Interactive Learning for Diverse Top-k Set

ABSTRACT

The top-k query is a representative multi-criteria decision-making operator that assists users in finding the best k tuples based on their criteria. However, it has certain limitations in both the query process and the final output. First, the query process requires knowing the users' criteria in advance, which may be difficult for some users to specify explicitly and accurately. Second, the final output often lacks diversity, potentially leading to user dissatisfaction. To address the limitations, in this paper, we enhance the top-k query by incorporating an interactive learning framework and a diversity mechanism, expecting to return a diverse output that aligns with the user's criterion, even if the criterion is not specified in advance.

We study our problem progressively. Initially, we examine a special case in which tuples are described by two scoring attributes. We present algorithm *TDIA* that is asymptotically optimal regarding the user effort needed for interaction. After that, we move on to the general case in which tuples are described by multiple scoring attributes. We propose algorithm *HDIA* that has a provable guarantee on the user effort needed for interaction. We also consider the situation where users may make mistakes during the interaction. Experiments were conducted on synthetic and real datasets, showing that our algorithms can return a diverse output by costing less user effort than existing ones.

ACM Reference Format:

. 2018. Interactive Learning for Diverse Top-k Set. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

One major task of database systems is to help users search for tuples in the database that align with their criteria, such as purchasing a car, admitting college students, and renting an apartment [61, 63, 65, 68]. Take the scenario of purchasing a car for illustration. Consider a car database as shown in Table 1. Each car is described by some attributes, e.g., price, horsepower, and brand. Suppose that a user named Alice wants to buy a car. It can be a daunting task for her to manually review each car one by one in the database. However, if the database system could recommend personalized candidates for Alice, it would significantly streamline her search process. This would not only save her valuable time by filtering out unsuitable tuples, but also assist her in making a well-informed decision.

In the literature, various operators [33, 42, 45, 56, 57], known as the *multi-criteria decision-making* operators, are proposed for this scenario. They characterize each user's criterion by a *utility function* $f_{\boldsymbol{u}}$. Each tuple $\boldsymbol{p}$ is then associated with a *utility* $f_{\boldsymbol{u}}(\boldsymbol{p})$ (i.e., a function score), reflecting how well the tuple aligns with the user's criterion. Consequently, the operators find tuples as the output guided by these utilities.

Table 1: The Car Database

Car	Price	Horsepower	Brand	$f_{\boldsymbol{u}_1}(\cdot)$
$\boldsymbol{p}_1$	\$5000	450hp	Tesla	5.05
$\boldsymbol{p}_2$	\$4000	400hp	Ford	5.60
$\boldsymbol{p}_3$	\$6000	500hp	Tesla	4.50
$\boldsymbol{p}_4$	\$3500	350hp	Ford	5.65
$\boldsymbol{p}_5$	\$4500	420hp	Tesla	5.28
$\boldsymbol{p}_6$	\$3600	380hp	Ford	5.82

There are two critical limitations of these operators. First, they rely on the assumption that the user's utility function is known in advance [41, 66]. However, in real-world scenarios, users often face difficulty quantifying their individual criteria precisely. For instance, Alice might struggle to articulate her exact trade-off between a car's price and horsepower. This uncertainty in user criteria presents a significant challenge to these operators, limiting their effectiveness in making accurate recommendations. Second, these operators tend to overlook the importance of diversity in the output. In Table 1, cars $\boldsymbol{p}_2$, $\boldsymbol{p}_4$, and $\boldsymbol{p}_6$ are the three cars with the highest utilities. If they are recommended to Alice, she might feel disappointed because these cars are from the same brand. The need for diversity is pervasive in many real-life scenarios. For example, fellowship programs might demand diverse nominees to ensure representation from underrepresented demographic groups in a particular field [1, 2]. Therefore, it is desirable if the database system can search for a diverse set of tuples with high utilities, even if the user's criterion is not known in advance.

In this paper, we study how a well-established *interactive learning framework* [41, 61, 66] and a widely used *diversity mechanism* [46, 74] could help to address these two limitations. In line with prior work [5, 58], we distinguish the tuple attributes into two types: *scoring attributes* which are used by the utility function in the interactive learning framework (e.g., price and horsepower); and *sensitive attributes* which are used by the diversity mechanism (e.g., brand).

The interactive learning framework contains a series of questions. Each question presents the scoring attributes of two tuples, which are selected from the database based on the user's answers to previous questions. From the two presented tuples, the user is asked to pick the one s/he prefers. Then, according to the user's feedback, the user's utility function is learned implicitly. This kind of interactive mode naturally appears in our daily life. For example, a car seller might show Alice two cars and ask her to select the preferred one. A real estate agency may present two apartments and ask Alice: which one would you like to rent? It is important to note that the tuples in each question are selected strategically so that the number of questions asked is manageable, since excessive questions may make users lose the plot and become excessively

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

disappointed, affecting the interaction results, as shown in the literature of the marketing research [39, 48]. Therefore, we strive to ask as few questions as possible.

The diversity mechanism operates by partitioning tuples into groups based on the values of their sensitive attributes. Given a tuple set, its diversity may be assessed in many ways, for example, by whether the distribution of these groups within the set closely mirrors their distribution within the entire database [22]. While our techniques are adaptable to various settings of diversity, we follow the popular method of defining diversity requirements as constraints on the number of tuples selected from each group. To illustrate, consider Table 1. The cars can be partitioned into two groups with the *Tesla* brand and the *Ford* brand, respectively. Let us restrict the output to include at least one car from each group. Then, set $\{\boldsymbol{p}_1, \boldsymbol{p}_2\}$ is considered diverse, since it contains one tuple $\boldsymbol{p}_1$ from the group with the *Tesla* brand and one tuple $\boldsymbol{p}_2$ from the group with the *Ford* brand.

By incorporating the *interactive learning framework* and the *diversity mechanism*, we propose problem *Interactive Learning for Diverse Top-k Set (problem IDT)*, which interacts with a user to find the diverse set of k tuples that have the highest utilities by minimizing the number of questions asked to the user.

To the best of our knowledge, we are the first to study problem IDT. Some closely related studies [5, 30, 41, 47, 66, 74] involve interactive learning or consider the diversity issue, but are significantly different from us. [30] aims to find the ranking of tuples by interacting with users, and [47] proposes to learn the user's utility function via interaction. These studies learn the user criteria through interactive learning but do not incorporate a *diversity* constraint. Thus, some questions they need to ask might be redundant in our setting due to diversity considerations. For example, in our setting, we do not need to ask Alice to compare two cars if we already know they cannot both be part of the diverse output set, but this comparison could be needed by [30, 47]. [41, 66] target to find a single tuple such that the *regret ratio*, evaluating user dissatisfaction with the tuple, is minimized via interaction. Their output of just one tuple, compared with our goal of recommending a diverse set of k tuples, overlooks the need for diversity. Other studies such as [5, 35, 74] concentrate on providing a diverse tuple set to users. However, [5, 35] require knowing the users' criteria in advance, which is not always practical. [74] considers all possible users' criteria collectively to recommend tuples, leading to less personalized recommendations since they aim at accommodating a broad range of user criteria rather than being tailored to individual users.

Contributions. Our contributions are listed as follows.

- To the best of our knowledge, we are the first to propose the problem of returning the diverse top-k set by interacting with the user (problem IDT). We prove a lower bound $\Omega(\log_2 n)$ on the number of questions asked to a user.
- We propose an algorithm *TDIA* for a special case of problem IDT where tuples are described by two scoring attributes. It asks an asymptotically optimal number of questions.
- We propose an algorithm *HDIA* for the general case of problem IDT where tuples are described by multiple scoring attributes. Algorithm *HDIA* has a provable guarantee on the number of questions asked to a user and performs well empirically. Based on algorithm *HDIA*, we also consider the situation where users may answer questions mistakenly during the interaction.
- We conducted experiments to demonstrate the superiority of our algorithms. The results show that our algorithms are able to return the diverse top-k set by asking approximately 40% fewer questions than existing ones under typical settings.

In the following, we discuss the related work in Section 2 and formally define our problem in Section 3. In Section 4, we propose an asymptotically optimal algorithm *TDIA* for a special case. In Section 5, we propose an algorithm *HDIA* for the general case that performs well theoretically and empirically. Section 6 presents our experimental results and Section 7 concludes this paper.

2 RELATED WORK

There are various studies concerned with the realm of multi-criteria decision-making. In the following, we focuses on two main sections. The first section elaborates several studies that involve interactive learning, and the second section centers around a variety of studies addressing diversity issues.

Interaction-based Learning. Many queries [7–10, 34, 41, 52, 66, 73] integrate an interactive learning framework into their query process. The central idea is to learn the users' criteria by interacting with them and then return tuples according to the learned criteria.

[7, 8, 34] propose the interactive skyline query. The skyline query aims to find various tuples satisfying *all* possible user criteria, usually resulting in a large output size. This motivates [7, 8, 34] to *narrow* the range of possible users' criteria by interacting with users, reducing the output size. However, the interactive skyline query is limited to learn the user's criterion on the attribute values (e.g., a user prefers the value of the color attribute *red* to *yellow*), and misses the user's criterion between attributes (e.g., attribute price is more important than horsepower). Due to this limitation, even if the criterion on all attribute values is obtained, the output size could still be arbitrarily large [42].

[9, 10, 52] propose the interactive similarity query. It learns the user's query tuple and distance function via user interaction, and then returns tuples with the smallest distances to the query tuple. The weakness of this query is that, during the interaction, it requires a user to assign *relevance scores* to hundreds or thousands of tuples to learn how close the tuples are to the query tuple, which is too demanding in practice. In contrast, our problem IDT interacts with users by a few easy questions (i.e., selecting one from two tuples).

Authors of [41] propose the interactive regret-minimizing query, which aims to reduce the *regret ratio* while maintaining a small output size by interacting with the user. Here, the regret ratio is a measurement used to evaluate the returned tuples. However, this query displays *fake tuples* during the interaction, which are artificially constructed (not selected from the database). This might produce unrealistic tuples (e.g., a car with 10 dollars and 50000 horsepower), leading to potential user disappointment [66]. To overcome the defect, [66] proposes the strongly truthful interactive regret minimizing query, which utilizes *real tuples* (selected from the database). However, it requires heavy user effort during the interaction, i.e., asking many questions. Besides, since both [41, 66] focus on finding the user's (close to) best tuple, to some extent, their studied problems can be seen as a special case of our problem

IDT when $k = 1$. To reduce the user effort, [62] reduces the quality of the output. Instead of finding the (close to) best tuple, it only searches for one of the user's top-k tuples. In this way, it lessens the precision required to learn the user's criterion, and thus, reduces the number of questions asked to the user. Further developments can be seen in [61, 63], which explore the integration of different types of attributes into the interactive learning framework. Note that all studies in [41, 61–63, 66] overlook the diversity in the output.

In the field of machine learning, our problem is related to the problem of *learning to rank* [26, 30, 37, 40] and *preference learning* [47]. Problem *learning to rank* learns the ranking of tuples by interacting with the user, and problem *preference learning* approximates the user criterion with the help of user interaction. However, most of the existing algorithms [26, 37, 40] do not utilize the inter-relation of tuples (where attribute "price" is an example of an inter-relation showing that if two tuples are the same in all attributes except for price, the tuple with \$200 is better than the one with \$500 since \$200 is lower) to consider the user's criterion, and thus, require more user effort during the interaction [66]. The algorithms proposed by [30, 47] consider the inter-relation between tuples. However, they either learn the total ranking of tuples or approximate the user criterion by interacting with the user, which may require asking some questions that are unnecessary to our problem. For example, if Alice prefers car p_1 to both p_2 and p_3, her criterion between p_2 and p_3 is less interesting in our problem IDT, but this additional comparison might be needed in [30, 47].

Compared with existing studies, our problem IDT has several advantages. (1) We use real tuples during the interaction, unlike the query proposed in [41] which incorporates fake tuples. (2) We require low user effort during the interaction. Firstly, contrary to existing studies that ask many questions to learn either a total ranking [26, 37, 40] or an exact user criterion [47], we only search for a set of k tuples. Secondly, unlike [26, 37, 40], we utilize the inter-relation between tuples, and thus, reduce user effort. Thirdly, it is easier to answer our designed questions and it is more effective in collecting the information of the user's criteria compared with [9, 52, 73]. (3) We put the diversity issue into consideration, addressing potential biases found in other studies [61–63, 66].

Diversity. Query result diversification [49, 59, 60, 75] is a line of study commonly used in information retrieval to provide a diverse output that covers different aspects or interpretations of the query, rather than returning multiple similar tuples [13, 18, 25, 38, 43, 50, 53, 64, 69, 70]. There are various definitions of diversity. The three main categories [14, 19, 20, 75] of diversity definitions in query result diversification are: (1) *content-based*, to include dissimilar tuples in the output; (2) *novelty-based*, to add tuples with new information that did not exist in previous output; and (3) *coverage-based*, to retrieve tuples of different categories or from different interpretations of the query [21].

One representative coverage-based query is the *group diversity query* [5, 46, 74], which is closely related to our problem IDT. It partitions tuples in the database into groups by the values of sensitive attributes. For example, the students with the same gender are in the same group. The goal is to ensure that (1) the proportion of each group in the output is identical or similar to that in the whole database (e.g., if there are 40% female students in the database, the output should include about 40% female students), or (2) the number of tuples of each group in the output should be in a given range (e.g., the number of female students in the output should be within a given range of 10-20). Our problem IDT utilizes the second type since the first one can be seen as a special case of the second type.

There are many algorithms proposed to enhance the diversity of output while maintaining relevance [14, 16, 20]. The algorithms in [14, 20] are designed in a greedy manner. In each round, they extract the tuple with the highest score. Initially, the score is based on the query only (e.g., the user's criterion). Subsequently, the score involves both the query and the diversity conditioned on tuples already selected. [16] proposes a dynamic programming-based algorithm. It proceeds by solving a sequence of sub-problems that relax the constraints of query and diversity. Some other algorithms improve diversity by modifying the query. Recent studies [35, 36, 55] aim to minimally refine the query to satisfy constraints on the size of specific groups in the output. Authors in [3, 4] also aim to satisfy cardinality constraints through minimal query refinement.

This line of study differs from our problem IDT because they return tuples that are relevant to a *pre-defined* query (e.g., the user's criterion). If the query is unknown or not explicitly defined, the proposed algorithms are not applicable. In contrast, our problem IDT can learn the query with the help of user interaction, making it adaptable to situations where the query is initially *unspecified*.

3 PROBLEM DEFINITION

We first introduce several concepts and define problem IDT in Section 3.1. Then, the characteristic of IDT is discussed in Section 3.2.

3.1 Problem IDT

Data. The input dataset $\mathcal{D}$ of our problem contains n tuples, i.e., $|\mathcal{D}| = n$. Each tuple $\boldsymbol{p}$ is described by d non-negative scoring attributes. We denote the value of $\boldsymbol{p}$ in the i-th scoring attribute by $p[i]$, where $i \in [1, d]$. Without loss of generality, following [63, 66, 67], we assume that each scoring attribute is normalized to $(0, 1]$ and a larger attribute value is preferred.

