# 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. They first learn the user preference by interacting with the user and then return tuples based on the learned preference. Specifically, they interact with a user by asking questions. In each question, they present two tuples and ask the user to select the one s/he prefers. Based on the feedback, the user preference is learned implicitly and the best tuple w.r.t. the learned preference is returned. However, existing queries only consider the dataset with ordered attributes (e.g., price), where there exists a trivial order on the attribute values. In practice, the dataset can be also 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 guarantee and perform well empirically. Experiments were conducted on synthetic and real datasets, showing that our algorithms outperform existing algorithms both on the execution time and the number of questions asked. 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, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

## 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, 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 since 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 so that it could carry the entire family or a single user may favor a car with few seats for saving fuel consumption. Thus, the user's expected number of seats could vary.

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* [8] and *the $k$ nearest neighbours query (kNN)* [25]. The first operator is the relative skyline query. It 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 [17]. The second operator is the $k$ nearest neighbours query (kNN). It measures the user preference on tuples by the distance from 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 [17, 29]. If the user's expected tuple is not known, kNN cannot be applied in practice.

Motivated by this 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 (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 several questions. Following [20, 27, 29], each question *consists of two tuples and asks the user to pick the one s/he prefers*. Based on the user selection, the user's expected tuple is implicitly learned, and the user's favorite tuple is returned. This kind of interaction naturally appears in our daily life. A real estate agent might show Alice two houses and let her pick the one she prefers. A seller may present two red wines and ask Alice: which wine more suits your taste?

Problem IOU involves a number of questions asked to a user. In the literature of the marketing research [15, 21], it is pivotal to keep the questions tally low. Otherwise, users may lose the plot and turn


Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

out to be 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 learn the user's expected tuple more clearly to distinguish its $k$ (instead of 1) nearest points. The experimental results of [27] verified that recommending multiple tuples (e.g., 20 tuples) requires obtaining more information from users. It may need to ask 4-10 times more questions than recommending one tuple. The user study of [27] also shows that instead of obtaining more recommended tuples, users are willing to answer fewer questions. 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 the $k$ nearest tuples. We will show the extension in Section 5.3.

To the best of our knowledge, we are the first to study problem IOU. There are some closely related problems [11, 17, 27, 29] involving user interaction, but they are different from ours. [11] aims at obtaining the ranking of all tuples (the favorite tuple ranks the first). Since the ranking of all tuples has to be determined, there are a lot of questions asked. [17, 29] propose to find the user's (close to) favorite tuple. However, they only consider the case that the tuples are described by ordered attributes. Besides, [17] utilizes "fake" tuples (i.e., tuples not in the dataset) in the questions during the interaction, which suffers the *truthfulness problem* to be described in Section 2. In contrast, we utilize "real" tuples (i.e., tuples in the dataset). [27] searches for one of the user's favorite $k$ tuples. Similar to [17, 29], it only works on the dataset with ordered attributes, which cannot be applied in many scenarios.

Unfortunately, none of the existing algorithms can 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, on the dataset with 4 ordered attributes and 6 unordered attributes, our algorithms ask fewer than 28 questions within 7 seconds. The existing algorithms either ask 2-10 times more questions ([11, 17, 27]) or execute 10-20 times longer ([11, 29]).

**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 from the dataset 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 to determine the user's favorite tuple in the dataset, where $n$ is the number of tuples in the dataset.
- 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 with respect to 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 guarantee on the number of questions asked and perform well empirically.
- We conducted experiments to demonstrate the superiority of our algorithms. The results show that under typical settings, our algorithms ask up to 10 times fewer questions and spend several orders of magnitude less time than existing algorithms.

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

## 2 RELATED WORK

There are various queries proposed to assist the multi-criteria decision-making. Section 2.1 briefly overviews the *preference-based queries* that return tuples based on the expected tuple given by a user. Section 2.2 illustrates the *interactive queries* that involve user interaction during the query processing.

