due to its geometry property. In our experiments, it took about a few minutes to answer a k-regret query, which is more efficient. The *StoredList* algorithm is a *materialization* version of *GeoGreedy*, which further speeds up the query time with some materialized storage. In our experiments, with the materialized storage, it took within a second to answer a k-regret query, which is extremely efficient.

The contributions of this paper are summarized as follows. Firstly, we give some theoretical properties of the k-regret query. Based on one of the properties, we propose the concept of "happy points" which are considered as candidate points for the query, which has not been studied in the literature. This result is very fundamental and beneficial to not only all existing algorithms but also any new algorithms to be developed for the k-regret query. Secondly, we propose two algorithms called *GeoGreedy* and *StoredList* which has their time complexities better than the best-known fastest algorithm, *Greedy*. Lastly, we conducted comprehensive experiments for both algorithms, verifying their superiority over existing algorithms.

The rest of the paper is organized as follows. Section II gives the problem definition and Section III gives some theoretical properties of this problem. Section IV presents two proposed algorithms, *GeoGreedy* and *StoredList*. Section V shows the experimental results. Section VI gives the related work and Section VII gives some discussions. Finally, Section VIII concludes the paper.

II. Problem Definition

We are given a set D of n points where each point can be regarded as a tuple in the database. Each point in D is a d-dimensional non-negative real vector. Assume that $d \geq 2$. The value of a point p on the i-th dimension is denoted by $p[i]$. Without loss of generality, each dimension is normalized to (0, 1] such that there exists a point p in D with its value on the i-th dimension equal to 1 for each $i \in [1, d]$. We assume that a larger value on each dimension is more preferable. If it is not true for a dimension, we can multiply all values in this dimension by -1. Here, in order to avoid several complicated, yet uninteresting, "boundary cases", in this paper, we consider that each dimension value of a point in D is equal to non-zero. Obviously, when this assumption is not fulfilled, we can always add a very small positive value to each dimension so that this assumption is fulfilled.

TABLE I: Car Database

Car	(Normalized) MPG	(Normalized) HP
(p_1) BMW M3 GTS	0.94	0.8
(p_2) Chevrolet Camaro SS	0.76	0.93
(p_3) Ford Shelby GT500	0.67	1.00
(p_4) Nissan 370Z coupe	1.00	0.72

Given a positive integer k, a k-regret problem is to find at most k points from D such that the *maximum regret ratio* of these selected points is minimized.

In this paper, we assume that $k \geq d$ in order to simplify the discussion. A discussion about the case when $k < d$ can be found in Section VII.

A utility function is a mapping $f : \mathbb{R}^d_+ \rightarrow \mathbb{R}_+$. The *utility* of a point p for a user with the utility function f is denoted by $f(p)$. Given a set S of points in D and a utility function f, the *maximum utility* of S for f, denoted by $U_{max}(S, f)$, is defined to be $\max_{p \in S} f(p)$. The point p in S where $f(p) = U_{max}(S, f)$ is said to be a *maximum utility point* of S for f.

Note that for any set $S \subseteq D$, $U_{max}(S, f) \leq U_{max}(D, f)$. In some cases, it is also possible that there is more than one maximum utility point of a set S for a utility function f.

Before we define the *maximum regret ratio*, we define the *regret ratio* as follows.

Definition 1 (Regret Ratio): Given a set S of points in D and a utility function f, the *regret ratio* of S, denoted by $rr(S, f)$, is defined to be $1 - \frac{U_{max}(S,f)}{U_{max}(D,f)}$. ■

The regret ratio ranges from 0 to 1. Intuitively, when the maximum utility of a set S is closer to the maximum utility of D, the regret ratio becomes smaller.

However, since it is difficult to obtain a user's utility function, same as [12], [13], we assume that all users' utility functions belong to a function class, denoted by $\mathcal{F}$. A function class is defined to be a set of functions which share some common characteristics. The *maximum regret ratio* of a set S is defined according to $\mathcal{F}$, representing the maximum regret ratio of S for every utility function from $\mathcal{F}$.

Definition 2 (Maximum Regret Ratio): Given a set S of points in D, the *maximum regret ratio* of S, denoted by $mrr(S)$, is defined to be $\max_{f \in \mathcal{F}} rr(S, f)$. The function f in $\mathcal{F}$ where $rr(S, f) = mrr(S)$ is said to be a *maximum regret ratio function* of S. ■

Example: In the following, we present an example for the illustration. Consider the car database containing 4 cars

TABLE II: CAR UTILITIES

Car	$f_{(0.3,0.7)}$	$f_{(0.5,0.5)}$	$f_{(0.7,0.3)}$
p_1	0.842	0.870	0.898
p_2	0.879	0.845	0.811
p_3	0.901	0.835	0.769
p_4	0.804	0.860	0.916

as shown in Table I where the attributes MPG and HP are normalized to (0, 1]. The utility function class $\mathcal{F}$ is $\{f_{(0.3,0.7)}, f_{(0.5,0.5)}, f_{(0.7,0.3)}\}$ where $f_{(a,b)} = a \cdot MPG + b \cdot HP$. The utilities of cars for the utility functions in $\mathcal{F}$ are shown in Table II. Let the selection set $S = \{p_2, p_3\}$. Then, $rr(S, f_{(0.3,0.7)}) = 1 - \frac{0.901}{0.901} = 0$, $rr(S, f_{(0.5,0.5)}) = 1 - \frac{0.845}{0.870} = 0.029$, and $rr(S, f_{(0.7,0.3)}) = 1 - \frac{0.811}{0.916} = 0.115$. So, $mrr(S) = \max\{0, 0.029, 0.115\} = 0.115$. ■

We formulate the problem as follows.

Problem. Given a positive integer k, the maximum regret ratio minimization (MRRM) problem is to find a set S containing at most k points such that $mrr(S)$ is minimized.

The above problem is also named as a k-regret query. In the following, we assume that $k \geq d$. We will discuss the scenario when $k < d$ in Section VII.

Linear Utility Function: Same as [12], [13], we focus on the function class $\mathcal{F}$ containing all possible linear functions due to its popularity for modeling user preferences. Specifically, each linear function f is associated with a *weight vector* ω which is a d-dimensional non-negative real vector where $\omega[i]$ denotes the importance of the i-th dimension according to the user's preference for $i \in [1, d]$. Specifically, a linear function f is expressed as: $f(p) = \sum_{i=1}^{d} \omega[i]p[i]$. It can also be written as a dot product as: $f(p) = \omega \cdot p$.

In this function class, two different utility functions can return the same ranking result on any set of d-dimensional points. If the feature space of a linear utility function in $\mathcal{F}$ could be any element in $\mathbb{R}^d$, there are more than one utility function returning the same ranking result given a set of points in a d-dimensional space. For example, when $d = 2$, $\omega = (0.3, 0.7)$ and $\omega' = (0.03, 0.07)$ can be regarded as two weight vectors representing two utility functions which return the same ranking result on any set of d-dimensional points.

Motivated by this, we focus on the function class $\mathcal{F}'$ which is a subset of $\mathcal{F}$ such that $\mathcal{F}'$ is *concise* and *complete* with respect to $\mathcal{F}$. $\mathcal{F}'$ is said to be *concise* with respect to $\mathcal{F}$ if and only if any two different utility functions in $\mathcal{F}'$ do not return the same ranking result on any set of d-dimensional points. $\mathcal{F}'$ is said to be *complete* with respect to $\mathcal{F}$ if and only if for each utility function $f \in \mathcal{F}$, there exists $f' \in \mathcal{F}'$ such that f and f' return the same ranking result on any set of d-dimensional points. Interestingly, the function class containing all possible linear functions whose weight vectors have their *norm* equal to 1 is concise and complete with respect to $\mathcal{F}$. In the following, for clarity, we just focus on this function class and we refer this function class simply as $\mathcal{F}$.

III. THEORETICAL PROPERTY

In Section III-A, we first present an interesting geometry property of a k-regret query which can be used to compute the maximum regret ratio of a given selection set more easily. In Section III-B, we give the definition of "happy points" and show some properties about "happy points". One of the properties to be shown is that happy points are candidate points for a k-regret query.

A. Geometry Properties

In this section, we present the geometry property which can be used to compute the maximum regret ratio of a given selection set more easily. Before we introduce the geometry property, we introduce some concepts which will be used in our geometry property.

The first concept is "*boundary point*". Given a set S of points in D, for each $i \in [1, d]$, a point $p \in S$ is said to be an *i-th dimension boundary point* of S if $p[i]$ is the greatest among all points in S. Consider our running example as shown in Figure 1 showing a set D of 7 data points, namely $p_1, p_2, ..., p_7$, in a two-dimensional space where the first dimension is X_1 and the second dimension is X_2. If $S = D$, then p_6 is the first dimension boundary point of S and p_7 is the second dimension boundary point of S. If $S = \{p_1, p_3, p_6\}$, then p_6 is the first dimension boundary point of S and p_1 is the second dimension boundary point of S.