Utility Function. Following [6, 62, 66], we model the user's criterion by a linear scoring function, called the *utility function*, which is a popular and effective representation for modeling users' criteria [23, 31]. As verified in [47, 63], it can effectively capture how real users assess the multi-attribute tuples. Formally, the utility function is defined as follows.

$$f_{\boldsymbol{u}}(\boldsymbol{p}) = \boldsymbol{u} \cdot \boldsymbol{p} = \sum_{i=1}^{d} u[i]p[i]$$

Here, $\boldsymbol{u} = (u[1], u[2], ..., u[d])$ is a d-dimensional vector, called the *utility vector*. Each $u[i]$ denotes the importance of the i-th scoring attribute to the user, where $i \in [1, d]$. Without loss of generality, following [41, 62, 66], we assume that $\sum_{i=1}^{d} u[i] = 1$ and each $u[i] \geq 0$. Function score $f_{\boldsymbol{u}}(\boldsymbol{p})$ is called the *utility* of $\boldsymbol{p}$ w.r.t. $\boldsymbol{u}$. It represents to what extent a user prefers tuple $\boldsymbol{p}$, where a higher utility means that $\boldsymbol{p}$ is more preferred.

Consider Table 2. Suppose the utility function is $f_{\boldsymbol{u}}(\boldsymbol{p}) = 0.4 \cdot p[1] + 0.6 \cdot p[2]$, i.e., $\boldsymbol{u} = (0.4, 0.6)$. The utility of $\boldsymbol{p}_3$ w.r.t. $\boldsymbol{u}$ is $f_{\boldsymbol{u}}(\boldsymbol{p}_3) = 0.4 \times 0.8 + 0.6 \times 0.3 = 0.5$. Similarly, for the other tuples.

Diversity Model. We adopt a well-established diversity model in the existing literature [15, 71, 74]. Except for the scoring attributes,

each tuple $\boldsymbol{p}$ is also described by several sensitive attributes. Let us denote the value of $\boldsymbol{p}$ in the j-th sensitive attribute by $ps[j]$. Table 2 shows an example. It contains four tuples with two scoring attributes and one sensitive attribute. We define a *group* as a set of tuples in $\mathcal{D}$ that have the same value in at least one sensitive attributes. For example, $\{\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$ is a group since the tuples in it have the same value A_2.

Given a dataset $\mathcal{D}$ and c groups $\mathbf{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_c\}$, the *diversity constraint* is defined by specifying a lower bound l_j and an upper bound b_j for each group $\mathcal{G}_j$, where $j \in [1, c]$. Formally, a subset $\mathcal{S} \subseteq \mathcal{D}$ is considered diverse in terms of $\mathbf{G}$ if and only if

$$l_j \leq |\mathcal{S} \cap \mathcal{G}_j| \leq b_j$$

for each group $\mathcal{G}_j \in \mathbf{G}$. Consider an example in Table 2. Suppose that there are two groups $\mathbf{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, where the tuples in $\mathcal{G}_1$ have the same value A_1 and the tuples in $\mathcal{G}_2$ have the same value A_2. Let $l_1 = l_2 = 1$ and $b_1 = b_2 = 5$. Set $\mathcal{S} = \{\boldsymbol{p}_1, \boldsymbol{p}_3\}$ is a diverse set since $|\mathcal{S} \cap \mathcal{G}_1| = |\{\boldsymbol{p}_1\}| \geq 1$ and $|\mathcal{S} \cap \mathcal{G}_2| = |\{\boldsymbol{p}_3\}| \geq 1$.

Based on the diversity constraint, we define the *diverse top-k set* of a utility function $f_{\boldsymbol{u}}(\cdot)$ to be a set $\mathcal{S}^* \subseteq \mathcal{D}$ of k tuples that have the highest utilities and satisfy the diversity constraint, i.e.,

$$\mathcal{S}^* = \underset{\mathcal{S} \subseteq \mathcal{D}: |\mathcal{S}|=k}{\arg\max} \sum_{\boldsymbol{p} \in \mathcal{S}} f_{\boldsymbol{u}}(\boldsymbol{p}) \text{ s.t. } l_j \leq |\mathcal{S} \cap \mathcal{G}_j| \leq b_j, \forall \mathcal{G}_j \in \mathbf{G}$$

Given a utility vector $\boldsymbol{u}$, there are existing algorithms [16, 20, 71] proposed to find set $\mathcal{S}^*$. We apply the one discussed in [71] to our work. Let us denote it by $\nabla_{\cdot}(\cdot)$, i.e., $\mathcal{S}^* = \nabla_{\boldsymbol{u}}(\mathcal{D})$.

Problem Definition. Our objective is to find the user's diverse top-k set with the help of user interaction. Specifically, our interactive learning framework follows [62, 63, 66]. The system interacts with a user in rounds. In each interactive round, (1) (*Tuple Selection*) it adaptively selects two tuples and asks the user to pick the one s/he prefers; (2) (*Information Maintenance*) based on the user feedback, the information maintained for finding the user's diverse top-k set is updated; (3) (*Stopping Condition*) it checks if the stopping condition is satisfied. If yes, the interaction process stops, and the diverse top-k set found is returned to the user. Otherwise, it starts another interactive round. Formally, we are interested in the problem below.

Problem 1. *(**Interactive Learning for Diverse Top-k Set (IDT**)) Given a dataset $\mathcal{D}$, an integer k, and a diversity constraint, we want to interact with a user in as few rounds as possible to find the user's diverse top-k set.*

The following theorem provides a lower bound on the number of interactive rounds needed to find the user's diverse top-k set. Due to the lack of space, the proofs of some theorems/lemmas can be found in Appendix C.

Theorem 3.1. *There exists a dataset $\mathcal{D}$ of size n such that any algorithm needs to interact with a user in $\Omega(\log_2 n)$ rounds to find the user's diverse top-k set.*

Proof. Consider a dataset $\mathcal{D}$, where (1) $\forall j \in [2, c], |\mathcal{G}_j \cap \mathcal{D}| = l_j$ and (2) each tuple in $\mathcal{G}_1$ can have the highest utility w.r.t. at least a utility function. In this way, there are at least $n - k$ different diverse top-k sets $\mathcal{S}$. The l_j tuples from group $\mathcal{G}_j$ in different $\mathcal{S}$ are the same, where $j \in [2, c]$, while the tuples from group $\mathcal{G}_1$ are different.

Any algorithm that aims to identify the user's diverse top-k set must conduct in the form of a binary tree. Each internal node in the tree corresponds to an interactive round, and each leaf node maps to a diverse top-k set. Since there are $\Omega(n - k)$ diverse top-k sets, there would be $\Omega(n-k)$ leaf nodes. The height of the tree should be $\Omega(\log_2(n - k))$. Since $k \ll n$, any algorithm needs to interact with a user in $\Omega(\log_2 n)$ rounds to find the user's diverse top-k set. □

3.2 Problem Characteristics

In this section, we formalize our problem IDT from a geometric perspective. With a slight abuse of notations, we use p to represent the scoring-attribute part of tuple $\boldsymbol{p}$, i.e., $p = (p[1], p[2], ..., p[d])$. In a d-dimensional geometric space $\mathbb{R}^d$, p can be regarded as a point. Each dimension corresponds to a scoring attribute. Consider any pair of tuples $\boldsymbol{p}_i, \boldsymbol{p}_j \in D$, based on the scoring attributes, we can build a hyper-plane as shown in Figure 1.

$$h_{i,j} = \{\boldsymbol{r} \in \mathbb{R}^d_+ \mid \boldsymbol{r} \cdot (p_i - p_j) = 0\}$$

Hyper-plane $h_{i,j}$ passes through the origin with its unit normal in the same direction as vector $p_i - p_j$. It divides space $\mathbb{R}^d$ into two halves: *positive half-space* $h^+_{i,j}$ and *negative half-space* $h^-_{i,j}$ [17]. The positive half-space (resp. negative half-space) contains all utility vectors $\boldsymbol{u} \in \mathbb{R}^d$ such that $\boldsymbol{u} \cdot (p_i - p_j) > 0$ (resp. $\boldsymbol{u} \cdot (p_i - p_j) < 0$).

A *polyhedron* $\mathcal{P}$ is defined to be the intersection of a finite number of half-spaces [17]. In space $\mathbb{R}^d$, each utility vector can be seen as a point. Recall that we assume that (1) $u[i] \geq 0$ for each dimension and (2) $\sum_{i=1}^{d} u[i] = 1$. The domain of the utility vector, called the *utility space* and denoted by $\mathcal{U}$, is a polyhedron in space $\mathbb{R}^d$, e.g., a triangle when $d = 3$ as shown in Figure 1 or a line segment when $d = 2$ as shown in Figure 2. If a hyper-plane $h_{i,j}$ intersects with the utility space $\mathcal{U}$, we denote the intersection by $\wedge_{i,j}$. The lemma below establishes our foundation for learning the user's utility function, more specifically, the user's utility vector.

Lemma 3.2. *Given $\mathcal{U}$ and two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, if a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, the user's utility vector must be in $h^+_{i,j} \cap \mathcal{U}$.*

Proof. Denote the user's utility vector by $\boldsymbol{u}$. If a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, we have $f_{\boldsymbol{u}}(\boldsymbol{p}_i) > f_{\boldsymbol{u}}(\boldsymbol{p}_j)$, i.e., $\boldsymbol{u} \cdot (p_i - p_j) > 0$. This implies that $\boldsymbol{u} \in h^+_{i,j}$ (or $\boldsymbol{u} \in h^-_{j,i}$), and thus, $\boldsymbol{u}$ must be in $h^+_{i,j} \cap \mathcal{U}$. □

Corollary 3.3. *Given $\mathcal{U}$ and two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, if a user's utility vector is in $h^+_{i,j} \cap \mathcal{U}$, the user must prefer $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$.*

Proof. Denote the user's utility vector by $\boldsymbol{u}$. If $\boldsymbol{u}$ is in $h^+_{i,j} \cap \mathcal{U}$, we have $\boldsymbol{u} \cdot (p_i - p_j) > 0$, i.e., $f_{\boldsymbol{u}}(\boldsymbol{p}_i) > f_{\boldsymbol{u}}(\boldsymbol{p}_j)$, This implies that the user must prefer $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$. □

Based on Lemma 3.2, our idea is to interact with a user to narrow down the utility space. When the narrowed utility space is sufficiently small, the diverse top-k set can be determined.

4 SPECIAL CASE OF IDT

We consider a special case of problem IDT, where each tuple $\boldsymbol{p}$ is described by two scoring attributes. In this case, the utility vector $\boldsymbol{u} = (u[1], u[2])$ is a two-dimensional vector, and the utility space $\mathcal{U}$ is a line segment in space $\mathbb{R}^2$ based on our assumption (i.e., $u[1], u[2] \geq 0$ and $u[1] + u[2] = 1$). We propose a two-dimensional

Table 2: Dataset ($\boldsymbol{u} = (0.4, 0.6)$)

$\boldsymbol{p}$	$p[1]$	$p[2]$	$ps[1]$	$f_{\boldsymbol{u}}(\boldsymbol{p})$
$\boldsymbol{p}_1$	0.0	1.0	A_1	0.60
$\boldsymbol{p}_2$	0.7	0.8	A_2	0.76
$\boldsymbol{p}_3$	0.8	0.3	A_2	0.50
$\boldsymbol{p}_4$	1.0	0	A_2	0.40

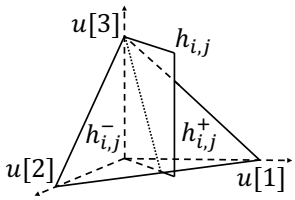


Figure 1: Hyper-plane

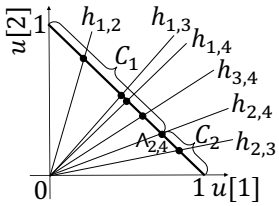


Figure 2: Algorithm TDIA

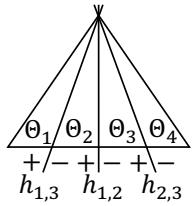


Figure 3: Partitions in $\mathbb{R}^d$

interactive algorithm *TDIA* that finds the user's diverse top-k set within an asymptotically optimal number of interactive rounds.

Algorithm *TDIA* consists of two phases: *exploration* and *interaction*. In the exploration phase, it scans all utility vectors $\boldsymbol{u}$ along the utility space $\mathcal{U}$ from one side to another, and records the diverse top-k sets w.r.t. $\boldsymbol{u}$ in a list $\mathbf{C}$ based on the scanning sequence. Note that if adjacent utility vectors correspond to the same diverse top-k set, we only record that set once in $\mathbf{C}$. Subsequently, in the interaction phase, it interacts with a user to identify which one in $\mathbf{C}$ is the user's diverse top-k set.

Exploration. Since there are infinite utility vectors in utility space $\mathcal{U}$, when scanning utility vectors, it is impractical to check the diverse top-k set w.r.t. each utility vector one by one. The following lemma provides a theoretical foundation that makes it feasible to consider several utility vectors together.

Lemma 4.1. *Given a line segment $[\boldsymbol{u}_1, \boldsymbol{u}_2]$, where $\boldsymbol{u}_1, \boldsymbol{u}_2 \in \mathcal{U}$, if $\nexists \boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$ such that hyper-plane $h_{i,j}$ intersects with $[\boldsymbol{u}_1, \boldsymbol{u}_2]$, the diverse top-k sets w.r.t. any utility vectors in $[\boldsymbol{u}_1, \boldsymbol{u}_2]$ are the same.*

Proof. Consider any hyper-plane $h_{i,j}$ of two tuples $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$. If it does not intersect with line segment $[\boldsymbol{u}_1, \boldsymbol{u}_2]$, based on the definition of hyper-plane, we know $\boldsymbol{u}\cdot(p_i - p_j) > 0$ or $\boldsymbol{u}\cdot(p_i - p_j) < 0$ for any utility vectors $\boldsymbol{u} \in [\boldsymbol{u}_1, \boldsymbol{u}_2]$. As a result, the rankings of tuples w.r.t. any utility vectors $\boldsymbol{u} \in [\boldsymbol{u}_1, \boldsymbol{u}_2]$ are the same, and thus, the diverse top-k sets w.r.t. any $\boldsymbol{u} \in [\boldsymbol{u}_1, \boldsymbol{u}_2]$ are the same. □

Following Lemma 4.1, during the scanning process, we can maintain the diverse top-k set w.r.t. the currently scanned utility vector. The set maintained may change only if the scanned utility vector $\boldsymbol{u}$ is the intersection of a hyper-plane $h_{i,j}$ and the utility space, i.e., $\boldsymbol{u} = \wedge_{i,j}$, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$.

