Interactive Mining with Ordered and Unordered Attributes

Weicheng Wang, Raymond Chi-Wing Wong
Hong Kong University of Science and Technology
wwangby@connect.ust.hk,raywong@cse.ust.hk

ABSTRACT

There are various queries proposed to assist users in finding their favorite tuples from a dataset with the help of user interaction. Specifically, they interact with a user by asking questions. Each question presents two tuples, which are selected from the dataset based on the user's answers to the previous questions, and asks the user to select the one s/he prefers. Following the user feedback, the user preference is learned implicitly, and the best tuple w.r.t. the learned preference is returned. However, existing queries only consider datasets with ordered attributes (e.g., price), where there exists a trivial order on the attribute values. In practice, a dataset can also be described by unordered attributes, where there is no consensus about the order of the attribute values. For example, the size of a laptop is an unordered attribute. One user might favor a large size because s/he could enjoy a large screen, while another user may prefer a small size for portability. In this paper, we study how to find a user's favorite tuple from the dataset that has both ordered and unordered attributes by interacting with the user.

We study our problem progressively. First, we look into a special case in which the dataset is described by one ordered and one unordered attributes. We present algorithm *DI* that is asymptotically optimal in terms of the number of questions asked. Then, we dig into the general case in which the dataset has several ordered and unordered attributes. We propose two algorithms *BS* and *EDI* that have provable performance guarantees and perform well empirically. Experiments were conducted on synthetic and real datasets, showing that our algorithms outperform existing algorithms in the number of questions asked and the execution time. Under typical settings, our algorithms ask up to 10 times fewer questions and take several orders of magnitude less time than existing algorithms.

PVLDB Reference Format:
Weicheng Wang, Raymond Chi-Wing Wong. Interactive Mining with Ordered and Unordered Attributes. PVLDB, 15(11): 2504 - 2516, 2022.
doi:10.14778/3551793.3551810

1 INTRODUCTION

Given a dataset described by several attributes, the attributes could be either ordered or unordered, where "ordered" means that there is a trivial order on the attribute values without considering the impact of other attributes, while "unordered" implies that users have various preferences on the attribute values. For example, in a car dataset, each car could be described by two attributes price and the number of seats. Price is an ordered attribute. If it is considered alone, users are always willing to spend as little money as possible. The number of seats is an unordered attribute. A user with a family might prefer a car with multiple seats to carry the entire family, or a single user may favor a car with a few seats to save fuel consumption. Thus, the user's expected number of seats could vary. Note that a trivial order is based on our common cognition or obtained from pre-knowledge. It is possible that a trivial order is not suitable for a particular user. For example, it is generally considered that a car's maximum speed should be as high as possible. However, some users may think that it is necessary to keep a car at a safe speed and do not pursue a high maximum speed. Thus, when needed, users can specify whether an attribute is ordered or unordered so that the type of attributes must exactly match the user preference.

Many operators have been proposed to assist users in finding their favorite tuples from a dataset with both ordered and unordered attributes. Such operators, regarded as *multi-criteria decision-making tool*, can be applied in various scenarios, including purchasing a car, buying a house, and picking a red wine. For example, Alice wants to buy a car. She might have an *expected* car in her mind, e.g., a cheap car with multiple seats. Based on her expected car, the operators search the dataset and recommend cars to Alice.

There are two representative operators: *the relative skyline query* [9] and *the k nearest neighbors query (kNN)* [29]. The relative (or dynamic) skyline query returns all tuples that are not *dominated* by other tuples, i.e., all the possible nearest tuples. A tuple p *dominates* another tuple q if p in each attribute is no farther from the user's expected tuple than q, and strictly closer in at least one attribute. Unfortunately, the output size of the relative skyline query is uncontrollable, and it often overwhelms users with excessive results [20]. The k nearest neighbors query (kNN) measures the user preference on tuples by the Euclidean distance of each tuple to the user's expected tuple. A smaller distance means that the tuple is more favored by the user. kNN returns the k tuples, called k nearest tuples, that have the smallest distance to the user's expected tuple. Unlike the relative skyline query, kNN fixes the output size to a number k. However, most users have difficulties in specifying their expected tuples explicitly [20, 35]. If the user's expected tuple is not known, kNN cannot be applied in practice.

Motivated by the limitation, we study how *user interaction* would help to learn the user's expected tuple. Formally, we propose a problem called *Interactive Mining with Ordered and Unordered Attributes (Problem IOU)*, which learns the user's expected tuple with the help of user interaction and finds the user's favorite tuple (i.e., the tuple with the smallest distance to the learned expected tuple). Specifically, we interact with a user by asking questions. Following [23, 32, 35], each question presents two tuples and asks the user to pick the one s/he prefers. The presented tuples are selected based on the user's answers to the previous questions. According to the user feedback, the user's expected tuple is implicitly learned, and the user's favorite tuple is returned.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551810

Problem IOU involves many questions asked to a user. In literature, there are two types of questions that are widely used: the *attribute-level question* and the *tuple-level question*. The former type displays part of the attributes of tuples in each question. For instance, it may present the price or the number of seats of several cars in one question (instead of all the attributes together). This could help users focus on a few attributes centrally. However, since a user only sees a subspace of attributes instead of the entire space, it is easy to miss the connection between attributes. For example, consider the scenario where Alice purchases a car. The price that Alice accepts is closely related to several attributes, e.g., the horsepower, the size, and the number of seats. It is necessary to learn the trade-off between price and the other attributes. If we only present part of the attributes, e.g., the price and the horsepower, we cannot discover the combined effect of all the other attributes on price and thus, inaccurately predict user preference. Moreover, the attribute-level question suffers the *untruthful issue* [20, 35]. Users may be disappointed since they cannot see the full picture of tuples.

Therefore, we utilize the tuple-level question that displays tuples (i.e., all the attributes) to users [13, 23, 32]. Note that we only present *two* tuples in each question since a small number can make the selection of the presented tuples more targeted. If the required number of presented tuples increases, there might be tuples that are not so qualified to be presented. In cognitive psychology, Thurstone's Law of Comparative Judgement indicates that pairwise comparison is a more effective way to learn the user preference than other methods [1, 23]. The user study in [23] also verified that pairwise comparison could effectively capture how real users assess multi-attributes tuples. Although presenting multiple tuples may lower the number of questions asked since it could learn more information of the user preference in each question, it does not reduce user effort essentially because picking the favorite one among multiple tuples is equal to conducting the pairwise comparison several times. Besides, showing two tuples prevents users from considering multiple tuples simultaneously and thus, makes the questions easier to be answered. This type of question naturally appears in our daily life. An agent gives Alice two houses to select. A seller presents two red wines and asks Alice: which wine more suits your taste?

In the literature of the marketing research [17, 25], it is pivotal to keep the questions tally low. Otherwise, users may lose the plot and become excessively disappointed, affecting the interaction results. In this paper, we follow the relative skyline query to find the nearest tuple instead of the kNN that finds k nearest tuples, since the latter case requires asking too many questions. Intuitively, the latter case needs to collect more information about the user's expected tuple to distinguish its k (instead of 1) nearest points. The experimental results of [32] verified that recommending multiple tuples (e.g., 20 tuples) requires obtaining more information from users. It asked 4-10 times more questions than recommending one tuple. The user study of [32] also showed that users are willing to answer fewer questions rather than obtaining more recommended tuples. Therefore, we reduce the number of questions asked by just focusing on returning the user's favorite tuple, which is sufficient in many applications, to strike a balance between the user effort and the output size. Consider a scenario where Alice goes on a business trip and plans to rent a car for a week. Since it is a short-term need, it is not necessary for her to spend a lot of effort cherry-picking. She could be frustrated due to the long selection process even if finally, several candidate cars are returned. Note that our proposed algorithms can be easily extended to returning k nearest tuples. Due to the lack of space, the extension is presented in the technical report [31].

To the best of our knowledge, we are the first to study problem IOU. There are some closely related studies [13, 20, 32, 35] involving user interaction, but they are distinct to ours by only covering ordered attributes. Our problem IOU takes both ordered and unordered attributes into account. In this sense, it can be seen as a more general case of existing studies. For the existing studies, they are hard to be adapted to solve our problem IOU satisfactorily. They either ask many questions or execute for a long time, which is quite troublesome. For example, as shown in Section 6, on the dataset with 4 ordered attributes and 6 unordered attributes, the existing algorithms either ask 2-10 times more questions than our algorithms ([13, 20, 32]) or execute 10-20 times longer ([13, 35]).

Contributions. Our contributions are described as follows.

- To the best of our knowledge, we are the first to propose the problem of finding the user's favorite tuple with ordered and unordered attributes by interacting with the user.
- We show a lower bound $\Omega(\log_2 n)$ on the number of questions asked, where n is the dataset size.
- We propose algorithm *DI* for the special case of IOU, where the dataset is described by one ordered attribute and one unordered attribute. *DI* asks $O(\log_2 n)$ questions, which is asymptotically optimal w.r.t. the number of questions asked.
- We propose two algorithms *BS* and *EDI* for the general case of IOU, where the dataset can be described by an arbitrary number of ordered attributes and unordered attributes. Algorithms *BS* and *EDI* have provable guarantees on the number of questions asked and perform well empirically.
- We conducted experiments to demonstrate the superiority of our algorithms. Under typical settings, our algorithms can ask up to 10 times fewer questions and spend several orders of magnitude less time than existing algorithms.

We discuss the related work in Section 2. The formal problem definition and relevant preliminaries are shown in Section 3. Section 4 describes algorithm *DI* for the special case of IOU. Section 5 presents algorithms *BS* and *EDI* for the general case of IOU. Experiments are shown in Section 6. Section 7 concludes our paper.

2 RELATED WORK

There are various queries proposed to assist the multi-criteria decision-making. The *preference-based queries* return tuples based on the expected tuple given by a user and the *interactive queries* involve user interaction during the query processing.

Preference-based Queries. The skyline query returns all tuples that are not dominated by other tuples [5, 9]. A tuple p *is dominated by* another tuple q if p is not better than q in each attribute and strictly worse in at least one attribute. The skyline query can be either absolute or relative. The absolute skyline query is based on the attribute values of tuples. If an attribute value of tuple p is smaller than that of tuple q, p is better than q in that attribute. Such formulation limits the absolute skyline query to dealing only with ordered attributes since the values of unordered attributes are

not the smaller, the better. The relative (or dynamic) skyline query can handle both ordered and unordered attributes. It measures the coordinate-wise distance between tuples and considers the user's expected tuple. Specifically, p is better than q in an attribute if p is closer to the user's expected tuple in that attribute than q. The deficiency of the skyline query is that its output size is uncontrollable. It is possible that the whole dataset is returned as the answer [21, 37].