The second concept is "*convex hull*". For each point $p \in D$, we define the *orthotope set* of p, denoted by $Orth(p)$, to be a set of 2^d d-dimensional points constructed by $\{0, p[1]\} \times \{0, p[2]\} \times ... \times \{0, p[d]\}$. That is, for each $i \in [1, d]$, the i-dimensional value of a point in $Orth(p)$ is equal to either 0 or $p[i]$. Consider our running example as shown in Figure 1. $Orth(p_1) = \{(p_1[1], p_1[2]), (p_1[1], 0), (0, p_1[2]), (0, 0)\}$. Note that there are 4 ($= 2^2$) points in $Orth(p_1)$. The points in $Orth(p_1)$ are shown in Figure 2 where $p'_1 = (p_1[1], 0)$ and $p''_1 = (0, p_1[2])$. Similarly, the points in $Orth(p_6)$ are shown in the same figure. Given a set S of points in D, we define $D_{orth}(S)$ to be $\cup_{p \in S} Orth(p)$.

Given a set S of points in D, we denote $Conv(S)$ to be the *convex hull* of $D_{orth}(S)$ [14]. For example, Figure 3 shows $Conv(D)$ in our running example where the points on $Conv(D)$ except the origin are $p''_7, p_7, p_1, p_3, p_5, p_6$ and p'_6 where $p''_7 = (0, p_7[2])$ and $p'_6 = (p_6[1], 0)$. When $S = \{p_1, p_3, p_6\}$, the points on $Conv(S)$ except the origin are p''_1, p_1, p_3, p_6 and p'_6, as shown in Figure 4. Given a set S of points in D, we denote $|Conv(S)|$ to be the number of extreme points of $Conv(S)$ which are in S. For example, $|Conv(D)| = 5$. If $S = \{p_1, p_3, p_6\}$, then $|Conv(S)| = 3$.

Consider that $S = \{p_1, p_3, p_6\}$. There are six *faces* of $Conv(S)$ (i.e., (1) the line segment between the origin and p''_1, (2) the line segment between p''_1 and p_1, (3) the line segment between p_1 and p_3, (4) the line segment between p_3 and p_6, (5) the line segment between p_6 and p'_6 and (6) the line segment between p'_6 and the origin) as shown in Figure 4. Two of them pass through the origin but the remaining four do not. In the following, we focus on the faces not passing through the origin. When we refer a face of the convex hull, we mean the face not passing through the origin. Let $F(S)$ be a set of all faces of $Conv(S)$. Note that a face of $Conv(S)$ is on a

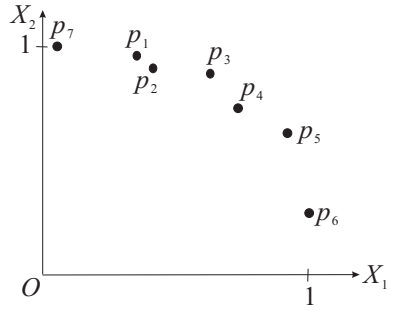


Fig. 1: A Running Example in a 2-dimensional Space

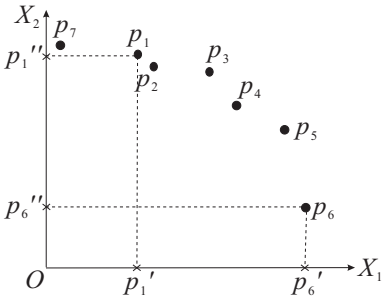


Fig. 2: $Orth(p_1)$ and $Orth(p_6)$

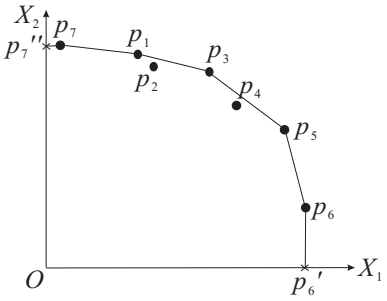


Fig. 3: $Conv(D)$

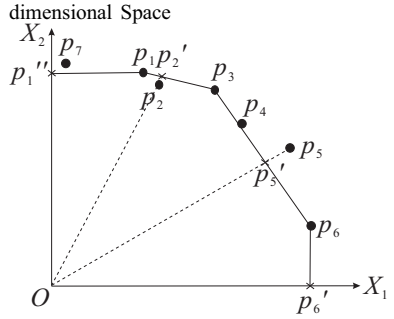


Fig. 4: $Conv(\{p_1, p_3, p_6\})$

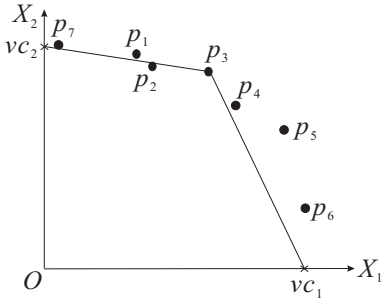


Fig. 5: $Conv(\{p_3\} \cup VC)$

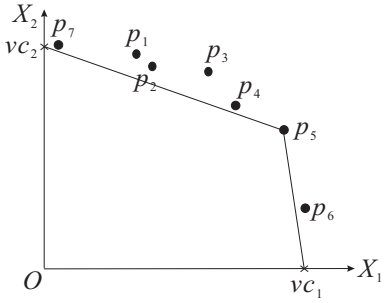


Fig. 6: $Conv(\{p_5\} \cup VC)$

hyperplane and is a portion of this hyperplane, and thus we say that this hyperplane *contains* this face.

In the following, we assume that $|Conv(D)| > k$. The case of $|Conv(D)| \leq k$ is simple to be handled since we know that all extreme points of $Conv(D)$ which are in D form the set S giving $mrr(S) = 0$, and thus S corresponds to the optimal solution.

The third concept is "*critical point*". Given a point $p \in D$ and a set S of points in D, we define the *p-critical point for* S (or the *critical point* of p for S) to be the intersection point between a face of $Conv(S)$ and the line passing through the origin and point p. In Figure 4, consider that $S = \{p_1, p_3, p_6\}$. p_5' is the p_5-critical point for S because p_5' is the intersection point between a face of $Conv(S)$ (i.e., the line segment between p_3 and p_6) and the line passing through the origin and p_5. Besides, p_2' is the p_2-critical point for S because p_2' is the intersection point between a face of $Conv(S)$ (i.e., the line segment between p_1 and p_3) and the line passing through the origin and p_2. Similarly, p_3 (itself) is the p_3-critical point for S.

Definition 3 (Critical Ratio): Let S be a set of points in D. Given a point $p \in D$, the *critical ratio* of p for S, denoted by $cr(p, S)$, is defined to be $\frac{\|p'\|}{\|p\|}$ where p' is the p-critical point for S. ■

In Figure 4, let $S = \{p_1, p_3, p_6\}$. Then, $cr(p_5, S)$ is equal to $\frac{\|p_5'\|}{\|p_5\|} = \frac{|p_5'o|}{|p_5o|}$ where p_5' is the p_5-critical point for S. Similarly, $cr(p_2, S)$ is equal to $\frac{\|p_2'\|}{\|p_2\|} = \frac{|p_2'o|}{|p_2o|}$ where p_2' is the p_2-critical point for S. Note that $cr(p_5, S) < 1$ and $cr(p_2, S) > 1$. Besides, $cr(p_3, S)$ is equal to $\frac{\|p_3\|}{\|p_3\|} = 1$.

Note that for each point p on $Conv(S)$ which is in S, $cr(p, S) = 1$. Thus, if S is non-empty, there exists a point p in S such that $cr(p, S) = 1$.

We have introduced all concepts needed. Now, we are ready to give our geometry property.

Lemma 1 (Geometry Property): Let S be a set of points in D. If $r = \min_{p \in D} cr(p, S)$, then $mrr(S) = 1 - r$. ■

Lemma 1 gives us a powerful tool to compute the maximum regret ratio of a given selection set by using the concept of "critical ratio". Specifically, it states that given a set S of points, the maximum regret ratio of S (i.e., $mrr(S)$) is exactly equal to $1 - r$ where r is the smallest critical ratio of a point in D for S. Note that r is at most 1 (since there exists a point in S such that $cr(p, S) = 1$).

For example, in Figure 4, let $S = \{p_1, p_3, p_6\}$. p_5 gives the smallest critical ratio for S, says r. Thus, $mrr(S)$ is equal to $1 - r$.

B. Happy Point Properties

In this section, we give the definition of "happy points" and show some properties about "happy points". Before that, we introduce some concepts first.