The exploration phase works as follows. Initially, we rank all the intersections $\wedge_{i,j}$, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$, along the utility space from one endpoint $(0, 1)$ to the other $(1, 0)$. The scan begins at utility vector $(0, 1)$. We find the diverse top-k set w.r.t. this utility vector by $\nabla_{(0,1)}(\mathcal{D})$ and insert it into $\mathbf{C}$. Note that the latest set inserted into $\mathbf{C}$ also denotes the diverse top-k set w.r.t. the currently scanned utility vector. Then, we scan the ranked intersections sequentially. Suppose that we reach an intersection $\wedge_{i,j}$ and line segment $[\wedge_{i,j}, (1, 0)] \subseteq h^-_{i,j}$ (similarly for $[\wedge_{i,j}, (1, 0)] \subseteq h^-_{j,i}$). Since $[\wedge_{i,j}, (1, 0)] \subseteq h^-_{i,j}$, we have $f_{\boldsymbol{u}}(\boldsymbol{p}_i) < f_{\boldsymbol{u}}(\boldsymbol{p}_j)$ w.r.t. any $\boldsymbol{u} \in [\wedge_{i,j}, (1, 0)]$. Let C be the latest diverse top-k set inserted into $\mathbf{C}$. If $\boldsymbol{p}_i \in C$ and $\boldsymbol{p}_j \notin C$, we check whether set $C' = C \cup \{\boldsymbol{p}_j\} \setminus \{\boldsymbol{p}_i\}$ satisfies the diversity constraint. If it satisfies, set C' is a new diverse top-k set. We append C' to $\mathbf{C}$. The scan terminates when all the intersections have been scanned.

Consider Table 2 and Figure 2. Assume that $\mathbf{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, where the tuples in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) have the same value A_1 (resp. A_2). Let $k = 2, l_1 = l_2 = 1$, and $b_1 = b_2 = 5$. Initially, we find the diverse top-k set $C_1 = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$ w.r.t. utility vector $(0, 1)$ and insert it into $\mathbf{C}$. Then, we reach the first intersection $\wedge_{1,2}$. Since tuples $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are in set C_1, we directly move to the second intersection $\wedge_{1,3}$. Here, given that $\boldsymbol{p}_1 \in C_1$ and $\boldsymbol{p}_3 \notin C_1$, we build set $C' = \{\boldsymbol{p}_2, \boldsymbol{p}_3\}$. However, $|C' \cap \mathcal{G}_1| < b_1$ makes C' unable to meet the diversity constraint. We move to the next intersection $\wedge_{1,4}$. The scan continues until intersection $\wedge_{2,3}$ is scanned. The final result is $\mathbf{C} =< C_1, C_2 >$, where $C_1 = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$ and $C_2 = \{\boldsymbol{p}_1, \boldsymbol{p}_4\}$.

Theorem 4.2. *The exploration phase runs in $O(n^2 \log n + Y_1 + Y_2 n^2)$ time, where Y_1 is the time complexity of $\nabla_{\cdot}(\cdot)$ [71] and Y_2 is the time complexity of checking if a set satisfies the diversity constraint [5].*

Proof. Since $|\mathcal{D}| = n$, there are $O(n^2)$ hyper-planes, and thus, there are $O(n^2)$ intersections. We need $O(n^2 \log n)$ time to rank all the intersections. For the scanning process, we need Y_1 time to find the diverse top-k set of utility vector $(0, 1)$ and Y_2 time to check the satisfaction of diversity constraint for each intersection. Thus, the total time complexity is $O(n^2 \log n + Y_1 + Y_2 n^2)$. □

Interaction. Let $\mathbf{C} =< C_1, C_2, C_3, \ldots >$ be all diverse top-k sets found in the exploration phase. They are listed based on the scanning sequence. The interaction phase determines which one is the user's diverse top-k set in $\mathbf{C}$ by *a binary search*. In each interactive round, (1) (*Tuple Selection*) it finds the median sets $C_x, C_{x+1} \in \mathbf{C}$ and presents a user with two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, where $\boldsymbol{p}_i$ is in C_x but not in C_{x+1}, and $\boldsymbol{p}_j$ is in C_{x+1} but not in C_x. (2) (*Information Maintenance*) If the user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, based on Lemma 4.3, half of the sets $< C_{x+1}, C_{x+2}, C_{x+3}, \ldots >$ in $\mathbf{C}$ can be deleted. Otherwise, the other half can be deleted from $\mathbf{C}$. (3) (*Stopping Condition*) The interaction process stops when $|\mathbf{C}| = 1$ and the diverse top-k set finally left in $\mathbf{C}$ is returned as the output.

Lemma 4.3. *If a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, the user's diverse top- k set cannot be the one among $< C_{x+1}, C_{x+2}, C_{x+3}, \ldots >$.*

Proof. Based on our construction of $\mathbf{C}$, each set in $< C_{x+1}, C_{x+2}, C_{x+3}, \ldots >$ is a diverse top-k set w.r.t. at least a utility vector in $h^-_{i,j} \cap \mathcal{U}$. Following Lemma 3.2, if a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, the user's utility vector must be in $h^+_{i,j} \cap \mathcal{U}$. Thus, none of the set in $< C_{x+1}, C_{x+2}, C_{x+3}, \ldots >$ is the diverse top-k set w.r.t. the user's utility vector. □

In Figure 2, initially, $\mathbf{C} =< \{\boldsymbol{p}_1, \boldsymbol{p}_2\}, \{\boldsymbol{p}_1, \boldsymbol{p}_4\} >$. We present a user with tuples $\boldsymbol{p}_2$ and $\boldsymbol{p}_4$. Assume that the user prefers $\boldsymbol{p}_2$ to $\boldsymbol{p}_4$. $\mathbf{C}$ is updated to be $< \{\boldsymbol{p}_1, \boldsymbol{p}_2\} >$. Since there is only one set left in $\mathbf{C}$, we stop the interaction process and return set $\{\boldsymbol{p}_1, \boldsymbol{p}_2\}$.

Summary and Analysis. The pseudocode of our algorithm *TDIA* is shown in Algorithm 1. Initially, all intersections are ranked (line 3). Based on these ranked intersections, we scan the utility space to obtain all diverse top-k sets $\mathbf{C} = \{C_1, C_2, \ldots\}$ (lines 4-8). Then, we

Algorithm 1: Algorithm *TDIA*

1 **Input:** Dataset $\mathcal{D}$, set **G**, parameters k, l_1, l_2, ..., b_1, b_2, ...
2 **Output:** The user's diverse top-k set
3 Initialize $\mathbf{C} = \emptyset$ and rank all the intersections;
4 Insert the diverse top-k set w.r.t. utility vector $(0, 1)$ into **C**;
5 **foreach** *intersection* $\wedge_{i,j}$ **do**
6 Build set C' based on set C, $\boldsymbol{p}_i$, and $\boldsymbol{p}_j$;
7 **if** *set* C' *satisfies the diversity constraint* **then**
8 Append C' to list **C**;
9 **while** $|\mathbf{C}| > 1$ **do**
10 Find the median sets $C_x, C_{x+1} \in \mathbf{C}$;
11 Display $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ to the user;
12 **if** *the user prefers* $\boldsymbol{p}_i$ *to* $\boldsymbol{p}_j$ **then**
13 $\mathbf{C} \leftarrow < ..., C_{x-2}, C_{x-1}, C_x >$;
14 **else**
15 $\mathbf{C} \leftarrow < C_{x+1}, C_{x+2}, C_{x+3} ... >$;
16 **return** The set finally left in **C**

interact with a user to delete half of the sets in **C** in each interactive round (lines 10-15). When there is only one set left in **C** (line 9), we stop and return this set as the output (line 16).

Theorem 4.4. *Algorithm* TDIA *solves the special case of problem IDT by interacting with a user within* $O(\log_2 n)$ *rounds.*

Proof. As discussed, the utility space is a line segment when $d = 2$. With $|\mathcal{D}| = n$, there are $O(n^2)$ hyper-planes, and thus, $O(n^2)$ intersections. In the exploration phase, since a new diverse top-k set may appear only if the scan reaches an intersection, there could be $O(n^2)$ diverse top-k sets in **C**. In the interaction phase, half of the sets in **C** can be deleted in each interactive round. Therefore, after $O(\log_2 n)$ rounds, there is only one set left in **C**. □

Corollary 4.5. *Algorithm* TDIA *is asymptotically optimal in terms of the interactive rounds for the special case of problem IDT.*

5 GENERAL CASE OF IDT

We consider the general case of problem IDT, where each tuple is described by d scoring attributes ($d \geq 2$). In this case, the utility vector $\boldsymbol{u}$ is a high-dimensional vector, and the utility space $\mathcal{U}$ is a polyhedron in a high-dimensional space $\mathbb{R}^d$, e.g., a triangle when $d = 3$ as shown in Figure 1. We propose a high-dimensional interactive algorithm *HDIA* that finds the user's diverse top-k set within an asymptotically optimal number of interactive rounds in expectation.

5.1 Algorithm HDIA

At a high level, our algorithm *HDIA* follows and extends the framework of algorithm *TDIA*. It maintains a polyhedron $\mathcal{R} \subseteq \mathcal{U}$, called *utility range*, which contains the user's utility vector. Initially, $\mathcal{R}$ is set to be the entire utility space, i.e., $\mathcal{R} = \{\boldsymbol{u} \in \mathbb{R}^d_+ \mid \sum_{i=1}^{d} u[i] = 1\}$. Algorithm *HDIA* repeats two phases in cycles: *exploration* and *interaction*. In the exploration phase, it divides utility range $\mathcal{R}$ into several smaller polyhedrons, termed partitions. Then, in the interaction phase, it interacts with the user to delete the partitions that cannot contain the user's utility vector. Utility range $\mathcal{R}$ maintains the partitions left. If there is only one partition left in $\mathcal{R}$, another cycle proceeds. Algorithm *HDIA* terminates if all utility vectors in $\mathcal{R}$ correspond to the same diverse top-k set C.

There are several challenges. The first challenge lies in how to divide utility range $\mathcal{R}$ into partitions; the second challenge involves how to strategically select tuples for user interaction; and the third challenge concerns how to determine whether all utility vectors in $\mathcal{R}$ correspond to the same diverse top-k set. In the following, we address these challenges respectively.

Challenge 1. We randomly sequence all hyper-planes $h_{i,j}$, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$, and separate them into batches. Each batch contains γ hyper-planes. In each cycle, a single batch of hyper-planes is utilized to divide utility range $\mathcal{R}$ into partitions. Each partition, denoted by Θ, is an intersection of $O(\gamma)$ positive or negative half-spaces. For example, as shown in Figure 3, suppose that the current utility range $\mathcal{R}$ is the whole utility space and there are three hyper-planes in the current batch. Utility range $\mathcal{R}$ is divided by these three hyper-planes into four partitions. Partition Θ_4 is the intersection of three negative half-spaces, i.e., $\Theta_4 = h^-_{1,2} \cap h^-_{1,3} \cap h^-_{2,3}$. Similarly for the other partitions. For the batch size γ, a small one might necessitate more cycles, prolonging the whole process, while a large one could lead to a large number of partitions, causing a high computational cost. We will discuss the setting of γ in Section 6.

Our strategy of dividing the utility range results in simple relationships between partitions and hyper-planes. For a partition Θ and a hyper-plane h, there are only two simple relationships between them: $\Theta \subseteq h^+$ or $\Theta \subseteq h^-$. To learn the relationship, it suffices to check the relationship of a utility vector $\boldsymbol{u} \in \Theta$ and hyper-plane h. If $\boldsymbol{u} \in h^+$ (resp. $\boldsymbol{u} \in h^-$), then $\Theta \subseteq h^+$ (resp. $\Theta \subseteq h^-$).

Challenge 2. In the exploration phase, utility range $\mathcal{R}$ is divided into partitions by the hyper-planes in the current batch. Then, in the interaction phase, we expect to delete partitions until there is only one left with the minimal number of interactive rounds. Let us formalize the interaction phase by a binary tree, called *decision tree* or *D-Tree* for short. In the D-Tree,

- Each internal node N contains i) two tuples $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$, ii) a partition set $\boldsymbol{\Theta}(N)$, and iii) two children N_1 and N_2. If N is the root node, $\boldsymbol{\Theta}(N)$ contains all the partitions divided by the hyper-planes in the current batch. The hyper-plane $h_{i,j}$ of $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ decides the partition sets in children N_1 and N_2. The partition set $\boldsymbol{\Theta}(N_1)$ (resp. $\boldsymbol{\Theta}(N_2)$) in child N_1 (resp. N_2) contains all the partitions in $\boldsymbol{\Theta}(N)$ that are in positive half-space $h^+_{i,j}$ (resp. negative half-space $h^-_{i,j}$), i.e., $\boldsymbol{\Theta}(N_1) = \{\Theta \in \boldsymbol{\Theta}(N) | \Theta \subseteq h^+_{i,j}\}$ (resp. $\boldsymbol{\Theta}(N_2) = \{\Theta \in \boldsymbol{\Theta}(N) | \Theta \subseteq h^-_{i,j}\}$).
- Each leaf N contains a partition set $\boldsymbol{\Theta}(N)$, where $|\boldsymbol{\Theta}(N)| = 1$.

Figure 4 shows a D-Tree that is based on the partitions and hyper-planes in Figure 3 (set $\mathcal{H}(N)$ in the internal node is used for construction only, which will be introduced later). For the root node N, it contains two tuples $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ and its partition set $\boldsymbol{\Theta}(N)$ includes all the partitions $\Theta_1, \Theta_2, \Theta_3$, and Θ_4. Based on hyper-plane $h_{1,2}$, the partitions can be separated into two sets $\Theta_1, \Theta_2 \subseteq h^+_{1,2}$ and $\Theta_3, \Theta_4 \subseteq h^-_{1,2}$. Thus, for child N_1, $\boldsymbol{\Theta}(N_1) = \{\Theta_1, \Theta_2\}$; for child N_2, $\boldsymbol{\Theta}(N_2) = \{\Theta_3, \Theta_4\}$. Similarly for the other nodes.

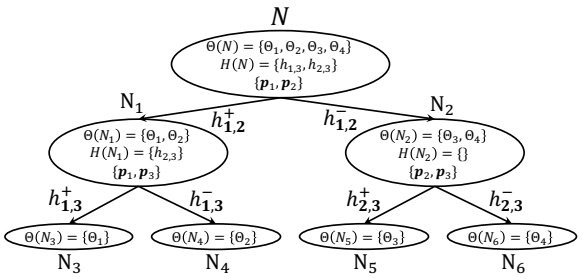


Figure 4: D-Tree

The interaction phase can proceed with the D-Tree in a top-down manner by starting from the root node. In each interactive round, we interact with the user by using the two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ that are contained in the current node N. Following the user feedback, based on Lemma 3.2, we learn that the user's utility vector is in one of the half-spaces, i.e., either $h^+_{i,j}$ or $h^-_{i,j}$. Then, we move to the child of N, where the partitions contained in the child are in the inferred half-space. For example, in the first interactive round (i.e., at the root node), if the user prefers $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$, the user's utility vector must be in $h^+_{1,2}$, and thus, we move to node N_1. The interaction process stops when we reach a leaf. The partition contained in the leaf must include the user's utility vector.