The k nearest neighbors query (kNN) [29] avoids this problem. It uses the Euclidean distance function to measure the distance from each tuple in the dataset to the user's expected tuple, and returns the k tuples with the smallest distance to the user's expected tuple. Another relevant query is the similarity query [28]. It defines a more complicated distance function and finds tuples close to the user's expected tuple w.r.t. the distance function. However, both the kNN and the similarity query rely on the assumption that the user's expected tuple is known in advance [3]. In practice, the expected tuple is not always known by a user. Even if it is known, the user needs to spend additional effort specifying it.

Interactive Queries. The interactive queries involve user interaction [1–4, 14, 20, 27, 35, 39]. They learn the user preference by asking the user questions and return tuples based on the learned preference. [1, 2, 14] propose the interactive skyline query that tries to reduce the output size of the skyline query. Specifically, it learns the user preference on the attribute values (e.g., a user prefers *red* to *yellow* in the color attribute), and then determines whether a tuple is dominated by the other tuples. However, the output size could be still arbitrarily large even if the user preference on all attribute values is obtained, [21]. Consider two cars: a cheap car p in yellow and an expensive car q in red. Suppose a user prefers red to yellow. p does not dominate q and vice versa, since p is better than q in the price attribute and worse than q in the color attribute. Thus, although the user preference on all attribute values is known, p and q are returned in the output [21, 37].

[20] proposes the interactive regret minimizing query. It returns a fixed set of tuples with a small *regret ratio* which evaluates returned tuples and represents how regretful a user is when s/he sees the returned tuples instead of the whole dataset. However, it displays fake tuples, which are artificially constructed (not selected from the dataset), in each question to interact with a user. This might produce unrealistic tuples (e.g., a car with \$10 and 1000 seats) and the user can be disappointed if the displayed tuples with which s/he is satisfied do not exist [35]. To overcome the defect, [35] proposes the strongly truthful regret minimizing query, which displays *real tuples* (selected from the dataset) during the interaction. However, it asks users too many questions, which causes the effectiveness issue in practice. To reduce the number of questions asked, [39] changes the way of asking questions. It asks users to sort the displayed tuples based on their preferences. However, this does not reduce the user effort essentially since sorting is equivalent to picking the favorite tuple several times.

There are alternative approaches [23, 32] which also involve user interaction. [23] approximates the user preference. Nevertheless, it aims at learning the user preference rather than returning tuples, which results in asking the user many questions [35]. For example, if Alice prefers car p_1 to both p_2 and p_3, her preference between p_2 and p_3 is less interesting in our case, but this additional comparison might be useful in [23]. [32] returns one of the top-k tuples. It models user preference as a utility function. With the help of user interaction, it learns the utility function and returns one of the k tuples that have the highest function value among all tuples. However, its defined utility function cannot be applied to the dataset with both ordered and unordered attributes.

[3, 4, 27] propose the interactive similarity query. It returns tuples close to a query tuple w.r.t. a distance function, where the query tuple and the distance function are learned by interacting with a user. However, during the interaction, it requires a user to assign *relevance scores* for hundreds or thousands of tuples to learn how close the tuples are to the query tuple. From the user's perspective, requiring the user to give accurate scores a lot of times is too demanding in practice. Besides, it estimates the query tuple at the beginning of the interaction and continually modifies it during the interaction based on the relevance scores given by the user. It is challenging to initialize the query tuple which significantly affects the final output. In comparison, we ask easy questions with little user effort and do not rely on an initial query tuple.

In the literature of machine learning, the problem of *learning to rank* [10, 13, 16, 18] also involves user interaction. It learns the ranking of tuples by interacting with a user. However, most of the existing methods [10, 16, 18] only consider the relations between tuples (where a relation means that a tuple is preferable to another tuple) and neglect their inter-relations (where attribute "price" is an example of an inter-relation showing that \$200 is better than \$500 since \$200 is cheaper). Thus, they require more feedback from users [32, 35]. Algorithm *ActiveRanking* [13] considers the inter-relations between tuples and learns the ranking of tuples by interacting with the user. However, it assumes that all tuples are in the *general position* [26], which could not be applied in many cases. Besides, it focuses on deriving the order of all pairs of tuples, which requires asking many questions due to the similar reason stated for [23].

Our work focuses on returning the user's favorite tuple with the help of user interaction on the dataset described by ordered and unordered attributes. It avoids the weaknesses of existing studies. (1) We do not require an exact expected tuple provided by a user (required by the kNN) or estimate a query tuple (required by the interactive similarity query). (2) We return the user's favorite tuple (but the skyline query has an uncontrollable output size). (3) We only use real tuples during the interaction (unlike [20] which utilizes fake tuples). (4) We can handle the dataset with both ordered and unordered attributes (while the existing interactive queries cannot deal with unordered attributes). (5) We only involve a few easy questions. Firstly, existing studies ask many questions since they require learning either a total ranking [10, 16, 18] or an exact user preference [23], while we only return the user's favorite tuple. Secondly, [10, 16, 18] do not utilize the inter-relation between tuples and thus, involve some unnecessary interaction. Thirdly, compared with [3, 27, 39], our designed questions are easier to answer and more effective in collecting the information of user preference.

3 PROBLEM DEFINITION

3.1 Terminologies

We consider that tuples are represented as d-dimensional points $p = (p[1], p[2], ..., p[d])$ in a dataset D. The first d_o dimensions,

called ordered dimensions, correspond to the ordered attributes of tuples and the last d_u ($d_u = d - d_o$) dimensions, called unordered dimensions, correspond to the unordered attributes of tuples. In the rest of the paper, we use "point/tuple" and "dimension/attribute" interchangeably. For the ordered dimensions, we make the convention that the larger values, the better, yet our findings could be easily adapted to the attributes that are to be minimized. Note that we assume that all the attributes are numerical. The categorical attributes (e.g., string and text) can be mapped to numerical values using the standard SVM convention, which is widely used in the machine learning area [19, 23].

Following [7, 12, 24, 30, 33], we model the user preference in the form of a function $f(p) = (\sum_{i=1}^{d}(p[i] - e[i])^2)^{\frac{1}{2}}$, namely *distance function*, denoted by $f(p) = ||p - e||$ for simplicity. Point e, called *expected point*, represents the user's expected tuple. For each ordered dimension $i \in [1, d_o]$, since the larger value the better, we assume that $e[i] = \max_{p \in D} p[i]$. For each unordered dimension $j \in [d_o + 1, d]$, since there does not exist a trivial order on its values, we assume that $e[j] \in [\min_{p \in D} p[j], \max_{p \in D} p[j]]$. The domain of e is called the *expected space*, denoted by $\mathcal{E}$, which is a *hyper-rectangle* [8] in a d-dimensional geometric space. For example, when $d_o = 1$ and $d_u = 1$, as shown in Figure 1, the expected space is a line segment (represented as a bold vertical line segment). $f(p)$ denotes the *distance* between p and e. It represents how much a user favors point p. A smaller distance means that the point is more preferred by the user. Given an expected point $e \in \mathcal{E}$, a point p is the *nearest point* of e among D if $p = \arg\min_{q \in D} f(p)$. We also call point p the user's *favorite point* in the whole dataset.

One may notice that the importance of different dimensions to a user may vary [36, 38] (i.e., different attributes may have different priorities) and thus, each dimension contributes to the distance variously. To involve this potential indicator, we could learn the importance of dimensions with the help of existing methods and scale up or down each dimension accordingly [6, 15, 23, 35]. The more important dimension is scaled up more so that it could contribute more to the distance. In this paper, we do not focus on the way to learn the importance. For the ease of illustration, we assume that all the dimensions are equally important and they are normalized to $[0, 1]$.

Example 3.1. Consider Table 1. Assume that $d_o = 1$ and $d_u = 1$. Let $f(p) = ((p[1] - 1)^2 + (p[2] - 0.5)^2)^{1/2}$ (i.e., $e = (1, 0.5)$). The distance from p_2 to e is $f(p_2) = ((0.8 - 1)^2 + (0.4 - 0.5)^2)^{1/2} = 0.22$. The distance of other points to e can be computed similarly. Since $f(p_2)$ is the smallest, p_2 is the user's favorite point.

3.2 Problem IOU

Our interactive framework follows [32, 35] and works on the dataset with ordered and unordered dimensions. Specifically, we interact with a user for rounds until we can find the user's favorite point. In each round, we process as follows. (1) (**Point selection**) Based on the user's answers to the previous questions, we present two points to the user and ask him/her to pick the one s/he prefers. The points are selected carefully, hoping to collect as much information about the user preference as possible. (2) (**Information maintenance**) According to the user feedback, we update the maintained information for learning the user's expected point. (3) (**Stopping condition**) If the stopping condition is satisfied, we terminate the interaction and return the result. Otherwise, we start another interactive round. Formally, we are interested in the following problem. Due to the lack of space, the proofs of some theorems/lemmas in this paper can be found in the technical report [31].

Problem 1. *(**Interactive Mining with Ordered and Unordered Attributes (IOU)**) Given a point set D that is described by ordered dimensions and unordered dimensions, we are to ask a user as few questions as possible to determine the user's favorite point in D.*

Theorem 3.2. *For any dimensionality d, there is a dataset of n d-dimensional points such that any algorithm needs to ask $\Omega(\log n)$ questions to determine the user's favorite point.*

Proof Sketch. Consider a dataset D such that each $p \in D$ could be the user's favorite point. Any algorithm that utilizes pairwise comparison must identify points in the form of a binary tree. Each leaf corresponds to a point and each internal node corresponds to a question asked to a user. Since there are n leaves, the height of the tree is $\Omega(\log_2 n)$. Thus, any algorithm needs to ask $\Omega(\log_2 n)$ questions to determine the user's favorite point. □

3.3 Problem Characteristics

In a d-dimensional geometric space $\mathbb{R}^d$, for any pair of points $p, q \in D$, we could build a *hyper-plane* (also called *bisect*) $h_{p,q} : (e - \frac{p+q}{2}) \cdot (p - q) = 0$, which passes through the middle point of p and q with its unit norm in the same direction as $p - q$ [8]. $h_{p,q}$ divides $\mathbb{R}^d$ into two half-spaces. The half-space above (resp. below) $h_{p,q}$, denoted by $h^+_{p,q}$ (resp. $h^-_{p,q}$), contains all the expected points e such that $(e - \frac{p+q}{2}) \cdot (p - q) > 0$, i.e., $f(p) < f(q)$ (resp. $(e - \frac{p+q}{2}) \cdot (p - q) < 0$, i.e., $f(p) > f(q)$). In geometry, a *polyhedron* $\mathcal{P}$ is the intersection of a set of half-spaces. The hyper-planes that bound $\mathcal{P}$ are called the boundaries of $\mathcal{P}$. The corner points in $\mathcal{P}$ are called the extreme points of $\mathcal{P}$. The following lemmas give our intuition of learning the user's expected point and determining the user's favorite point.