### 2.1 Preference-based Queries

The skyline query [8] returns all tuples that are not dominated by another tuple. A tuple $q$ *dominates* another tuple $p$ 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 only based on the attribute values of tuples. For example, if an attribute value of tuple $p$ is smaller than that of tuple $q$, we say that $p$ is better than $q$ in that attribute. This limits it to deal only with ordered attributes since the values of unordered attributes are not the smaller the better. In comparison, the relative skyline query is able to handle both ordered and unordered attributes. It is based on the coordinate-wise distance between two tuples and takes the user's expected tuple into account. Specifically, a tuple $p$ is better than $q$ in an attribute if the attribute value of $p$ is closer to that of the user's expected point than that of $q$. The deficiency of the skyline query is that its output size is uncontrollable. It is possible that the whole dataset is return as the answer [18, 30].

The $k$ nearest neighbors query (kNN) [25] avoids this problem. Based on the user's expected tuple, it defines a distance function which measures the distance from each tuple in the dataset to the user's expected point. The kNN returns the $k$ tuples, which have the smallest distance to the user's expected point. Another relevant query is the similarity query [24]. It defines a more complicated distance function and finds tuples that are close to the expected tuple w.r.t. the distance function. However, both the kNN and the similarity query rely on an assumption that the user's expected tuple is known in advance [3]. In practice, a user does not always know the expected tuple. Even if the expected tuple is known, the user needs to spend additional effort specifying it.

### 2.2 Interactive Queries

The interactive queries involves user interaction [1–4, 12, 17, 23, 29, 31]. They learn the user preference by asking the user questions and return tuples based on the learned preference. [1, 2, 12] proposed the interactive skyline query that tries to reduce the output size of the skyline query by interacting with the user. Specifically, it learns

the user preference on the attributes 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, it is possible that even if the user preference on all attribute values is obtained, the output size is still arbitrarily large [18]. Let us 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 [18, 30].

[17] proposed the interactive regret minimizing query. It targets to return a small size of output with a small *regret ratio* by interacting with the user. The *regret ratio* evaluates returned tuples and represents how regretful a user is when s/he sees the returned tuples instead of the whole dataset. During the interaction, [17] asks a user several questions, each of which consists of several tuples, and asks the user to pick the one s/he prefers. However, the displayed tuples are *fake tuples*, which are artificially constructed (not selected from the dataset). This might produce unrealistic tuples (e.g., a car with 10 dollars and 1000 seats) and the user can be disappointed if the displayed tuples with which s/he is satisfied do not exist [29]. To overcome the defect, [29] proposed the strongly truthful regret minimizing query, which displays *real tuples* (selected from the dataset) during the interaction. However, it requires users to answer too many questions, which causes the effectiveness problem in real-life applications. To reduce the number of questions asked, [31] changed the way of asking questions. It asks users to sort the displayed tuples based on their preference. However, this does not reduce the user effort essentially since sorting the displayed tuples is equivalent to picking the favorite tuple several times.

There are alternative approaches [20, 27] which also take user interaction into account. [20] approximated the user preference by interacting with the user. Nevertheless, it aims at learning the user preference rather than returning tuples. This might result in asking the user many questions [29]. 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 [20]. [27] returns one of the top-$k$ tuples by interacting with the user. In detail, it models the user preference as a utility function. With the help of user interaction, it learns the utility function and returns one of the $k$ tuples with 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, 23] proposed the interactive similarity query. It returns tuples that are close to a query tuple w.r.t. a distance function, where the query tuple and the distance function are learned by interacting with the 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 that affects the final output significantly. 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* [9, 11, 14, 16] also involves user interaction. It learns the ranking of tuples. However, most of the existing methods [9, 14, 16] only consider the relations between tuples (where a relation means that a tuple is preferable to another tuple) and do not utilize their inter-relation (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 [27, 29]. Algorithm *Active-Ranking* [11] considers the inter-relation between tuples and learns the ranking of tuples by interacting with the user. However, it assumes that all tuples are in the *general position* [22], which could not be applied in many cases. Besides, it focuses on deriving the order for all pairs of tuples, which requires asking many questions due to the similar reason stated for [20].

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. This 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 [17] 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 either require to learn a full ranking [9, 14, 16] or an exact user preference [20], while we only return the user's favorite tuple. Secondly, [9, 14, 16] do not utilize the inter-relation between tuples and thus, involve some unnecessary interaction. Thirdly, compared with [3, 23, 31], our designed questions are easier to answer and more efficient to collect the information of the user preference.

