Interactive Learning for Diverse Top-k Set

Weicheng Wang[1], Raymond Chi-Wing Wong[1], Jinyang Li[2], H.V. Jagadish[2]
[1]*Hong Kong University of Science and Technology* [2]*University of Michigan*
wwangby@connect.ust.hk raywong@cse.ust.hk jinyli@umich.edu jag@umich.edu

***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 the query process and the final output. First, the query process requires users to specify their criteria explicitly and accurately in advance, which may be difficult for some users. Second, the final output often lacks diversity, which potentially leads to user dissatisfaction. To address these limitations, in this paper, we propose an enhanced 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 where tuples are described by two scoring attributes. We present the TDIA algorithm that is asymptotically optimal regarding the user effort needed for interaction. Then, we move on to the general case where tuples are described by multiple scoring attributes. We propose the HDIA algorithm which is asymptotically optimal w.r.t. the number of questions asked in expectation. Experiments were conducted on synthetic and real datasets. The results show that our algorithms can return a diverse output while requiring less user effort than existing ones.

***Index Terms*—top-k query, interactive query, diversity**

I. 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 clothes, admitting college students, and selecting job candidates [1]–[4]. Take the scenario of purchasing clothes for illustration. Consider a clothes database in Table I. Each clothes tuple is described by some attributes, e.g., quality, recyclability, and brand. Suppose that a user named Alice wants to buy some clothes. It can be a daunting task for her to manually review each tuple in the database. 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 clothes but also assist her in making a well-informed decision.

In the literature, various operators [5]–[8], referred to as the *multi-criteria decision-making* operators, have been proposed for this scenario. They characterize each user's criterion by a *utility function* $f_{\boldsymbol{u}}$. Each tuple $\boldsymbol{p}$ in the database 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. Guided by these utilities, the operators find tuples as the output.

There are two critical limitations of these operators. First, they assume that users' utility functions are known in advance. However, in real-world scenarios, users often face difficulty in precisely quantifying their individual criteria. For instance, Alice might struggle to articulate her exact trade-off between the quality and recyclability attributes. This uncertainty presents a significant challenge for these operators in returning accurate recommendations. Second, these operators tend to overlook the importance of *diversity* in the output. In Table I, tuples $\boldsymbol{p}_2$, $\boldsymbol{p}_3$, and $\boldsymbol{p}_4$ have the highest utilities but belong to the same brand. If they are recommended to Alice, she might feel dissatisfied due to the lack of variety. In market research, this is commonly known as *brand fatigue* [9], [10], where repeated exposure to the same brand diminishes user interest, even if the tuples meet users' criteria. 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 [11].

Table I: The Clothes Database ($\boldsymbol{u} = (0.7, 0.3)$)

Clothes	Quality	Recyclability	Brand	$f_{\boldsymbol{u}}(\cdot)$
$\boldsymbol{p}_1$	4.0	5.0	Nike	4.30
$\boldsymbol{p}_2$	4.7	4.8	H&M	4.73
$\boldsymbol{p}_3$	4.8	4.3	H&M	4.65
$\boldsymbol{p}_4$	5.0	4.0	H&M	4.70

Given these limitations, 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* [4], [12], [13] and a widely used *diversity mechanism* [14], [15] could help to address these limitations. In line with prior work [16], [17], we distinguish the attributes of tuples into two types: (1) *scoring attributes* which are used by the utility function in the interactive learning framework (e.g., quality and recyclability) and (2) *sensitive attributes* which are used by the diversity mechanism (e.g., brand).

The interactive learning framework engages users through a series of questions. Each question presents two tuples (with scoring attributes only) and asks the user to pick the one s/he prefers, where the two tuples are selected from the database based on the user's answers to previous questions. Following the user's answers, the user's utility function is learned.

The diversity mechanism operates by grouping tuples based on their sensitive attributes and restricting the number of tuples from each group in the output. For example, consider Table I. The clothes can be divided into two groups: one with the *Nike* brand and the other with the *H&M* brand. Let us restrict the output to include at least one clothes from each group. Then, set $\{\boldsymbol{p}_1, \boldsymbol{p}_2\}$ is considered diverse with respect to brand since it contains one clothes $\boldsymbol{p}_1$ from the *Nike* group and one clothes $\boldsymbol{p}_2$ from the *H&M* group. Note that diversity can be assessed in various ways [18], [19]. In this paper, we primarily follow

the popular method of restricting the number of returned tuples from each group. However, our algorithms proposed later can be adapted to different diversity methods (see Section VII-A).

By incorporating the *interactive learning framework* and the *diversity mechanism*, we propose problem *Interactive Learning for Diverse Top-k Set (Problem IDT)*. The goal is to interact with a user to find a *diverse top-k set* while minimizing the number of questions asked to the user. The diverse top-k set, following [14], [20], is defined as a set of k tuples that (1) have the highest utilities and (2) meet the diversity constraints, i.e., the number of tuples from each group satisfies the given restrictions. Besides, it is crucial that the questions should be as few as possible since excessive questioning can overwhelm users, leading to frustration and negatively impacting the interaction results, as shown in marketing research [21], [22].

Our proposed algorithms are built upon two phases: exploration and interaction. In the exploration phase, they process the database to gather relevant information for interaction. Subsequently, in the interaction phase, they select tuples based on this information to interact with the user and then implicitly learn the user's utility function. Once the utility function is sufficiently learned, they derive the top-ranked tuples and identify the user's diverse top-k set based on the diversity constraints.

A key insight behind our algorithms is that we can construct *hyper-planes* based on any two tuples in the database (details are postponed in Section III-B). These hyper-planes divide the domain of all possible utility functions, guiding the selection of tuples for interaction. Moreover, the hyper-planes based on the tuples presented in each question progressively narrow the domain of the utility function. When the narrowed domain is sufficiently small, the diverse top-k set can be determined.

To the best of our knowledge, we are the first to study problem IDT. While some closely related studies [12]–[14], [16], [23]–[25] involve interactive learning or consider the diversity issue, they differ significantly from us. [23] aims to rank tuples by interacting with users, and [24] proposes to learn the user's utility function via interaction. [12], [13] target to find a single tuple such that the *regret ratio*, evaluating user dissatisfaction with the tuple, is minimized by user interaction. These studies learn the user criteria through interactive learning but do not incorporate a *diversity* constraint. As a result, some questions they ask might be redundant in our setting due to diversity considerations. For example, we do not need to ask Alice to compare two clothes if they are known to be excluded from the diverse output set, but this comparison might be needed by [23], [24]. Other studies such as [14], [16], [25] concentrate on providing a diverse tuple set to users. However, [16], [25] assume the users' criteria are known in advance, which is often impractical. [14] 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 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 algorithm HDIA for the general case of problem IDT where tuples are described by multiple scoring attributes. It is asymptotically optimal w.r.t. the number of questions asked in expectation.
- 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 30% fewer questions than existing ones under typical settings.

In the following, we discuss the related work in Section II and formally define our problem in Section III. In Section IV, we propose an asymptotically optimal algorithm TDIA for a special case. In Section V, we propose an algorithm HDIA for the general case that performs well theoretically and empirically. Section VII presents our experimental results and Section VIII concludes this paper.

II. Related Work

Utility-based Query. The regret-minimizing query [26]–[29] does not require prior knowledge of the user's utility function. It considers all possible utility functions and returns a small set of tuples, expecting that each user can find at least one tuple that aligns with his/her criterion. Specifically, it minimizes a criterion called *regret ratio* that evaluates the user's dissatisfaction when comparing the returned set to the entire database. However, it is hard to achieve a small output size and a small regret ratio simultaneously. When the output size is small, the regret ratio is typically large [12], [28], [30]. Besides, since the query considers all possible utility functions, it returns the same tuples to all users. In contrast, our problem IDT requires the returned tuples to align with each user individually.