Lemma 3.3. *Given $\mathcal{E}$ and two points p and q presented to a user, if the user prefers p to q, the user's expected point must be in $h^+_{p,q} \cap \mathcal{E}$.*

Proof. If a user prefers p to q, p must be closer to the user's expected point e than q. We have $f(p) < f(q)$, i.e., $(e - \frac{p+q}{2}) \cdot (p - q) > 0$, which implies that $e \in h^+_{p,q}$ or $e \in h^-_{q,p}$. □

Based on Lemma 3.3, we could narrow down the range in which the user's expected point is located. Let us denote the range by $\mathcal{R}$, which is an intersection of a set of half-spaces $h^+_{p,q}$ (or $h^-_{q,p}$) and $\mathcal{E}$. Based on $\mathcal{R}$, some points in D can be determined not to be the user's favorite point and put out of consideration.

Lemma 3.4. *Given $\mathcal{R}$, point p can be put out of consideration, if $\forall e \in \mathcal{R}$, $\exists q \in D$ such that $||p - e|| > ||q - e||$, i.e., $f(p) > f(q)$.*

Intuitively, the points that cannot be the nearest point of any $e \in \mathcal{R}$ are left out of account. The verification of a point p satisfying Lemma 3.4 can be achieved by the Linear Programming (LP) algorithm. We define a variable x and set the objective function to be $\max x$. For each $q \in D \setminus \{p\}$, we build a constraint $(e - \frac{p+q}{2}) \cdot (p - q) > x$ (which is equal to $||p - e|| + x < ||q - e||$), where $e \in \mathcal{R}$. If the result $x < 0$, it implies that $\forall e \in \mathcal{R}$, $\exists q \in D \setminus \{p\}$

Table 1: Dataset ($e = (1, 0.5)$)

p	$p[1]$	$p[2]$	$f(p)$
p_1	1	0	0.50
p_2	0.8	0.4	0.22
p_3	0.6	0.6	0.41
p_4	0.7	0.8	0.42
p_5	0.2	1	0.94

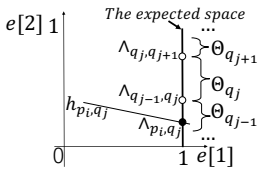


Figure 1: Scan Case 1

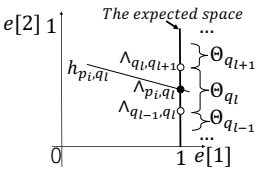


Figure 2: Scan Case 2

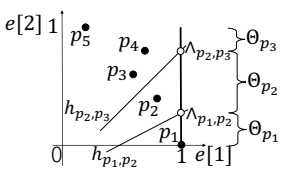


Figure 3: Division of S_3

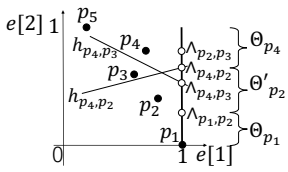


Figure 4: Division of S_4

such that $||p - e|| > ||q - e||$. Due to the lack of space, the detailed LP formulation is shown in the technical report [31]. Since there are n points in space $\mathbb{R}^d$, there are $O(n)$ constraints and $O(d)$ variables in LP. An LP solver (e.g., Simplex [5]) needs $O(dn^2)$ time in practice. It might be time-consuming to proceed an LP calculation if there are many points. The following lemma gives a sufficient condition to reduce the number of points requiring an LP calculation.

Lemma 3.5. *Given $\mathcal{R}$, point p can be put out of consideration, if $\exists q \in D$ such that $\forall e \in \mathcal{R}$, $||p - e|| > ||q - e||$, i.e., $f(p) > f(q)$.*

Lemma 3.5 compares p with each point $q \in D \backslash \{p\}$ to see whether q is closer to any $e \in \mathcal{R}$ than p. Suppose there are v extreme points e_v in $\mathcal{R}$. Each comparison takes $O(\mathrm{v})$ time to check whether all the extreme points satisfy $||p - e_v|| > ||q - e_v||$.

Note that Lemma 3.4 may find a set of points. Each point is closer to some $e \in \mathcal{R}$ than p. Nevertheless, Lemma 3.5 searches for only one point that is closer to all $e \in \mathcal{R}$ than p.

4 SPECIAL CASE OF IOU

We begin with a special case of IOU. Each point has one ordered and one unordered dimensions. We propose algorithm *DI* that is asymptotically optimal in terms of the number of questions asked.

Without loss of generality, assume that the first dimension is ordered and the second dimension is unordered. In a 2-dimensional geometric space $\mathbb{R}^2$, as shown in Figure 1, the expected space is a vertical line segment, where $e[1] = \max_{p \in D} p[1]$ and $e[2] \in [\min_{p \in D} p[2], \max_{p \in D} p[2]]$. Our algorithm *DI* consists of two parts: dividing the expected space and interacting with a user. Intuitively, we first divide the expected space into several disjoint smaller line segments, called partitions and denoted by Θ. Each partition Θ corresponds to a point in D, which is the nearest point of any $e \in \Theta$. Then, we interact with a user to locate the partition that contains the user's expected point. The point that corresponds to the located partition is returned to the user as the answer. The pseudocode of algorithm *DI* is shown in Algorithm 1.

4.1 Dividing the Expected Space

Since the fewer partitions, the easier to locate which partition the user's expected point is in, our method divides the expected space into the fewest partitions. At a high-level, we build the division of the expected space $\mathcal{E}$ progressively. Every set of points $S \subseteq D$ decides a division of $\mathcal{E}$. Our idea is to put the points into consideration one by one and then update the division of $\mathcal{E}$.

In space $\mathbb{R}^2$, as discussed in Section 3.3, for any pair $p, q \in D$, we can build a hyper-plane $h_{p,q}$ (i.e., a line in $\mathbb{R}^2$). If $h_{p,q}$ intersects $\mathcal{E}$, we denote the intersection by $\wedge_{p,q}$. The line segment connecting any two intersections $\wedge_1$ and $\wedge_2$ is represented by $[\wedge_1, \wedge_2]$. For example, consider Figure 1. The black dot is the intersection $\wedge_{p_i,q_j}$ of h_{p_i,q_j} and $\mathcal{E}$. The line segment that connects $\wedge_{q_{j-1},q_j}$ and $\wedge_{q_j,q_{j+1}}$ (the two white dots) is denoted by $[\wedge_{q_{j-1},q_j}, \wedge_{q_j,q_{j+1}}]$.

We sort all the points based on their second dimension in non-decreasing order. Let $< p_1, p_2, ..., p_n >$ denote the sorted list. Every set of points $S_{i-1} =< p_1, p_2, ..., p_{i-1} >$ decides a division of $\mathcal{E}$, where $i \in [2, n]$. Note that not every point in S_{i-1} necessarily corresponds to a partition in the division. Assume that $< q_1, q_2, ..., q_k >$ are the points in S_{i-1} that correspond to partitions $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_k} >$ from bottom to top. Point q_j is the nearest point of any $e \in \Theta_{q_j}$ among S_{i-1}, where $j \in [1, k]$ and $\Theta_{q_j} = [\wedge_{q_{j-1},q_j}, \wedge_{q_j,q_{j+1}}]$ ($\wedge_{q_0,q_1} = (1, 0)$ and $\wedge_{q_k,q_{k+1}} = (1, 1)$). We add p_i and update the division of $\mathcal{E}$ by scanning points from q_k to q_1 until we reach a point q_l in $< q_1, q_2, ..., q_k >$ such that $\wedge_{p_i,q_l}$ is above $\wedge_{q_{l-1},q_l}$ as shown in Figure 2.

Lemma 4.1. *The nearest point of any $e \in [\wedge_{q_0,q_1}, \wedge_{p_i,q_l}]$ among $S_i = S_{i-1} \cup \{p_i\}$ is the same as that among S_{i-1}. Point p_i is nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{q_k,q_{k+1}}])$ among S_i.*

Proof Sketch. For the ease of illustration, let $\wedge_1 \prec \wedge_2$ represent that $\wedge_1$ is below $\wedge_2$. According to our scanning strategy, $\forall j \in [l + 1, k]$, $\wedge_{p_i,q_j} \prec \wedge_{q_{j-1},q_j}$. We have $\Theta_{q_j} \subseteq h^+_{p_i,q_j}$ as shown in Figure 1. Based on the definition of $h^+_{p_i,q_j}$, p_i must be the nearest point of any $e \in \bigcup_{j=l+1}^{k} \Theta_j$ (i.e., any $e \in [\wedge_{q_l,q_{l+1}}, \wedge_{q_k,q_{k+1}}]$) among S_i.

The above conclusion indicates that $\wedge_{p_i,q_l} \prec \wedge_{q_l,q_{l+1}}$. Moreover, since $\wedge_{q_{l-1},q_l} \prec \wedge_{p_i,q_l}$, we have $\wedge_{p_i,q_l} \in [\wedge_{q_{l-1},q_l}, \wedge_{q_l,q_{l+1}}]$ as shown in Figure 2. Since $[\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}] \subseteq h^+_{p_i,q_l}$, p_i must be the nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}]$ among S_i. Because $[\wedge_{q_0,q_1}, \wedge_{p_i,q_l}] \subseteq h^-_{p_i,q_l}$, point q_l is closer to any $e \in [\wedge_{q_0,q_1}, \wedge_{p_i,q_l}]$ than p_i. The nearest point of any $e \in [\wedge_{q_0,q_1}, \wedge_{p_i,q_l}]$ among S_i is the same as that among S_{i-1}. □

Based on Lemma 4.1, when we reach point q_l, the division of the expected space $\mathcal{E}$ is updated as follows. (1) $< q_1, q_2, ..., q_l, q_{l+1}, ..., q_k >$ is updated to be $< q_1, q_2, ..., q_l, p_i >$. (2) $< \Theta_{q_1}, ..., \Theta_{q_{l-1}}, \Theta_{q_l}, ..., \Theta_{q_k} >$ is updated to be $< \Theta_{q_1}, ..., \Theta_{q_{l-1}}, \Theta'_{q_l}, \Theta_{p_i} >$, where $\Theta'_{q_l} = [\wedge_{q_{l-1},q_l}, \wedge_{p_i,q_l}]$ and $\Theta_{p_i} = [\wedge_{p_i,q_l}, \wedge_{q_k,q_{k+1}}]$.