It is easy to see that the number of interactive rounds depends on the height of the D-Tree. Thus, to achieve the best tuple selection strategy, i.e., the minimal number of interactive rounds, we need to ensure that the height of the D-Tree is the smallest.

We employ the recursive manner to construct the shortest D-Tree. Specifically, for each node N, we add two data structures for construction: a number L_N and a hyper-plane set $\mathcal{H}(N)$. L_N denotes the length of the longest path from a node N to any of its reachable leaves. $\mathcal{H}(N)$ contains the hyper-planes that can be used to separate the partitions in $\boldsymbol{\Theta}(N)$.

The construction starts at the root node and gradually builds nodes downwards. Initially, the root node contains all hyper-planes in the current batch and all partitions divided by these hyper-planes. Suppose that the construction process reaches a node N. If $|\boldsymbol{\Theta}(N)| = 1$, L_N is set to 0. Otherwise, we derive L_N based on the hyper-planes in $\mathcal{H}(N)$. Specifically, for each hyper-plane $h \in \mathcal{H}(N)$, we build two children N_1 and N_2 such that (1) the partition set $\boldsymbol{\Theta}(N_1)$ and $\boldsymbol{\Theta}(N_2)$ contain the partitions $\Theta \in \boldsymbol{\Theta}(N)$ in h^+ and h^-, respectively, and (2) $\mathcal{H}(N_1) = \mathcal{H}(N_2) = \mathcal{H}(N) \setminus \{h\}$. Then, we set

$$L_N = \min_{h \in \mathcal{H}(N)} \max\{L_{N_1}, L_{N_2}\} + 1.$$

Intuitively, $\max\{L_{N_1}, L_{N_2}\}$ represents the length of the longest path (from node N to any of its reachable leaves) if we choose a hyper-plane h to separate the partitions in node N. $\min_{h \in \mathcal{H}(N)}$ means that we want to find the hyper-plane in $\mathcal{H}(N)$ so that the longest path can be the shortest. Node N only remains the two children which lead to the smallest L_N.

Building D-Trees recursively can be time-consuming. To accelerate, we propose several strategies in the following. Our first focus is to reduce the construction of the D-Tree. To illustrate, consider Figure 4. Let us only build nodes N, N_1, and N_2 of the D-Tree. Utilizing the incomplete D-Tree, we can interact with the user. Without loss of generality, suppose that we move to a node, say N_1, based on the user feedback. Then, we can use node N_1 as a root to construct a new D-Tree, which only includes nodes N_3 and N_4 (without N_5 and N_6). Based on this idea, we do not wait for the complete D-Tree before starting the interaction. Let us relax the second requirement in the definition of the D-Tree.

- Each leaf N contains a partition set $\boldsymbol{\Theta}(N)$, where $|\boldsymbol{\Theta}(N)|$ is smaller than or equal to a given threshold (instead of 1).

Our strategy is to construct several small D-Trees in iterations, each of which is part of the complete D-Tree. Every time a small D-Tree is constructed, we interact with the user based on it. Specifically, in each iteration, assume that the root node has m partitions. We construct a D-Tree by allowing each leaf to contain at most m/β partitions (instead of one partition). The setting of β is discussed in Section 6. Then, we follow the D-Tree to interact with the user, leading to a leaf node N. If $\boldsymbol{\Theta}(N) = 1$, we stop. Otherwise, we proceed another iteration by using node N as a root to build a new D-Tree and interacting with the user.

Our second focus is to reduce the number of hyper-planes in $\mathcal{H}(N)$ that are considered to derive L_N for each node N. We limit our consideration to a few hyper-planes that divide the partitions in $\boldsymbol{\Theta}(N)$ the most evenly. The idea behind this is that if each half-space of the hyper-plane contains half of the partitions in $\boldsymbol{\Theta}(N)$, the number of partitions in each child can be reduced by half, potentially leading to a significant reduction in the height of the D-tree. To implement, we define M_h to be $\min\{M(h^+), M(h^-)\}$, where $M(h^+)$ (resp. $M(h^-)$) represents the number of partitions in half-space h^+ (resp. h^-). Then, we only consider α hyper-planes with the largest M_h. The setting of α will be discussed in Section 6.

We also propose a lower bound for L_N. If we find a hyper-plane which can achieve the lower bound, we do not need to check the remaining hyper-planes.

Lemma 5.1. *For any node N, we have $L_N \geq \lceil \log_2 \frac{\beta|\boldsymbol{\Theta}(N)|}{m} \rceil$, where m is the number of partitions in the root node and β is a parameter. m/β indicates the number of partitions allowed in leaves.*

Proof. Let us build an optimal sub-tree in which node N is considered the root. For each internal node N' in the optimal sub-tree, there is a hyper-plane that separates the partitions in $|\boldsymbol{\Theta}(N')|$ into two equal sets of partitions, i.e., the numbers of partitions in the positive and negative half-spaces are the same. In this case, the height of the optimal sub-tree is $\lceil \log_2 \frac{\beta|\boldsymbol{\Theta}(N)|}{m} \rceil$. □

Challenge 3. We verify if the diverse top-k sets w.r.t. any utility vectors in $\mathcal{R}$ are the same based on the *extreme utility vectors* of $\mathcal{R}$. The extreme utility vectors are the corner points of a polyhedron. For example, in Figure 2, the extreme utility vectors of the utility space are $(0, 1)$ and $(1, 0)$. The diverse top-k set w.r.t. each extreme utility vector e of $\mathcal{R}$ can be obtained by $\nabla_e(\mathcal{D})$.

Lemma 5.2. *If the diverse top-k sets w.r.t. any extreme utility vectors of utility range $\mathcal{R}$ are the same, all utility vectors in $\mathcal{R}$ correspond to the same diverse top-k set.*

Proof. Suppose that all the extreme utility vectors of $\mathcal{R}$ correspond to the same diverse top-k set C, and there is a utility vector in $\mathcal{R}$ corresponding to a diverse top-k set C' such that $C \neq C'$. There must be two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ whose ranks change, where $\boldsymbol{p}_i$

in C but not in C' and $\boldsymbol{p}_j$ in C' but not in C. Thus, hyper-plane $h_{i,j}$ intersects $\mathcal{R}$. Since utility range $\mathcal{R}$ is a polyhedron, it is convex. The extreme utility vectors of $\mathcal{R}$ must be separated by $h_{i,j}$, i.e., there will be extreme utility vectors $e_1 \in h^+_{i,j}$ and $e_2 \in h^-_{i,j}$. For $e_2 \in h^-_{i,j}$, $\boldsymbol{p}_j$ must rank higher than $\boldsymbol{p}_i$, which is a contradiction since C is the diverse top-k set w.r.t. e_2 ($\boldsymbol{p}_i$ ranks higher than $\boldsymbol{p}_j$). □

Summary and Analysis. We summarize our algorithm *HDIA* by combining the strategies presented. The pseudocode is shown in Algorithm 2. In the beginning, utility range $\mathcal{R}$ is set to be the entire utility space. All hyper-planes are sequenced randomly and separated into batches (lines 3-4). For each batch H_i, where $i \geq 1$, we conduct two phases: exploration and interaction. In the exploration phase, the hyper-planes in H_i divide utility range $\mathcal{R}$ into partitions $\boldsymbol{\Theta}_i = \{\Theta_1, \Theta_2, \Theta_3, ...\}$ (line 6). Then, in the interaction phase, we build D-Trees and interact with the user.

For the first D-Tree, we initialize the root node to contain all hyper-planes in H_i and all partitions in $\boldsymbol{\Theta}_i$ (lines 7-8). Assume that the construction process reaches a node N. If $|\boldsymbol{\Theta}(N)| \leq \beta^*$ (β^* represents the number of partitions allowed in the leaf), we set N to be a leaf and $L_N = 0$ (lines 20-21). Otherwise, we select α hyper-planes in $\mathcal{H}(N)$ (line 22). Based on the selected hyper-planes, we build children for N and derive L_N (lines 23-37). Note that we can skip the checking of some hyper-planes based on Lemma 5.1 (lines 34-35). After constructing the D-Tree, we interact with the user by conducting a top-down traverse on the D-Tree. Suppose that we are at a node N'. (1) If it is a leaf and $\boldsymbol{\Theta}(N') = 1$, we turn to the next batch of hyper-planes. (2) If it is a leaf and $\boldsymbol{\Theta}(N') > 1$, we turn to build a new D-Tree by using node N' as the root node (line 8). (3) If N' is an internal node, we use the two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ contained in N' to interact with the user (*Tuple Selection*). When obtaining the user feedback, following Lemma 3.2, we build a hyper-plane $h_{i,j}$ and update $\mathcal{R}$ (either $\mathcal{R} \leftarrow \mathcal{R} \cap h^+$ or $\mathcal{R} \leftarrow \mathcal{R} \cap h^-$). In this way, $\mathcal{R}$ becomes smaller (*Information Maintenance*). We move to the child N'' of N', where the partitions contained in $\boldsymbol{\Theta}(N')$ are in $\mathcal{R}$ (lines 11-16). If all the extreme utility vectors in $\mathcal{R}$ correspond to the same diverse top-k set, the algorithm stops and the diverse top-k set is returned (lines 17-18) (*Stopping Condition*). The theoretical analysis is presented in Theorem 5.3.

Take Figures 3 and 4 to illustrate. Assume that there is only one batch of hyper-planes $\mathcal{H} = \{h_{1,2}, h_{1,3}, h_{2,3}\}$. In the exploration phase, utility range $\mathcal{R}$ is divided into four partitions $\boldsymbol{\Theta} = \{\Theta_1, \Theta_2, \Theta_3, \Theta_4\}$. Then, in the interaction phase, we set the root node N of the D-Tree by $\boldsymbol{\Theta}(N) = \{\Theta_1, \Theta_2, \Theta_3, \Theta_4\}$ and $\mathcal{H}(N) = \{h_{1,2}, h_{1,3}, h_{2,3}\}$. Assume that the number of partitions allowed in the D-Tree is 2. Consider hyper-plane $h_{1,2}$. It can separate the partitions in $\boldsymbol{\Theta}(N)$ into two sets. We build two children N_1 and N_2 for N, where $\boldsymbol{\Theta}(N_1) = \{\Theta_1, \Theta_2\}$, $\boldsymbol{\Theta}(N_2) = \{\Theta_3, \Theta_4\}$, and $\mathcal{H}(N_1) = \mathcal{H}(N_2) = \{h_{1,3}, h_{2,3}\}$. Since $|\boldsymbol{\Theta}(N_1)| \leq 2$ and $|\boldsymbol{\Theta}(N_2)| \leq 2$, nodes N_1 and N_2 are set to be leaves. We have $L_{N_1} = L_{N_2} = 0$. Consequently, $L_{N_1} = 1$. Since $|\boldsymbol{\Theta}(N)| = 4$, L_N achieves lower bound $\lceil \log_2 |\boldsymbol{\Theta}(N)|/2 \rceil$. We do not need to consider the other hyper-planes to derive L_N for N. We stop the D-Tree construction and then interact with the user based on the D-Tree. The root node contains tuples $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. We present them to the user as a question. Suppose that the user prefers $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$, we move to node N_1 and update $\mathcal{R}$ to be $\mathcal{R} \cap h^+_{1,2}$. Since N_1 is a leaf and $|\boldsymbol{\Theta}(N_1)| > 1$, we use node N_1 as a root to build a new D-Tree and interact with the user continually. The algorithm stops when all the extreme utility vectors in $\mathcal{R}$ correspond to the same diverse top-k set.

Algorithm 2: Algorithm *HDIA*

1 **Input:** Dataset $\mathcal{D}$, set $\mathbf{G}$, parameters k, l_1, l_2, ..., b_1, b_2, ...
2 **Output:** The user's diverse top-k set
3 Randomly sequence all hyper-planes $h_{i,j}$, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$;
4 Separate the hyper-planes into batches; $\mathcal{R} \leftarrow \mathcal{U}$;
5 **foreach** *batch* H_i **do**
6 Divide $\mathcal{R}$ into partitions $\boldsymbol{\Theta}_i = \{\Theta_1, \Theta_2, ...\}$ based on H_i;
7 Set root N by $L_N \leftarrow \infty$, $\mathcal{H}(N) \leftarrow H_i$, $\boldsymbol{\Theta}(N) \leftarrow \boldsymbol{\Theta}_i$;
8 **while** $\boldsymbol{\Theta}(N) > 1$ **do**
9 **SearchLN**(N, $|\boldsymbol{\Theta}(N)|/\beta$);
10 $N' \leftarrow$ the root of the D-Tree;
11 **while** N' *is not a leaf* **do**
12 Use $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ in N' to interact with a user;
13 **if** *the user prefers* $\boldsymbol{p}_i$ *to* $\boldsymbol{p}_j$ **then**
14 $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{i,j}$; $N' \leftarrow N_1$;
15 **else**
16 $\mathcal{R} \leftarrow \mathcal{R} \cap h^-_{i,j}$; $N' \leftarrow N_2$;
17 **if** *the stopping condition is satisfied* **then**
18 **return** The diverse top-k set;
19 $N \leftarrow N'$;

SearchLN(node N, β^*)
20 **if** $|\boldsymbol{\Theta}(N)| \leq \beta^*$ **then**
21 $L_N \leftarrow 0$; **return**;
22 Select α hyper-planes $h \in \mathcal{H}(N)$ with the largest M_h.
23 **foreach** *selected hyper-plane* $h_{i,j}$ **do**
24 **if** $h_{i,j}$ *separates the partitions in* $\boldsymbol{\Theta}$ *into two sets* **then**
25 Build two children N_1 and N_2 for N;
26 $\boldsymbol{\Theta}(N_1) \leftarrow \{\Theta \in \boldsymbol{\Theta}(N) \mid \Theta \subseteq h^+_{i,j}\}$; $L_{N_1} \leftarrow \infty$;
27 $\boldsymbol{\Theta}(N_2) \leftarrow \{\Theta \in \boldsymbol{\Theta}(N) \mid \Theta \subseteq h^-_{i,j}\}$; $L_{N_2} \leftarrow \infty$;
28 $\mathcal{H}(N_1) \leftarrow \mathcal{H}(N) \setminus \{h_{i,j}\}$
29 $\mathcal{H}(N_2) \leftarrow \mathcal{H}(N) \setminus \{h_{i,j}\}$;
30 **SearchLN(**N_1**,** β^***)**; **SearchLN(**N_2**,** β^***)**;
31 **if** $L_N > \max\{L_{N_1}, L_{N_2}\} + 1$ **then**
32 $L_N = \max\{L_{N_1}, L_{N_2}\} + 1$;
33 Remove the children except N_1 and N_2;
34 **if** $L_N \leq \lceil \log_2 |\boldsymbol{\Theta}(N)|/\beta^* \rceil$ **then**
35 break;
36 **else**
37 Remove the two children N_1 and N_2;

Theorem 5.3. *Algorithm* HDIA *solves the general case of problem IDT by interacting with a user within* $O(cd \log_2 n)$ *rounds in expectation, where* $c > 1$ *is a constant.*

Corollary 5.4. *Algorithm* HDIA *is asymptotically optimal in terms of the interactive rounds for the general case of problem IDT in expectation if d is fixed.*

5.2 Algorithm HDIA with Interactive Error

Our algorithm *HDIA* follows the assumption that users always provide *correct* feedback in each interactive round. Here, consider two tuples $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$ and suppose that $f_{\boldsymbol{u}}(\boldsymbol{p}_i) > f_{\boldsymbol{u}}(\boldsymbol{p}_j)$ w.r.t. the user's utility vector $\boldsymbol{u}$. We say that a user's feedback is *correct* if the user tells that s/he prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$. However, in real-world scenarios, users may provide *incorrect* feedback (i.e., the user tells that s/he prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$) due to some reasons, e.g., a mis-click. Such incorrect feedback could result in the generation of wrong half-spaces, which in turn leads to a wrong update of utility range $\mathcal{R}$ (e.g., even if $f_{\boldsymbol{u}}(\boldsymbol{p}_i) > f_{\boldsymbol{u}}(\boldsymbol{p}_j)$, utility range $\mathcal{R}$ is updated to be $\mathcal{R} \cap h^-_{i,j}$), affecting the output of the algorithm.