Consider a non-negative d-dimensional vector ω. Let h be a hyperplane represented by an equation "$\omega \cdot x = c$" where c is a constant. A point p is said to be *below* hyperplane h perpendicular to ω if $\omega \cdot p < c$. A point p is said to be *above* hyperplane h perpendicular to ω if $\omega \cdot p > c$. A point p is said to be *on* hyperplane h perpendicular to ω if $\omega \cdot p = c$. For instance, in Figure 4, let h be the line passing p_3 and p_6. p_5 is above h, p_3 is on h and p_2 is below h.

For each $i \in [1, d]$, we define the *i-th dimension virtual corner point*, denoted by vc_i, to be a d-dimensional point p where its i-th dimension value (i.e., $p[i]$) is set to 1 and each of the other dimension values is set to 0. For example, the first dimension virtual corner point is $(1, 0, ..., 0)$. Let VC be the set of all virtual corner points (i.e., $VC = \cup_{i=1}^{d} vc_i$). Note that there are d points in VC. For example, in a 2-dimensional space, $vc_1 = (1, 0)$ and $vc_2 = (0, 1)$. Thus, $VC = \{vc_1, vc_2\}$.

The two virtual corner points are shown in Figure 5. Note that for each $vc_i \in VC$, $Orth(vc_i)$ contains two points only, namely vc_i and the origin.

Consider a point $p \in D$. By definition, we know that $Conv(\{p\} \cup VC)$ is the convex hull of $D_{orth}(\{p\} \cup VC)$. For example, Figure 5 shows $Conv(\{p_3\} \cup VC)$ and Figure 6 shows $Conv(\{p_5\} \cup VC)$.

Given a point $p \in D$, we define $Y(p)$ to be a set of all possible hyperplanes, each of which contains a face of $Conv(\{p\} \cup VC)$. For example, in Figure 5, $Y(p_3)$ is equal to the set of two lines, namely the line passing $vc_1(=(1,0))$ and p_3, and the line passing p_3 and $vc_2(=(0,1))$.

We have introduced all notations needed. Now, we are ready to give the definition of "*happy point*".

Definition 4 (Subjugation and happy points): Given a point $p \in D$ and another point $p' \in D$, p is said to be *subjugated* by p' if and only if p is on or below each hyperplane in $Y(p')$ and p is below at least one hyperplane in $Y(p')$. If there does not exist any point in D which subjugates a point p, p is said to be a *happy* point in D. ■

In Figure 5, $Y(p_3)$ is a set of two lines, namely the line passing vc_1 and p_3, and the line passing p_3 and vc_2. Since p_2 is below each of the hyperplanes (or lines) in $Y(p_3)$, p_2 is subjugated by p_3 and thus p_2 is not a happy point in D. However, p_1 is a happy point in D since there does not exist any point in D which subjugates p_1. The other happy points in D are p_3, p_4, p_5, p_6 and p_7.

We have just given the definition of "happy points". In the following, we show the first property that happy points are candidate points for a k-regret query.

Lemma 2: Let mrr_o be the optimal maximum regret ratio of the MRRM problem. There exists a subset S of the set of all happy points in D such that $|S| = k$ and $mrr(S) = mrr_o$. ■

The above lemma suggests that it is sufficient to focus on all happy points in D to find the optimal solution of MRRM.

Next, we state more properties about happy points.

Let D_{conv} be the set of all extreme points of $Conv(D)$ in D, D_{sky} be the set of all skyline points in D [10], and D_{happy} be the set of all happy points in D.

Consider the same running example in Figure 1. It is easy to verify that $D_{conv} = \{p_1, p_3, p_5, p_6, p_7\}$ and $D_{sky} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$. As we described before, $D_{happy} = \{p_1, p_3, p_4, p_5, p_6, p_7\}$.

Interestingly, these three sets have the relationship shown in the following lemma.

Lemma 3 (Relationship): $D_{conv} \subseteq D_{happy} \subseteq D_{sky}$. ■

The above lemma states that (1) each point in D_{conv} is a happy point in D and (2) each happy point in D is a skyline point in D. It is interesting to ask the following two questions about their relationships: Does there exist a happy point in D which is not in D_{conv}? Does there exist a skyline point in D which is not a happy point in D? The answers of these two questions are "yes", which can be verified from our example.

Lemma 4: There exists a set D of objects such that $D_{happy} \setminus D_{conv} \neq \emptyset$ and $D_{sky} \setminus D_{happy} \neq \emptyset$. ■

One may also ask a question about the optimality related to D_{conv}: Is it true that each point in the optimal set of the MRRM problem is in D_{conv} even if the size of D_{conv} is at least k? Unfortunately, the answer is "no" by the following lemma.

Lemma 5: Let S_o be the optimal set of the MRRM problem. In general, there exists a point $p \in S_o$ such that $p \notin D_{conv}$. ■

Consider the same running example. We know that when $k = 3$, the optimal solution S_o of the MRRM problem is $\{p_1, p_4, p_6\}$. Note that $D_{conv} = \{p_1, p_3, p_5, p_6, p_7\}$ but p_4 is in S_o but not in D_{conv}.

Algorithm for Finding Happy Points in D: We are interested in finding happy points in D so that the algorithms designed for the MRRM problem are executed on the happy points only. One straightforward approach for finding a set D_{happy} of all happy points is to check whether each point in D is subjugated by the other points in D. All the points not subjugated by the other points in D are the happy points. We can see that there are $O(n^2)$ pairwise comparisons between two points. For each comparison between a point p and another point p', we first construct d hyperplanes in $Y(p')$ where each hyperplane can be constructed in $O(d)$ time, and then determine whether p is below each of the d constructed hyperplane which can be done in $O(d)$ time. Thus, each comparison takes $O(d^2)$ time. Since there are $O(n^2)$ comparisons, the overall time complexity of finding all happy points is $O(d^2n^2)$.

IV. Algorithms

In the following, we propose two algorithms, namely *GeoGreedy* and *StoredList*. In the previous section, we know that the solution of MRRM is a subset of D_{happy}. In the following, when we write D, we mean D_{happy}.

A. The Geometry Greedy Algorithm

In this section, we present a heuristic greedy algorithm called *GeoGreedy* using the geometry property we discussed in the previous section. *GeoGreedy* is exactly the best-known algorithm called *Greedy* [12] but the computation of the maximum regret ratio of a selection set used in each iteration is replaced by our geometry property described in Lemma 1. This algorithm has two features. The first feature is that it is more efficient than the best-known algorithm called *Greedy* [12]. The second feature is that it returns the same solution as [12] with a very small maximum regret ratio in practice.

GeoGreedy has the same greedy skeleton as *Greedy*. In order to describe *GeoGreedy* better, we describe the details of *GeoGreedy* directly in the following. We know that given a set S of points in D, we can calculate $mrr(S)$ according to Definition 2. There exists a function $f_o \in \mathcal{F}$ such that $f_o = \arg\max_{f' \in \mathcal{F}} rr(S, f')$. We know that $mrr(S) = rr(S, f_o)$. From $rr(S, f_o)$, according to Definition 1, there exists a point $p \in S$ and another point $q \in D$ such that $mrr(S) = 1 - \frac{f(p)}{f(q)}$ (since $f(p) = U_{max}(S, f)$ and $f(q) = U_{max}(D, f)$). It is easy to see that if $mrr(S) \neq 0$, then q is not equal to p and is in $D \setminus S$. In this case, we say that q *contributes* to the maximum

Algorithm 1 The *GeoGreedy* Algorithm

Input: D and k
Output: A set S of k points in D
1: $S \leftarrow \emptyset$
2: **for** each $i := 1$ to d **do**
3: $p \leftarrow$ an i-th dimension boundary point of D
4: $S \leftarrow S \cup \{p\}$
5: **for** each $i := 1$ to $k - d$ **do**
6: find a point $p \in D \setminus S$ with the smallest critical ratio for S (i.e., $cr(p, S)$)
7: **if** $cr(p, S) \geq 1$ **then**
8: **return** S
9: **else**
10: $S \leftarrow S \cup \{p\}$
11: **return** S

regret ratio of S. This point can be regarded as the "worst" point in $D \setminus S$ contributing to the maximum regret ratio of S. Note that according to our geometry property described in Lemma 1, q is also the point with the smallest critical value for S.

Now, we are ready to present our *GeoGreedy* algorithm as shown in Algorithm 1. Initially, we initialize S with d boundary points in D. Then, we iteratively select a point p which contributes to the maximum regret ratio of S. This can be done efficiently by our geometry property. That is, the point p to be found is the point with the smallest critical ratio for S. If the critical ratio found is at least 1, then the maximum regret ratio of S is equal to 0 and thus there is no need to find any other points to be inserted into S. Otherwise, we insert p into S. We iteratively perform the above steps until S contains k points.