Example 4.2. Assume that $S_3 =< p_1, p_2, p_3 >$ decides a division of the expected space $< \Theta_{p_1}, \Theta_{p_2}, \Theta_{p_3} >$ as shown in Figure 3. Let us add point p_4 by scanning points from p_3 to p_1 in Figure 4. Since $\wedge_{p_4,p_3}$ is below $\wedge_{p_2,p_3}$ and $\wedge_{p_4,p_2}$ is above $\wedge_{p_1,p_2}$, we update the division as follows. (1) $< p_1, p_2, p_3 >$ is updated to be $< p_1, p_2, p_4 >$; (2) $< \Theta_{p_1}, \Theta_{p_2}, \Theta_{p_3} >$ is updated to be $< \Theta_{p_1}, \Theta'_{p_2}, \Theta_{p_4} >$, where $\Theta'_{p_2} = [\wedge_{p_1,p_2}, \wedge_{p_4,p_2}]$ and $\Theta_{p_4} = [\wedge_{p_4,p_2}, (1, 1)]$.

Theorem 4.3. *The expected space can be divided into the fewest partitions in $O(n \log n)$ time.*

Proof Sketch. Suppose there are m partitions in the optimal case (i.e., the fewest partitions case). Use $\Theta'_i = [\wedge'_{q_{i-1},q_i}, \wedge'_{q_i,q_{i+1}}]$ and $\Theta_i = [\wedge_{q_{i-1},q_i}, \wedge_{q_i,q_{i+1}}]$ to denote the i-th partition of the optimal case and the i-th partition obtained by our algorithm, respectively ($i \in [1, m]$). Let $\wedge_{q_i,q_{i+1}} \geq \wedge'_{q_i,q_{i+1}}$ represents that $\wedge_{q_i,q_{i+1}}$ is not below $\wedge'_{q_i,q_{i+1}}$. We prove that $\forall i \in [1, m]$, $\wedge_{q_i,q_{i+1}} \geq \wedge'_{q_i,q_{i+1}}$ with the help of mathematical induction. Since $\wedge_{q_m,q_{m+1}} \geq \wedge'_{q_m,q_{m+1}} = (1, 1)$, the number of partitions obtained by our algorithm will not be more than that of the optimal case. As for the time complexity, we need $O(n \log n)$ time to sort all the points. To update the division of the expected space, we conclude that each point needs $O(1)$ time. Since there are n points, we require $O(n)$ time. Thus, the total time complexity is $O(n \log n)$. □

4.2 Interacting with A User

Section 4.1 obtains a division of $\mathcal{E}$ decided by D. Denote the set of partitions by $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_m} >$ from bottom to top with their corresponding points $< q_1, q_2, ..., q_m >$. For any pair of points q_j and q_{j+1}, where $j \in [1, m-1]$, hyper-plane $h_{q_j,q_{j+1}}$ intersects the expected space at $\wedge_{q_j,q_{j+1}}$ and separates the partitions into two sets $\mathcal{S}_1 =< \Theta_{q_1}, ..., \Theta_{q_j} >$ and $\mathcal{S}_2 =< \Theta_{q_{j+1}}, ..., \Theta_{q_m} >$. $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) contains all the partitions such that for any expected point e in the partition, $||q_j - e|| < ||q_{j+1} - e||$ (resp. $||q_j - e|| > ||q_{j+1} - e||$). If a user prefers q_j to q_{j+1} (resp. q_{j+1} to q_j), the user's expected point must be located in the partitions in $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) and the user's favorite point must be in $< q_1, q_2, ..., q_j >$ (resp. $< q_{j+1}, q_{j+2}, ..., q_m >$).

Our algorithm maintains a point set C initialized to be $< q_1, q_2, ..., q_m >$ and interacts with a user for rounds to find the user's favorite point in a binary search manner. Specifically, in each round, our algorithm asks a question by presenting the user with the middle points q_j and q_{j+1} in C. If a user prefers q_j to q_{j+1}, C is updated to be the first half of C. Otherwise, C is updated to be the remaining half of C. The process continues until $|C| = 1$ and the point finally left in C is returned to the user as the answer.

Example 4.4. Following Example 4.2, point set C is initialized to be $< p_1, p_2, p_4 >$. We present a user with the middle points p_1 and p_2. If the user prefers p_1 to p_2, C is updated to be $< p_1 >$. Since $|C| = 1$, the interaction process stops and point p_1 is returned.

Theorem 4.5. *Algorithm* DI *determines the user's favorite point by asking the user $O(\log n)$ questions.*

Proof Sketch. We prove that candidate set C could be initialized to contain n points in the worst case. Since we reduce C by half in each round, $|C|$ can be reduced to 1 in $O(\log n)$ rounds. □

Corollary 4.6. *Algorithm* DI *is asymptotically optimal in terms of the number of questions asked.*

5 GENERAL CASE OF IOU

We are ready to describe our algorithms *BS* and *EDI* for the general case of IOU. In the following, we show how we address each of the three components of the interactive framework in the algorithms.

5.1 Algorithm BS

In this section, we present algorithm *BS* that performs the best in the experiments w.r.t. the number of questions asked. Intuitively, we intend to follow the idea in Section 4. To divide the expected space, a simple approach is to compute the Voronoi Diagram [8]. For each $p \in D$, its Voronoi cell c_p contains all the points in space $\mathbb{R}^d$ such that p is their nearest point. If c_p intersects with the expected space $\mathcal{E}$, p must be the nearest point of any $e \in c_p \cap \mathcal{E}$ and thus, $c_p \cap \mathcal{E}$ must be the partition that corresponds to p. However, the computation of (1) the Voronoi Diagram and (2) the intersection of c_p and $\mathcal{E}$ are costly. Besides, the partitions are not simply located linearly when $d_u \geq 2$. To interact with a user, although we can still find as a question the two points whose corresponding partitions are neighboring, it would be difficult to conduct in a binary search manner. Based on these challenges, we propose algorithm *BS* (Binary Search) which efficiently divides the expected space and selects points as questions following the idea of binary search.

Algorithm 1: Algorithm *DI*

Input: A point set D
Output: The user's favorite point

```
Sort all points based on their second dimension
Θ ←< Θ_{p_1} >, C ←< p_1 >
for i ← 2 to n do
    for j ← k to 1 do
        if ∧_{p_i,q_j} is above ∧_{q_{j-1},q_j} then
            l ← j, C ←< q_1, q_2, ..., q_l, p_i >
            Θ ←< Θ_{q_1}, ..., Θ_{q_{l-1}}, Θ'_{q_l}, Θ_{p_i} >
            break
left ← 1, right ← m
while |C| > 1 do
    Present the middle points q_j and q_{j+1} in C to the user
    if q_j is preferable to q_{j+1} then
        right ← j
    else
        left ← j + 1
    C ←< q_left, ..., q_right >
return The point finally left in C
```

5.1.1 Information Maintenance & Stopping Condition. Algorithm *BS* maintains 3 data structures (1) a polyhedron $\mathcal{R} \subseteq \mathcal{E}$ that contains the user's expected point; (2) a point set C that stores the user's favorite point; and (3) a hyper-plane set $\mathcal{H}$ used for point selection. Initially, $\mathcal{R}$ is set to be the whole expected space $\mathcal{E}$. C contains all the points in D, each of which is the nearest point of at least one expected point in $\mathcal{E}$. $\mathcal{H}$ is constructed based on C (which will be discussed later). Then, we interact with a user for rounds. In each round, we select a hyper-plane $h_{p,q} \in \mathcal{H}$ and present the points p and q to the user as a question (shown in Section 5.1.2). Based on the user feedback, we update $\mathcal{R}$ to be $\mathcal{R} \cap h^+_{p,q}$ or $\mathcal{R} \cap h^-_{p,q}$, and delete the points in C that cannot be the user's favorite point (shown in Section 3.3). The interaction process stops when $|C| = 1$. The point finally left in C is returned to the user as the answer.

A naive idea to initialize C is to find the nearest point of each $e \in \mathcal{E}$. However, it is unachievable in practice due to the infinite expected points. Thus, we try to divide $\mathcal{E}$ and utilize the connection

between points in D. Consider a point $p \in D$. Let $\mathcal{E}_p$ denote the *maximal* polyhedron in $\mathcal{E}$ such that p is the nearest point of any $e \in \mathcal{E}_p$. The maximum means that there does not exist a polyhedron $\mathcal{E}'_p \subseteq \mathcal{E}$, where $\mathcal{E}_p \subset \mathcal{E}'_p$ and p is the nearest point of any $e \in \mathcal{E}'_p$. Note that there is at most one $\mathcal{E}_p \subseteq \mathcal{E}$ for each point $p \in D$ and it is possible that $\mathcal{E}_p = \emptyset$, i.e., p is not the nearest point of any $e \in \mathcal{E}$.

Lemma 5.1. *There does not exist two maximal polyhedrons $\mathcal{E}'_p, \mathcal{E}''_p \subseteq \mathcal{E}$ such that (1) $\mathcal{E}'_p \cap \mathcal{E}''_p = \emptyset$ and (2) p is the nearest point of any $e \in \mathcal{E}'_p \cup \mathcal{E}''_p$.*

Proof Sketch. The polyhedron $\mathcal{E}'_p$ or $\mathcal{E}''_p$ must be the intersection of half-spaces, i.e., $\bigcap_{q \in D \setminus \{p\}} h^+_{p,q}$. In geometry, the intersection of half-spaces is a polyhedron or an empty space [8]. □

Lemma 5.2. *If $\mathcal{E}_p \neq \emptyset$ and $h_{p,q}$ is a boundary of $\mathcal{E}_p$, q must be the nearest point of at least one expected point in $\mathcal{E}$, i.e., $\mathcal{E}_q \neq \emptyset$.*

Proof. Since $\mathcal{E}_p \neq \emptyset$ and $h_{p,q}$ is one of the boundaries of $\mathcal{E}_p$, $h_{p,q} \cap \mathcal{E}_p \neq \emptyset$. Let $e_{p,q}$ be an expected point in $h_{p,q} \cap \mathcal{E}_p$. Since p is the nearest point of any $e \in \mathcal{E}_p$, p must be the nearest point of $e_{p,q}$. According to the definition of $h_{p,q}$, p and q have the same distance to any $e \in h_{p,q}$. Thus, q must be the nearest point of $e_{p,q}$. □

Our method initializes C as well as $\mathcal{H}$ based on Lemma 5.2. Intuitively, when finding a qualified point p, we can know some other qualified points q based on the boundaries of $\mathcal{E}_p$. Then we can also find others based on the boundaries of $\mathcal{E}_q$. Specifically, we maintain a queue Q storing the points that will be inserted into C. In the beginning, we randomly select an expected point $e \in \mathcal{E}$ and put the nearest point of e into Q. Then, we continually pop out points in Q until Q is empty. For each popped out point p, we insert it into C and find the maximal polyhedron $\mathcal{E}_p \subseteq \mathcal{E}$ such that p is the nearest point of any $e \in \mathcal{E}_p$. For each boundary $h_{p,q}$ of $\mathcal{E}_p$, $h_{p,q}$ is inserted into $\mathcal{H}$ if $h_{p,q} \notin \mathcal{H}$ and point q is put into Q if $q \notin C \cup Q$.