Suppose that the user starts to provide incorrect feedback to the questions in the $(t+1)$-th cycle, i.e., the questions generated based on the first t batches of hyper-planes are correctly answered, where $t \geq 1$. Let us analyze the distance from any utility vector in the final $\mathcal{R}$ to the user's utility vector in the best case [47], which is achieved when the utility space is uniformly divided into the maximum number of partitions by all hyper-planes in the first t batches.

Lemma 5.5. *The distance from any utility vector in the final $\mathcal{R}$ to the user's utility vector is smaller than $(d(\frac{(\sqrt{2})^{d-1}}{(d-1)!m})^{2/d})^{1/2}$ in the best case, where $m = C^0_{\gamma t} + C^1_{\gamma t} + ... + C^{d-1}_{\gamma t}$ and γ denotes the number of hyper-planes in each batch.*

To handle the incorrect feedback, our strategy follows the idea of existing work [30]. It asks the same questions several times and chooses the majority feedback. Although there might exist incorrect feedback, we expect the accumulated feedback to be correct. Due to the lack of space, the details (including the corresponding experiments) can be found in Appendix A and Appendix B.

6 EXPERIMENT

We conducted experiments on a machine with 3.10GHz CPU and 16GB RAM. All programs were implemented in C/C++.

Datasets. The experiments were conducted on synthetic and real datasets that were commonly used in existing studies [27, 32, 44, 66]. The synthetic datasets were *anti-correlated* and were constructed by the generator developed for *skyline* operators [11, 44]. Following [74], we separated tuples into g equal-sized groups, where $g \geq 1$. The real datasets were *Lawschs* and *Adult* [63, 74]. Dataset *Lawschs* contains 56,233 student tuples from 25 law schools after the tuples with missing values are deleted. Each record is described by one sensitive (gender) and two scoring attributes (LSAT and GPA). Dataset *Adult* consists of 32, 561 individual's tuples, each of which is described by two sensitive attributes (gender and race) and five scoring attributes (education years, capital gain, capital loss, work hours per week, and overall weight). The tuples in the two real datasets were separated into groups based on their sensitive attributes.

For all the datasets, each scoring attribute is normalized to $(0, 1]$. Note that existing studies [12, 66] preprocessed datasets to contain skyline tuples only (which are all possible top-1 tuples w.r.t. at least a utility function) since they look for the (close to) top-1 tuple. Consistent with their setting, we also preprocessed all the datasets to enable a fair comparison of our algorithms with existing ones. For each group of the dataset, we only included k-skyband tuples (which are all possible top-k tuples w.r.t. at least a utility function) [27] since we were interested in the diverse top-k set.

Algorithms. We evaluated our 2-dimensional algorithm *TDIA*, and d-dimensional algorithm *HDIA*. The competitor algorithms are: *ActiveRanking* [30], *UH-Simplex* [66], *SinglePass* [72], *Preference-Learning* (denoted by *Pref-Learning* for short in the following) [47], and *RH* [62]. Since none of the existing algorithms are designed to solve our problem directly, we adapted them as follows:

- Algorithm *ActiveRanking* focuses on learning the full ranking of tuples by interacting with the user. We find the diverse top-k set by an existing algorithm $\nabla_{\boldsymbol{u}}(\mathcal{D})$ when the ranking is obtained.
- Algorithms *UH-Simplex* and *SinglePass* are proposed to return the user's (close to) top-1 tuple by interacting with the user. We maintain a set $\mathcal{S}$ which is initialized by $\mathcal{S} = \emptyset$. Each algorithm is iterated several times by using dataset $\mathcal{D} \setminus \mathcal{S}$. After the i-th iteration, we can obtain the top-1 tuple of dataset $\mathcal{D} \setminus \mathcal{S}$, which is the top-$i$ tuple of dataset $\mathcal{D}$. We put the tuple into $\mathcal{S}$ and check if there exist k tuples in $\mathcal{S}$ that satisfy the diversity constraint. If yes, we return the k tuples; otherwise we start another iteration.
- Algorithm *Pref-Learning* approximates the user's utility vector by interacting with the user. After the approximated utility vector $\boldsymbol{u}$ is obtained, we find the diverse top-k set based on the approximated utility vector $\boldsymbol{u}$ by an existing algorithm $\nabla_{\boldsymbol{u}}(\mathcal{D})$.
- Algorithms *RH* is proposed to achieve two goals based on its designed stopping conditions: returning desired tuples and learning the ranking of tuples by interacting with the user. We follow its design for the second goal. After the ranking is obtained, we find the diverse top-k set by an existing algorithm $\nabla_{\boldsymbol{u}}(\mathcal{D})$.

Parameter Setting. We evaluated the performance of each algorithm by varying different parameters. (1) The dataset size n. (2) The number of scoring attributes d. (3) The number of groups g (as shown in Section 3, the tuples can be divided into groups based on their sensitive attributes). Unless stated explicitly, following [62, 66, 74], the default setting of parameters on synthetic datasets is $n = 100,000$, $d = 4$, and $g = 3$. (4) The parameter k which decides the size of the output. We set $k = 10$ by default. For the diversity constraint, following [24, 74], we set the bounds proportionally, i.e., we require the proportion of each group in the output to be approximately equal to that in the dataset $\mathcal{D}$. Specifically, for each group $\mathcal{G}_i$, where $i \in \{1, 2, ..., g\}$, we set $l_i = \lfloor (1-\lambda)k \cdot \frac{|\mathcal{G}_i|}{|\mathcal{D}|} \rfloor$ and $b_i = \lceil (1+\lambda)k \cdot \frac{|\mathcal{G}_i|}{|\mathcal{D}|} \rceil$, where $\lambda = 0.1$ by default. (5) We also varied parameter λ.

Performance Measurement. We evaluated the performance of each algorithm by two measurements: (1) *the execution time* which is the processing time; (2) *the number of questions asked* which is the number of rounds interacting with the user. Each algorithm was conducted 10 times with different randomly generated user utility vectors and the average performance was reported.

6.1 Performance on Synthetic Datasets

Parameter Setting of *HDIA*. We explored several parameters' setting of algorithm *HDIA* on a 4-dimensional synthetic dataset.

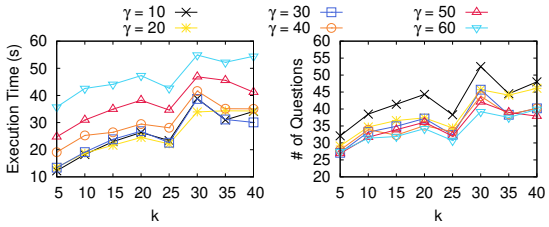


Figure 5: Batch Size γ

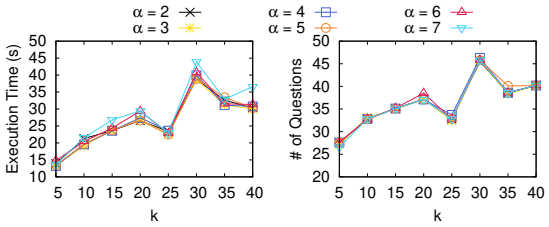


Figure 6: Parameter α

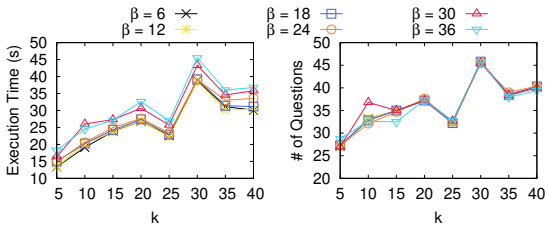


Figure 7: Parameter β

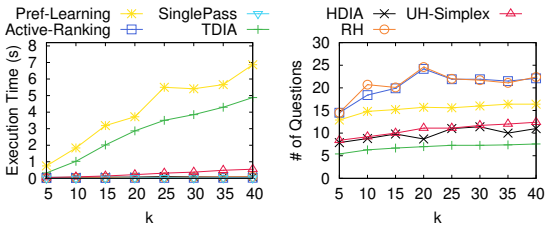


Figure 8: 2D

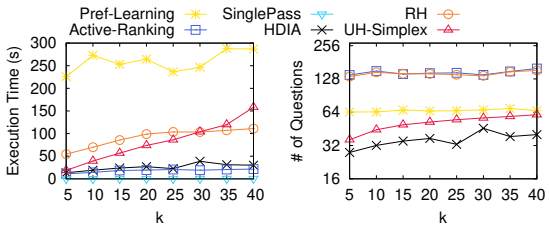


Figure 9: 4D

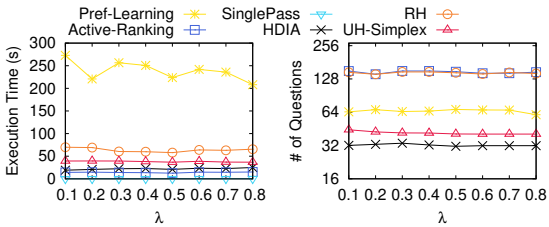


Figure 10: Vary λ

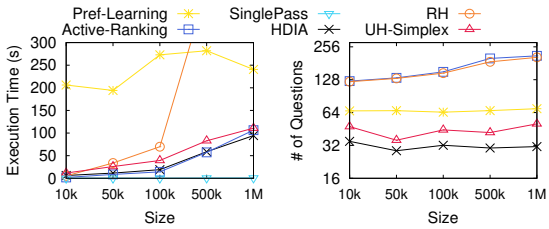


Figure 11: Vary Size n

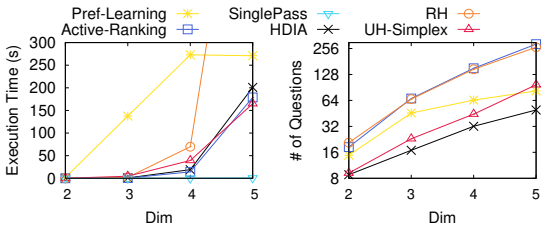


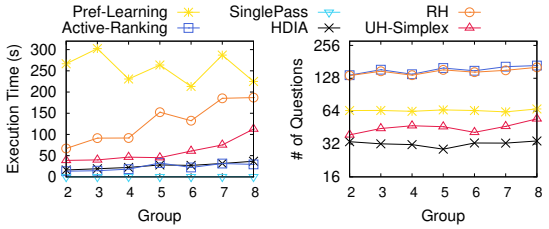
Figure 12: Vary # of scoring attributes d

Parameter γ. In Figure 5, we varied parameter γ, which decides the batch size (introduced in Section 5), from 10 to 60, and evaluated the execution time and the number of questions asked of the algorithm. The results show that when γ increases, the execution time rises but the number of questions asked decreases. This is because a large batch of hyper-planes provides a broader selection of candidate hyper-planes for tuple selection, which enhances the likelihood of asking effective questions to users, and thus, reduces the overall number of questions asked. However, it also results in a great number of partitions, causing a high computational cost. To achieve a balance, we set $\gamma = 30$ in the rest of our experiments.

Parameter α. As shown in Section 5, parameter α determines, for each node N, the number of hyper-planes in $\mathcal{H}(N)$ that can be used to derive L_N. In Figure 6, we varied parameter α from 2 to 7, and evaluated the execution time and the number of questions asked. Both measurements fluctuate slightly, which indicates that we only need to consider a small set of hyper-planes to derive L_N for each node N. Thus, we set $\alpha = 3$ in the rest of our experiments.

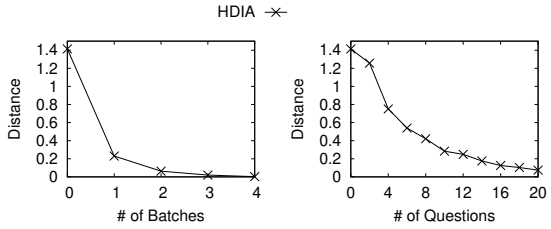
Parameter β. As shown in Section 5, parameter β is related to the number of partitions allowed in the leaf of the D-Tree. The partitions in the leaf should be no more than $1/\beta$ partitions in the root node. In Figure 7, we varied β from 6 to 36 and evaluated the execution time and the number of questions asked. As we can see, the execution time rises with the increasing β, while the number of questions asked remains almost the same. This indicates that constructing several small D-Trees is better than constructing a big completed D-Tree. Thus, we set $\beta = 6$ in the rest of our experiments.

Vary k. We studied the impact of parameter k on all algorithms.