To overcome the deficiencies, several existing studies [12], [13] incorporate user interaction. [13] proposes the interactive regret-minimizing query, which aims to reduce the *regret ratio* while maintaining a small size of output by interacting with the user. 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 priced at 10 dollars with $50,000$ horsepower), leading to potential user disappointment [12]. To resolve this issue, [12] proposes the strongly truthful interactive regret-minimizing query, which utilizes *real tuples* (selected from the database). However, this query requires heavy user effort during the interaction, i.e., asking many questions. Besides, since both [13] and [12] 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 user effort, [31] 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. This 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 [3], [4], which explore the integration of different types of attributes into the interactive learning framework. Nevertheless, none of these studies [3], [4], [12], [13], [31] consider the diversity in the output.

In the field of machine learning, the problem of *learning to rank* [23] and *preference learning* [24] are similar to our problem IDT. However, they either learn the full ranking of tuples or approximate the user's criterion, which may require asking some questions that are unnecessary to our problem. For instance, if Alice prefers tuple 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 [23], [24].

Compared to existing studies, our problem IDT offers several advantages. (1) We use real tuples during the interaction, unlike [13] which incorporates fake tuples. (2) We require low user effort during the interaction. Contrary to existing studies that ask many questions to learn either a full ranking [23] or an exact user criterion [24], we only search for a set of k tuples. (3) We consider the diversity of output, addressing potential biases overlooked in other studies [3], [4], [12], [31].

Diversity-based Query. Query result diversification [32]–[35] is a line of study in information retrieval, aiming to provide a diverse output that covers different aspects or interpretations of the query, rather than returning multiple similar tuples [36]–[45]. There are two widely considered categories [33], [46]–[48] of diversity definitions in query result diversification: *content-based*, to include dissimilar tuples in the output; and *coverage-based*, to retrieve tuples of different categories or from different interpretations of the query [49].

Content-based diversity [18], [19] focuses on minimizing the similarity between tuples in the output. This is achieved through pairwise similarity measures, with the goal of ensuring that the aggregated similarity among the tuples in the output is as low as possible. There are multiple algorithms proposed, ranging from heuristics to dynamic programming.

For coverage-based query, a representative one is the *group diversity query* [14]–[16], which is closely related to our problem IDT. It groups tuples in the database based on the 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 contain 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 [47], [48], [50]. The algorithms in [47], [48] are designed in a greedy manner. In each round, they extract the tuple with the highest *score*. Initially, the score is determined by the query only (e.g., the user's criterion). Subsequently, the score involves both the query and the diversity conditioned on the selected tuples. [14], [50] propose dynamic programming-based algorithms. Some recent studies [25], [51], [52] aim to minimally refine the query to satisfy constraints on the size of specific groups in the output.

This line of study differs from our problem IDT since they return tuples relevant to a *pre-specified* 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 via user interaction, making it adaptable to cases where the query is initially *unspecified*.

III. Problem Definition

A. Problem IDT

Data. The input dataset $\mathcal{D}$ 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 [3], [12], [53], we assume that each scoring attribute is normalized to $(0, 1]$ and a larger attribute value is preferred.

Utility Function. Following [12], [31], [54], 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 [55], [56]. As verified in [3], [24], it can effectively capture how real users assess the multi-attribute tuples. Formally, the utility function is defined as follows.

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

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 [12], [13], [31], 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 $\boldsymbol{p}$. A higher utility means that $\boldsymbol{p}$ is more preferred.

Consider Table I. Suppose that $f_{\boldsymbol{u}}(\boldsymbol{p}) = 0.7p[1] + 0.3p[2]$, i.e., $\boldsymbol{u} = (0.7, 0.3)$. The utility of $\boldsymbol{p}_3$ w.r.t. $\boldsymbol{u}$ is $f_{\boldsymbol{u}}(\boldsymbol{p}_3) = 0.7 \times 4.8 + 0.3 \times 4.3 = 4.65$. Similarly for the other tuples.

Diversity Model. We adopt a well-established diversity model in the existing literature [14], [20], [57]. Except for the scoring attributes, each tuple $\boldsymbol{p}$ is also described by several sensitive attributes. Table I shows an example. It contains four tuples with two scoring attributes and one sensitive attribute. In the diversity model, a *group* is a set of tuples in $\mathcal{D}$ with the same value in at least one sensitive attribute. For instance, $\{\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$ is a group in Table I since the tuples in it have the same value H&M. Note that if the sensitive attributes are continuous, we can handle them by discretizing their values through established methods [52], [58]. For example, consider attribute *age* as a sensitive attribute. It can be discretized into three values: Young (< 30), Middle (30-60), and Old (> 60).

Given a dataset $\mathcal{D}$ and c groups $\mathbf{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_c\}$, the *diversity constraint* in the diversity model [14], [20], [57] is established 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 $\mathcal{G}_j \in \mathbf{G}$. For example, consider Table I. Assume that $\mathbf{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, where the tuples in $\mathcal{G}_1$ have the same value Nike and the tuples in $\mathcal{G}_2$ have the same value H&M. 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 $|\mathcal{S} \cap \mathcal{G}_2| = |\{\boldsymbol{p}_3\}| \geq 1$.

Given groups $\mathbf{G}$, 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 [20], [48], [50] finding set $\mathcal{S}^*$. We apply the one discussed in [20] to our work. Let us denote it by $\nabla_{\cdot}(\cdot)$, i.e., $\mathcal{S}^* = \nabla_{\boldsymbol{u}}(\mathcal{D})$. Note that there are various ways to define the diverse top-k set. Our definition follows [14], [20], which has gained wide acceptance in the literature. We also adapted our algorithms to accommodate different diversity models. The comparison can be found in Section VII-A.

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 [3], [12], [31]. The system interacts with a user for 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 it is satisfied, the interaction process ends, and the diverse top-k set is returned to the user. Otherwise, another interactive round begins. Formally, we are interested in the problem below.

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

B. Problem Characteristics

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 half-spaces. The *positive half-space* $h^+_{i,j}$ (resp. *negative half-space* $h^-_{i,j}$) 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 hyper-planes and half-spaces [59]. 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 shaded triangle when $d = 3$ as shown in Figure 1 or a bolded 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. Due to the lack of space, the proofs for some theorems and lemmas can be found in our technical report [60].

Lemma 1. ([31]) Given $\mathcal{U}$ and tuples $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$, if and only if a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, the user's utility vector is in $h^+_{i,j} \cap \mathcal{U}$.

Based on Lemma 1, 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.

IV. Special Case of IDT

We consider a special case of problem IDT, where tuples are 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 $\mathbb{R}^2$ (since $u[1], u[2] \geq 0$ and $u[1] + u[2] = 1$). We propose a two-dimensional 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 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 multiple utility vectors together.

Lemma 2. 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 $[\boldsymbol{u}_1, \boldsymbol{u}_2]$, the diverse top-$k$ sets w.r.t. any $\boldsymbol{u} \in [\boldsymbol{u}_1, \boldsymbol{u}_2]$ are the same.

Following Lemma 2, during the scanning process, we maintain the diverse top-k set w.r.t. the currently scanned utility vector. The set 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 inserted set in $\mathbf{C}$ 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 $[\wedge_{i,j}, (1, 0)] \subseteq h^-_{i,j}$

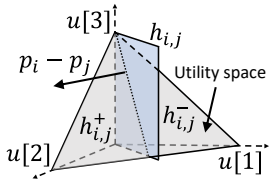


Figure 1: Hyper-plane

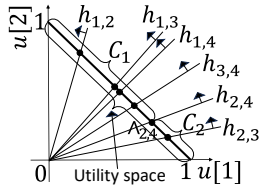


Figure 2: Algorithm TDIA

Table II: The Car Database

$\boldsymbol{p}$	HP ($\times 10^2$)	MPG ($\times 10$)	Torque ($\times 10^2$)	Brand
$\boldsymbol{p}_1$	2.6	2.4	2.3	Tesla
$\boldsymbol{p}_2$	3.4	1.6	2.3	Ford
$\boldsymbol{p}_3$	2.8	1.8	2.3	Ford

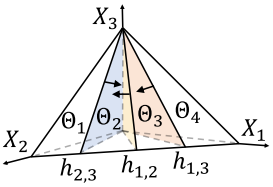


Figure 3: Partitions in $\mathbb{R}^d$

(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 $\mathcal{C}$ be the latest diverse top-k set inserted into $\mathbf{C}$. If $\boldsymbol{p}_i \in \mathcal{C}$ and $\boldsymbol{p}_j \notin \mathcal{C}$, we check whether set $\mathcal{C}' = \mathcal{C} \cup \{\boldsymbol{p}_j\} \setminus \{\boldsymbol{p}_i\}$ satisfies the diversity constraint. If it satisfies, set $\mathcal{C}'$ is a new diverse top-k set. We append $\mathcal{C}'$ to $\mathbf{C}$. This process terminates when all the intersections have been scanned.