We find the polyhedron $\mathcal{E}_p \subseteq \mathcal{E}$ with the help of the Linear Programming algorithm (LP). The techniques are the same as the LP formulation for Lemma 3.4 except that $\mathcal{R}$ is set to be the whole expected space, i.e., $\mathcal{R} = \mathcal{E}$. If constraint $(e - \frac{p+q}{2}) \cdot (p - q) > x$ bounds the feasible region of the LP, $h_{p,q}$ is a boundary of $\mathcal{E}_p$.

Example 5.3. Consider Table 1. Assume that each point is described by two unordered dimensions. Suppose that p_3 is popped out from Q. We insert p_3 into C and find the maximal polyhedron $\mathcal{E}_{p_3}$ which is a shaded area shown in Figure 5. Since hyper-planes h_{p_2,p_3}, h_{p_3,p_4} and h_{p_3,p_5} are the boundaries of $\mathcal{E}_{p_3}$, we insert h_{p_2,p_3}, h_{p_3,p_4} and h_{p_3,p_5} into $\mathcal{H}$, and put p_2, p_4 and p_5 into Q.

5.1.2 Point Selection. Our strategy is to select hyper-planes in $\mathcal{H}$ that can reduce the size of C rapidly. In a high-level, let $U = \{\mathcal{R}_p | p \in C\}$, where $\mathcal{R}_p \subseteq \mathcal{R}$ is the *maximal* polyhedron such that p is the nearest point of any $e \in \mathcal{R}_p$. Similarly, the maximum means that there does not exist a polyhedron $\mathcal{R}'_p \subseteq \mathcal{R}$, where $\mathcal{R}_p \subset \mathcal{R}'_p$ and p is the nearest point of any $e \in \mathcal{R}'_p$. Assume that we could find a hyper-plane $h_{p,q} \in \mathcal{H}$ that divides U into two equal subsets, i.e., $h^+_{p,q}$ and $h^-_{p,q}$ contain half of the polyhedrons in U, respectively. Based on the user preference on p and q, $\mathcal{R}$ is updated to be $\mathcal{R} \cap h^+_{p,q}$ or $\mathcal{R} \cap h^-_{p,q}$. Then, half of the polyhedrons in U that are not in the updated $\mathcal{R}$ can be put out of consideration and thus, half of the points p in C that correspond to the neglected polyhedrons $\mathcal{R}_p$ can be pruned.

Algorithm 2: Algorithm *BS*

Input: A point set D
Output: The user's favorite point

```
R ← E, C = {p ∈ D | ∃e ∈ E, p is the nearest point of e}
H contains all the boundaries of all E_p
while |C| > 1 do
    Select hyper-plane h_{p,q} ∈ H with the highest priority
    Present points p and q to the user
    Update R and C based on the user's feedback
return The point finally left in C
```

Following this idea, we select the hyper-plane $h_{p,q} \in \mathcal{H}$ that divides the most equally the set of polyhedrons $\mathcal{R}_p$ in U, and present point p and q to the user as a question. However, it is time-consuming to check the geometric relation between a polyhedron and a hyper-plane. We present a heuristic approach for relation checking that performs well empirically in the following.

Consider a point $p \in C$ and its polyhedron $\mathcal{R}_p$. Denote by r_p the middle point of all the extreme points of $\mathcal{R}_p$. Let us first estimate the distance from r_p to the boundary of $\mathcal{R}_p$. Before the user provides any information, $\mathcal{R} = \mathcal{E}$ and C is initialized. Suppose that all the points in C are uniformly distributed, i.e., the polyhedron of each point in C can be approximated to be an equal size hyper-square. Use $\mathcal{V}(\cdot)$ to represent the volume of a polyhedron. The volume of each hyper-square is $\mathcal{V}(\mathcal{R})/|C|$. Thus, the length of each side of a hyper-square is $\ell = (\mathcal{V}(\mathcal{R})/|C|)^{\frac{1}{d_u}}$. In this way, the distance from r_p to the boundary of $\mathcal{R}_p$ can be estimated to be $d_{NN} = \ell/2$. For any hyper-plane $h \in \mathcal{H}$, let $dist(r_p, h)$ denote the distance from r_p to h. If $r_p \in h^+$ (resp. $r_p \in h^-$) and $dist(r_p, h) \geq d_{NN}$, we consider that $\mathcal{R}_p \subseteq h^+$ (resp. $\mathcal{R}_p \subseteq h^-$). Based on the relation checking approach, we define the *priority* of a hyper-plane. It evaluates how equally the hyper-plane divides the set of polyhedrons $\mathcal{R}_p$ in U.

Definition 5.4 (Priority). Given a hyper-plane h and a set U of polyhedrons $\mathcal{R}_p$, the priority of h is defined to be $\min\{N_+, N_-\} + \beta \min\{NN_+, NN_-\}$, where $\beta > 1$ is a balancing parameter.

Notation N_+ (resp. N_-) denotes the number of $\mathcal{R}_p$ such that $r_p \in h^+$ (resp. $r_p \in h^-$) and $dist(r_p, h) < d_{NN}$. Notation NN_+ (resp. NN_-) denotes the number of $\mathcal{R}_p$ such that $r_p \in h^+$ (resp. $r_p \in h^-$) and $dist(r_p, h) \geq d_{NN}$. Intuitively, a higher priority means that the hyper-plane divides the set of polyhedrons more equally. The first term $\min\{N_+, N_-\}$ considers $\mathcal{R}_p$ that may be on one side of h. The second term $\beta \min\{NN_+, NN_-\}$ gives an award for $\mathcal{R}_p$ that are in large possibility on one side of h. Note that if $\mathcal{R}_p$ interacts with h, $\mathcal{R}_p$ cannot be pruned no matter whether $\mathcal{R}$ is updated to be either $\mathcal{R} \cap h^+$ or $\mathcal{R} \cap h^-$. Thus, we expect both sides of h contain as many $\mathcal{R}_p$ as possible. In each interactive round, we present the user with points p and q, where hyper-plane $h_{p,q} \in \mathcal{H}$ has the highest priority.

Example 5.5. Figure 6 shows $U = \{\mathcal{R}_{p_1}, \mathcal{R}_{p_2}, \mathcal{R}_{p_3}, \mathcal{R}_{p_4}, \mathcal{R}_{p_5}\}$. Consider h_{p_2,p_3} and let $\beta = 2$. If the distance from the middle points of $\mathcal{R}_{p_1}, \mathcal{R}_{p_2}, \mathcal{R}_{p_3}$ and $\mathcal{R}_{p_5}$ to h_{p_2,p_3} are larger than d_{NN}, the priority of h_{p_2,p_3} is $\min\{0, 1\} + \beta \min\{2, 2\} = 4$.

5.1.3 Summary. We summarize *BS* by involving all the techniques illustrated previously. The pseudocode is presented in Algorithm 2.

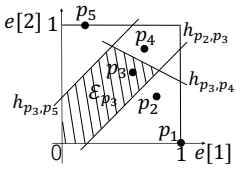


Figure 5: Polyhedron $\mathcal{E}_{p_3}$

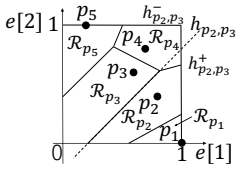


Figure 6: Polyhedrons

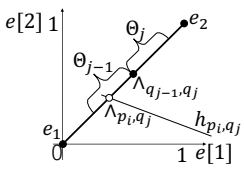


Figure 7: Case 1

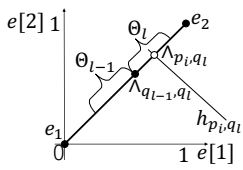


Figure 8: Case 2

At the beginning, $\mathcal{R}$, C and $\mathcal{H}$ are initialized (lines 1-2). In each interactive round, *BS* selects the hyper-plane $h_{p,q}$ in $\mathcal{H}$ that has the highest priority (line 4). Points p and q are presented as a question asked to a user (line 5). Based on the user feedback, $\mathcal{R}$ becomes a smaller polyhedron $\mathcal{R} \cap h^+_{p,q}$ or $\mathcal{R} \cap h^-_{p,q}$ and there are points pruned from C (line 6). If there is only one point p in C (line 3), we stop the interaction and return p as the answer (line 7).

Theorem 5.6. *Algorithm* BS *solves IOU in $O(n)$ rounds.*

Proof Sketch. In each interactive round, we can prune at least one point from C (one of the presented points). Since there are n points, there will be only one point left in C after $O(n)$ rounds. □

In practice, *BS* can prune multiple points from C in each round and thus, its performance is much better than $O(n)$. Suppose there are at least $\alpha\%$ points reduced in each round. *BS* needs to ask $O(\log_{\frac{1}{1-\alpha\%}} n)$ questions so that $|C| = 1$. In our experiments (Section 6), we explored that $\alpha \geq 25$ and in most cases, it is up to $40-50$.

5.2 Algorithm EDI

We present algorithm *EDI*, an extension of algorithm *DI* for the general case of IOU. Intuitively, when $d_u \geq 2$, the expected space $\mathcal{E}$ is not a line segment. Algorithm *EDI* processes several iterations. In each iteration, we find a line segment in $\mathcal{E}$ and proceed similar to *DI*. The process stops when we cannot find a qualified line segment in $\mathcal{E}$.

5.2.1 Information Maintenance & Stopping Condition. We maintain 3 data structures: (1) a polyhedron $\mathcal{R} \subseteq \mathcal{E}$ that contains the user's expected point; (2) a point set C that stores the user's favorite point; and (3) a point set $\mathcal{P}$ that stores the candidate points for point selection. Initially, $\mathcal{R} = \mathcal{E}$ and C contains all the points in D, each of which is the nearest point of at least one expected point in $\mathcal{E}$. $\mathcal{P}$ starts with some of the points in C. During the interaction, we select two points in $\mathcal{P}$ and present them to the user as a question (shown in Section 5.2.2). Based on the user feedback, $\mathcal{R}$ and C are updated accordingly (shown in Section 3.3). Meanwhile, $\mathcal{P}$ is also updated. The interaction stops when $|C| = 1$ and the point finally left in C is returned. The processing of $\mathcal{R}$ and C is in the same way as *BS*. In the following, we show how to initialize and update $\mathcal{P}$.