# 3 PROBLEM DEFINITION

## 3.1 Terminologies

We consider that the 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.

Following [10, 28], we model the user preference in the form of a function $f(p) = (\sum_{i=1}^{d}(p[i] - e[i])^2)^{\frac{1}{2}}$, called *distance function*, denoted by $f(p) = ||p-e||$ for simplicity. Point $e$ is an *expected point* that 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* [7] in a $d$-dimensional geometric space. For example, if there are one ordered dimension and one unordered dimension, 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 closer to the user's expected point, i.e., 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 and thus, each dimension might contribute 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, 13, 20, 29]. In the rest of the paper, we assume that all the dimensions are equally important and are normalized to $[0, 1]$.

*Example 3.1.* Consider Table 1. Assume that each point is described by one ordered dimension and one unordered dimension. 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 Interactive Framework

Our interactive framework follows [27, 29] and works on the dataset with ordered dimensions 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.

- (**Point selection**) We select two points presented to the user and ask the user to pick the one s/he prefers.
- (**Information maintenance**) Based on the feedback, the maintained information for learning the expected point is updated.
- (**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.

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 $D$.*

## 3.3 Lower bound

We present a lower bound of the number of questions asked on IOU. Due to the lack of space, the proofs of some theorems/lemmas in this paper can be found in the technical report [26].

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. We show that any algorithm needs to ask $\Omega(\log_2 n)$ questions to determine the user's favorite point. □

## 3.4 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 $p - q$ [7]. Hyper-plane $h_{p,q}$ divides space $\mathbb{R}^d$ 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 halfspaces. 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 $h^+_{p,q}$ (or $h^-_{p,q}$) 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 algorithm Linear Programming (LP). We define a variable $x$ and set the objective function to be $\max x$. For each point $q \in D \setminus \{p\}$, 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 $\forall e \in \mathcal{R}$, $\exists q \in D \setminus \{p\}$ such that $||p - e|| > ||q - e||$. Due to the lack of space, the detailed LP formulation is shown in the technical report [26]. Since there are $O(n)$ points in a $d$-dimensional space, 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 essentially.

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 $q \in D \setminus \{p\}$ 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 that 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

| $p$ | $p[1]$ | $p[2]$ | $f(p)$ |
|---|---|---|---|
| $p_1$ | 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 |

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

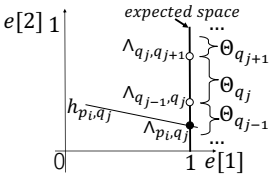



**Figure 1: Scan Case 1**

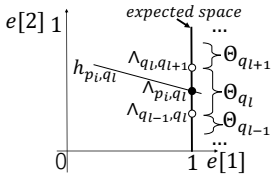



**Figure 2: Scan Case 2**

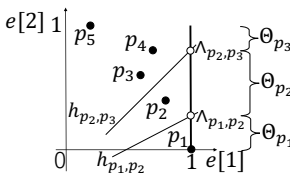



**Figure 3: Division of $S_3$**

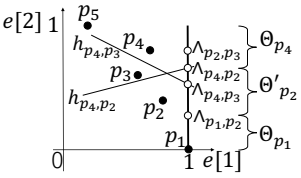



**Figure 4: Division of $S_4$**

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, the expected space is first divided into several disjoint smaller line segments, called partitions and denoted by $\Theta$. Each partition $\Theta$ corresponds to a point that 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

We present our method to divide the expected space into the fewest partitions. The result is called the *division* of the expected space. In a 2-dimensional space $\mathbb{R}^2$, as discussed in Section 3.4, for any pair of points $p, q \in D$, we could build a hyper-plane $h_{p,q}$ (i.e., a line in $\mathbb{R}^2$) which divides $\mathbb{R}^2$ into two half-spaces (i.e., two half-planes in $\mathbb{R}^2$). If $h_{p,q}$ intersects the expected space, the intersection is denoted by $\wedge_{p,q}$. The line segment that connects any two intersections $\wedge_1$ and $\wedge_2$ is represented by $[\wedge_1, \wedge_2]$.