Consider Table I. For each pair of clothes tuples in it, we build a hyper-plane, as shown in Figure 2. The norms of the hyper-planes are towards their positive half-spaces. Assume $\mathbf{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, where the tuples in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) have the same brand Nike (resp. H&M). Let $k = 2$, $l_1 = l_2 = 1$ (i.e., each brand should have at least one tuple), and $b_1 = b_2 = 2$ (i.e., each brand should have at most two tuples). Initially, we find the diverse top-k set $\mathcal{C}_1 = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$ w.r.t. utility vector $(0,1)$ and insert it into $\mathbf{C}$. Then, we reach the first intersection $\wedge_{1,2}$. Since $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are in set $\mathcal{C}_1$, we directly move to the second intersection $\wedge_{1,3}$. Here, given that $\boldsymbol{p}_1 \in \mathcal{C}_1$ and $\boldsymbol{p}_3 \notin \mathcal{C}_1$, we build set $\mathcal{C}' = \{\boldsymbol{p}_2, \boldsymbol{p}_3\}$. However, $|\mathcal{C}' \cap \mathcal{G}_1| < l_1$ makes $\mathcal{C}'$ unable to meet the diversity constraint. We do not insert $\mathcal{C}'$ into $\mathbf{C}$. For the other interactions, we perform similar steps. For $\wedge_{1,4}$, we build set $\mathcal{C}'' = \{\boldsymbol{p}_2, \boldsymbol{p}_4\}$. Since $\mathcal{C}''$ does not meet the diversity constraint, it is not inserted into $\mathbf{C}$. For $\wedge_{3,4}$, since $\boldsymbol{p}_3, \boldsymbol{p}_4 \notin \mathcal{C}_1$, there is no set built. For $\wedge_{2,4}$, since $\boldsymbol{p}_2 \in \mathcal{C}_1$ and $\boldsymbol{p}_4 \notin \mathcal{C}_1$, set $\mathcal{C}_2 = \{\boldsymbol{p}_1, \boldsymbol{p}_4\}$ is built and inserted into $\mathbf{C}$. For $\wedge_{2,3}$, since $\boldsymbol{p}_2, \boldsymbol{p}_3 \notin \mathcal{C}_2$, there is no set built. The final result is $\mathbf{C} =< \mathcal{C}_1, \mathcal{C}_2 >$, where $\mathcal{C}_1 = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$ and $\mathcal{C}_2 = \{\boldsymbol{p}_1, \boldsymbol{p}_4\}$.

Remark. Although there exist a few scan algorithms [31], [54], they pursue entirely different objectives and cannot be easily adapted to our problem. Note that the exploration phase can be precomputed before the interaction phase. It does not affect the real-time interaction experience.

Interaction. Let $\mathbf{C} =< \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, ... >$ 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 by *a binary search*. In each interactive round, (1) (*Tuple Selection*) it finds the median sets $\mathcal{C}_x, \mathcal{C}_{x+1} \in \mathbf{C}$ and presents a user with two tuples $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, where $\boldsymbol{p}_i$ is in $\mathcal{C}_x$ but not in $\mathcal{C}_{x+1}$, and $\boldsymbol{p}_j$ is in $\mathcal{C}_{x+1}$ but not in $\mathcal{C}_x$. (2) (*Information Maintenance*) If the user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, based on Lemma 3, half of the sets $< \mathcal{C}_{x+1}, \mathcal{C}_{x+2}, \mathcal{C}_{x+3}, ... >$ 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 3. If a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, the user's diverse top-k set cannot be the one among $< \mathcal{C}_{x+1}, \mathcal{C}_{x+2}, \mathcal{C}_{x+3}, ... >$.

Algorithm 1: Algorithm TDIA

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

In Figure 2, initially, set $\mathbf{C} =< \{\boldsymbol{p}_1, \boldsymbol{p}_2\}, \{\boldsymbol{p}_1, \boldsymbol{p}_4\} >$. We present a user with $\boldsymbol{p}_2$ and $\boldsymbol{p}_4$ (because $\mathcal{C}_x = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$, $\mathcal{C}_{x+1} = \{\boldsymbol{p}_1, \boldsymbol{p}_4\}$, $\boldsymbol{p}_2 \in \mathcal{C}_x$ (not $\mathcal{C}_{x+1}$) and $\boldsymbol{p}_4 \in \mathcal{C}_{x+1}$ (not $\mathcal{C}_x$)). Assume that the user's utility vector is $\boldsymbol{u} = (0.7, 0, 3)$. The user will tell that $\boldsymbol{p}_2$ is preferred. Following the feedback, set $\mathbf{C}$ is updated to be $< \{\boldsymbol{p}_1, \boldsymbol{p}_2\} >$. Since $|\mathbf{C}| = 1$, we stop the interaction process and return set $\{\boldsymbol{p}_1, \boldsymbol{p}_2\}$.

Summary. The pseudocode of TDIA is shown in Algorithm 1. Initially, set $\mathbf{C}$ is empty and all intersections are ranked (line 3). Based on these intersections, we scan the utility space to obtain all diverse top-k sets $\mathbf{C} = \{\mathcal{C}_1, \mathcal{C}_2, ...\}$ (lines 4-9). Then, we interact with a user to delete half of the sets in $\mathbf{C}$ in each interactive round (lines 11-16). When there is only one set left in $\mathbf{C}$ (line 10), we stop the interaction and return this set as output (line 17). The theoretical analysis is shown in Section VI.

V. 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 is a high-dimensional vector, and the utility space 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.

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 *iterations*: *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 $\mathcal{R}$ is the union of the left partitions. If there is only one partition left in $\mathcal{R}$, another iteration proceeds. Algorithm HDIA terminates if all utility vectors in $\mathcal{R}$ correspond to the same diverse top-k set $\mathcal{C}$.

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 $h_{i,j}$ and separate them into batches, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$. Each batch contains γ hyper-planes. In each iteration, 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. The randomized strategy ensures that each batch contains a balanced distribution of hyper-planes, leading to a more uniform division of the utility range.

For example, Table II contains three cars that are described by three scoring and one sensitive attribute. Suppose that the current utility range $\mathcal{R}$ is the whole utility space, and there are three hyper-planes in the current batch based on the car tuples in Table II. As shown in Figure 3, utility range $\mathcal{R}$ is divided into four partitions by these three hyper-planes whose norms are towards their positive half-spaces. Partition Θ_4 is the intersection of one positive and two negative half-spaces, i.e., $\Theta_4 = h^-_{1,2} \cap h^-_{1,3} \cap h^+_{2,3}$. Similarly for the other partitions.

To learn if a partition is in either positive or negative half-spaces (i.e., $\Theta \subseteq h^+$ or $\Theta \subseteq h^-$), 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. $\Theta \subseteq h^-$).

For the batch size γ, a small one might necessitate more iterations (i.e., repeating more times the exploration and interaction phases), 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 VII.

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 a 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_{pos} and N_{neg}. 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_{pos} and N_{neg}. The partition set $\boldsymbol{\Theta}(N_{pos})$ (resp. $\boldsymbol{\Theta}(N_{neg})$) in child N_{pos} (resp. N_{neg}) 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., $\boldsymbol{\Theta}(N_{pos}) = \{\Theta \in \boldsymbol{\Theta}(N) | \Theta \subseteq h^+_{i,j}\}$ (resp. $\boldsymbol{\Theta}(N_{neg}) = \{\Theta \in \boldsymbol{\Theta}(N) | \Theta \subseteq h^-_{i,j}\}$).
- Each leaf N has 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). Each leaf node contains a partition set $\boldsymbol{\Theta}(N)$ with $|\boldsymbol{\Theta}(N)| = 1$. 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 Θ_1, Θ_2, Θ_3, and Θ_4. Based on the hyper-plane $h_{1,2}$ of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, 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.