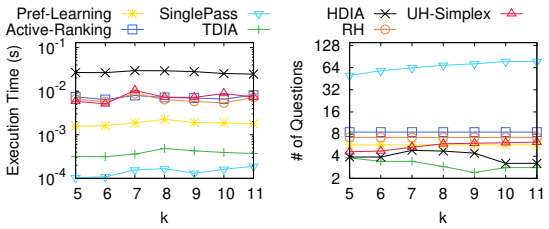
(a) (b)

Figure 13: Vary Group g



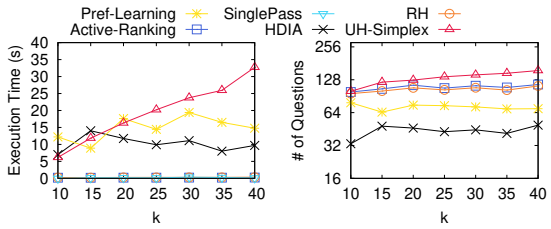
(a) (b)

Figure 14: Distance



(a) (b)

Figure 15: Lawschs



(a) (b)

Figure 16: Adult

2D Dataset. In Figure 8, we present the performance of algorithms on a 2-dimensional synthetic dataset by varying k from 5 to 40. The other parameters were set by default. Figure 8(a) depicts the execution time. All algorithms can finish within a few seconds, indicating overall efficiency. Our algorithm *TDIA* and the existing algorithm *Pref-Learning* spend slightly longer times than the others. This can be attributed to certain costly steps: algorithm *HDIA* needs to find all the possible diverse top-k sets in the exploration phase; algorithm *Pref-Learning* requires building a *spherical tree* data structure at the beginning. Nevertheless, these steps only extend the execution time a few more seconds and are completed before the interaction phase, ensuring that the marginally longer execution time does not impact the user interaction. Figure 9(b) presents the number of questions asked. We do not show algorithm *SinglePass* since it asks at least 1838 questions in all cases, which is two orders of magnitude more than those of other algorithms. Our algorithm *TDIA* asks the fewest questions. Compared with the best existing algorithm *UH-Simplex*, it asks at least 31% fewer questions. This performance demonstrates its superior effectiveness in minimizing the number of questions asked for arbitrary k.

4D Dataset. Figure 9 demonstrates the performance of algorithms on a 4-dimensional synthetic dataset by varying k from 5 to 40. The other parameters were set by default. Algorithm *SinglePass* achieves significantly shorter execution times. It takes approximately two orders of magnitude less time than other algorithms. However, this speed comes at the cost of a large number of questions (more than 2052 questions), which is two to three orders of magnitude more than those of other algorithms. Given that the number of questions mainly determines the user satisfaction during interaction (as discussed in Section 1), this trade-off results in a less favorable balance. To better highlight the performance of the remaining algorithms, *SinglePass* is omitted from Figure 9(b). Algorithms *ActiveRanking* and *RH* also ask many questions since they need to learn the complete ranking of tuples. For example, when $k = 40$, they ask 159.1 and 152.8 questions, respectively. Besides, the number of questions these two algorithms ask escalates when parameter k increases. This trend can be attributed to the preprocessing step applied to the datasets. Because the size of the processed dataset rises with the increasing k, both algorithms *ActiveRanking* and *RH* need to learn the ranking of more tuples. The number of questions asked by algorithm *Pref-Learning* remains consistent across different k. This is because *Pref-Learning* is designed to approximate the user's utility vector, which is independent of parameter k.

Algorithm *UH-Simplex* asks the fewest questions among existing algorithms. However, its execution time can extend to hundreds of seconds. In contrast, our algorithm *HDIA* requires much shorter execution time than *UH-Simplex* for arbitrary k. For instance, algorithm *HDIA* takes 30.08 seconds when $k = 40$, while *UH-Simplex* needs 158.34 seconds. Additionally, algorithm *HDIA* asks fewer questions than existing ones. Compared with the best algorithm *UH-Simplex*, it asks 15.4 fewer questions on average. This performance underscores the effectiveness of algorithm *HDIA*.

Vary λ. In Figure 10, we studied the impact of the tightness of the diversity constraint on the algorithms. We varied parameter λ from 0.1 to 0.8, which is used to restrict the proportion of each group in the output as introduced at the beginning of Section 6. We excluded algorithm *SinglePass* from Figure 10(b) since it asks more than two thousands of questions in all cases. The performances of algorithms *Pref-Learning*, *ActiveRanking*, and *RH* remain unaffected with the variation of parameter λ. This is because these algorithms either learn the ranking of tuples or approximate the user's utility vector, which are independent of the diverse constraints. For the other algorithms, when λ increases, the number of questions asked shows a downward trend. For example, algorithm *SinglePass* drops from 4579.9 to 2909.4. The underlying reason is that a larger λ corresponds to a more relaxed constraint, and thus, the algorithms are required to collect less information about the user's utility vector to identify the diverse top-k set. Our algorithm *HDIA* is affected slightly and it consistently asks the fewest questions in all cases, demonstrating its effectiveness in different diversity constraints.

Scalability. We evaluated the scalability of algorithms by varying three parameters n, d, and g, respectively.

Varying n. In Figure 11, we studied the scalability on dataset size n by varying the dataset size from 10k to 1M on 4-dimensional datasets. Figure 11(a) shows the execution time. Except for algorithms *Pref-Learning* and *RH*, the other algorithms display relatively similar execution times. Figure 11(b) demonstrates the number of questions asked by each algorithm. We omit algorithm *SinglePass* since it asks more than 3471.8 questions. Our algorithm *HDIA* scales the best. The number of questions it asks is significantly small, up to two orders of magnitude smaller than existing algorithms. For instance, algorithm *HDIA* asks 38 questions when $n = 500,000$, whereas algorithm *SinglePass* asks 5087.50 questions. The results highlight the effectiveness of *HDIA* in handling large datasets.

Varying d. In Figure 12, we evaluated the scalability on the number of scoring attributes d by varying d from 2 to 5 on 4-dimensional datasets. Algorithm *SinglePass* consistently spends the shortest execution time across all dimensions. However, *SinglePass* asks a significantly large number of questions, which is two to three orders of magnitude more than those of the other algorithms. For better demonstrations, we omit *SinglePass* in Figure 12(b). For the other algorithms, the execution time and the number of questions asked increase with high dimensions as expected. This can be attributed to the increased complexity of learning the user's utility vector in high-dimensional spaces. Our algorithm *HDIA* consistently outperforms others regarding the number of questions asked across all dimensions. For example, when $d = 5$, *HDIA* asks 49.6 questions, while the best existing algorithm *Pref-Learning* asks 82.60 questions. The results show the usefulness of *HDIA* in high-dimensional spaces.

Varying g. In Figure 13, we examined the performance of algorithms by varying the number of groups g from 2 to 8 on 4-dimensional datasets. We excluded *SinglePass* from Figure 13(b) since it asks thousands of questions (more than 2789.4 questions in all cases). The results show that except for algorithm *Pref-Learning*, both the execution time and the number of questions asked by each algorithm reduce when the number of groups decreases. This trend is caused by the complexity of diversity constraints. When the number of groups decreases, algorithms are required to collect less information of the user's utility vector to accurately identify the diverse top-k set. Conversely, algorithm *Pref-Learning* is designed to approximate the user's utility vector directly. Its core mechanism is not influenced by the diversity constraints associated with g. Our algorithm *HDIA* scales the best among all algorithms. For instance, when $g = 8$, *HDIA* asks 34.1 questions while the best existing algorithm *UH-Simplex* asks 54.4 questions. The results underscore *HDIA*'s adaptability in different diversity constraints.

Distance. Recall that in Lemma 5.5, we provide a theoretical analysis of the maximum distance of any two utility vectors in utility range $\mathcal{R}$ during the interaction in the best case. To augment this analysis, in Figure 14, we conducted experiments by considering the general case. In Figure 14(a), we recorded the maximum distance of any two utility vectors in utility range $\mathcal{R}$ after processing each batch of hyper-planes. The results reveal a rapid decrease. For instance, after the first batch, the maximum distance drops from 1.41 to 0.23. To demonstrate the how individual questions influence this decrease more clearly, in Figure 14(b), we recorded the maximum distance of any two utility vectors in utility range $\mathcal{R}$ after processing each question. The results also show a substantial decrease in distance after each question. These findings suggest that if the first few questions are correctly answered, even if mistakes occur in the subsequent user feedback, the difference between the learned utility vector and the user's utility vector remains small, and thus, the overall quality of the output is only marginally impacted.

6.2 Performance on Real Datasets

We studied the performance of our algorithms against existing ones on two real datasets, by varying parameter k. Figures 15 and 16 show the results on datasets *Lawschs* and *Adult*, respectively. Note that we only included algorithm *TDIA* on dataset *Lawschs* since *TDIA* only works for the datasets with two scoring attributes. We also excluded *SinglePass* from Figure 16(b) since it asks thousands of questions (more than 4048.4 questions). The results show that our algorithms perform well regarding the execution time and the number of questions asked. On dataset *Lawschs*, our algorithm *TDIA* consistently asks the fewest questions across various k. For example, when $k = 9$, while the best existing algorithm *Pref-Learning* asks 5.7 questions, *TDIA* only requires 2.4 questions, reducing the questions by 58%. Similarly, on dataset *Adult*, our algorithm *HDIA* outperforms others in terms of the number of questions asked for any given k. When $k = 10$, *Pref-Learning* poses 78.8 questions, while *HDIA* asks 33.2 questions, making a 57.9% reduction. These findings verify the effectiveness of our algorithms in real-life scenarios.

6.3 Summary

The experiments showed the superiority of our algorithms over the best-known existing ones: (1) We are effective and efficient. Our algorithms *TDIA* and *HDIA* ask fewer questions within less time than existing algorithms (e.g., on a 4-dimensional dataset with $k = 10$, while existing algorithms ask at least 78.8 questions, *HDIA* requires only 33.2 questions). (2) Our algorithms scale well on the dataset size, the number of dimensions, and the number of groups (e.g., ours ask 54 questions when $d = 5$, while existing algorithms ask at least 82.60 questions). (3) Our algorithms show great promise for real-world applications (e.g., on dataset *Adult* with $k = 10$, *HDIA* achieves a 57.9% reduction in the number of questions compared to existing algorithms). In summary, *TDIA* asks the fewest questions in a 2-dimensional space with a small execution time. In a d-dimensional space, *HDIA* runs within a few seconds and asks the fewest questions.

7 CONCLUSION

In this paper, we incorporate an interactive learning framework and a diversity mechanism, presenting interactive algorithms for finding the user's diverse top-k set. For the special case, where the dataset is described by two scoring attributes, we propose algorithm *TDIA*, which is asymptotically optimal w.r.t. the number of questions asked. For the general case, where the dataset is described by multiple scoring attributes, we present algorithm *HDIA*, which is asymptotically optimal w.r.t. the number of questions asked in expectation. Extensive experiments showed that our algorithms are efficient and effective. As for future work, we consider the dynamic datasets and data stream.

REFERENCES

[1] [n. d.]. https://www.microsoft.com/en-us/research/academic-program/phd-fellowship/canada-us/.

[2] [n. d.]. https://research.google/outreach/faq/?category=phd.

[3] Chiara Accinelli, Barbara Catania, Giovanna Guerrini, and Simone Minisi. 2021. covRew: a Python Toolkit for Pre-Processing Pipeline Rewriting Ensuring Coverage Constraint Satisfaction.. In *EDBT*. 698–701.

[4] Chiara Accinelli, Simone Minisi, and Barbara Catania. 2020. Coverage-based Rewriting for Data Preparation.. In *EDBT/ICDT Workshops*.

[5] Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, and Gautam Das. 2019. Designing Fair Ranking Schemes *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA.

[6] Abolfazl Asudeh, Azade Nazi, Nan Zhang, Gautam Das, and H. V. Jagadish. 2019. RRR: Rank-Regret Representative. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 263–280.

[7] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. 2007. Eliciting Matters – Controlling Skyline Sizes by Incremental Integration of User Preferences. In *Advances in Databases: Concepts, Systems and Applications*. Springer, Berlin, Heidelberg, 551–562.

[8] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. 2007. User Interaction Support for Incremental Refinement of Preference-Based Queries. In *Research Challenges in Information Science*. 209–220.

[9] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. 2014. Domination in the Probabilistic World: Computing Skylines for Arbitrary Correlations and Ranking Semantics. *ACM Transactions on Database Systems* 39, 2 (2014), 1–45.

[10] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. 2001. FeedbackBypass: A New Approach to Interactive Similarity Query Processing. In *Proceedings of the 27th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 201–210.

[11] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *Proceedings of the International Conference on Data Engineering*. 421–430.

[12] Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan. 2017. k-Regret Minimizing Set: Efficient Algorithms and Hardness. In *20th International Conference on Database Theory*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 11:1–11:19.

[13] Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. 2011. Efficient diversification of web search results. *arXiv preprint arXiv:1105.4255* (2011).

[14] Jaime Carbonell and Jade Goldstein. 1998. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Melbourne, Australia) *(SIGIR '98)*. Association for Computing Machinery, New York, NY, USA, 335–336.

[15] L. Elisa Celis, Lingxiao Huang, and Nisheeth K. Vishnoi. 2018. Multiwinner Voting with Fairness Constraints. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (Stockholm, Sweden) *(IJCAI'18)*. AAAI Press, 144–151.

[16] L Elisa Celis, Damian Straszak, and Nisheeth K Vishnoi. 2017. Ranking with fairness constraints. *arXiv preprint arXiv:1704.06840* (2017).

[17] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. 2008. *Computational geometry: Algorithms and applications*. Springer Berlin Heidelberg.

[18] Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang Nejdl. 2010. DivQ: diversification for keyword search over structured databases. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 331–338.

[19] Ting Deng and Wenfei Fan. 2013. On the complexity of query result diversification. *Proceedings of the VLDB Endowment* 6, 8 (2013), 577–588.

[20] Marina Drosou, Hosagrahar V Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. 2017. Diversity in big data: A review. *Big data* 5, 2 (2017), 73–84.

[21] Marina Drosou and Evaggelia Pitoura. 2010. Search result diversification. *ACM SIGMOD Record* 39, 1 (2010), 41–47.

[22] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through Awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 214–226.

[23] James Dyer and Rakesh Sarin. 1979. Measurable Multiattribute Value Functions. *Operations Research* 27 (08 1979), 810–822.

[24] Marwa El Halabi, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakab Tardos, and Jakub Tarnawski. 2020. Fairness in Streaming Submodular Maximization: Algorithms and Hardness. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 1142, 14 pages.

[25] Bahaeddin Eravci and Hakan Ferhatosmanoglu. 2013. Diversity based relevance feedback for time series search. *Proceedings of the VLDB Endowment* 7, 2 (2013), 109–120.

[26] Brian Eriksson. 2013. Learning to Top-k Search Using Pairwise Comparisons. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, Vol. 31. PMLR, Scottsdale, Arizona, USA, 265–273.

[27] Yunjun Gao, Qing Liu, Baihua Zheng, Li Mou, Gang Chen, and Qing Li. 2015. On processing reverse k-skyband and ranked reverse skyline queries. *Information Sciences* 293 (2015), 11–34.

[28] Chungwu Ho and Seth Zimmerman. 2006. On the number of regions in an m-dimensional space cut by n hyperplanes. *Terry Tao wins the Fields Medal* 294 (2006).

[29] David R Hunter. 2004. MM algorithms for generalized Bradley-Terry models. *The annals of statistics* 32, 1 (2004), 384–406.

[30] Kevin G. Jamieson and Robert D. Nowak. 2011. Active Ranking Using Pairwise Comparisons. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2240–2248.

[31] Ralph Keeney, Howard Raiffa, and David Rajala. 1979. Decisions with Multiple Objectives: Preferences and Value Trade-Offs. *Systems, Man and Cybernetics, IEEE Transactions on* 9 (08 1979), 403 – 403.