Note that the only difference between *Greedy* and *GeoGreedy* is line 6 where *Greedy* finds a point $p \in D \setminus S$ such that p contributes to the maximum regret ratio of S via time-consuming constrained programming, and *GeoGreedy* find a point $p \in D \setminus S$ with the smallest critical ratio for S via an efficient geometry method (which will be described next). Since this difference is only related to how to implement this step and the final points found by *Greedy* and *GeoGreedy* in this step are the same, *Greedy* and *GeoGreedy* have the same output (if there are no ties).

Indexing for GeoGreedy Algorithm: Next, we present an indexing method for our *GeoGreedy* algorithm. This indexing method is used for efficiently computing the critical ratio of a point p for S (line 6 of Algorithm 1).

Specifically, whenever we compute the critical ratio of a point $p \in D \setminus S$ for the current selection set S, we have to determine the *particular* face of $Conv(S)$ crossed by the line passing through the origin and p, and find the intersection point p' on this face. Thus, determining which face of $Conv(S)$ crossed by the line passing through the origin and p *efficiently* is very important. In the following, we propose to build an index based on all faces of $Conv(S)$ so that for each point $p \in D \setminus S$, with this index, we can determine which face crossed by the line passing through the origin and p efficiently.

Before we describe the index, we first introduce a *ray shooting query* [15], which is used in our index.

Definition 5 ([15]): Let H be a set of m hyperplanes and $P(H)$ be its polytope. Given a ray ρ from a point v towards a direction where $v \in P(H)$, a *ray shooting query* is to find the first hyperplane $h \in H$ hit by ρ.

Consider a single iteration of our *GeoGreedy* algorithm. In our setting, since the origin o lies inside the d-dimensional convex hull $Conv(S)$, we can adopt the indexing technique in [15] for our indexing. Specifically, for each face of $Conv(S)$, we construct a hyperplane containing this face and insert it into H, initialized to $\emptyset$. Then, we build an index as in [15] for H. Whenever we need to find the face of $Conv(S)$ crossed by the line passing through the origin and a point p, we issue a ray shooting query where the input is the ray from the origin o to point p. The hyperplane returned by the ray shooting query corresponds to the face of $Conv(S)$ crossed by the line passing through the origin and the point p.

Next, we describe how we adopt this method in *GeoGreedy* with *multiple* iterations. Note that in *GeoGreedy*, the set S expands and changes over time. A straightforward implementation of this indexing method is to construct the index from *scratch* for each iteration. That is, for an iteration where S is the current set, we construct the index based on all faces of $Conv(S)$. Obviously, this implementation is time-consuming.

Instead, we propose an *incremental* manner for this indexing method based on the following important observation. Consider the current iteration which select a point p_o to insert into the current set S. Let S' be the set S after p_o is inserted. That is, $S' = S \cup \{p_o\}$. Since S' is the selection set in the next iteration, we know that for the next iteration, the face of $Conv(S)$ crossed by the line passing through the origin and p_o should be removed and some new faces (of $Conv(S')$) containing p_o should be created. The above observation allows us to construct the incremental index efficiently since we do not need to re-determine the face of $Conv(S')$ crossed by the line passing through the origin and *all* points in $D \setminus S$.

Specifically, consider the current iteration. Suppose that for each point $p \in D \setminus S$, we know which face of $Conv(S)$ crossed by the line passing through the origin and p and thus we can find the p-critical point (for S) on this face. Next, consider the next iteration. Let f be the face of $Conv(S)$ crossed by the line passing through the origin and p_o. We know that the critical points of *only* the points which have their lines from the origin crossing the face f have to be re-computed (since f is removed in the next iteration). Note that the critical points of all other points remain unchanged since the other faces remain unchanged. In order to re-compute the critical point of each point which has its line from the origin crossing f, we just need to check which of the newly constructed faces (of $Conv(S')$) containing p_o is crossed by the line from the origin to this point. Since all other original faces do not need to be considered for this re-computation, our algorithm is very efficient.

TABLE III: REAL DATASETS

Dataset	No. of Dim.	Size	$\lvert D_{sky}\rvert$	$\lvert D_{happy}\rvert$	$\lvert D_{conv}\rvert$
household	6	903,077	9,832	1,332	927
nba	5	21,962	447	75	65
color	9	68,040	1,023	151	124
stocks	5	122,574	3,042	449	396

B. The StoredList Algorithm

In this section, we present an algorithm called the *StoredList* algorithm, a variation of the previous *GeoGreedy* algorithm, with a pre-computation step so that the query time can be enhanced significantly. Specifically, there are two phases. The first phase is the preprocessing phase. Firstly, we find a set D_{happy} of happy points. Secondly, we create a list L which is to store the happy points in D_{happy} next. Thirdly, we execute Algorithm 1 by setting the input parameter D to D_{happy} and the input parameter k to $|D_{happy}|$. In Algorithm 1, whenever we insert a happy point into the output set S, we insert it into our list L. This list L is used in the query phase for fast query processing. The second phase is the query phase. Whenever we encounter a k-regret query with a parameter k, we returns the first k points in L as the answer set. It is easy to verify that the answer set returned by *StoredList* is exactly the same as that returned by *GeoGreedy*. But, *StoredList* has to maintain a list containing all happy points in a particular order but *GeoGreedy* does not.

V. EMPIRICAL STUDIES

We conducted experiments on a workstation with 1.60GHz CPU and 8GB RAM. All programs were implemented by C++ and the experiments. Most of our experiments follow the same settings as those in [12].

We conducted experiments on both real datasets and synthetic datasets. We adopted four real datasets, namely, *household* (http://www.ipums.org), *nba* (http://www.basketballreference.com), *color* (http://kdd.ics.uci.edu) and *stocks* (http://pages.swcp.com/stocks). The number of dimensions and the size each of these real datasets can be found in Table III. Besides, we conducted experiments on synthetic datasets [16]. Following many existing papers about skyline queries we adopt the dataset generator developed by [16] to generate anti-correlated datasets. Unless stated explicitly, for synthetic datasets, the number of tuples is set to 10,000 (i.e., $n = 10,000$), the dimensionality is set to 6 (i.e., $d = 6$), and k is set to 10.

We compared the performance of our two proposed algorithms, namely *GeoGreedy* and *StoredList*, with the best-known fastest algorithm proposed in [12], namely *Greedy*. *GeoGreedy* corresponds to Algorithm 1 and *StoredList* corresponds to the algorithm described in Section IV-B. We evaluated each of the algorithms in terms of both its maximum regret ratio, its *query time* and its *total time* (to be described in detail next). We regard the step of finding a set of happy points (or a set of skyline points) as the *pre-processing* step. The *query time* of an algorithm corresponds to the time the algorithm takes excluding the time of the pre-processing step. The *total time* of an algorithm corresponds to the time the algorithm takes including the time of the pre-processing step.

We conducted three kinds of experiments. The first experiment (Section V-A) is to study some statistics of three sets, namely D_{sky}, D_{happy} and D_{conv}. The second experiment (Section V-B) is to study the performance of each algorithm on real datasets. The third experiment (Section V-C) is to study the performance of each algorithm on synthetic datasets.

A. Study on D_{sky}, D_{happy} and D_{conv}

In this experiment, we would like to study some statistics of three sets, namely D_{sky}, D_{happy} and D_{conv}. For each real dataset, we find the number of skyline points ($|D_{sky}|$), the number of happy points ($|D_{happy}|$) and the number of points on the convex hull ($|D_{conv}|$), as shown in Table III. In this table, we observe that for each real dataset, $|D_{sky}| > |D_{happy}| > |D_{conv}|$, which is consistent with our theoretical result in Lemma 3. According to this result, we find that $|D_{sky}|$ is significantly larger than $|D_{happy}|$. In particular, in dataset *household*, the ratio of $|D_{sky}|$ to $|D_{happy}|$ (i.e., $|D_{sky}|/|D_{happy}|$) is about 8.77, which is quite large. As we know, from Lemma 2, the optimal solution of MRRN is a subset of D_{happy}. The large ratio difference suggests that D_{happy} is very useful for MRRN compared with D_{sky} because all existing algorithms for MRRN can be executed based on D_{happy} with a very small size (compared with D_{sky}). Note that the experimental studies in [12] are based on D_{sky} which is much larger than D_{happy}.

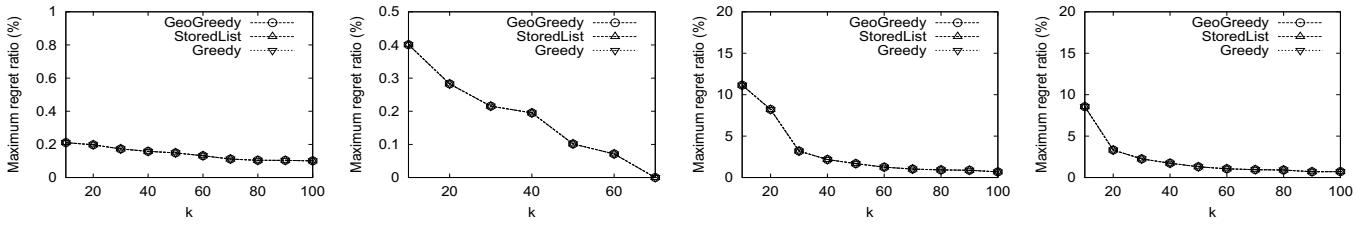
B. Results on Real Datasets

We study the effect of k on each of the real datasets in terms of the maximum regret ratio, the query time and the total time of each algorithm. In the experiments, we vary k from 10 to 100.