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 1, 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. 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 a recursive method 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 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_{pos} and N_{neg} such that (1) the partition sets $\boldsymbol{\Theta}(N_{pos})$ and $\boldsymbol{\Theta}(N_{neg})$ contain the partitions $\Theta \in \boldsymbol{\Theta}(N)$ in h^+ and h^-, respectively, and (2) $\mathcal{H}(N_{pos}) = \mathcal{H}(N_{neg}) = \mathcal{H}(N) \setminus \{h\}$. Then, we set

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

Intuitively, $\max\{L_{N_{pos}}, L_{N_{neg}}\}$ 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.

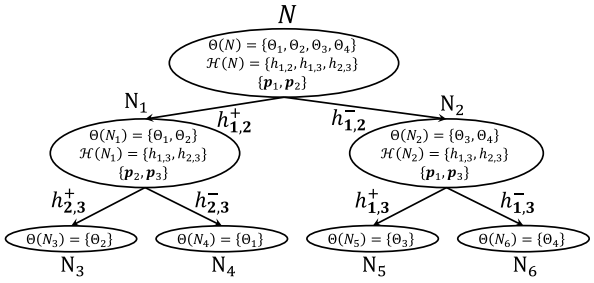


Figure 4: D-Tree

Since building D-Trees recursively can be time-consuming, we propose several strategies in the following to accelerate. 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 (threshold ≥ 1).

Our strategy is to construct several small D-Trees in loops, 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 loop, 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) where β is a user parameter and is a positive integer. The setting of β is discussed in Section VII. 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 with another loop 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 of N 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 $M(h^+)$ and $M(h^-)$ represents the number of partitions in half-space h^+ and h^-, respectively. Then, we only consider α hyper-planes with the largest M_h, where α is a user parameter and is a positive integer. The setting of α will be discussed in Section VII.

We also propose a lower bound for L_N. Suppose that we find a hyper-plane h. If using h to divide the partitions in N results in L_N reaching the lower bound, we can skip checking the remaining hyper-planes. This is because using other hyper-planes to divide the partitions in N will not make L_N smaller.

Lemma 4. 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 (which is a positive integer). Note that m/β indicates the number of partitions allowed in leaves.

Algorithm 2: Algorithm HDIA

Input: Dataset $\mathcal{D}$, set $\mathbf{G}$, parameters k, l_1, l_2, ..., b_1, b_2, ...
Output: The user's diverse top-k set
Randomly sequence all $h_{i,j}$, where $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$;
Separate the hyper-planes into batches; $\mathcal{R} \leftarrow \mathcal{U}$;
foreach *batch* H_i **do**
  Divide $\mathcal{R}$ into partitions $\boldsymbol{\Theta}_i = \{\Theta_1, ...\}$ based on H_i;
  Set root node N with $L_N \leftarrow \infty$, $\mathcal{H}(N) \leftarrow H_i$, and $\boldsymbol{\Theta}(N) \leftarrow \boldsymbol{\Theta}_i$;
  while $\boldsymbol{\Theta}(N) > 1$ **do**
    SearchLN(N, $|\boldsymbol{\Theta}(N)|/\beta$);
    $N' \leftarrow$ the root of the D-Tree;
    while N' *is not a leaf* **do**
      Use $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ in N' to interact with a user;
      if *the user prefers* $\boldsymbol{p}_i$ *to* $\boldsymbol{p}_j$ **then**
        $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{i,j}$; $N' \leftarrow N_{pos}$;
      else
        $\mathcal{R} \leftarrow \mathcal{R} \cap h^-_{i,j}$; $N' \leftarrow N_{neg}$;
      if *the stopping condition is satisfied* **then**
        return The diverse top-k set;
    $N \leftarrow N'$;

SearchLN(node N, β^*)
if $|\boldsymbol{\Theta}(N)| \leq \beta^*$ **then**
  $L_N \leftarrow 0$; **return**;
Select α hyper-planes $h \in \mathcal{H}(N)$ with the largest M_h.
foreach *selected hyper-plane* $h_{i,j}$ **do**
  if $h_{i,j}$ *separates the partitions in* $\boldsymbol{\Theta}$ *into two sets* **then**
    Build two children N_{pos} and N_{neg} for N;
    $\boldsymbol{\Theta}(N_{pos}) \leftarrow \{\Theta \in \boldsymbol{\Theta}(N) \mid \Theta \subseteq h^+_{i,j}\}$;
    $L_{N_{pos}} \leftarrow \infty$;
    $\mathcal{H}(N_{pos}) \leftarrow \mathcal{H}(N) \setminus \{h_{i,j}\}$
    SearchLN(N_{pos}, β^*);
    $\boldsymbol{\Theta}(N_{neg}) \leftarrow \{\Theta \in \boldsymbol{\Theta}(N) \mid \Theta \subseteq h^-_{i,j}\}$;
    $L_{N_{neg}} \leftarrow \infty$;
    $\mathcal{H}(N_{neg}) \leftarrow \mathcal{H}(N) \setminus \{h_{i,j}\}$;
    SearchLN(N_{neg}, β^*);
    if $L_N > \max\{L_{N_{pos}}, L_{N_{neg}}\} + 1$ **then**
      $L_N = \max\{L_{N_{pos}}, L_{N_{neg}}\} + 1$;
      Remove children except N_{pos} and N_{neg};
      if $L_N \leq \lceil \log_2 |\boldsymbol{\Theta}(N)|/\beta^* \rceil$ **then**
        break;
    else
      Remove the two children N_{pos} and N_{neg};

Proof. Let us build an optimal sub-tree that is rooted at node N. 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 utility range $\mathcal{R}$ 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. 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.

Summary. 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 randomly sequenced and separated into batches (lines 3-4). For each batch H_i ($i \geq 1$), we conduct two phases: exploration and interaction. In the exploration phase, the hyper-planes in H_i divide utility range $\mathcal{R}$ into partitions $\mathbf{\Theta}_i = \{\Theta_1, \Theta_2, ...\}$ (line 6). Then, in the interaction phase, we build D-Trees and use them for interaction.

For the first D-Tree, we initialize the root node to contain all hyper-planes in H_i and all partitions in $\mathbf{\Theta}_i$ (line 7). Assume that the construction process reaches a node N. If $|\mathbf{\Theta}(N)| \leq \beta^*$ (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 23-40). Note that we can skip the checking of some hyper-planes based on Lemma 4 (lines 37-38).

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 $\mathbf{\Theta}(N') = 1$, we turn to the next batch of hyper-planes. (2) If it is a leaf and $\mathbf{\Theta}(N') > 1$, 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 1, we build a hyper-plane $h_{i,j}$ and update $\mathcal{R}$ (either $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{i,j}$ or $\mathcal{R} \leftarrow \mathcal{R} \cap h^-_{i,j}$). In this way, $\mathcal{R}$ becomes smaller (*Information Maintenance*). We move to the child N'' of N', where the partitions contained in $\mathbf{\Theta}(N'')$ 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 Section VI.