We divide the expected space based on the hyper-planes that consist of points in $D$. Specifically, 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 the expected space, where $i \in [2, n]$. Our strategy is to add $p_i$ and then update the division of the expected space. Note that not every point in $S_{i-1}$ will necessarily correspond to a partition in the division. Assume that $< q_1, q_2, ..., q_k >$ are the corresponding points in $S_{i-1}$ of the partitions $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_k} >$ from bottom to top. Point $q_j$, where $j \in [1, k]$, is the nearest point of any $e \in \Theta_{q_j}$ among $S_{i-1}$. The lower boundary and upper boundary of $\Theta_{q_j}$ are the intersections $\wedge_{q_{j-1},q_j}$ and $\wedge_{q_j,q_{j+1}}$, respectively, i.e, $\Theta_{q_j} = [\wedge_{q_{j-1},q_j}, \wedge_{q_j,q_{j+1}}]$ (where $\wedge_{q_0,q_1} = (1, 0)$ and $\wedge_{q_k,q_{k+1}} = (1, 1)$). 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 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. According to our scanning strategy, for each point $p_j$, where $j \in [l+1, k]$, $\wedge_{p_i,q_j}$ is below $\wedge_{q_{j-1},q_j}$. As shown in Figure 1, if $\wedge_{p_i,q_j}$ is below $\wedge_{q_{j-1},q_j}$, $\Theta_{q_j} \subseteq h^+_{p_i,q_j}$. Note that point $q_j$ is the nearest point of any $e \in \Theta_{q_j}$ among $S_{i-1}$. Since $p_i$ is closer to any $e \in h^+_{p_i,q_j}$ than $q_j$, $p_i$ must be the nearest point of any $e \in \Theta_{q_j}$ among $S_i$. Thus, $p_i$ is the nearest point of any $e \in \bigcup_{j=l+1}^{k} \Theta_j = [\wedge_{q_l,q_{l+1}}, \wedge_{q_k,q_{k+1}}]$ among $S_i$.

Since $p_i$ is the nearest point of any $e \in [\wedge_{q_l,q_{l+1}}, \wedge_{q_k,q_{k+1}}]$ among $S_i$, $\wedge_{p_i,q_l}$ should be below $\wedge_{q_l,q_{l+1}}$. Otherwise, $[\wedge_{q_l,q_{l+1}}, \wedge_{p_i,q_l}] \subseteq h^-_{p_i,q_l}$ and $q_l$ is closer to any $e \in [\wedge_{q_l,q_{l+1}}, \wedge_{p_i,q_l}]$ than $p_i$. Because $\wedge_{p_i,q_l}$ is below $\wedge_{q_l,q_{l+1}}$, $[\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}] \subseteq h^+_{p_i,q_l}$. Point $p_i$ is closer to any $e \in [\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}]$ than $q_l$. Since $q_l$ is the nearest point of any $e \in \Theta_{q_l}$ among $S_{i-1}$, $p_i$ must be the nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}]$ among $S_i$. Thus, $p_i$ is nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{q_k,q_{k+1}}]$ among $S_i$.

Since $\wedge_{p_i,q_l}$ is above $\wedge_{q_{l-1},q_l}$, $\wedge_{p_i,q_l} \in [\wedge_{q_{l-1},q_l}, \wedge_{q_l,q_{l+1}}]$. Note that $[\wedge_{q_0,q_1}, \wedge_{p_i,q_l}] \subseteq h^-_{p_i,q_l}$. Point $q_l$ is closer to any $e \in [\wedge_{q_0,q_1}, \wedge_{p_i,q_l}]$ than $p_i$. This means that $p_i$ will not be the nearest point of any $e \in [\wedge_{q_0,q_1}, \wedge_{p_i,q_l}]$ among $S_i$. The nearest point of any $e \in [\wedge_{q_0,q_1}, \wedge_{q_{l-1},q_l}]$ 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 is updated as follows. (1) $< q_1, q_2, ..., q_k >$ 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 updated 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. We prove that our algorithm can obtain the fewest partitions with the help of mathematical induction. As for the time complexity, we need $O(n \log n)$ time to sort all the points. Then, when updating 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 the expected space decided by $S_n$. 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. $< q_{j+1}, q_{j+2}, ..., q_m >$).