Maximum Regret Ratio: Since the solution of MRRN is a subset of D_{happy} and is a subset of D_{sky}, we conducted two sets of experiments. The first experiment is to execute each algorithm based on D_{happy}. The second experiment is to execute each algorithm based on D_{sky}.

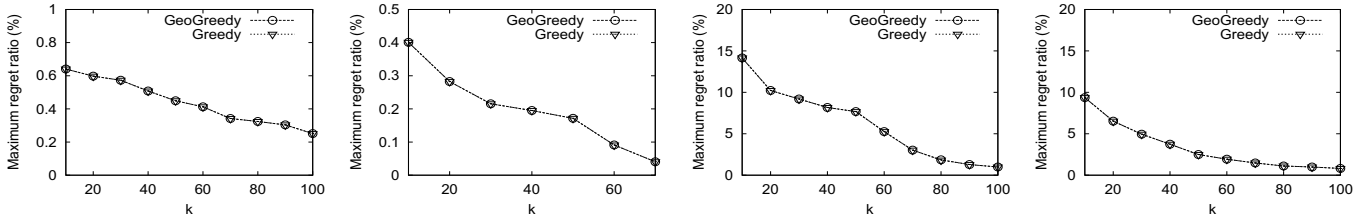
Figure 7 shows the results based on D_{happy}. Since all algorithms return the same answer set (because they follows the same greedy skeleton as discussed before), their maximum regret ratios are the same. When k increases, the maximum regret ratios of all algorithms decrease. This is because when more points/tuples are selected in the final selection set, the maximum regret ratio of an algorithm decreases.

Figure 8 shows the results based on D_{sky}. Here, since *StoredList* is based on D_{happy} instead of D_{sky}, we do not include it in the figure. We observe a similar trend in the figure. However, in general, the maximum regret ratio of each algorithm based on D_{sky} is larger than the one based on D_{happy}. This is because since D_{sky} is larger than D_{happy}, it is more likely that each of the algorithms selects points in some points in D_{sky} which cannot be found in D_{happy}. It is very likely that these points cannot be found in the optimal



(a) household (b) nba (c) color (d) stocks

Fig. 7: Effect on Maximum Regret Ratio Based on D_{happy} (Real Datasets)



(a) household (b) nba (c) color (d) stocks

Fig. 8: Effect on Maximum Regret Ratio Based on D_{sky} (Real Datasets)

solution of MRRM and increase the maximum regret ratio of the final selection set. According to this result, happy points can be regarded as some "better" candidates for the solution set of MRRM (compared with skyline points).

Query Time: Similarly, we conducted two kinds of experiments. The first one is based on D_{happy} and the second one is based on D_{sky}. For each experiment, we measured the query time of each algorithm.

Figure 9 shows the experimental results about the effect on the query time based on D_{happy}. In all sub-figures of Figure 9, as expected, *StoredList* is the fastest since it has a pre-computed list and it just returns the first k points/tuples immediately for each query issued. The query time of *GeoGreedy* is smaller than that of *Greedy* since *GeoGreedy* makes use of the geometry propery and thus its efficency can be improved compared with *Greedy*.

Figure 10 shows the experimental results about the effect on the query time based on D_{sky}. We observe similar trends. However, the query time of each algorithm based on D_{sky} is slightly larger than that based on D_{happy} since D_{sky} is larger than D_{happy}.

Total Time: Similarly, we conducted two experiments and measured the total time of each algorithm. Figure 11 shows the experimental results about the effect on the total time based on D_{happy}. We have a similar trend as the results for the query time. Since the total time of *StoredList* includes both the time of finding D_{happy} and the time of materializing the whole stored list, which is time consuming, the total time of *StoredList* is the largest. Compared with the query time, the total time of each algorithm is larger since the total time of each algorithm includes the time of finding D_{happy}.

We also conducted experiments for the second experiment based on D_{sky}. We observe similar trends. For the sake of space, we do not show the results here.

C. Results on Synthetic Datasets

We also conducted experiments on synthetic datasets. Since the experimental results in the previous section show that the results based on D_{happy} are better than those based on D_{sky}, in the following, we focus on experiments based on D_{happy} only. Besides, since we know that *StoredList*, developed from *GeoGreedy*, is very efficient compared with the two other algorithms (i.e., *Greedy* and *GeoGreedy*), in the following experiments, we focus on comparing between *Greedy* and *GeoGreedy*.

Maximum Regret Ratio: Figure 12(a) shows that when d increases, the maximum regret ratio of each algorithm increases. Figure 12(b) shows that when n increases, the maximum regret ratio of each algorithm increases slightly. We conclude that the size of the dataset is not a crucial factor that affects the maximum regret ratio of an algorithm. In Figure 12(c), the maximum regret ratio of each algorithm decreases when k increases.

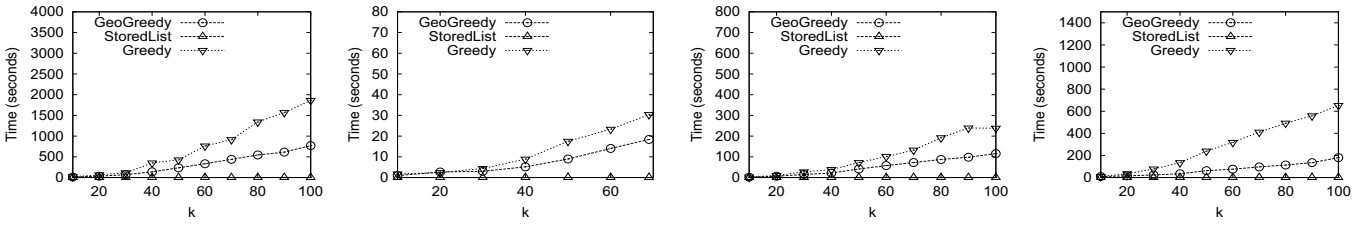
We would like to see how small the maximum regret ratio of each of the algorithms when k is very large. Figure 12(d) shows that when k is very large, the maximum regret ratio of each of these algorithm is very small (smaller than 9%).

Query Time: Figure 13 shows the corresponding query time of each algorithm for Figure 12. Figures 13(a), (b) and (c) show that the query time of each algorithm increases with d, n and k. Figure 13(d) shows that the query time of *GeoGreedy* is much smaller than that of *Greedy*. We conducted experiments on a large dataset containing 5 million tuples. We found that *Greedy* took 3 hours, *GeoGreedy* took within a few minutes and *StoredList* took within a second for a k-regret query where k is set to 100.

Total Time: We also conducted experiments to measure the total time of each algorithm. The results are similar to results on real datasets. For the sake of space, we do not include them in this paper.

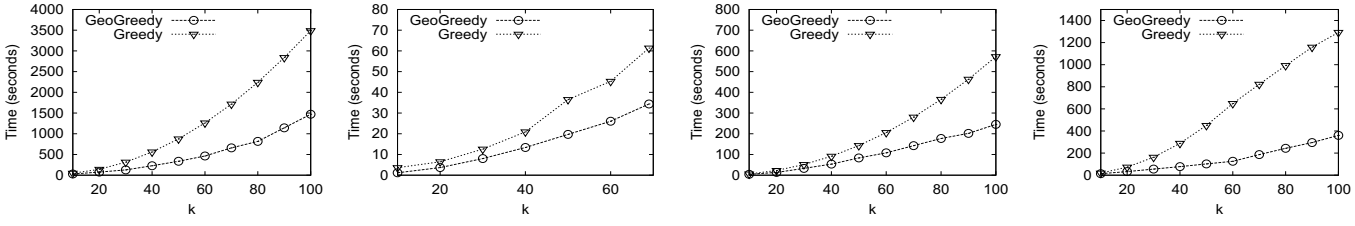
D. Summary

StoredList has the shortest query time due to its materialized list storing all pre-computed answers. Besides, *GeoGreedy* performs more efficiently than the best-known fastest algorithm, *Greedy*, due to its geometry property in all datasets.