Consider Figures 3 and 4. Suppose that there is only one batch of hyper-planes $H = \{h_{1,2}, h_{1,3}, h_{2,3}\}$ based on the car tuples in Table II. In the exploration phase, the utility range is divided into four partitions $\mathbf{\Theta} = \{\Theta_1, \Theta_2, \Theta_3, \Theta_4\}$ by these hyper-planes. Then, in the interaction phase, we set the root node N of the D-Tree by $\mathbf{\Theta}(N) = \{\Theta_1, \Theta_2, \Theta_3, \Theta_4\}$ and $\mathcal{H}(N) = \{h_{1,2}, h_{1,3}, h_{2,3}\}$. Assume that the number of partitions allowed in the leaf node is 2. Consider hyper-plane $h_{1,2}$. It can separate the partitions in $\mathbf{\Theta}(N)$ into two sets. We build two children N_1 and N_2 for N, where $\mathbf{\Theta}(N_1) = \{\Theta_1, \Theta_2\}$, $\mathbf{\Theta}(N_2) = \{\Theta_3, \Theta_4\}$, and $\mathcal{H}(N_1) = \mathcal{H}(N_2) = \{h_{1,3}, h_{2,3}\}$. Since $|\mathbf{\Theta}(N_1)| \leq 2$ and $|\mathbf{\Theta}(N_2)| \leq 2$, nodes N_1 and N_2 are set to be leaves. We have $L_{N_1} = L_{N_2} = 0$. Consequently, $L_N = 1$. Because $|\mathbf{\Theta}(N)| = 4$, L_N achieves lower bound $\lceil \log_2 |\mathbf{\Theta}(N)|/2 \rceil = 1$. We do not need to consider other hyper-planes to derive L_N for N. The D-Tree construction stops.

Then, we interact with the user based on the D-Tree. Let $k = 2$, $l_1 = l_2 = 1$ (i.e., each brand should have at least one car), and $b_1 = b_2 = 2$ (i.e., each brand should have at most two cars). The root node contains car tuples $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. We present them to the user as a question. Assume that the user's utility vector is $\boldsymbol{u} = (0.6, 0, 2, 0.2)$. The user will tell that $\boldsymbol{p}_2$ is preferred. We move to node N_2 and update $\mathcal{R}$ to be $\mathcal{R} \cap h^-_{1,2}$, i.e., $\Theta_3 \cup \Theta_4$. Since any extreme utility vector in $\Theta_3 \cup \Theta_4$ corresponds to the same diverse top-k set $\{\boldsymbol{p}_1, \boldsymbol{p}_3\}$, we stop the interaction and return the set as the output.

VI. Theoretical Analysis

We analyze the complexity of the problem IDT and establish theoretical guarantees for our proposed algorithms.

Problem IDT. We provide a lower bound on the number of interactive rounds needed to find the user's diverse top-k set.

Theorem 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.

TDIA. The theoretical analysis of TDIA is shown as follows.

Theorem 2. The exploration phase of TDIA runs in $O(n^2 \log n + Y_1 + Y_2 n^2)$ time, where Y_1 is the time complexity of $\nabla_\cdot(\cdot)$ [20] and Y_2 is the time complexity of checking if a set satisfies the diversity constraint [16].

Theorem 3. Algorithm TDIA solves the special case of problem IDT by interacting with a user within $O(\log_2 n)$ rounds.

Corollary 1. Algorithm TDIA is asymptotically optimal in terms of the number of interactive rounds for the special case of problem IDT.

HDIA. The theoretical analysis of HDIA is shown as follows.

Theorem 4. Constructing a D-Tree needs $O((2m)^\kappa \frac{\gamma!}{(\gamma-\kappa)!})$ time, where m is the total number of partitions, γ is the size of the hyper-plane batch, and κ is the height of the D-Tree.

Note that although the time complexity appears large due to the exponential expression, it remains manageable in practice. First, our strategy only constructs several small D-Trees (instead of a complete one), each representing a part of the complete D-Tree. This keeps the height κ of the small D-Tree low (e.g., 3). Second, we introduce a lower bound for L_N. It helps eliminate certain hyper-planes from consideration (instead of considering all γ hyper-planes). As a result, the D-Tree construction remains efficient, as shown in Section VII-A.

Theorem 5. 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 parameter used to bound the size of partitions.

Corollary 2. Algorithm HDIA is asymptotically optimal in terms of the number of interactive rounds for the general case of problem IDT in expectation if c and d are fixed.

VII. EXPERIMENT

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

Datasets. The experiments were conducted on synthetic and real datasets that were commonly used in existing studies [12], [30], [61], [62]. The synthetic datasets were *anti-correlated*. They were constructed by the generator developed for *skyline* operators [61], [63]. Following [14], we separated tuples into g equal-sized groups, where $g \geq 1$. The real datasets were *Adult* [14], *Bank* [64] and *Car* [65]. Dataset *Adult* consists of $32,561$ individual tuples, each of which is described by two sensitive attributes (gender and race) and five scoring attributes (education years, capital gain, etc.) Dataset *Bank* consists of $45,211$ individual tuples, each of which is described by one sensitive attribute (marital) and six scoring attributes (age, balance, duration, etc.). Dataset *Car* consists of $1,294,759$ individual car tuples. We use one sensitive attribute (transmission type: manual, automatic, and others) and three scoring attributes (mileage, power, and price). 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. 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) [62] since we were interested in the diverse top-k set.

Algorithms. We evaluated our algorithms TDIA and HDIA against existing ones: ACTIVERANKING [23], UH-SIMPLEX [12], SINGLEPASS [67], PREFERENCE-LEARNING (denoted by PREF-LEARNING for short in the following) [24], and RH [31]. 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 $\nabla_{\boldsymbol{u}}(\mathcal{D})$ 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 $\mathcal{S}$ that is initialized to be $\emptyset$ and perform the algorithm iteratively. At each iteration, the algorithm learns the top-1 tuple of dataset $\mathcal{D} \setminus \mathcal{S}$ and adds it to $\mathcal{S}$. This is achieved by re-using the information about the utility function that is learned from previous iterations and interacting with the user to refine the information based on dataset $\mathcal{D} \setminus \mathcal{S}$. After the iteration, we check whether $\mathcal{S}$ contains k tuples 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 $\nabla_{\boldsymbol{u}}(\mathcal{D})$.
- 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 $\nabla_{\boldsymbol{u}}(\mathcal{D})$.

Table III: Parameters and Values

Parameter	Meaning	Default Value
n	The dataset size.	100,000
d	The number of scoring attributes.	4
g	The number of groups.	3
k	The size of output.	10
λ	It controls the upper/lower bounds of the diversity constraint.	0.1
γ	The batch size.	30
α	The number of hyper-planes in $\mathcal{H}(N)$ that can be used to derive L_N.	3
β	It is related to the number of partitions allowed in the leaf of the D-Tree.	6

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 discussed in Section III, tuples can be divided into groups based on their sensitive attributes). Unless stated explicitly, following [12], [14], [31], 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 [14], [68], we set the bounds proportionally. That is, 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. Table III summarizes the parameters used in the experiments.

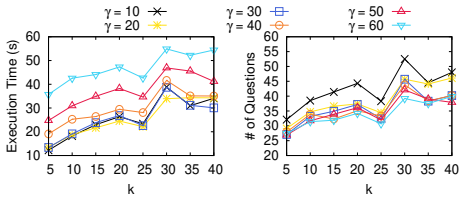
Performance Measurement. We evaluated algorithms with two measurements: (1) *the execution time*, which is the processing time; (2) *the number of questions asked*, which is the number of rounds interacting with users. Each algorithm was conducted 10 times with different randomly generated user utility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

A. Performance on Synthetic Datasets

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

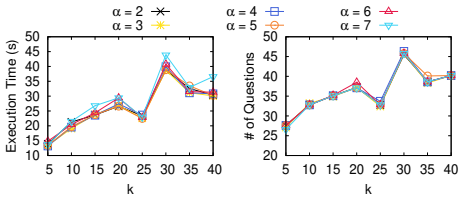
Parameter γ. In Figure 5, we varied parameter γ from 10 to 60, and evaluated the execution time and the number of questions asked. Parameter γ decides the batch size (introduced in Section V). The results show that when γ increased, the execution time rose, but the number of questions asked decreased. 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 V, 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



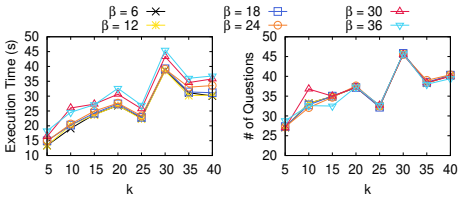
(a) (b)