Initialization of $\mathcal{P}$. Recall that *DI* divides $\mathcal{E}$ into partitions and utilizes the points that correspond to the partitions for interaction. We intend to follow *DI* to find points and store them in $\mathcal{P}$ for point selection. Since $\mathcal{E}$ is not a line segment if $d_u > 1$, we are to find a line segment in $\mathcal{E}$ to proceed. Let e and e' be any two points in $\mathcal{E}$. With a slight abuse of notations, we denote the line segment that connects e and e' by $[e, e']$. We select two extreme points e_1 and e_2 of $\mathcal{R}$ ($\mathcal{R} \subseteq \mathcal{E}$) (the way of selecting extreme points is shown in Section 5.2.2) and divide line segment $[e_1, e_2]$ into the fewest partitions. Each partition Θ_p corresponds to a point $p \in C$ that is the nearest point of any $e \in \Theta_p$. Set $\mathcal{P}$ stores all these corresponding points.

Specifically, for each $p \in C$, we define the length t_p of p w.r.t. $[e_1, e_2]$ to be the projection of vector $p - e_1$ on the direction of vector $e_2 - e_1$, i.e., $t_p = \frac{(p-e_1)\cdot(e_2-e_1)}{||e_2-e_1||}$. We first sort all the points in C based on their t_p in ascending order. Let $< p_1, p_2, ..., p_{|C|} >$ represent the sorted list. Similar to *DI*, every set of points $S_{i-1} =< p_1, p_2, ..., p_{i-1} >$ decides a division of $[e_1, e_2]$, where $i = 2, 3, ..., |C|+1$. We are to add p_i and update the division of $[e_1, e_2]$. Note that not every point in S_{i-1} corresponds to a partition in the division. Assume that $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_k} >$ is the division of $[e_1, e_2]$ decided by S_{i-1} and $< q_1, q_2, ..., q_k >$ are the corresponding points. Denote by $\wedge_{p,q}$ the intersection of $[e_1, e_2]$ and hyper-plane $h_{p,q}$. The boundary of each pair of adjacent partitions Θ_{j-1} and Θ_j is $\wedge_{q_{j-1},q_j}$, where $j \in [2, k]$. We add p_i by scanning points from q_k to q_1 until we reach a point q_l in $< q_1, q_2, ..., q_k >$ such that $\wedge_{p_i,q_l}$ is farther away from e_1 than $\wedge_{q_{l-1},q_l}$. Figure 8 shows such case when $d_u = 2$.

Lemma 5.7. *The nearest point of any $e \in [e_1, \wedge_{p_i,q_l}]$ among $S_i = S_{i-1} \cup \{p_i\}$ is the same as that among S_{i-1}. Point p_i is nearest point of any $e \in [\wedge_{p_i,q_l}, e_2]$ among S_i.*

Proof Sketch. The proof is similar to that of Lemma 4.1. Based on our scanning strategy, $\forall j \in [l+1, k]$, $\wedge_{p_i,q_j}$ is closer to e_1 than $\wedge_{q_{j-1},q_j}$. Figure 7 shows such case when $d_u = 2$. p_i must be the nearest point of any $e \in \Theta_j$ among S_i. For p_l, $\wedge_{p_i,q_l}$ is farther away from e_1 than $\wedge_{q_{l-1},q_l}$. Figure 8 shows such case when $d_u = 2$. The nearest point of any $e \in [e_1, \wedge_{p_i,q_l}]$ among S_i is the same as that among S_{i-1} and p_i is nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{p_l,q_{l+1}}]$ among S_i. □

Lemma 5.7 is an extension of Lemma 4.1 in a high dimensional space. Following the lemma, when we reach point q_l, we update the division of $[e_1, e_2]$ as follows. (1) $< q_1, q_2, ..., q_l, q_{l+1}, ..., q_k >$ is updated to be $< q_1, q_2, ..., q_l, p_i >$; and (2) $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_{l-1}}, \Theta_{q_l}, ..., \Theta_{q_k} >$ is updated to be $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_{l-1}}, \Theta'_{q_l}, \Theta_{p_i} >$, where $\Theta'_{q_l} = [\wedge_{q_{l-1},q_l}, \wedge_{p_i,q_l}]$ and $\Theta_{p_i} = [\wedge_{p_i,q_l}, e_2]$.

Update of $\mathcal{P}$. Let $\mathcal{P} =< p_1, p_2, ..., p_m >$, where the points are sorted based on their corresponding partitions from e_1 to e_2. Suppose we select two nearby points $p_i, p_{i+1} \in \mathcal{P}$ as a question asked to a user. If the user prefers p_i to p_{i+1}, the user's expected point must be in $h^+_{p_i,p_{i+1}}$ according to Lemma 3.3. Since $\forall j \in [i+1, m], \Theta_{p_j} \nsubseteq h^+_{p_i,p_{i+1}}$, we delete the corresponding points $< p_{i+1}, p_{i+2}, ..., p_m >$ from $\mathcal{P}$. After the pruning, if $|\mathcal{P}| \leq \gamma$, we select two new extreme points $e_3, e_4 \in \mathcal{R}$ and partition line segment $[e_3, e_4]$ to initialize $\mathcal{P}$ with C again. As for parameter γ, if it is too large, $\mathcal{P}$ may need to be initialized many times. On the contrary, the lack of candidates for point selection may affect the performance of the algorithm. The setting of parameter γ will be discussed in Section 6.

5.2.2 Point selection. We are ready to show how to select points from $\mathcal{P}$ as questions, and pick extreme points of $\mathcal{R}$ to initialize $\mathcal{P}$.

Selecting Points from $\mathcal{P}$. Recall that we select a line segment $[e_1, e_2]$ in $\mathcal{R}$ and partition it by considering the points in C to initialize $\mathcal{P}$. Let $\mathcal{P} =< p_1, p_2, ..., p_m >$, where points are sorted based on their corresponding partitions from e_1 to e_2. Our thought is to reduce $\mathcal{P}$ by half in each interactive round. Specifically, we select the two median points p_i and p_{i+1} in $\mathcal{P}$ and present them to a user, where hyper-plane $h_{p_i,p_{i+1}}$ divides the partitions into two equal halves $< \Theta_1, ..., \Theta_i >$ and $< \Theta_{i+1}, ..., \Theta_m >$.

Picking Extreme Points of $\mathcal{R}$. Our idea is to select the pair of extreme points in $\mathcal{R}$ so that it could help to prune as many points in C as possible in each interactive round. In detail, we randomly select a set $\mathcal{S}$ of extreme points of $\mathcal{R}$. For each pair $e_1, e_2 \in \mathcal{S}$, we divide $[e_1, e_2]$ into partitions by considering C. Each partition Θ corresponds to a point in C that is the nearest point of any $e \in \Theta$ among C. Denote the corresponding points of the partitions by $\mathcal{P}_{e_1,e_2} =< p_1, p_2, ..., p_m >$. We find the median points p_i and p_{i+1} of $\mathcal{P}_{e_1,e_2}$ and check the priority of hyper-plane $h_{p_i,p_{i+1}}$ (defined in Definition 5.4). If the priority is the highest, the pair e_1 and e_2 is picked to initialize $\mathcal{P}$. Note that if $|\mathcal{P}_{e_1,e_2}| = 1$ for all pairs $e_1, e_2 \in \mathcal{S}$, we randomly select a new set $\mathcal{S}$ of extreme points of $\mathcal{R}$. The following lemma guarantees the soundness of our strategy used to pick the extreme points to initialize $\mathcal{P}$. Before reaching the stopping condition, we can always find points in $\mathcal{P}$ as a question asked to a user.

Lemma 5.8. *If $|C| \geq 2$, we could initialize $\mathcal{P}$ such that $|\mathcal{P}| \geq 2$.*

Proof Sketch. If $|C| \geq 2$, we show that there exists a pair of extreme points $e_1, e_2 \in \mathcal{R}$ such that line segment $[e_1, e_2]$ will be divided into at least two partitions. Thus, there are at least two points inserted into $\mathcal{P}$ that correspond to the partitions. □

5.2.3 Summary. The pseudocode of *EDI* is shown in Algorithm 3. $\mathcal{R}$, C and $\mathcal{P}$ are first initialized (lines 1-2). In each interactive round, *EDI* selects the two median points in $\mathcal{P}$ as a question asked to a user (line 4). Based on the user feedback, $\mathcal{R}$ and C become smaller. $\mathcal{P}$ is also reduced by half (line 5). If $|\mathcal{P}| \leq \gamma$ and the interaction does not achieve the stopping condition (i.e., $|C| > 1$), $\mathcal{P}$ is initialized again with $|\mathcal{P}| \geq 2$ based on Lemma 5.8 (lines 6-7). If $|C| = 1$, we stop the interaction (line 3) and return the point finally left in C (line 8). Theorem 5.9 shows the theoretical analysis. c is the number of times initializing $\mathcal{P}$. We will study c in Section 6 in detail.

Theorem 5.9. *Algorithm* EDI *finds the user's favorite point in $O(c \log n)$ rounds. c is the number of times initializing $\mathcal{P}$ and $c \leq n$.*

Proof Sketch. Since $|D| = n$, $\mathcal{P}$ is initialized with $|\mathcal{P}| \leq n$. Since $\mathcal{P}$ can be reduced by half in each round, $|\mathcal{P}|$ becomes 1 ($\leq \gamma$) in $O(\log n)$ rounds in each iteration. Because $|D| = n$, $c \leq n$. □

Algorithm 3: Algorithm *EDI*

Input: A point set D
Output: The user's favorite point
1 $\mathcal{R} \leftarrow \mathcal{E}$, $C = \{p \in D | \exists e \in \mathcal{E}, p \text{ is the nearest point of } e\}$
2 $\mathcal{P}$ contains the corresponding points of partitions in $[e_1, e_2]$.
3 **while** $|C| > 1$ **do**
4 Select the median points of $\mathcal{P}$ and present them to a user
5 Update $\mathcal{R}$, C and $\mathcal{P}$ based on the user feedback
6 **if** $|\mathcal{P}| \leq \gamma$ *and* $|C| > 1$ **then**
7 Select extreme points in $\mathcal{R}$ and initialize $\mathcal{P}$
8 **return** The point finally left in C

6 EXPERIMENT

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

Datasets. We conducted experiments on synthetic and real datasets that were commonly used in existing studies [22, 32, 35]. The synthetic datasets were *anti-correlated*, *correlated*, *independent* and *clustered* [5, 11]. The real datasets were *Car*, *Flight*, *Air quality*, *House*, *NBA* and *Wine*. *Car* contained 20,186 cars after we filtered the cars whose attribute values were not in a normal range. The filtering step followed [32]. *Car* had 3 ordered attributes (price, year of production, and used mileage) and 3 unordered attributes (length, width, and the number of seats). There were cars whose attribute values differed from those of the other cars largely (e.g., the year of production was 1927). Thus, we removed the records with price outside the range \$27000 – \$5215000 and the records with year of production outside the range 1997 – 2020. *Flight* contained 439,645 records with 3 ordered attributes (taxi-out, taxi-in, and actual-elapsed-time) and 2 unordered attributes (air-time and distance). *Air quality* contained 2,470,487 records with 2 ordered attributes (PM 2.5 and PM 10) and 1 unordered attribute (temperature). Due to the lack of space, the description and results of the other real datasets are shown in our technical report [31]. Note that existing studies [32, 35] preprocessed datasets to only contain skyline points p (i.e., $\nexists q$ which is better than p w.r.t. any user preference). Consistent with their setting, we preprocessed all the datasets to only contain the points p such that $\nexists q$ which is closer than p to any $e \in \mathcal{E}$. After the preprocessing, the datasets contained about $10^2 - 10^4$ records.