**Algorithm 1:** Algorithm *DI*

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

Sort all points based on their second dimension
$\Theta \leftarrow < \Theta_{p_1} >, C \leftarrow < p_1 >$
**for** $i \leftarrow 2$ *to* $n$ **do**
  **for** $j \leftarrow k$ *to* 1 **do**
    **if** $\wedge_{p_i,q_j}$ *is above* $\wedge_{q_{j-1},q_j}$ **then**
      $l \leftarrow j, C \leftarrow < q_1, q_2, ..., q_l, p_i >$
      $\Theta \leftarrow < \Theta_{q_1}, ..., \Theta_{q_{l-1}}, \Theta'_{q_l}, \Theta_{p_i} >$
      break
$left \leftarrow 1, right \leftarrow m$
**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**
    $right \leftarrow j$
  **else**
    $left \leftarrow j + 1$
  $C \leftarrow < q_{left}, ..., q_{right} >$
**return** The point finally left in $C$

Our algorithm interacts with a user for rounds. It maintains a point set $C$ that contains the user's favorite point. $C$ is initialized to be the set of points $< q_1, q_2, ..., q_m >$ obtained from Section 4.1. In each round, our algorithm asks a question by presenting to 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 final left point in $C$ is returned to the user as the answer.

*Example 4.4.* Following Example 4.2, suppose that $C$ is initialized to be $< p_1, p_2, p_4 >$. Our algorithm presents the 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 to the user as the answer.

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. Both algorithms have provable guarantee on the number of questions asked and preform well empirically. 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 with respect to the number of questions asked.

*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 $\mathcal{E}$ and $C$ contains all the points in $D$ that are 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 deleted the points in $C$ that cannot be the user's favorite point (shown in Section 3.4). 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 expected point in $\mathcal{E}$. However, it is unachievable in practice due to the infinite expected points. Thus, we try to 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 $p \in D$.

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 $e \in \mathcal{E}'_p \cup \mathcal{E}''_p$.*

Note that it is possible that $\mathcal{E}_p = \emptyset$, i.e., $p$ is not the nearest point of any $e \in \mathcal{E}$. Now, let us consider a pair of points $p, q \in D$.

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 $q$ has 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 $\mathcal{H}$ as follows. 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 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 $\mathcal{E}_{p_3}$

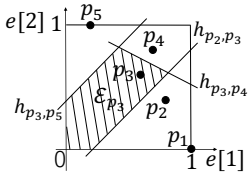



**Figure 5: Polyhedron $\mathcal{E}_{p_3}$**

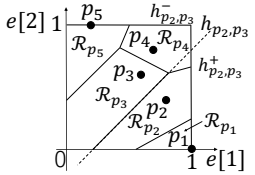



**Figure 6: Polyhedrons**

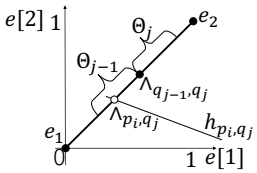



**Figure 7: Case 1**

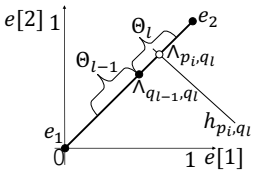



**Figure 8: Case 2**

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 $\mathcal{H}$ so that the size of $C$ can be rapidly reduced during the interaction. In a high level, let set $U = \{\mathcal{R}_p | p \in C\}$, 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 corresponds to the neglected polyhedrons $\mathcal{R}_p$ can be pruned.

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. In the following, we present a heuristic approach for the relation checking that performs well empirically.

Consider a point $p \in C$ and its polyhedron $\mathcal{R}_p$. Denote by $r_p$ the middle points 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 a 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 $\mathcal{R}$ 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.