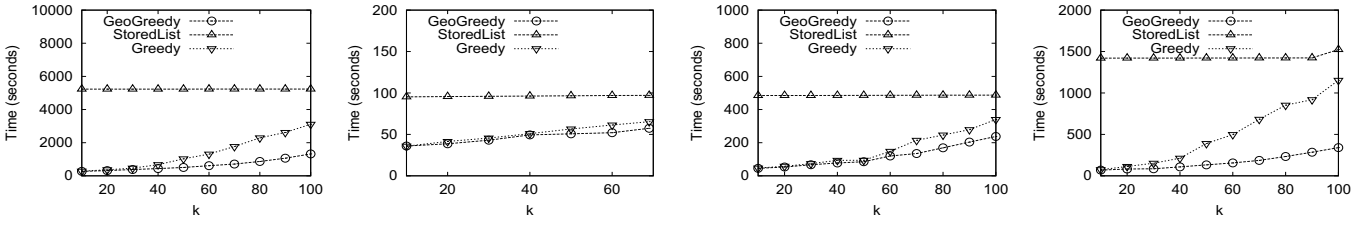
(a) household (b) nba (c) color (d) stocks

Fig. 9: Effect on Query Time Based on D_{happy} (Real Datasets)



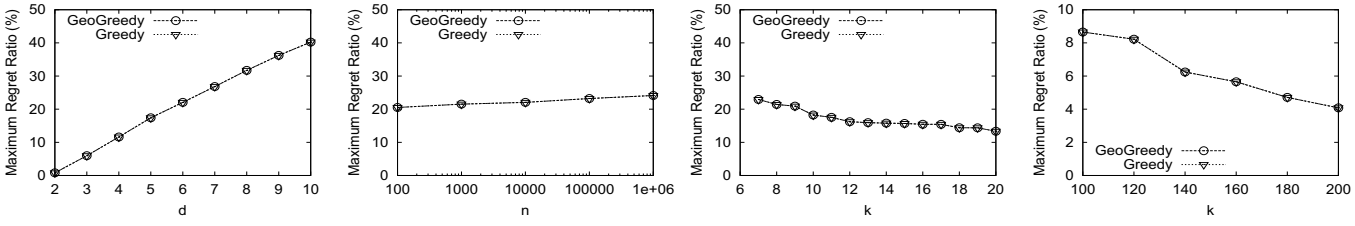
(a) household (b) nba (c) color (d) stocks

Fig. 10: Effect on Query Time Based on D_{sky} (Real Datasets)



(a) household (b) nba (c) color (d) stocks

Fig. 11: Effect on Total Time Based on D_{happy} (Real Datasets)



(a) Effect of d (b) Effect of n (c) Effect of k (d) Effect of Large k

Fig. 12: Effect on Maximum Regret Ratio on D_{happy} (Synthetic Datasets)

All algorithms returns the same maximum regret ratios in all cases since they follow the same greedy skeleton. Thus, both *StoredList* and *GeoGreedy* are better than *Greedy*.

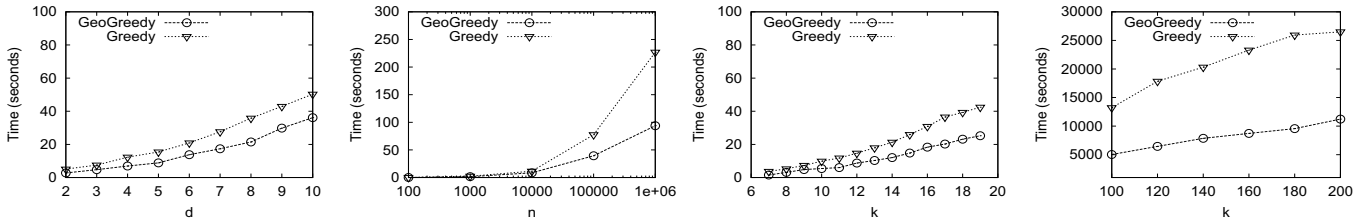
VI. Related work

Top-k queries and skyline queries are two popular and fundamental queries, which support users for decision-making. However, as discussed in [12], there are some deficiencies of these two types of queries. Top-k queries require users to specify utility functions and skyline queries suffer from a potential large output problem. Due to the deficiencies of these queries, a number of alternatives were proposed recently.

Firstly, some existing studies [8], [11], developed based on skyline queries, studied how to reduce the output size of a skyline query. One kind of queries is called a *k-representative skyline* query [8], [11]. A k-representative skyline query [8], [11] outputs a set of k "representative" points from a set of the skyline points. Different papers have different definitions about the concept of "representative" points. [8] proposed to find k points which dominate the greatest number of points as "representative" points. [11] proposed to find k points such that each of these points is far away from the other points.

Secondly, some existing papers, developed based on top-k queries, proposed alternative ways to users to specify utility functions for top-k queries. [9] requested users to specify a small number of possible weights each denoting the importance of a dimension. [2] requested users to specify some pairwise comparisons between two dimensions to indicate whether a dimension is more important than the other dimension for a comparison. However, these studies require users to specify some kinds of utilities/preferences.

Thirdly, there are other studies which were developed based on both top-k queries and skyline queries. For example, these studies found k points in the result such that each point in the result is not dominated by other points in the result. [1] is an example studying a top-k skyline select query and a top-k skyline join query based on a user-defined utility function. [17] is another example studying an ϵ-skyline query which returns a set of points whose size is controllable according to ϵ when a user specify a utility function. However, in these studies, users still need to specify some kinds of utilities/preferences.



(a) Effect of d (b) Effect of n (c) Effect of k (d) Effect of Large k

Fig. 13: Effect on Query Time on D_{happy} (Synthetic Datasets)

Fourthly, some existing studies proposed queries which do not heavily rely on top-k queries and skyline queries. The first example is a k-*regret* query first proposed in [12]. As we described before, it is quite attractive because users do not need to specify any utility function and k points are returned to users. The second example is a diversified top-k query [18], [19], [20] studied recently. Given a query point q with some keywords, a diversified top-k query is to return a set of k points such that k points are "relevant" to q and each of these k points is not quite similar to the other points. Different from a k-regret query, a query point is needed in this query.

VII. Discussion

It is also interesting to consider the case when $k < d$. Our theoretical result stressed that the size of the subset should be at least the dimensionality of the database. If the size of the subset is smaller than the dimensionality, even the maximum regret ratio in the *optimal solution* is unbounded. We give a very simple example for illustration. Consider the case that there are 4 points in the database, each of which can be represented as a 4-dimensional vector. Specifically, the corresponding vector of these four points are $(\delta, \delta, \delta, 1)$, $(\delta, \delta, 1, \delta)$, $(\delta, 1, \delta, \delta)$ and $(1, \delta, \delta, \delta)$ where δ is a very small positive number near to 0. When a user issues a k-regret query by setting $k = 3$, no matter which three points we select from these four points on the boundary, the maximum regret ratio of the selection set is nearly equal to 1.

The inherent reason for the unboundedness in the case of $k < d$ is that when each tuple in the database is represented as a point in a higher dimensional space, the possibility of choosing a point which maximizes the user's underlying utility function reduces exponentially if we remain the same output size. Because of the *curse of dimensionality*, k should normally be much larger than d. Therefore, it is impractical to consider a small k for a relative large d through this perspective.

VIII. Conclusions

We studied a k-regret problem in this paper. We proposed two algorithms called *GeoGreedy* and *StoredList*. *GeoGreedy* is a greedy algorithm using some geometry properties for speeding up the search. *StoredList* is a materialization version of *GeoGreedy* which can further speed up the query time with some materialized storage. We conducted experiments to verify the effectiveness and the efficiency of the proposed algorithms.

There are many promising future directions. Firstly, we can consider the problem of minimizing the *average* regret of a set of points instead of the maximum regret studied in this paper. Secondly, we can study how to adapt our geometry properties in an *interactive* setting studied in [13]. That is, a specific user is asked to choose a "best" point from a set of selected points iteratively such that his/her utility function can be learnt finally.

Acknowledgement

We are grateful to the anonymous reviewers for their constructive comments on this paper. This research is supported by grant FSGRF14EG34.

References

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

[2] J. Lee, G. You, and S. Hwang, "Personalized top-k skyline queries in high-dimensional space," *Information Systems*, vol. 34, no. 1, pp. 45–61, 2009.

[3] X. Lian and L. Chen, "Top-k dominating queries in uncertain databases," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 660–671.

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

[5] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang, "Top-k query processing in uncertain databases," in *ICDE*, 2007, pp. 896–905.

[6] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *SIGMOD*, 2006, pp. 503–514.

[7] ——, "On high dimensional skylines," *Advances in Database Technology-EDBT 2006*, pp. 478–495, 2006.

[8] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *ICDE*, 2007, pp. 86–95.

[9] D. Mindolin and J. Chomicki, "Discovering relative importance of skyline attributes," *VLDB*, vol. 2, no. 1, pp. 610–621, 2009.

[10] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *TODS*, vol. 30, no. 1, pp. 41–82, 2005.

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

[12] D. Nanongkai, A. Sarma, A. Lall, R. Lipton, and J. Xu, "Regret-minimizing representative databases," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1114–1124, 2010.