Algorithms. We evaluated our algorithms *DI*, *BS* and *EDI* with existing interactive algorithms *ActiveRanking* [13], *UtilityApprox* [20], *UH-Random* [35], and *RH* [32]. The existing ones cannot solve our problem directly since they are designed only for ordered attributes. They model the user preference by a *utility function* and learn the utility function by interacting with users. We adapted them by replacing the utility function with our distance function and following their idea to select points as questions for interaction. Besides, we made a few more adaptations for each of them. *ActiveRanking* learns the ranking of points. We returned the first ranked point after obtaining the full ranking. *UtilityApprox* and *UH-Random* find a point such that a criterion called the *regret ratio* [35] is minimized. They stop the interaction when they can find a point whose regret ratio satisfies a given threshold ϵ. We set $\epsilon = 0$ since this guarantees that the returned point is the user's favorite point. *RH* returns one of the user's favorite k points. To obtain the user's favorite point, we set $k = 1$. Note that [32, 35] also proposed algorithms *UH-Simplex* and *HD-PI*. However, both algorithms cannot be adapted to solve our problem IOU since they utilize several properties of the utility function that the distance function does not have.

Parameter Setting. We evaluated the performance of algorithms by varying several parameters: (1) parameter β, used in the priority

shown in Section 5.1.2; (2) parameter γ, deciding the update of $\mathcal{P}$ shown in Section 5.2.1; (3) the dataset size n; (4) the number of ordered dimensions d_o; (5) the number of unordered dimensions d_u; and (6) the number of questions we can ask. Unless stated explicitly, following the default setting of [32, 35], the synthetic datasets were anti-correlated. Besides, $n = 100,000$, $d_o = 4$ and $d_u = 4$ by default.

Performance Measurement. We evaluated algorithms by the following measurements: (1) *preprocessing time*, which is the time cost before the user interaction; (2) *interaction time*, which is the time cost of the user interaction; (3) *the number of questions asked*; and (4) *candidate size*, which is the size of C during the interaction. We reported the percentage of the remaining points in C in each interactive round. Each algorithm was conducted 10 times with different distance functions and the average performance was reported.

6.1 Results on Synthetic Datasets

Parameter β. In Figure 9, we studied parameter β, which is used in the *priority* (shown in Definition 5.4), by varying it from 2 to 7 and evaluating the interaction time and the number of questions asked. When $\beta = 4$, the two measurements reach the smallest. Thus, we stick to $\beta = 4$ in the rest of our experiments.

Parameter γ. Figure 9 shows our exploration on parameter γ, which decides the update of $\mathcal{P}$ in *EDI*. We varied γ from 1 to 6 and evaluated the interaction time and the number of questions asked. Both measurements fluctuate slightly. The number of questions asked is around 18 and the interaction time is comparably shorter when $\gamma = 3$ and 4. Thus, we set $\gamma = 4$ in the rest of our experiments.

Ratio α. We studied α in Figure 10, which denotes the percentage of points in C that *BS* reduces in each round (discussed in Section 5.1.3). i–i means that the dataset has i ordered and i unordered dimensions, where $i \in [1, 4]$. It can be seen that $\alpha \geq 25$ and α is around 40-50 in most cases. This indicates that *BS* can effectively reduces C and thus, it will only ask users a few questions.

Parameter c. In Figure 11, we studied parameter c used in the theoretical bound of the number of questions asked by *EDI* (discussed in Section 5.2.3). $i - i$ means the dataset has i ordered and i unordered dimensions, where $i \in [1, 4]$. The results show that c is a small value. This supports that *EDI* will only ask users a few questions.

Progress Study. We demonstrated how the algorithms progress during the interaction process. Varying the number of questions we can ask, we reported the interaction time and the candidate size. The second measurement is an indicator showing the number of questions asked by algorithms. If the candidate size can be reduced effectively, we can quickly achieve the stopping condition. We also evaluated the preprocessing time. Algorithms *BS* and *EDI* were compared against existing algorithms on 2 synthetic datasets. For completeness, we also included the simple approach, namely *Baseline*, in our experiments, which utilizes the Voronoi Diagram (introduced at the beginning of Section 5.1).

In the first dataset, $d_o = 1$ and $d_u = 1$. It contained around 400 points after the preprocessing. Our algorithms *DI* were also involved. Figure 13 shows the preprocessing time where the x-axis label is "1-1". All the algorithms preprocess nearly the same time except *UtilityApprox*. Note that *UtilitApprox* does not preprocess the dataset since it uses fake points (i.e., points not selected from the dataset) during the interaction. Although it has a slight advantage on the preprocessing time, it encounters several problems as mentioned in Section 2. Figure 12 shows the results of the other two measurements. All the algorithms can finish within 0.1 seconds. In particular, *DI* performs the best among all the algorithms. It only takes 10^{-3} seconds. Besides, our algorithms *DI*, *BS* and *EDI* reduce the candidate size the most effectively. They reduce the candidate size to less than 3% after asking 3 questions.

In the second dataset, $d_o = 4$ and $d_u = 4$. It contained about 1300 points after the preprocessing. Figure 13 shows the preprocessing time where the x-axis label is "4-4". *Baseline* is costly since it utilizes the Voronoi Diagram which is time-consuming to compute. It takes 3 orders of magnitude longer than our algorithms. Except for *Baseline*, the preprocessing times of all the other algorithms are short. Since the preprocessing times of algorithms are short (except *Baseline*) and have little impact on the user interaction, we do not show them later. Figure 14(a) shows the interaction time. All the algorithms only take a few seconds. *ActiveRanking* and *RH* run slightly faster than our algorithms since they do not prune points during the interaction. The lack of pruning points may neglect the connection between points and result in asking more questions. Although our algorithms spend slightly more time, their interaction times are short and reasonable, given that they can effectively reduce the candidate size and obtain the user's favorite point by asking a few questions. Figure 14(b) shows the candidate size. Since *ActiveRanking* only focuses on learning the ranking of points and *UtilityApprox* does not include points from datasets during the user interaction, they fail to provide any reduction on the candidate size. *UH-Random* and *Baseline* do not reduce the candidate size effectively. After asking 5 questions, they contain about 5 times and 2 times more points in C than our algorithms, respectively. Since *Baseline* performed badly (i.e., a long preprocessing time and an ineffective candidate size reduction), we did not include it in the later experiments.

Scalability. We studied the scalability of algorithms by varying the type of datasets, d_o, d_u and n. Each algorithm was measured by the interaction time and the number of questions asked.

Varying type. In Figure 15, we ran algorithms on four kinds of datasets: anti-correlated (Anti), correlated (Cor), independent (Indep), and clustered (Clu). They contained about 1300, 2100, 2000, and 1800 points after the preprocessing. All the algorithms run within 3 seconds. Our algorithms ask the fewest questions. They can ask up to 89% fewer questions than existing algorithms.

Varying d_o. In Figure 16, we fixed $d_u = 4$ and varied d_o from 3 to 6. After the preprocessing, the datasets contained 700-3000 records for different d_o. The results show that all the algorithms are fast. They can finish within 3.4 seconds. Besides, our algorithms ask $8\% - 88\%$ fewer questions than existing algorithms. This verifies that our algorithms perform well when we scale the ordered attributes.

Varying d_u. In Figure 17, we fixed $d_o = 4$ and varied d_u from 3 to 6. After the preprocessing, the datasets contained 400-8000 records for different d_u. Our algorithms perform the best in terms of the number of questions asked. When $d_u = 6$, they ask at most 28.4 questions, while the best existing algorithm *UH-Random* asks 32 questions. The interaction times of *UH-Random* and *ActiveRanking* increase largely when d_u increases. When $d_u = 6$, they run 1-2 orders of magnitude longer than our algorithms.