**Algorithm 2:** Algorithm *BS*

**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 $\mathcal{H}$ contains all the boundaries of all $\mathcal{E}_p$
3 **while** $|C| > 1$ **do**
4  Select hyper-plane $h_{p,q} \in \mathcal{H}$ with the highest priority
5  Present points $p$ and $q$ to the user
6  Update $\mathcal{R}$ and $C$ based on the user's feedback
7 **return** The point finally left in $C$

*Example 5.5.* Figure 6 demonstrates $U = \{\mathcal{R}_{p_1}, \mathcal{R}_{p_2}, \mathcal{R}_{p_3}, \mathcal{R}_{p_4}, \mathcal{R}_{p_5}\}$. Consider hyper-plane $h_{p_2,p_3}$ 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.* Our algorithm *BS* is summarized by involving all the techniques illustrated previously. The pseudocode is presented in Algorithm 2. At the beginning, $\mathcal{R}$, $C$ and $\mathcal{H}$ are initialized (line 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).

Lemma 5.6. *After each interactive round, $\mathcal{R}$ and $C$ are updated to be strictly smaller.*

Proof. Consider two points $p_1$ and $p_2$ that are presented in an interactive round. If a user prefers $p_1$ to $p_2$, we learn that $p_2$ cannot be the user's favorite point. $\mathcal{R}$ will be updated to be $\mathcal{R} \cap h^+_{p_1,p_2}$ so that $\mathcal{R}_{p_2} = \emptyset$, where $\mathcal{R}_{p_2} \subseteq \mathcal{R}$ is the maximal polyhedron such that $p_2$ is the nearest point of any $e \in \mathcal{R}_{p_2}$. Thus, $\mathcal{R}$ will be strictly smaller. Since $\mathcal{R}_{p_2} = \emptyset$, we will prune $p_2$ from $C$. Thus, $C$ will be strictly smaller. □

Theorem 5.7. *Algorithm* BS *solves IOU in $O(n)$ rounds. In particular, if the selected hyper-plane can help to reduce $C$ by half in each round (i.e., the optimal case),* BS *can solve IOU in $O(\log n)$ rounds.*

Proof. Based on Lemma 5.6, we can remove at least one point from $C$ in each interactive round. Since there are $n$ points, there will be only one point left in $C$ after $O(n)$ rounds. If the selected hyper-plane can help to reduce $C$ by half in each round, there exists only one point in $C$ after $O(\log n)$ rounds. □

## 5.2 Algorithm EDI

In this section, we present algorithm *EDI* that is an extension of algorithm *DI* for the general case of IOU. It asks a user $O(c \log n)$ questions to determine the user's favorite point, where $c$ is a parameter that will be discussed later.

*5.2.1 Information Maintenance & Stopping Condition.* In the course of the whole algorithm, 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 $D$ that are 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, the user preference is learned implicitly, and $\mathcal{R}$ and $C$ are updated accordingly (shown in Section 3.4). Meanwhile, $\mathcal{P}$ is also updated. The interaction stops when there is only one point $p$ in $C$. Finally, $p$ is returned to the user as the answer. The processing of $\mathcal{R}$ and $C$ is in the same way as algorithm *BS*. In the following, we show how to initialize and update $\mathcal{P}$.

**Initialization of $\mathcal{P}$.** First, we select two extreme points $e_1$ and $e_2$ of $\mathcal{R}$ (the way of selecting extreme points is shown in Section 5.2.2). With a slight abuse of notations, denote by $[e, e']$ the line segment that connects any pair of points $e, e' \in \mathcal{E}$. Following the idea of algorithm *DI*, we divide $[e_1, e_2]$ into the least number of partitions. Each partition $\Theta_p$ 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 point $p \in C$, we define the length of $p$ w.r.t. $[e_1, e_2]$, denote by $t_p$, to be the projection of vector $p - e_1$ on the direction of vector $e_2 - e_1$, i.e., $t_p = \frac{(p-e_1)\cdot(e_2-e_1)}{||e_2-e_1||}$. At the beginning, all the points in $C$ are sorted based on their $t_p$ in non-decreasing order. Let $< p_1, p_2, ..., p_{|C|} >$ represent the sorted list. Similar to algorithm *DI*, every set of points $S_{i-1} =< p_1, p_2, ..., p_{i-1} >$ decides a division of $[e_1, e_2]$. We are to add $p_i$ and update the division of $[e_1, e_2]$. Note that not every point in $S_{i-1}$ necessarily 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 $\Theta_{j-1}$ and $\Theta_j$ 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 there are two unordered dimensions.