Figure 5: Batch Size γ



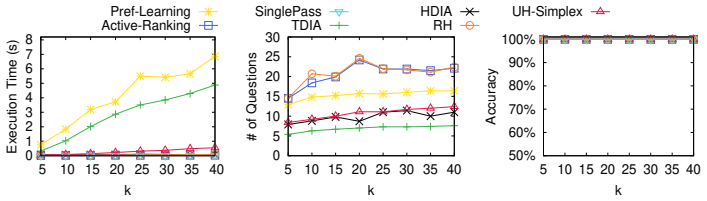
(a) (b)

Figure 6: Parameter α



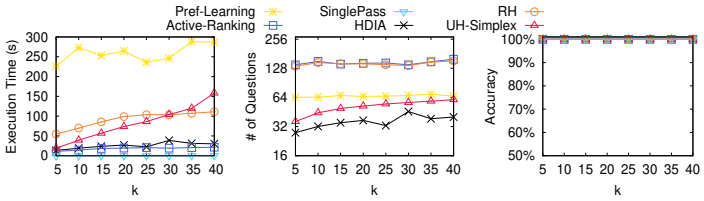
(a) (b)

Figure 7: Parameter β



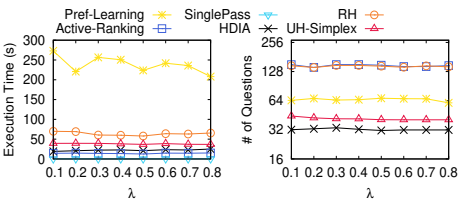
(a) (b) (c)

Figure 8: 2D Dataset



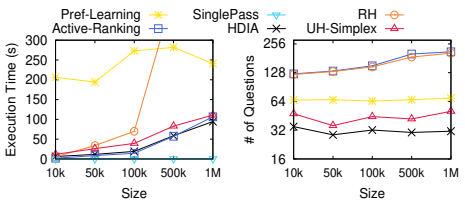
(a) (b) (c)

Figure 9: 4D Dataset



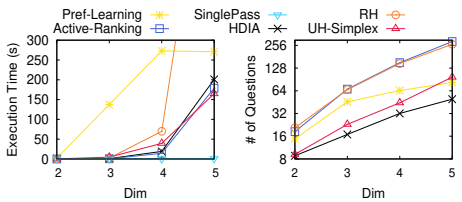
(a) (b)

Figure 10: Vary λ



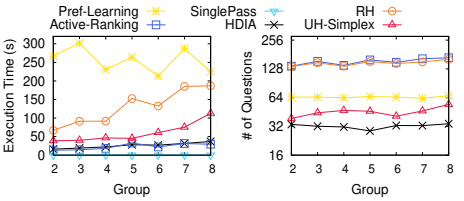
(a) (b)

Figure 11: Vary Size n



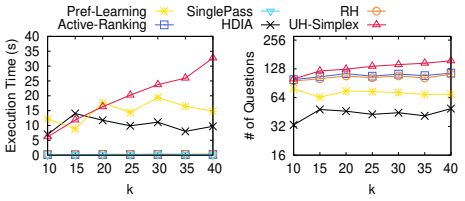
(a) (b)

Figure 12: Vary # of scoring attributes d



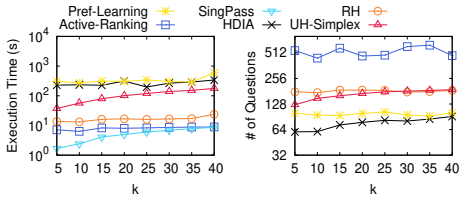
(a) (b)

Figure 13: Vary Group g



(a) (b)

Figure 14: Adult



(a) (b)

Figure 15: Bank

2 to 7, and evaluated the execution time and the number of questions asked. Both measurements fluctuated slightly, which indicated that we only needed 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 V, 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 can be seen, the execution time rose with the increasing β, while the number of questions asked remained almost the same. This indicated that constructing several small D-Trees was better than constructing a big completed D-Tree. Thus, we set $\beta = 6$ in the rest of our experiments.

Remark. When k increased, there was an upward trend in the execution time and the number of questions asked. This is expected since a larger k necessitates a smaller utility range to distinguish more top-ranked tuples. However, occasional dips may happen (e.g., when $k = 25$). This could be attributed to our dataset preprocessing, where the dataset was restricted to only include k-skyband tuples. When k increased, there were more tuples included. This introduced more hyper-planes, potentially leading to better questions for interaction.

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

2D Dataset. In Figure 8, we evaluate 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 could finish within 7 seconds, indicating high overall efficiency. TDIA and PREF-LEARNING took slightly longer times than the others. This is because TDIA has to find all possible diverse top-k sets in the exploration phase, and PREF-LEARNING requires building a *spherical tree* data structure at the beginning. Nevertheless, the additional execution time is minimal and occurs before the interaction phase, ensuring it does not affect the user

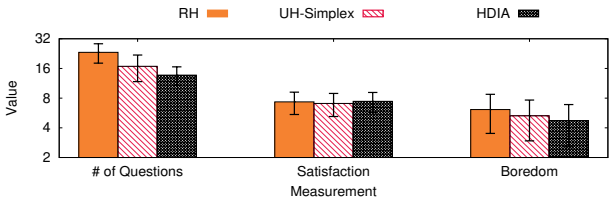


Figure 16: User Study

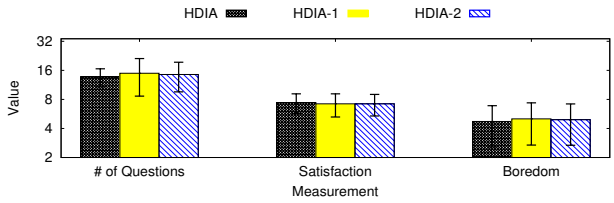


Figure 17: Diversity Study

interaction. Figure 8(b) presents the number of questions asked. We do not show SINGLEPASS since it asked over 1,838 questions in all cases, which was two orders of magnitude more than the others. Our algorithm TDIA asked the fewest questions. It reduced at least 31% questions compared to the best existing one UH-SIMPLEX. Figure 8(c) validates all algorithms returned the exact diverse top-k set. We compared the set $\mathcal{S}$ of tuples returned by each algorithm with the ground truth $\mathcal{S}^*$. The accuracy was defined as $|\mathcal{S}^* \cap \mathcal{S}|/|\mathcal{S}^*|$. We observed that all algorithms achieved 100% accuracy.

4D Dataset. Figure 9 presents the performance of algorithms on a 4-dimensional synthetic dataset by varying k from 5 to 40. The other parameters were set by default. SINGLEPASS achieved significantly shorter execution times. However, this speed came at the cost of a large number of questions (exceeding 2,052), which was two to three orders of magnitude more than the others. Given that the number of questions mainly determines user satisfaction during interaction (as discussed in Section I), 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). ACTIVERANKING and RH also asked many questions since they needed to learn the full ranking of tuples. For example, when $k = 40$, they asked 159.1 and 152.8 questions, respectively. Among existing algorithms, UH-SIMPLEX asked the fewest questions However, its execution time could extend to hundreds of seconds. In contrast, our algorithm HDIA required a much shorter execution time than UH-SIMPLEX for arbitrary k. For instance, it took 81% less time than UH-SIMPLEX when $k = 40$. Additionally, HDIA asked fewer questions than existing ones. Compared to the best algorithm UH-SIMPLEX, it asked 15.4 fewer questions on average. Figure 9(c) shows that all the algorithms returned the exact diverse top-k set.

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 VII. We excluded algorithm SINGLEPASS from Figure 10(b) since it asked more than two thousand questions in all cases. The performances of PREF-LEARNING, ACTIVERANKING, and RH remained 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 increased, the number of questions asked showed a downward trend. For example, algorithm SINGLEPASS dropped from 4,579.9 to 2,909.4. 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 was affected slightly, and it consistently asked the fewest questions in all cases.

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

Varying n. In Figure 11, we varied the dataset size n from 10k to 1M on 4-dimensional datasets. Figure 11(a) shows the execution time. Except for PREF-LEARNING and RH, the other algorithms displayed relatively similar execution times. Figure 11(b) displays the number of questions asked by each algorithm. We omitted SINGLEPASS since it asked more than 3,471.8 questions. Our algorithm HDIA scaled the best. The number of questions it asked is significantly small. For instance, HDIA asked 38 questions when $n = 500,000$, whereas SINGLEPASS asked 5,087.50 questions. The results highlight the effectiveness of HDIA in handling large datasets.