[32] Georgia Koutrika, Evaggelia Pitoura, and Kostas Stefanidis. 2013. Preference-Based Query Personalization. *Advanced Query Processing* (2013), 57–81.

[33] Jongwuk Lee, Gae-won You, and Seung-won Hwang. 2009. Personalized top-k skyline queries in high-dimensional space. *Information Systems* 34 (2009), 45–61.

[34] Jongwuk Lee, Gae-Won You, Seung-Won Hwang, Joachim Selke, and Wolf-Tilo Balke. 2012. Interactive skyline queries. *Information Sciences* 211 (2012), 18–35.

[35] Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and HV Jagadish. 2023. Query Refinement for Diversity Constraint Satisfaction. *Proceedings of the VLDB Endowment* 17, 2 (2023), 106–118.

[36] Jinyang Li, Alon Silberstein, Yuval Moskovitch, Julia Stoyanovich, and HV Jagadish. 2023. Erica: Query Refinement for Diversity Constraint Satisfaction. *Proceedings of the VLDB Endowment* 16, 12 (2023), 4070–4073.

[37] Tie-Yan Liu. 2010. Learning to Rank for Information Retrieval. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, 904.

[38] Ziyang Liu, Peng Sun, and Yi Chen. 2009. Structured search result differentiation. *Proceedings of the VLDB Endowment* 2, 1 (2009), 313–324.

[39] Alchemer LLC. 2022. https://www.alchemer.com/resources/blog/how-many-survey-questions/

[40] Lucas Maystre and Matthias Grossglauser. 2017. Just Sort It! A Simple and Effective Approach to Active Preference Learning. In *Proceedings of the 34th International Conference on Machine Learning*. 2344–2353.

[41] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino. 2012. Interactive Regret Minimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 109–120.

[42] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J. Lipton, and Jun Xu. 2010. Regret-Minimizing Representative Databases. In *Proceedings of the VLDB Endowment*, Vol. 3. VLDB Endowment, 1114–1124.

[43] Tu Ngoc Nguyen and Nattiya Kanhabua. 2014. Leveraging dynamic query subtopics for time-aware search result diversification. In *European Conference on Information Retrieval*. Springer, 222–234.

[44] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems* 30, 1 (2005), 41–82.

[45] Peng Peng and Raymong Chi-Wing Wong. 2015. K-Hit Query: Top-k Query with Probabilistic Utility Function. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 577–592.

[46] Evaggelia Pitoura, Kostas Stefanidis, and Georgia Koutrika. 2021. Fairness in Rankings and Recommendations: An Overview. *The VLDB Journal* 31, 3 (oct 2021), 431–458.

[47] Li Qian, Jinyang Gao, and H. V. Jagadish. 2015. Learning User Preferences by Adaptive Pairwise Comparison. In *Proceedings of the VLDB Endowment*, Vol. 8. VLDB Endowment, 1322–1333.

[48] QuestionPro. 2022. https://www.questionpro.com/blog/optimal-number-of-survey-questions/

[49] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. 2008. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*. 784–791.

[50] Davood Rafiei, Krishna Bharat, and Anand Shukla. 2010. Diversifying web search results. In *Proceedings of the 19th international conference on World wide web*. 781–790.

[51] J.-R. Sack and J. Urrutia. 2000. *Handbook of Computational Geometry*. North-Holland, Amsterdam.

[52] Gerard Salton. 1989. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., USA.

[53] Rodrygo LT Santos, Craig Macdonald, and Iadh Ounis. 2010. Selectively diversifying web search results. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. 1179–1188.

[54] Nihar B Shah and Martin J Wainwright. 2017. Simple, robust and optimal ranking from pairwise comparisons. *The Journal of Machine Learning Research* 18, 1 (2017), 7246–7283.

[55] Suraj Shetiya, Ian P Swift, Abolfazl Asudeh, and Gautam Das. 2022. Fairness-aware range queries for selecting unbiased data. In *Proc. of the Int. Conf. on Data Engineering, ICDE*.

[56] Mohamed A. Soliman and Ihab F. Ilyas. 2009. Ranking with uncertain scores. In *Proceedings of the International Conference on Data Engineering*. 317–328.

[57] Zhexuan Song and Nick Roussopoulos. 2001. K-Nearest Neighbor Search for Moving Query Point. In *International Symposium on Spatial and Temporal Databases*. Springer, Berlin, Heidelberg, 79–96.

[58] Julia Stoyanovich, Ke Yang, and H. V. Jagadish. 2018. Online Set Selection with Fairness and Diversity Constraints. In *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, Michael H. Böhlen, Reinhard Pichler, Norman May, Erhard Rahm, Shan-Hung Wu, and Katja Hose (Eds.). OpenProceedings.org, 241–252.

[59] Reinier H Van Leuken, Lluis Garcia, Ximena Olivares, and Roelof van Zwol. 2009. Visual diversification of image search results. In *Proceedings of the 18th international conference on World wide web*. 341–350.

[60] Marcos R Vieira, Humberto L Razente, Maria CN Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina, and Vassilis J Tsotras. 2011. On query result diversification. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 1163–1174.

[61] Weicheng Wang and Raymond Chi-Wing Wong. 2022. Interactive mining with ordered and unordered attributes. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2504–2516.

[62] Weicheng Wang, Raymond Chi-Wing Wong, and Min Xie. 2021. Interactive Search for One of the Top-k. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 13 pages.

[63] Min Xie Weicheng Wang, Raymond Chi-Wing Wong. 2023. Interactive Search with Mixed Attributes. In *In IEEE ICDE International Conference on Data Engineering*.

[64] Qiong Wu, Yong Liu, Chunyan Miao, Yin Zhao, Lu Guan, and Haihong Tang. 2019. Recent advances in diversified recommendation. *arXiv preprint arXiv:1905.06589* (2019).

[65] Min Xie, Tianwen Chen, and Raymond Chi-Wing Wong. 2019. FindYourFavorite: An Interactive System for Finding the User's Favorite Tuple in the Database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 2017–2020.

[66] Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. 2019. Strongly Truthful Interactive Regret Minimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 281–298.

[67] Min Xie, Raymond Chi-Wing Wong, Jian Li, Cheng Long, and Ashwin Lall. 2018. Efficient K-Regret Query Algorithm with Restriction-Free Bound for Any Dimensionality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 959–974.

[68] Min Xie, Raymond Chi-Wing Wong, Peng Peng, and Vassilis J. Tsotras. 2020. Being Happy with the Least: Achieving α-happiness with Minimum Number of Tuples. In *Proceedings of the International Conference on Data Engineering*. 1009–1020.

[69] Cong Yu, Laks Lakshmanan, and Sihem Amer-Yahia. 2009. It takes variety to make a world: diversification in recommender systems. In *Proceedings of the 12th international conference on extending database technology: Advances in database technology*. 368–378.

[70] Cong Yu, Laks VS Lakshmanan, and Sihem Amer-Yahia. 2009. Recommendation diversification using explanations. In *2009 IEEE 25th international conference on data engineering*. IEEE, 1299–1302.

[71] Meike Zehlike, Ke Yang, and Julia Stoyanovich. 2022. Fairness in Ranking, Part I: Score-Based Ranking. *ACM Comput. Surv.* 55, 6, Article 118 (dec 2022), 36 pages. https://doi.org/10.1145/3533379

[72] Guangyi Zhang, Nikolaj Tatti, and Aristides Gionis. 2023. Finding Favourite Tuples on Data Streams with Provably Few Comparisons. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (, Long Beach, CA, USA,) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 3229–3238.

[73] Jiping Zheng and Chen Chen. 2020. Sorting-Based Interactive Regret Minimization. In *Web and Big Data-4th International Joint Conference, APWeb-WAIM*. Springer, 473–490.

[74] Jiping Zheng, Yuan Ma, Wei Ma, Yanhao Wang, and Xiaoyang Wang. 2022. Happiness Maximizing Sets under Group Fairness Constraints. *Proc. VLDB Endow.* 16, 2 (oct 2022), 291–303.

[75] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. 2017. A survey of query result diversification. *Knowledge and Information Systems* 51, 1 (2017), 1–36.

Algorithm 3: Algorithm *HDIA-R*

1 Same as lines 1–11 in Algorithm 2;
2 Find tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ in node N; $Dif \leftarrow 0$;
3 **foreach** *extreme utility vector* $\boldsymbol{e} \in \mathcal{R}$ **do**
4 **if** $Dif < |f_{\boldsymbol{e}}(\boldsymbol{p}_i) - f_{\boldsymbol{e}}(\boldsymbol{p}_j)|$ **then**
5 $Dif \leftarrow |f_{\boldsymbol{e}}(\boldsymbol{p}_i) - f_{\boldsymbol{e}}(\boldsymbol{p}_j)|$;
6 Set $P(\boldsymbol{p}_i, \boldsymbol{p}_j)$ with Dif;
7 Interact with a user $O(\frac{2}{(2P(\boldsymbol{p}_i,\boldsymbol{p}_j)-1)^2} \ln \frac{2}{\epsilon})$ times;
8 Same as lines 13–37 in Algorithm 2;

Appendix A ALGORITHM HDIA-R

We propose a robust variant of algorithm *HDIA*, called *HDIA-R*, which considers the potential incorrect feedback during the interaction. We apply the Bradley–Terry model [29, 40, 54], which is a widely used probabilistic model for simulating the user feedback of tuple comparison. It is defined as follows. Let $\boldsymbol{u}$ be the user's utility vector and suppose that two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are presented to the user. The probability that a user provides a correct feedback is as follows.

$$P(\boldsymbol{p}_i, \boldsymbol{p}_j) = \frac{1}{1 + exp^{-|f_{\boldsymbol{u}}(\boldsymbol{p}_i) - f_{\boldsymbol{u}}(\boldsymbol{p}_j)|}}$$

Intuitively, the probability of providing correct feedback depends on the difference between the utilities of tuples. If the difference is large, the correct feedback happens in a high probability.

To handle the incorrect feedback, our strategy follows the idea of existing work [30]. It asks the same questions several times and chooses the majority feedback. Although there might exist incorrect feedback, we expect the accumulated feedback to be correct. Consider two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_i$. Since the probability that a user provides the correct feedback is concerned with the difference between the utilities of $\boldsymbol{p}_i$ and $\boldsymbol{p}_i$, i.e., $|f_{\boldsymbol{u}}(\boldsymbol{p}_i) - f_{\boldsymbol{u}}(\boldsymbol{p}_j)|$, the number of times asking the question (i.e., do you prefer $\boldsymbol{p}_i$ or $\boldsymbol{p}_i$?) also depends on the difference of utilities. The following lemma provides a theoretical guarantee for the number of times.

Lemma A.1. *We can obtain the correct user preference between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ with confidence $1-\epsilon$, if these two tuples are used to interact with the user in $O(\frac{1}{(2P(\boldsymbol{p}_i,\boldsymbol{p}_j)-1)^2} \ln \frac{2}{\epsilon})$ times.*

Proof. Given two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, let us denote the probability that a user provides incorrect feedback by $\tilde{P} = 1 - P(\boldsymbol{p}_i, \boldsymbol{p}_j)$. Suppose that we ask the user the same question t times and the number of correct feedback is t_c. To learn the correct user preference between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, we need $t_c > t/2$.

$$\begin{aligned} t_c > t/2 &\Rightarrow -t_c < -t/2 \\ &\Rightarrow (1-\tilde{P})t - t_c < (1-\tilde{P})t - t/2 \\ &\Rightarrow (1-\tilde{P})t - t_c < t/2 - \tilde{P}t \end{aligned}$$

Based on the Chernoff's bound, we have the probability below.

$$\mathbb{P}(|(1-\tilde{P})t - t_c| \geq t/2 - \tilde{P}t) \leq 2exp^{-2(1/2-\tilde{P})^2 t}$$

Let $\epsilon = \mathbb{P}(|(1-\tilde{P})t - t_c| \geq t/2 - \tilde{P}t) \leq 2exp^{-2(1/2-\tilde{P})^2 t}$, we have $t \leq -\frac{2}{(1-2\tilde{P})^2} \ln \frac{\epsilon}{2}$. Thus, we can obtain the correct user preference between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ with confidence $1-\epsilon$ if the question is asked $O(-\frac{2}{(1-2\tilde{P})^2} \ln \frac{\epsilon}{2}) = O(\frac{2}{(2P(\boldsymbol{p}_i,\boldsymbol{p}_j)-1)^2} \ln \frac{2}{\epsilon})$ times. □

Now, the next difficulty is how to learn the difference between the utilities of two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_i$, i.e., $|f_{\boldsymbol{u}}(\boldsymbol{p}_i) - f_{\boldsymbol{u}}(\boldsymbol{p}_j)|$ since the user's utility vector is unknown during the interaction. Recall that we maintain a utility range $\mathcal{R}$, which contains the user's utility vector. Thus, our strategy is to consider all the utility vectors in $\mathcal{R}$ and select the largest difference, i.e.,

$$\max_{\boldsymbol{u} \in \mathcal{R}} |f_{\boldsymbol{u}}(\boldsymbol{p}_i) - f_{\boldsymbol{u}}(\boldsymbol{p}_j)|$$

. Note that there are infinite utility vectors in $\mathcal{R}$. However, based on Lemma A.2, we only need to check the extreme utility vectors of utility range $\mathcal{R}$.

Lemma A.2. *The largest difference between the utilities of two tuples happens to an extreme utility vector of utility range $\mathcal{R}$.*

Proof. Let $E = \{\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_m\}$ be the set of the extreme utility vectors of $\mathcal{R}$. Since utility range $\mathcal{R}$ is a polyhedron, it is convex. For any extreme utility vector $\boldsymbol{u} \in \mathcal{R}$, there exist real numbers $a_1, a_2, ..., a_m$ such that $\boldsymbol{u} = a_1\boldsymbol{e}_1 + a_2\boldsymbol{e}_2 + ... + a_m\boldsymbol{e}_m$ and $a_1 + a_2 + ... + a_m = 1$. Suppose that $\hat{\boldsymbol{u}} = \max_{\boldsymbol{u} \in \mathcal{R}} |f_{\boldsymbol{u}}(\boldsymbol{p}_i) - f_{\boldsymbol{u}}(\boldsymbol{p}_j)|$. We have

$$\begin{aligned} |f_{\hat{\boldsymbol{u}}}(\boldsymbol{p}_i) - f_{\hat{\boldsymbol{u}}}(\boldsymbol{p}_j)| &= |\sum_{l=1}^{d} \hat{u}[i](p_i[l] - p_j[l])| \\ &= |\sum_{l=1}^{d} (\sum_{t=1}^{m} a_t \boldsymbol{e}_t[l](p_i[l] - p_j[l]))| \\ &\leq \sum_{t=1}^{m} a_t |\sum_{j=l}^{d} \boldsymbol{e}_t[l](p_i[l] - p_j[l])| \\ &\leq \max_{\boldsymbol{e} \in E} |\sum_{j=1}^{d} \boldsymbol{e}[l](p_i[l] - p_j[l])| \end{aligned}$$

Thus, the largest difference between the utilities of two tuples happens to an extreme utility vector of utility range $\mathcal{R}$. □

The pseudocode of algorithm *HDIA-R* is shown in Algorithm 3. When we find a pair of tuples, we check the number of times that we need to use these two tuples to interact with the user (lines 2-7).