Lemma 5.8. *The nearest points 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. For each point $q_j$, where $j \in [l+1, k]$, $\wedge_{p_i,q_j}$ is closer to $e_1$ than $\wedge_{q_{j-1},q_j}$. Figure 7 shows such case when there are two unordered dimensions. We prove that $p_i$ is the nearest point of any $e \in \Theta_j$ among $S_i$. For point $p_l$, $\wedge_{p_i,q_l}$ is farther away from $e_1$ than $\wedge_{q_{l-1},q_l}$. Figure 8 shows such case when there are two unordered dimensions. We prove that point $q_l$ is nearest point of any $e \in [\wedge_{q_{l-1},q_l}, \wedge_{p_i,q_l}]$ among $S_i$ and point $p_i$ is nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{p_l,q_{l+1}}])$ among $S_i$. □

Lemma 5.8 is an extension of Lemma 4.1 in high dimensional space. Following the lemma, when we reach point $q_l$, we update the division of $[e_1, e_2]$ as follows. (1) $< q_1, q_2, ..., q_k >$ 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 that 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 $\forall j \in [i+1, m]$, $\Theta_{p_j} \nsubseteq h^+_{p_i,p_{i+1}}$, points $< p_{i+1}, p_{i+2}, ..., p_m >$ cannot be the user's favorite point. Thus, we delete them from $\mathcal{P}$. If $|\mathcal{P}| \leq \gamma$, we select two new extreme points $e_3, e_4 \in \mathcal{R}$ and partition $[e_3, e_4]$ to initialize $\mathcal{P}$ with $C$ again. As for parameter $\gamma$, if it is too large, $\mathcal{P}$ may need to be initialized many times. On the contrary, the lack of candidates for points selection may affect the performance of algorithm. The setting of parameter $\gamma$ will be discussed in Section 6.

*5.2.2 Point selection.* We are ready to present our point selection strategy in the interactive framework. In particular, we first show how to select points from $\mathcal{P}$ and then discuss the way of picking 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 two consecutive 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.** 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 first 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 $\Theta$ 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 $\mathcal{P}$. The following lemma guarantees the soundness of our strategy used to pick the extreme points to initialize $\mathcal{P}$. Before reaching the stopping condition of the interactive framework, we can always find points in $\mathcal{P}$ as a question asked to a user.

Lemma 5.9. *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.* We summarize our algorithm *EDI* by combining the techniques presented in previous sections. The pseudocode is shown in Algorithm 3. $\mathcal{R}$, $C$ and $\mathcal{P}$ are first initialized (line 1-2). In each interactive round, *EDI* selects the two middle points in $\mathcal{P}$ as a question asked to a user (line 4). Based on the user feedback, $\mathcal{R}$ and

**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** $|\mathcal{P}| > 1$ **do**
4  Select the middle 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 $\mathcal{P}$

$C$ become strictly smaller (shown in Lemma 5.6). $\mathcal{P}$ is also reduced by half (line 5). If $|\mathcal{P}| \leq \gamma$ and the interaction does not achieve the stopping condition, $\mathcal{P}$ is initialized again (line 6-7). If $|\mathcal{P}| = 1$ after $\mathcal{P}$ is initialized, $|C| = 1$ based on Lemma 5.9 (line 3). We stop the interaction and return the user the finally left point in $C$ (line 8).

Theorem 5.10. *Algorithm* EDI *determine the user's favorite point in $O(c \log n)$ rounds, where $c$ is the number of times initializing $\mathcal{P}$.*

Proof. Since $|D| = n$, $\mathcal{P}$ is initialized with at most $n$ points. In each round, $\mathcal{P}$ can be reduced by half. $|\mathcal{P}|$ can be reduced to 1 ($\leq \gamma$) in $O(\log n)$ rounds. Since $\mathcal{P}$ will be initialized to be $c$ times, *EDI* can determine the user's favorite point in $O(c \log n)$ rounds. □

Corollary 5.11. *Algorithm* EDI *is asymptotically optimal in terms of the number of questions asked.*