Varying d. In Figure 12, we varied the number of scoring attributes (i.e., dimensions) d from 2 to 5 on 4-dimensional datasets. SINGLEPASS consistently spent the shortest execution time across all dimensions. However, SINGLEPASS asked a significantly large number of questions, which was two to three orders of magnitude more than the other algorithms. For better demonstrations, we omitted SINGLEPASS in Figure 12(b). For the other algorithms, the execution time and the number of questions asked increased 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 outperformed the others regarding the number of questions asked across all dimensions. For example, when $d = 5$, HDIA asked 40% fewer questions than the best existing algorithm PREF-LEARNING. The results show the usefulness of HDIA in high-dimensional spaces.

Varying g. In Figure 13, we varied the number of groups g from 2 to 8 on 4-dimensional datasets. We excluded SINGLEPASS from Figure 13(b) since it asked thousands of questions (more than 2,789.4 questions in all cases). The results show that except for PREF-LEARNING, both the execution time and the number of questions asked by each algorithm reduced when the number of groups decreased. This trend was caused by the complexity of diversity constraints. When the number of groups decreased, algorithms were required to collect less information on the user's utility vector to accurately identify the diverse top-k set. Conversely, 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 scaled the best among all algorithms. For instance, when $g = 8$, HDIA asked 37% fewer questions than the best existing algorithm UH-SIMPLEX. The results underscore HDIA's adaptability in different diversity constraints.

Diversity Models. Our algorithm HDIA can be adapted to different diversity models. We implemented and evaluated it with two widely-used diversity models proposed by [18] and [19]. Due to space limitations, the details can be found in [60].

B. Performance on Real Datasets

Vary k. We evaluated algorithms on the three real datasets by varying parameter k. Figures 14 and 15 show the results on datasets *Adult* and *Bank*, respectively. Due to the lack of space, the results on dataset *Car* can be found in [60]. We excluded SINGLEPASS from Figures 14(b) and 15(b) since it asked thousands of questions. The results show that our algorithm performed well regarding the execution time and the number of questions asked. For example, when $k = 10$, compared to the best existing algorithm PREF-LEARNING, HDIA made 57.9% and 36.0% reduction in terms of the number of questions asked on *Adult* and *Bank*, respectively.

User Study. We conducted a user study on dataset *Car* to show our effectiveness on real users. Following existing settings in [3], [12], [31], we randomly selected 1000 candidate cars from the dataset. Each car was described by four attributes: price, horsepower, used mileage, and transmission type. During the interaction, only the first three attributes were shown, while all attributes were shown in the final results. We recruited 30 participants and reported their average results.

To reduce the burden on participants, we compared our algorithm HDIA against two existing ones: RH and UH-SIMPLEX. Both existing algorithms asked very few questions. SINGLEPASS was excluded due to the excessive number of questions it required. PREF-LEARNING was excluded from the comparison since its stopping condition assumes that the user's utility function is known, making it unsuitable for user study (where the user's utility function is unknown). Fixing $k = 10$, each algorithm was evaluated by three measurements. (1) *The number of questions asked*. (2) *Satisfaction*. It is a score from 1 to 10 given by each participant (the higher, the better), indicating how satisfied the participant is by considering one aspect: the returned cars. (3) *Boredom*. It is a score from 1 to 10 given by each participant (the lower, the better), showing how bored the participant feels by considering two aspects: the returned cars and the number of questions asked.

Figure 16 shows the average results and standard deviations. Our algorithm HDIA achieved the smallest standard deviations. As for the average results, our algorithm HDIA performed the best across all measurements. The number of questions asked by HDIA was 13.7, while existing algorithms RH and UH-SIMPLEX asked 23.3 and 16.8 questions, respectively. The boredom of our algorithm was also better than existing ones by at least 10%. This verifies the effectiveness of our algorithms on real users.

Diversity Model Study. We conducted a study on dataset *Car* to compare our algorithms equipped with different diversity modes. For clarity, we denote our algorithm with the diversity models in [14], [18], and [19] by HDIA, HDIA-1, and HDIA, respectively. The settings and measurements were consistent with those used in our previous user study.

Figure 17 summarizes the average results along with their standard deviations. As shown there, all three versions of our algorithm asked only a few questions. This indicates the effectiveness of our algorithm regardless of the diversity model applied. The satisfaction and boredom across all three versions were close. Nevertheless, the standard deviations indicate individual participants expressed various preferences. This suggests that user preferences for diversity may be subjective. Given the adaptability of our algorithm to different diversity models, it has the potential for various scenarios.

C. 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 2-dimensional dataset with $k = 10$, while existing algorithm RH asks 20.7 questions, TDIA requires only 6.3 questions). (2) Our algorithms scale well on the dataset size, the number of dimensions, and the number of groups (e.g., HDIA asks 40% fewer questions than the best existing algorithm when $d = 5$). (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.

VIII. 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 case that users cannot answer the question or equally prefer the presented tuples during the interaction.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their constructive comments. The research is supported in part by NSF grant 2312931.

REFERENCES

[1] M. Xie, T. Chen, and R. C.-W. Wong, "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*. New York, NY, USA: ACM, 2019, p. 2017–2020.

[2] M. Xie, R. C.-W. Wong, P. Peng, and V. J. Tsotras, "Being happy with the least: Achieving α-happiness with minimum number of tuples," in *Proceedings of the International Conference on Data Engineering*, 2020, pp. 1009–1020.

[3] M. X. Weicheng Wang, Raymond Chi-Wing Wong, "Interactive search with mixed attributes," in *In IEEE ICDE International Conference on Data Engineering*, 2023.

[4] W. Wang and R. C.-W. Wong, "Interactive mining with ordered and unordered attributes," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2504–2516, 2022.

[5] J. Lee, G.-w. You, and S.-w. Hwang, "Personalized top-k skyline queries in high-dimensional space," *Information Systems*, vol. 34, pp. 45–61, 2009.

[6] M. A. Soliman and I. F. Ilyas, "Ranking with uncertain scores," in *Proceedings of the International Conference on Data Engineering*, 2009, pp. 317–328.

[7] P. Peng and R. C.-W. Wong, "K-hit query: Top-k query with probabilistic utility function," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 577–592.

[8] Z. Song and N. Roussopoulos, "K-nearest neighbor search for moving query point," in *International Symposium on Spatial and Temporal Databases*. Berlin, Heidelberg: Springer, 2001, pp. 79–96.

[9] D. Mandaliya, "The what, why, and how of brand fatigue," 2024. [Online]. Available: https://www.marketingmonk.so/p/the-what-why-and-how-of-brand-fatigue

[10] O. Bukun-Joseph, "Navigating brand weariness, and paving the path forward," 2024. [Online]. Available: https://tosinloluwa.medium.com/navigating-brand-weariness-and-paving-the-path-forward-8ffaf9bb9dcb

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

[12] M. Xie, R. C.-W. Wong, and A. Lall, "Strongly truthful interactive regret minimization," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2019, p. 281–298.

[13] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino, "Interactive regret minimization," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2012, p. 109–120.

[14] J. Zheng, Y. Ma, W. Ma, Y. Wang, and X. Wang, "Happiness maximizing sets under group fairness constraints," *Proc. VLDB Endow.*, vol. 16, no. 2, p. 291–303, oct 2022.

[15] E. Pitoura, K. Stefanidis, and G. Koutrika, "Fairness in rankings and recommendations: An overview," *The VLDB Journal*, vol. 31, no. 3, p. 431–458, oct 2021.

[16] A. Asudeh, H. V. Jagadish, J. Stoyanovich, and G. Das, "Designing fair ranking schemes," ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019.

[17] J. Stoyanovich, K. Yang, and H. V. Jagadish, "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*, M. H. Böhlen, R. Pichler, N. May, E. Rahm, S. Wu, and K. Hose, Eds. OpenProceedings.org, 2018, pp. 241–252.