[13] D. Nanongkai, A. Lall, and A. Sarma, "Interactive regret minimization," *Proceedings of the 2012 ACM SIGMOD*, 2012.

[14] D. Henderson and E. Moura, *Experiencing geometry: On plane and sphere*. Prentice Hall Upper Saddle River, NJ, 1996.

[15] P. K. Agarwal and J. Matoušek, "Ray shooting and parametric search," *SIAM Journal on Computing*, vol. 22, no. 4, pp. 794–806, 1993.

[16] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001.

[17] T. Xia, D. Zhang, and Y. Tao, "On skylining with flexible dominance relation," in *ICDE*, 2008.

[18] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proceedings of the Second ACM International Conference on Web Search and Data Mining*. ACM, 2009, pp. 5–14.

[19] P. Fraternali, D. Martinenghi, and M. Tagliasacchi, "Top-k bounded diversification," in *Proceedings of the 2012 international conference on Management of Data*. ACM, 2012, pp. 421–432.
[20] L. Qin, J. Yu, and L. Chang, "Diversifying top-k results," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1124–1135, 2012.
[21] P. Peng and R. C.-W. Wong, "Geometry approach for k-regret query (technical report)," in *http://www.cse.ust.hk/~raywong/paper/regret-technical.pdf*, 2014.

APPENDIX: PROOF OF LEMMAS/THEOREMS

Proof of Lemma 1: Before we give the proof, we first introduce some notations and a lemma which will be used in the proof.

Given two d-dimensional vectors ω and ω', the *angle* between ω and ω' is denoted by $\theta(\omega, \omega')$. With this notation, we know that the dot product between ω and ω', denoted by $\omega \cdot \omega'$, is equal to $\|\omega\| \cdot \|\omega'\| \cdot \theta(\omega, \omega')$.

Consider a set S of points. Since $Conv(S)$ is defined based on the *orthotope sets* of points in S, we know that for each hyperplane containing a face of $Conv(S)$, there exists a non-negative d-dimensional vector ω such that the hyperplane can be represented in the form of "$\omega \cdot x = \omega \cdot p$" where p is a point on the hyperplane. In the following, for clarity, when we write ω, we implicitly mean that ω is a non-negative d-dimensional vector.

Note that the norm of each weight vector ω, denoted by $\|\omega\|$, is equal to 1. Given a face f and a hyperplane h, we say that f is *below* h if each point on f is below h. Given a face f, a point p on f and a hyperplane h, we say that f excluding p is *below* h if each point on f excluding p is below h.

After we described the notations, we give the following lemma which will be used in our proof.

Lemma 6: Let h be a hyperplane containing a point p which is represented by equation "$\omega \cdot x = \omega \cdot p$" where ω is a d-dimensional vector with its norm equal to 1. Let l be the line passing through the origin and q. Let q_ω be the intersection between h and l. Then, $\frac{\|p\| \cdot \cos\theta(\omega,p)}{\|q\| \cdot \cos\theta(\omega,q)} = \frac{\|q_\omega\|}{\|q\|}$. ■

Proof: Since q_ω is on h, we have

$$\begin{aligned} \omega \cdot q_\omega &= \omega \cdot p \\ \|\omega\| \cdot \|q_\omega\| \cdot \cos(\omega, q_\omega) &= \|\omega\| \cdot \|p\| \cdot \cos(\omega, p) \\ \|p\| \cdot \cos(\omega, p) &= \|q_\omega\| \cdot \cos(\omega, q_\omega) \end{aligned} \quad (1)$$

Since q and q_ω are along line l, we have

$$\theta(\omega, q) = \theta(\omega, q_\omega) \quad (2)$$

By Equation (1) and Equation (2), we have

$$\frac{\|p\| \cdot \cos\theta(\omega,p)}{\|q\| \cdot \cos\theta(\omega,q)} = \frac{\|q_\omega\|}{\|q\|}$$

■

With the notations and the above lemma, we are ready to show that $mrr(S) = 1 - r$ next where $r = \min_{q \in D} cr(q, S)$. Let $\mathcal{W}$ be the set of the weight vectors for all possible utility functions in $\mathcal{F}$. According to Definition 2, we know that $mrr(S) = \max_{f \in \mathcal{F}} rr(S, f) = \max_{f \in \mathcal{F}}\{1 - \frac{U_{max}(S,f)}{U_{max}(D,f)}\} = \max_{f \in \mathcal{F}}\{1 - \frac{\max_{p \in S} f(p)}{\max_{q \in D} f(q)}\} = \max_{\omega \in \mathcal{W}}\{1 - \frac{\max_{p \in S} \omega \cdot p}{\max_{q \in D} \omega \cdot q}\} = 1 - \min_{\omega \in \mathcal{W}} \frac{\max_{p \in S} \omega \cdot p}{\max_{q \in D} \omega \cdot q} = 1 - \min_{\omega \in \mathcal{W}} \min_{q \in D} \frac{\max_{p \in S} \omega \cdot p}{\omega \cdot q} = 1 - \min_{q \in D} \min_{\omega \in \mathcal{W}} \frac{\max_{p \in S} \omega \cdot p}{\omega \cdot q}$.

Given a point $q \in D$, we define $g(q, S) = \min_{\omega \in \mathcal{W}} \frac{\max_{p \in S} \omega \cdot p}{\omega \cdot q}$. We have $mrr(S) = 1 - \min_{q \in D} g(q, S)$.

In the following, we show that $g(q, S) = cr(q, S)$. After we know that $g(q, S) = cr(q, S)$, we have $mrr(S) = 1 - \min_{q \in D} cr(q, S) = 1 - r$, which completes the proof. In the remaining of the proof, we only focus on showing that $g(q, S) = cr(q, S)$.

Suppose that $S = \{p_1, p_2, ..., p_k\}$. For each $i \in [1, k]$, we define $\mathcal{W}_i$ to be the set of all possible weight vectors in $\mathcal{W}$ such that p_i is a maximum utility point of S for the utility function with each of these weight vectors. Formally, for each $i \in [1, k]$, $\mathcal{W}_i$ is defined to be the set of all possible weight vectors ω in $\mathcal{W}$ such that for each $p \in S$, $\omega \cdot p_i \geq \omega \cdot p$. For each $i \in [1, k]$, if $\omega \in \mathcal{W}_i$, then $\max_{p \in S} \omega \cdot p$ can simply be written as $\omega \cdot p_i$. Note that $\mathcal{W} = \cup_{i=1}^{k} \mathcal{W}_i$.

Consider $g(q, S) = \min_{\omega \in \mathcal{W}} \frac{\max_{p \in S} \omega \cdot p}{\omega \cdot q} = \min_{i \in [1,k]} \min_{\omega \in \mathcal{W}_i} \frac{\max_{p \in S} \omega \cdot p}{\omega \cdot q} = \min_{i \in [1,k]} \min_{\omega \in \mathcal{W}_i} \frac{\omega \cdot p_i}{\omega \cdot q} = \min_{i \in [1,k]} \min_{\omega \in \mathcal{W}_i} \frac{\|\omega\| \cdot \|p_i\| \cdot \cos\theta(\omega, p_i)}{\|\omega\| \cdot \|q\| \cdot \cos\theta(\omega, q)}$. Thus, we have

$$g(q, S) = \min_{i \in [1,k]} \min_{\omega \in \mathcal{W}_i} \frac{\|p_i\| \cdot \cos\theta(\omega, p_i)}{\|q\| \cdot \cos\theta(\omega, q)} \quad (3)$$

Note that each face of $Conv(S)$ is a convex polygon with its vertices from S and is on a hyperplane containing these vertices. Let l be the line passing through the origin and q. We know that l intersects with a face of $Conv(S)$. There exists a point $p_{i_q} \in S$ on this face where $i_q \in [1, k]$. Note that this face is on a hyperplane containing p_{i_q} represented by equation "$\omega_q \cdot x = \omega_q \cdot p_{i_q}$" where ω_q is a d-dimensional vector with its norm equal to 1. Denote this hyperplane by h_{ω_q}.

Let q' be the q-critical point for S. Note that $cr(q, S) = \frac{\|q'\|}{\|q\|}$ and q' is the intersection between h_{ω_q} and l.

From Equation (3), there is a term "$\min_{\omega \in \mathcal{W}_i} \frac{\|p_i\| \cdot \cos\theta(\omega,p_i)}{\|q\| \cdot \cos\theta(\omega,q)}$" where $i \in [1, k]$. We want to find the expression for this term with the following two lemmas.