Theorem A.3. *Assume that the smallest probability of providing a correct feedback on two tuples during the interaction is P_{min}. We can find the user's diverse top-k set within $O(\frac{2cd}{(2P_{min}-1)^2} \ln(\frac{2n^2}{\epsilon}) \log n)$ interactive rounds with confidence $1-\epsilon$.*

Proof. Based on Lemma A.1, for any two tuples, we need to interact with the user by $O(\frac{2}{(2P_{min}-1)^2} \ln \frac{2}{\epsilon})$ times and the confidence of learning the correct preference is $1-\epsilon$. Theorem 5.3 shows that we need to interact with a user in $O(cd \log n)$ pairs of tuples. Thus, we can obtain a confidence of at least $1 - n^2\epsilon$, if we interact with a user within $O(\frac{2cd}{(2P_{min}-1)^2} \ln(\frac{2}{\epsilon}) \log n)$. Thus, we can find the user's diverse top-k set within $O(\frac{2cd}{(2P_{min}-1)^2} \ln(\frac{2n^2}{\epsilon}) \log n)$ interactive rounds with confidence $1-\epsilon$. □

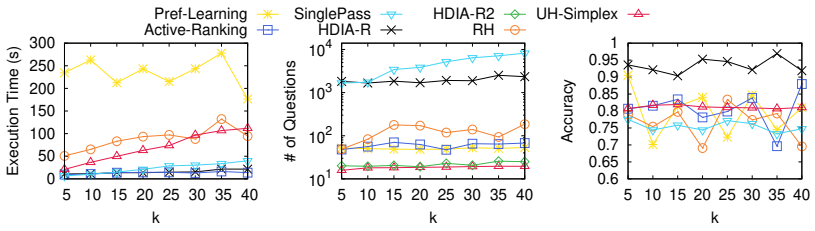


Figure 17: Error

Appendix B EXPERIMENTS

As discussed in Section A, users might make mistakes during the interaction. In this part, we compared our algorithm *HDIA-R* with existing ones, focusing on their robustness in handling user errors. We set parameter ϵ in algorithm *HDIA-R* to be 0.2. It's important to note that in algorithm *HDIA-R*, we employ $\max_{\boldsymbol{u}\in\mathcal{R}} |f_{\boldsymbol{u}}(\boldsymbol{p}_i) - f_{\boldsymbol{u}}(\boldsymbol{p}_j)|$ to determine the number of repetitions for each unique question. This approach, while effective, can sometimes result in many repetitions of the same question. To manage this, we set a limitation of 100 repetitions in the experiments to prevent excessive redundancy.

We apply the Bradley–Terry model to monitor the user feedback. To measure the performance of algorithms, in addition to the execution time and the number of questions asked, we follow [62] to include another measurement, called *accuracy*. Let $U(\mathcal{S})$ be the sum of utilities of tuples in set $\mathcal{S}$, i.e., $U(\mathcal{S}) = \sum_{\boldsymbol{p}\in\mathcal{S}} f(\boldsymbol{p})$. We define *accuracy* to be $U(\mathcal{S})/U(\mathcal{S}^*)$, where S is the output set returned by the algorithm and S^* is the ground truth.

Figure 17 shows the results. Our algorithm *HDIA-R* achieves the highest accuracy among all algorithms. In all cases, its accuracy surpasses 90%. Note that *HDIA-R* asks many questions. However, most of them are repetitions of the same questions. We believe that answering a previously encountered question requires less effort than answering a new one. This is because when a user encounters a question they've already answered, s/he is dealing with the same information and does not need to acquire new information. Moreover, the user is likely to have already formed opinions for the same question, simplifying the decision-making process. To provide a clear view of the actual user effort during the interaction, we use *HDIA-R2* to show the number of unique questions asked by *HDIA-R*. It can be seen that the number of unique questions is quite low, For instance, when $k = 20$, *HDIA-R* only asks 19.1 unique questions. This suggests that despite the large number of questions, the actual user effort during the interaction is manageable.

Appendix C PROOF

Proof of Corollary 4.5. Theorem 3.1 claims a lower bound $\Omega(\log_2 n)$ of the number of interactive rounds to find the user's diverse top-k set. As proved in Theorem 4.4, our algorithm *TDIA* can find the user's diverse top-k set within $O(\log_2 n)$ rounds. Thus, *TDIA* is asymptotically optimal in terms of the interactive rounds. □

Proof of Theorem 5.3. We consider that the algorithm terminates when it finishes processing all $\eta = \frac{n(n-1)}{2}$ hyper-planes $h_{i,j}$, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$, i.e., utility range $\mathcal{R}$ becomes one of the partitions divided by all η hyper-planes.

Since each hyper-plane $h_{i,j}$ and the utility space are $(d-1)$-dimensional, their intersection must lie in a $(d-2)$-dimensional hyper-plane. We denote the $(d-2)$-dimensional hyper-plane by $h'_{i,j}$. Consider the largest number of partitions that can be divided by η hyper-planes $h_{i,j}$. To achieve this, (1) all the hyper-planes $h_{i,j}$, where $p_i, p_j \in D$, intersect with the utility space; and (2) their intersections $h'_{i,j}$ are in general position [17]. Then, the utility space is divided into $N_{d-1}(\eta) = C_\eta^0 + C_\eta^1 + ... + C_\eta^{d-1}$ partitions [51]. In the following, we first assume that all partitions are in equal size and then consider the general case.

Special Case. Suppose that all $N_{d-1}(\eta)$ partitions are in equal size. In our algorithm, we randomly separate the hyper-planes into batches H_i. Consider the hyper-planes in the first $v-1$ batches, where $v \in [2, n]$. These hyper-planes divide the utility space into $N_{d-1}(\gamma(v-1))$ partitions, where γ is the batch size. Based on the Lemma 3 in [30], these partitions are also in equal size in expectation.

Assume that we have processed the first $v-1$ batches of hyper-planes. Utility range $\mathcal{R}$ becomes one of the partitions divided by these hyper-planes. Let us move to the v-th batch. Based on our interaction phase, we will select some hyper-planes as questions to ask the user. Consider any hyper-plane $h_{i,j} \in H_v$ selected. If it intersects with utility range $\mathcal{R}$, we need to ask the user a question. Consider the relationship between $h_{i,j}$ and the $N_{d-1}(\gamma(v-1))$ partitions divided by the first $v-1$ batches of hyper-planes. Hyper-plane $h'_{i,j}$ can be seen as being divided by the first $\gamma(v-1)$ hyper-planes $h'_{i,j}$ into $N_{d-2}(\gamma(v-1))$ disjoint polyhedrons. This means there are $N_{d-2}(\gamma(v-1))$ partitions among the $N_{d-1}(\gamma(v-1))$ partitions (divided by the first $v-1$ batches of hyper-planes $h'_{i,j}$) that intersect with $h'_{i,j}$, and thus, intersect with $h_{i,j}$.

Note that the $N_{d-1}(\gamma(v-1))$ partitions divided by the first $(v-1)$ batches of hyper-planes $h'_{i,j}$ are in equal size and utility range $\mathcal{R}$ is one of them before considering batch H_v. For each $h_{i,j} \in H_v$, since it intersects with $N_{d-2}(\gamma(v-1))$ partitions divided by the first $(v-1)$ batches of hyperplanes, the probability P_v that $h_{i,j}$ intersects with

$\mathcal{R}$ (i.e., the probability of asking a question) is as follows.

$$\begin{aligned} P_v &= \frac{N_{d-2}(\gamma(v-1))}{N_{d-1}(\gamma(v-1))} \\ &= \frac{C^0_{\gamma(v-1)} + C^1_{\gamma(v-1)} + ... + C^{d-2}_{\gamma(v-1)}}{C^0_{\gamma(v-1)} + C^1_{\gamma(v-1)} + ... + C^{d-1}_{\gamma(v-1)}} \\ &\leq \frac{C^0_{\gamma(v-1)} + C^1_{\gamma(v-1)} + ... + C^{d-2}_{\gamma(v-1)}}{C^{d-2}_{\gamma(v-1)} + C^{d-1}_{\gamma(v-1)}} \\ &= \frac{\sum_{\varphi=0}^{d-2} C^{\varphi}_{\gamma(v-1)}}{C^{d-1}_{\gamma(v-1)+1}} \end{aligned}$$

If $2(d-2) \leq \gamma(v-1)$, then $C^{\varphi}_{\gamma(v-1)} \leq C^{d-2}_{\gamma(v-1)}$, where $\varphi = 0, 1, ..., d-3$. Define $1 \leq \alpha \leq d-1$ such that $\alpha C^{d-2}_{\gamma(v-1)} \geq \sum_{\varphi=0}^{d-2} C^{\varphi}_{\gamma(v-1)}$. Then we have

$$P_v \leq \frac{\alpha C^{d-2}_{\gamma(v-1)}}{C^{d-1}_{\gamma(v-1)+1}} = \frac{\alpha(d-1)}{\gamma(v-1)+1} \leq \frac{\alpha(d-1)}{\gamma(v-1)}$$

If $2(d-2) > \gamma(v-1)$, the relation between $C^{\varphi}_{\gamma(v-1)}$ and $C^{d-2}_{\gamma(v-1)}$ varies, where $\varphi = 0, 1, ..., d-3$. For ease of calculation, we define $P_v = 1$. In this way, we define

$$P_v = \begin{cases} 1 & v \leq 2d \\ \frac{\alpha(d-1)}{\gamma(v-1)} & v > 2d \end{cases}$$

Because the probability that a hyper-plane in H_v intersects with utility range (i.e., the probability of asking a question) is P_v, the expected number of questions asked a user is $\sum_{v=1}^{m} \sum_{j=1}^{\gamma} P_v$, where $m = \eta/\gamma$ denotes the number of batches.

$$\begin{aligned} \sum_{v=1}^{m} \sum_{j=1}^{\gamma} P_v &= \sum_{v=1}^{2d} \sum_{j=1}^{\gamma} P_v + \sum_{v=2d+1}^{m} \sum_{j=1}^{\gamma} P_v \\ &\leq 2d\gamma + \sum_{v=2d+1}^{m} \sum_{j=1}^{\gamma} \frac{\alpha(d-1)}{\gamma(v-1)} \\ &\leq 2d\gamma + \sum_{v=2d+1}^{m} \frac{\alpha(d-1)}{(v-1)} \\ &\leq 2d\gamma \log_2(2d) + \alpha(d-1) \log_2(\frac{m}{2d+1-1}) \\ &\leq 2\alpha d\gamma \log_2 m \\ &\leq 4\alpha d\gamma \log_2 n \end{aligned}$$

Note that α and γ are two small constants. Thus, if the partitions divided by the η hyper-planes $h_{i,j}$ are in equal size, the expected number of questions asked is $\mathbb{E}_u = O(d \log_2 n)$.

Geneal Case. Consider the general case that all the partitions are in random size. Let $\mathbf{P}_i$ denote the proportion of the size of the i-partition to that of the utility space. Use $\mathcal{N}_i$ and $\mathbb{E}[\mathcal{N}_i]$ to represent the number of questions asked and the expected number of questions asked to locate the i-th partition if the user's utility point is in the i-th partition.

Denote by $\mathbb{E}_g$ the expected number of questions asked in the general case. We have

$$\mathbb{E}_g = \sum_{i=1}^{N_{d-1}(\eta)} \mathbb{E}[\mathcal{N}_i]$$

, where $\eta = n(n-1)/2$. Assume that $\mathbf{P}_i$ is bounded by $\mathbf{P}_i \leq \frac{c}{N_{d-1}(\eta)}$, for some constant $c > 1$. In order to obtain the largest $\mathbb{E}_g$, we distribute $\frac{c}{N_{d-1}(\eta)}$ to $j = \frac{N_{d-1}(\eta)}{c}$ partitions whose $\mathbb{E}[\mathcal{N}_i]$ is the largest. Without loss of generality, set $\mathbb{E}[\mathcal{N}_i] = \rho$ for these particular partitions (with the largest $\mathbb{E}[\mathcal{N}_i]$) and then the largest $\mathbb{E}_g$ should be ρ. For the remaining $N_{d-1}(\eta) - j$ partitions, each partition should be bounded by at least d hyper-planes. Since one question (showing points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ to the user) is needed for each bounded hyper-plane $h_{i,j}$, the number of questions asked for these partitions must be larger than d. Therefore, we have

$$\mathbb{E}_u = \frac{1}{N_{d-1}(\eta)} \sum_{i=1}^{N_{d-1}(\eta)} \mathbb{E}[\mathcal{N}_i] \geq \frac{\rho}{N_{d-1}(\eta)} j + d\frac{N_{d-1}(\eta) - j}{N_{d-1}(\eta)}$$

Then we obtain the largest $\mathbb{E}_g$

$$\mathbb{E}_g = \rho \leq c(\mathbb{E}_u - d\frac{N_{d-1}(\eta) - j}{N_{d-1}(\eta)}) \leq c\mathbb{E}_u$$

Therefore, in the general case, the expected number of questions asked is $\mathbb{E}_g = c\mathbb{E}_u = O(cd \log_2 n)$, where $c > 1$ is a constant. □

Proof of Corollary 5.4. Theorem 3.1 claims a lower bound $\Omega(\log_2 n)$ of the number of interactive rounds to find the user's diverse top-k set. As we proved in Theorem 5.3, our algorithm *HDIA* can find the user's diverse top-k set within $O(cd \log_2 n)$ rounds in expectation. Thus, if d is fixed, algorithm *HDIA* is asymptotically optimal in terms of the interactive rounds in expectation. □

Proof of Lemma 5.5. According to literature [17, 28, 51], the maximum number of partitions can be obtained if all γt hyper-planes intersect the utility space and these intersections are in *general position* [17, 30]. In this case, the utility space is divided into m partitions [17, 28, 51], where

$$m = C^0_{\gamma t} + C^1_{\gamma t} + ... + C^{d-1}_{\gamma t}.$$

Since the volume of the utility space $V_{\mathcal{U}} \leq \frac{(\sqrt{2})^{d-1}}{(d-1)!}$ and the utility space is uniformly divided into partitions, the volume of each partition is $V_\Theta = V_{\mathcal{U}}/m$. Note that the partitions are uniformly divided. Let us approximate each partition as a cube. The distance between any two utility vector in the partition would be smaller than $(dV_\Theta^{2/d})^{1/2}$. Because the final $\mathcal{R}$ must be in the partition, the distance from any utility vector in the final $\mathcal{R}$ to the user's utility vector is smaller than $(dV_\Theta^{2/d})^{1/2}$ in the best case. □