[18] P. Fraternali, D. Martinenghi, and M. Tagliasacchi, "Top-k bounded diversification," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 421–432.

[19] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *Proc. VLDB Endow.*, vol. 5, no. 11, p. 1124–1135, jul 2012.

[20] M. Zehlike, K. Yang, and J. Stoyanovich, "Fairness in ranking, part i: Score-based ranking," *ACM Comput. Surv.*, vol. 55, no. 6, dec 2022.

[21] "Alchemer llc," Last accessed on 02/2025. [Online]. Available: https://www.alchemer.com/resources/blog/how-many-survey-questions/

[22] "Questionpro," Last accessed on 02/2025. [Online]. Available: https://www.questionpro.com/blog/optimal-number-of-survey-questions/

[23] K. G. Jamieson and R. D. Nowak, "Active ranking using pairwise comparisons," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2011, p. 2240–2248.

[24] L. Qian, J. Gao, and H. V. Jagadish, "Learning user preferences by adaptive pairwise comparison," in *Proceedings of the VLDB Endowment*, vol. 8, no. 11. VLDB Endowment, 2015, p. 1322–1333.

[25] J. Li, Y. Moskovitch, J. Stoyanovich, and H. Jagadish, "Query refinement for diversity constraint satisfaction," *Proceedings of the VLDB Endowment*, vol. 17, no. 2, pp. 106–118, 2023.

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

[27] P. Peng and R. C.-W. Wong, "Geometry approach for k-regret query," in *Proceedings of the International Conference on Data Engineering*, 2014, pp. 772–783.

[28] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides, "Computing k-regret minimizing sets," in *Proceedings of the VLDB Endowment*, vol. 7, no. 5. VLDB Endowment, 2014, p. 389–400.

[29] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri, "Efficient algorithms for k-regret minimizing sets," *arXiv preprint arXiv:1702.01446*, 2017.

[30] G. Koutrika, E. Pitoura, and K. Stefanidis, "Preference-based query personalization," *Advanced Query Processing*, pp. 57–81, 2013.

[31] W. Wang, R. C.-W. Wong, and M. Xie, "Interactive search for one of the top-k," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2021.

[32] M. R. Vieira, H. L. Razente, M. C. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras, "On query result diversification," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 1163–1174.

[33] K. Zheng, H. Wang, Z. Qi, J. Li, and H. Gao, "A survey of query result diversification," *Knowledge and Information Systems*, vol. 51, no. 1, pp. 1–36, 2017.

[34] F. Radlinski, R. Kleinberg, and T. Joachims, "Learning diverse rankings with multi-armed bandits," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 784–791.

[35] R. H. Van Leuken, L. Garcia, X. Olivares, and R. van Zwol, "Visual diversification of image search results," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 341–350.

[36] C. Yu, L. V. Lakshmanan, and S. Amer-Yahia, "Recommendation diversification using explanations," in *2009 IEEE 25th international conference on data engineering*. IEEE, 2009, pp. 1299–1302.

[37] C. Yu, L. Lakshmanan, and S. Amer-Yahia, "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*, 2009, pp. 368–378.

[38] Q. Wu, Y. Liu, C. Miao, Y. Zhao, L. Guan, and H. Tang, "Recent advances in diversified recommendation," *arXiv preprint arXiv:1905.06589*, 2019.

[39] D. Rafiei, K. Bharat, and A. Shukla, "Diversifying web search results," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 781–790.

[40] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri, "Efficient diversification of web search results," *arXiv preprint arXiv:1105.4255*, 2011.

[41] R. L. Santos, C. Macdonald, and I. Ounis, "Selectively diversifying web search results," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010, pp. 1179–1188.

[42] Z. Liu, P. Sun, and Y. Chen, "Structured search result differentiation," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 313–324, 2009.

[43] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl, "Divq: diversification for keyword search over structured databases," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 2010, pp. 331–338.

[44] B. Eravci and H. Ferhatosmanoglu, "Diversity based relevance feedback for time series search," *Proceedings of the VLDB Endowment*, vol. 7, no. 2, pp. 109–120, 2013.

[45] T. N. Nguyen and N. Kanhabua, "Leveraging dynamic query subtopics for time-aware search result diversification," in *European Conference on Information Retrieval*. Springer, 2014, pp. 222–234.

[46] T. Deng and W. Fan, "On the complexity of query result diversification," *Proceedings of the VLDB Endowment*, vol. 6, no. 8, pp. 577–588, 2013.

[47] J. Carbonell and J. Goldstein, "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*, ser. SIGIR '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 335–336.

[48] M. Drosou, H. V. Jagadish, E. Pitoura, and J. Stoyanovich, "Diversity in big data: A review," *Big data*, vol. 5, no. 2, pp. 73–84, 2017.

[49] M. Drosou and E. Pitoura, "Search result diversification," *ACM SIGMOD Record*, vol. 39, no. 1, pp. 41–47, 2010.

[50] L. E. Celis, D. Straszak, and N. K. Vishnoi, "Ranking with fairness constraints," *arXiv preprint arXiv:1704.06840*, 2017.

[51] S. Shetiya, I. P. Swift, A. Asudeh, and G. Das, "Fairness-aware range queries for selecting unbiased data," in *Proc. of the Int. Conf. on Data Engineering, ICDE*, 2022.

[52] J. Li, A. Silberstein, Y. Moskovitch, J. Stoyanovich, and H. Jagadish, "Erica: Query refinement for diversity constraint satisfaction," *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 4070–4073, 2023.

[53] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall, "Efficient k-regret query algorithm with restriction-free bound for any dimensionality," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2018, p. 959–974.

[54] A. Asudeh, A. Nazi, N. Zhang, G. Das, and H. V. Jagadish, "Rrr: Rank-regret representative," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2019, pp. 263–280.

[55] J. Dyer and R. Sarin, "Measurable multiattribute value functions," *Operations Research*, vol. 27, pp. 810–822, 08 1979.

[56] R. Keeney, H. Raiffa, and D. Rajala, "Decisions with multiple objectives: Preferences and value trade-offs," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 9, pp. 403 – 403, 08 1979.

[57] L. E. Celis, L. Huang, and N. K. Vishnoi, "Multiwinner voting with fairness constraints," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 144–151.

[58] H. Liu, R. C.-W. Wong, Z. Zhang, M. Xie, and B. Tang, "Fair top-k query on alpha-fairness," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 2338–2350.

[59] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational geometry: Algorithms and applications*. Springer Berlin Heidelberg, 2008.

[60] W. Wang, R. C.-W. Wong, J. Li, and H. Jagadish, "Interactive learning for diverse top-k set," Tech. Rep., 2025. [Online]. Available: https://github.com/WANGWC1996/2025ICDE-Interactive-Learning-for-Diverse-Top-k-Set

[61] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems*, vol. 30, no. 1, p. 41–82, 2005.

[62] Y. Gao, Q. Liu, B. Zheng, L. Mou, G. Chen, and Q. Li, "On processing reverse k-skyband and ranked reverse skyline queries," *Information Sciences*, vol. 293, pp. 11–34, 2015.

[63] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of the International Conference on Data Engineering*, 2001, p. 421–430.

[64] "Dataset bank," Last accessed on 02/2025. [Online]. Available: https://archive.ics.uci.edu/dataset/222/bank+marketing

[65] "Dataset car," Last accessed on 11/2024. [Online]. Available: https://www.kaggle.com/datasets/ekibee/car-sales-information?select=region25_en.csv

[66] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R. C.-W. Wong, and W. Zhan, "k-Regret Minimizing Set: Efficient Algorithms and Hardness," in *20th International Conference on Database Theory*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 11:1–11:19.

[67] G. Zhang, N. Tatti, and A. Gionis, "Finding favourite tuples on data streams with provably few comparisons," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 3229–3238.

[68] M. El Halabi, S. Mitrović, A. Norouzi-Fard, J. Tardos, and J. Tarnawski, "Fairness in streaming submodular maximization: Algorithms and hardness," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.