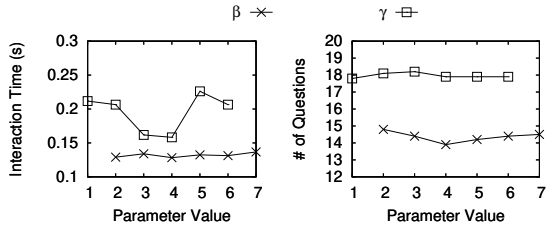


Figure 9: Parameter β and γ

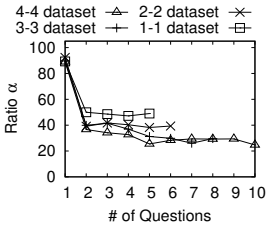


Figure 10: Ratio α

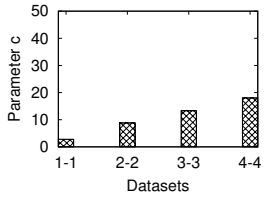


Figure 11: Parameter c

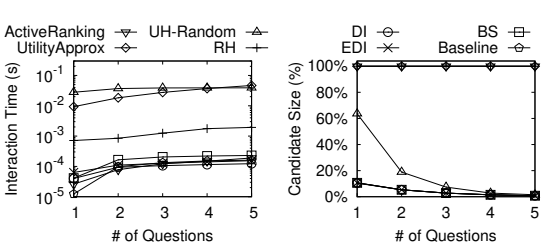


Figure 12: 1 Ordered & 1 Unordered

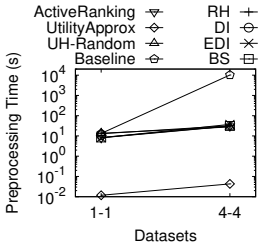


Figure 13: Preprocessing

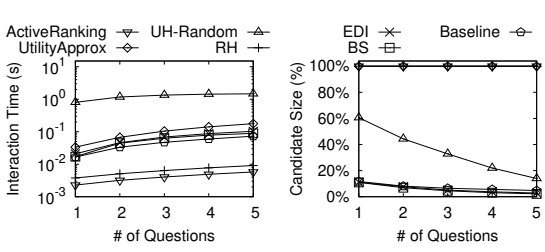


Figure 14: 4 Ordered & 4 Unordered

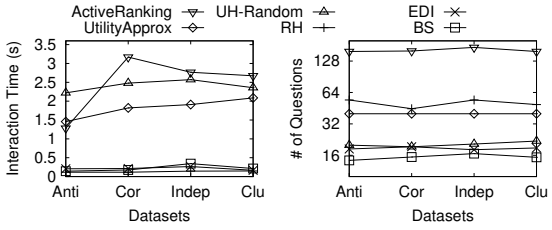


Figure 15: Vary Type

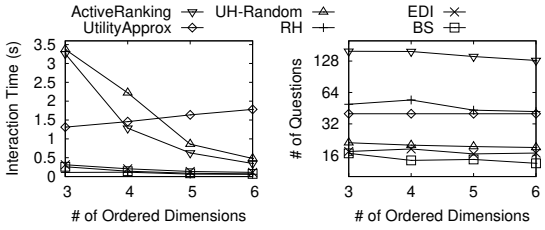


Figure 16: Vary d_o

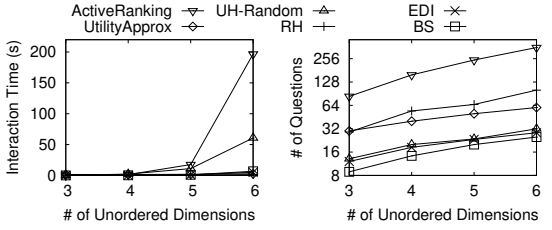


Figure 17: Vary d_u

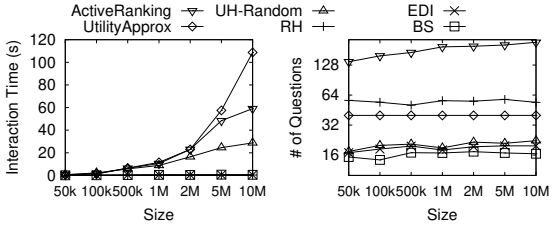
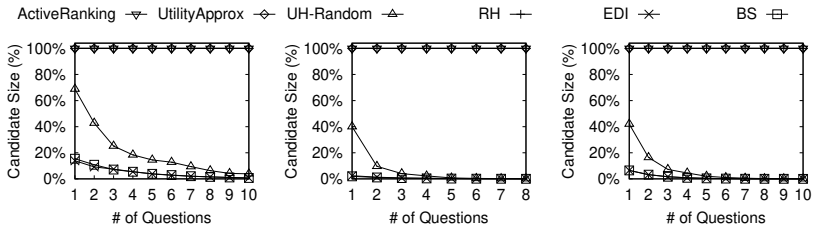


Figure 18: Vary n

Varying n. In Figure 18, we varied dataset size n from 50k to 10M. After the preprocessing, the datasets contained 1000-12,000 records for different n. Our algorithms ask 5% – 92% fewer questions than existing algorithms. They also scale well w.r.t. the interaction time. They spend less than 0.6 seconds even if n = 10M. Note that *RH* is slightly faster than our algorithms since it does not prune points during the interaction. However, this leads it to ask around 3 times more questions than our algorithms. Although our algorithms spend slightly more time, their interaction times are still short (within 0.6 seconds)



(a) Car (b) Flight (c) Air quality

Figure 19: Real Datasets

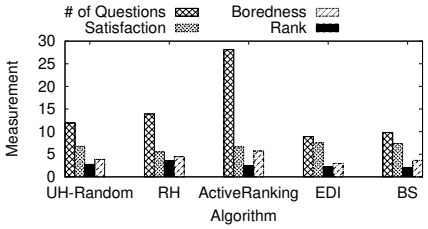


Figure 20: User Study

and they can effectively reduce the candidate size and obtain the user's favorite points by asking only a few questions.

6.2 Results on Real Datasets

We explored how the algorithms progress during the interaction process on real datasets. *Car*, *Flight* and *Air quality* contained about 3×10^2, 6×10^3 and 2×10^4 records after the preprocessing. We varied the number of questions we can ask and reported the candidate size. Due to the lack of space, the interaction time of each algorithm is shown in the technical report [31]. Figure 19 presents the candidate size by varying the number of questions we can ask. Our algorithms reduce the candidate size the most effectively. For example, on dataset *Car*, after asking 2 questions, our algorithms only contain less than 15% points in C. In comparison, the existing algorithms contain at least 68% points in C.

6.3 User Study

We conducted a user study on dataset *Car* to see the impact of user mistakes on the final results since users might make mistakes or provide inconsistent feedback during the interaction. Following the setting in [23, 32, 34, 35], 30 users were recruited, and their average result was reported. We compared our algorithms *BS* and *EDI* against 3 existing algorithms *Active-Ranking*, *UH-Random* and *RH*. We did not involve *UtilitApprox* since there are issues to apply it in real life scenarios [32, 35] as mentioned in Section 2. The adaptation of the existing algorithms shown previously was maintained.

The following measurements evaluated each algorithm. (1) *The number of questions asked.* (2) *Degree of satisfaction.* It is a score from 1 to 10 given by each user, indicating how satisfied the user is with the returned car. A larger score means the user is more satisfied with the returned car. (3) *Rank.* It is the rank of the algorithm given by each user based on its returned car. Since users sometimes gave the same degree of satisfaction for the cars returned by different algorithms, to distinguish them clearly, we asked each user to give a ranking of all the algorithms. (4) *Degree of boredness.* It is a score from 1 to 10 given by each user, indicating how bored the user feels when s/he sees the returned car after being asked several questions. 1 denotes the least bored and 10 means the most bored.

Figure 20 shows the results. Our algorithms *BS* and *EDI* ask 9.8 and 8.9 questions, respectively, while existing algorithms ask more than 11.9 questions. In particular, the number of questions asked by *ActiveRanking* is up to 28.1. Besides, our algorithms *BS* and *EDI* return the car with the highest satisfaction and rank the best. Their degrees of satisfaction are 7.4 and 7.5, respectively. In comparison, the degrees of satisfaction of existing algorithms are less than 6.7. The degree of satisfaction of the least satisfying algorithm *RH* is 5.5, especially. Our algorithms also obtain the least boredness score. Their degrees of boredness are 3.6 and 2.9, respectively, while the degrees of boredness of existing algorithms are more than 3.8.

We also conducted a user study to see the impact of difficult questions since there might be questions that users cannot answer. Due to the lack of space, it can be found in the technical report [31].

6.4 Summary

The experiments showed the superiority of our algorithms over the best-known existing ones. (1) We are effective and efficient. Compared with the existing algorithms, our algorithms achieve significant improvements in the interaction time and the number of questions asked (e.g., when $d_o = 4$ and $d_u = 4$, our algorithms ask up to 10 times fewer questions than *ActiveRanking*). (2) Our algorithms scale well on the type of dataset, the number of dimensions and the dataset size (e.g., our algorithm *BS* asks 25 questions in 6 seconds on the dataset with $d_o = 4$ and $d_u = 6$, while *UH-Random* asks 32 questions in 60 seconds). (3) The pruning strategy (Lemma 3.4) is useful (e.g., our algorithms reduce the candidate size to 15% by only asking 2 questions on datasets *Car*, while the existing algorithms can only reduce it to 68%). In summary, *DI* asks the fewest questions in the shortest time for the special case of IOU. *BS* and *EDI* ask the fewest questions for the general case of IOU.

7 CONCLUSION

In this paper, we present interactive algorithms for searching the user's favorite tuple on the dataset with ordered and unordered attributes. On the dataset in which $d_o = 1$ and $d_u = 1$, we propose algorithm *DI*, which is asymptotically optimal w.r.t. the number of questions asked. For the general cases, we present two algorithms *BS* and *EDI*, which perform well w.r.t. the number of questions asked theoretically and empirically. Extensive experiments showed that our algorithms are both efficient and effective. As for future work, we consider that users might have various priorities to different attributes and may make mistakes when answering questions.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Weicheng Wang and Raymond Chi-Wing Wong is supported by *PRP*/026/21*FX*.

REFERENCES

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

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

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

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

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

[6] Baoping Cai, Lei Huang, and Min Xie. 2017. Bayesian Networks in Fault Diagnosis. *IEEE Transactions on Industrial Informatics* 13, 5 (2017), 2227–2240.

[7] John Gerald Cleary. 1979. Analysis of an Algorithm for Finding Nearest Neighbors in Euclidean Space. *ACM Trans. Math. Softw.* 5, 2 (1979), 183–192.

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

[9] Evangelos Dellis and Bernhard Seeger. 2007. Efficient Computation of Reverse Skyline Queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 291–302.

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

[11] Clustered Dataset Generator. 2022. https://personalpages.manchester.ac.uk/staff/Julia.Handl/generators.html

[12] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.

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

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

[15] Wei Li, Pascal Poupart, and Peter van Beek. 2011. Exploiting Structure in Weighted Model Counting Approaches to Probabilistic Inference. *J. Artif. Int. Res.* 40, 1 (2011), 729–765.

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

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

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

[19] Boriana L Milenova, Joseph S Yarmus, and Marcos M Campos. 2005. SVM in Oracle Database 10g: Removing the Barriers to Widespread Adoption of Support Vector Machines. In *Proceedings of the 31st International Conference on Very Large Data Bases*. 1152–1163.

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

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

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

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

[24] Jianbin Qin, Wei Wang, Chuan Xiao, Ying Zhang, and Yaoshu Wang. 2021. High-Dimensional Similarity Query Processing for Data Science. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining*. Association for Computing Machinery, New York, NY, USA, 4062–4063.

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

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

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

[28] Thomas Seidl and Hans-Peter Kriegel. 1997. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 506–515.

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

[30] Yufei Tao, Jun Zhang, Dimitris Papadias, and Nikos Mamoulis. 2004. An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. *IEEE Trans. Knowl. Data Eng.* 16, 10 (2004), 1169–1184.

[31] Weicheng Wang and Raymond Chi-Wing Wong. 2022. *Interactive Mining with Ordered and Unordered Attributes*. Technical Report. https://cse.hkust.edu.hk/~raywong/paper/vldb22-interaction-mixedAttribute-technicalReport.pdf

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

[33] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 194–205.

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

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

[36] Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. 2020. An Experimental Survey of Regret Minimization Query and Variants: Bridging the Best Worlds between Top-k Query and Skyline Query. *VLDB Journal* 29, 1 (2020), 147–175.

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

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

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