Lemma 7: $[\min_{\omega \in \mathcal{W}_{i_q}} \frac{\|p_{i_q}\| \cdot \cos\theta(\omega,p_{i_q})}{\|q\| \cdot \cos\theta(\omega,q)}] = \frac{\|q'\|}{\|q\|}$. ■

Proof: Note that h_{ω_q} is a hyperplane containing point p_{i_q} represented by equation "$\omega_q \cdot x = \omega_q \cdot p_{i_q}$". Since q' is the intersection between h_{ω_q} and l, by Lemma 6, we have

$$\frac{\|p_{i_q}\| \cdot \cos\theta(\omega_q,p_{i_q})}{\|q\| \cdot \cos\theta(\omega_q,q)} = \frac{\|q'\|}{\|q\|} \quad (4)$$

Consider an arbitrary $\omega \in \mathcal{W}_{i_q}$. Let h_ω be the hyperplane containing p_{i_q} represented by "$\omega \cdot x = \omega \cdot p_{i_q}$". Let q_ω be the intersection between h_ω and l.

By Lemma 6, we derive that for each $\omega \in \mathcal{W}_{i_q}$,

$$\frac{\|p_{i_q}\| \cdot \cos\theta(\omega,p_{i_q})}{\|q\| \cdot \cos\theta(\omega,q)} = \frac{\|q_\omega\|}{\|q\|} \quad (5)$$

For each $\omega \in \mathcal{W}_{i_q}$, we know that for each $p \in S$, $\omega \cdot p_{i_q} \geq \omega \cdot p$, and thus p is below h_ω or on h_ω (because h_ω is represented by equation "$\omega \cdot x = \omega \cdot p_{i_q}$"). Since each face of $Conv(S)$ is a convex polygon with its vertices from S and is on a hyperplane containing these vertices, we deduce that each face of $Conv(S)$ excluding p_{i_q} is below h_ω or on h_ω.

Since q' is on a face of $Conv(S)$, we know that q' is below h_ω or on h_ω. Since q_ω and q' are along the same line l, and h_ω contains q_ω, we derive that for each $\omega \in \mathcal{W}_{i_q}$, $\|q_\omega\| \geq \|q'\|$. From Equation (5), we know that for each $\omega \in \mathcal{W}_{i_q}$, $\frac{\|p_{i_q}\| \cdot \cos\theta(\omega, p_{i_q})}{\|q\| \cdot \cos\theta(\omega,q)} \geq \frac{\|q'\|}{\|q\|}$. Since $\omega_q \in \mathcal{W}_{i_q}$, with Equation (4), we conclude that $[\min_{\omega \in \mathcal{W}_{i_q}} \frac{\|p_{i_q}\| \cdot \cos\theta(\omega, p_{i_q})}{\|q\| \cdot \cos\theta(\omega,q)}] = \frac{\|q'\|}{\|q\|}$. ■

Lemma 8: If $i \neq i_q$, then $[\min_{\omega \in \mathcal{W}_i} \frac{\|p_i\| \cdot \cos\theta(\omega, p_i)}{\|q\| \cdot \cos\theta(\omega,q)}] \geq \frac{\|q'\|}{\|q\|}$. ■

Proof: Consider an arbitrary $\omega \in \mathcal{W}_i$. Let h_ω be the hyperplane containing p_i represented by "$\omega \cdot x = \omega \cdot p_i$". Let q_ω be the intersection between h_ω and l.

By Lemma 6, we derive that for each $\omega \in \mathcal{W}_i$,

$$\frac{\|p_i\| \cdot \cos\theta(\omega, p_i)}{\|q\| \cdot \cos\theta(\omega,q)} = \frac{\|q_\omega\|}{\|q\|} \tag{6}$$

For each $\omega \in \mathcal{W}_i$, we know that for each $p \in S$, $\omega \cdot p_i \geq \omega \cdot p$, and thus p is below h_ω or on h_ω (because h_ω is represented by equation "$\omega \cdot x = \omega \cdot p_i$"). Since each face of $Conv(S)$ is a convex polygon with its vertices from S and is on a hyperplane containing these vertices, we deduce that each face of $Conv(S)$ excluding p_i is below h_ω or on h_ω. Since q' is on a face of $Conv(S)$, we know that q' is below h_ω or on h_ω. Since q_ω and q' are along the same line l, and h_ω contains q_ω, we derive that for each $\omega \in \mathcal{W}_i$,

$$\|q_\omega\| \geq \|q'\|$$

From Equation (6), we know that for each $\omega \in \mathcal{W}_i$,

$$\frac{\|p_i\| \cdot \cos\theta(\omega, p_i)}{\|q\| \cdot \cos\theta(\omega,q)} \geq \frac{\|q'\|}{\|q\|}$$

We conclude that

$$[\min_{\omega \in \mathcal{W}_i} \frac{\|p_i\| \cdot \cos\theta(\omega, p_i)}{\|q\| \cdot \cos\theta(\omega,q)}] \geq \frac{\|q'\|}{\|q\|}$$

■

By the above two lemmas (i.e., Lemma 7 and Lemma 8), from Equation (3), we have

$$g(q, S) = \frac{\|q'\|}{\|q\|} = cr(q, S)$$

which completes the proof. ■

Proof Sketch of Lemma 2: We prove by contradiction. Suppose that there does not exist any subset S' of the set of all happy points in D such that $|S'| = k$ and $mrr(S') = mrr_o$. Let S be the set of k points such that $mrr(S) = mrr_o$ and some of the k points are not happy points. We iteratively replace some non-happy points in S with some points in $D \setminus S$ with two steps until each point in S is a happy point. Specifically, the first step (Step I) is to iteratively replace each non-happy point in S which is a boundary point of S and is subjugated by a point in S with a point in $D \setminus S$ until each boundary point of S is not subjugated by any point in S. The second step (Step II) is to replace a single non-happy point p in S with the point $q \in D \setminus S$ subjugating p. After each of these steps, we show that the maximum regret ratio of the selection set does not increase. When the above iterative steps stop, we know that each point in S is a happy point and the maximum regret ratio of S is at most mrr_o. Since mrr_o is the smallest maximum regret ratio of a set containing k points, we conclude that the maximum regret ratio of this set is equal to mrr_o, which leads to a contradiction. A detailed proof can be found in [21].

Proof of Lemma 3: Firstly, we show that $D_{conv} \subseteq D_{happy}$. That is, we show that for each point $p \in D_{conv}$, p is also a happy point. In other words, each point $p \in D_{conv}$ is not subjugated by any point in D. We prove by contradiction. Suppose that there exists a point $p \in D_{conv}$ and a point $q \in D$ such that p is subjugated by q. Since p is subjugated by q, we know that p is on or below each of the hyperplanes in $Y(q)$ and is below at least one hyperplane in $Y(q)$.

Let P_q be the polyhedron formed by $Conv(\{q\} \cup VC)$. and P_D be the polyhedron formed by $Conv(D)$. Note that for each $i \in [1, d]$, there exists a point in D with its value on the i-th dimension equal to 1. Thus, we know that $Conv(D) = Conv(D \cup VC)$. Since $q \in D$, due to the convex property of $Conv(D)$, we know that P_D covers P_q. That is, each point in P_q is in P_D.

Note that each face of $Conv(\{q\} \cup VC)$ is on a hyperplane in $Y(q)$, and each hyperplane in $Y(q)$ contains a face of $Conv(\{q\} \cup VC)$. Besides, note that an extreme point of each face of P_q is q, a point in $Orth(q)$ or a virtual corner point in VC. Furthermore, note that one of the dimensions of a point in $Orth(q)$ is equal to 0 and one of the dimensions of a point in VC is 0. Since each dimensional value of p is positive, and p is on or below each of the hyperplanes in $Y(q)$ and is below at least one hyperplane in $Y(q)$, we deduce that p is in P_q and is not an extreme point of each face of P_q. Since P_D covers P_q, we know that p is also in P_D and is not an extreme point of each face of P_D. We deduce that $p \notin D_{conv}$, which leads to a contradiction.

Secondly, we show that $D_{happy} \subseteq D_{sky}$. That is, we show that for each point $p \in D_{happy}$, p is also a skyline point. In other words, each point $p \in D_{happy}$ is not dominated by any point in D. We prove by contradiction. Suppose that there exists a point $p \in D_{happy}$ and a point $q \in D$ such that p is dominated by q.

Consider an arbitrary hyperplane h in $Y(q)$. Since it contains q, it can be represented by equation "$\omega \cdot x = \omega \cdot q$" where ω is a d-dimensional vector.

Since p is dominated by q, we know that for each $i \in [1, d]$, $q[i] \geq p[i]$ and there exists an integer $i_o \in [1, d]$ such that $q[i_o] > p[i_o]$. We deduce that $\omega \cdot q \geq \omega \cdot p$. Thus, p is on or below hyperplane h.

Similarly, we conclude that p is on or below each hyperplane in $Y(q)$. Furthermore, we deduce that p is below at least one hyperplane in $Y(q)$ (since there exists an integer $i_o \in [1, d]$ such that $q[i_o] > p[i_o]$). So, p is subjugated by q. Thus, p is not a happy point, which leads to a contradiction. ■