### 5.3 Extension to k Nearest Points

In this section, we discuss the way to extend our proposed algorithms to return the user's favorite $k$ points. Our extension strategy is to iterate the algorithms $k$ times and output the tuples returned by each iteration. Recall that both *BS* and *EDI* maintain (1) a point set $C$ that stores the user's favorite point and (2) a polyhedron $\mathcal{R} \subseteq \mathcal{E}$ that contains the user's expected point. We maintain a point set $\mathcal{B}$ as the output. In the end of each iteration (i.e., when $|C| = 1$), we put the point in $C$ into $\mathcal{B}$. Then, we start a new iteration by keeping the polyhedron $\mathcal{R}$ of the last iteration and initializing $C$ to contain all the points in $D \setminus \mathcal{B}$ that are the nearest points of at least one expected point in $\mathcal{R}$. We continue this process until $|\mathcal{B}| = k$.

## 6 EXPERIMENT

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

**Datasets.** The experiments were conducted on synthetic and real datasets that are commonly used in existing studies [19, 27, 29]. Specifically, the synthetic datasets are *anti-correlated* [5] and the real datasets are *Car*, *House*, *NBA* and *Wine*. Dataset *Car* contains 20,186 cars after it is filtered by only keeping the cars whose attribute values are in a normal range. It has 3 ordered attributes (price, year of production and used mileage) and 3 unordered attributes (length, width, and number of seats). The category of an attribute is determined by considering whether there is a trivial user preference on the attribute values. For example, 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 few seats for fuel saving. Dataset *House* includes 21,067 houses described by 3 ordered attributes (price, grade, and built year) and 3 unordered attributes (number of bedrooms, number of bathrooms, and size). Dataset *NBA* involves 11,690 players after the records with missing values are deleted. Each player is described by 3 ordered attributes (point, rebound and assist) and 4 unordered attributes (usage, age, weight, and height). Dataset *Wine* contains 1,599 red wines that are described by 1 ordered attribute (quality) and 3 unordered attributes (fixed acidity, residual sugar, and alcohol). Note that existing studies [27, 29] preprocessed datasets to only contain skyline points $p$ (i.e., there does not exist a point $q$ 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 there does not exist a point $q$ which is closer than $p$ to any expected point in the expected space.

**Algorithms.** We evaluated our algorithms *DI*, *BS* and *EDI* with existing algorithms *ActiveRanking* [11], *UtilityApprox* [17], *UH-Random* [29], and *RH* [27]. The existing algorithms cannot solve our problem directly, since they are designed for datasets only described by ordered attributes. They model the user preference by a *utility function* and try to learn the utility function with the help of user interaction. In order to enable the existing algorithms to work on our problem IOU, we replaced the utility function with our distance function and followed their idea to select points as questions to interact with the user. Besides, we made a few adaptations for each of them as follows.

- Algorithm *ActiveRanking* learns the ranking of points by interacting with the user. We return the point that ranks the first after obtaining the full ranking.
- Algorithm *UtilityApprox* and algorithm *UH-Random* finds a point such that a criterion called the *regret ratio*, evaluating how "regretful" a user is when s/he sees the resulting point instead of the whole dataset, is minimized by interacting with the user. It stops the interaction when it can find a point whose regret ratio satisfies a given threshold $\epsilon$. We set $\epsilon = 0$, since this guarantees that the returned point is the user's favorite point.
- Algorithm *RH* returns one of the top-$k$ points (i.e., one of the user's favorite $k$ points) by interacting with the user. To obtain the user's favorite point, we set $k = 1$.

Note that [27, 29] also proposed algorithms *UH-Simplex* and *HD-PI*. However, both algorithms cannot be adapted to 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 each algorithm by varying different parameters: (1) parameter $\beta$, which is used in the priority shown in Section 5.1.2; (2) parameter $\gamma$, which decides 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 [27, 29], for each synthetic dataset, the dataset size was set to 100,000 (i.e., $n = 100,000$) and the number of ordered dimensions and unordered dimensions were set to 4 (i.e., $d_o = 4$ and $d_u = 4$).

**Performance Measurement.** We evaluated the performance of each algorithm 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*, which is the number of rounds interacting with the user; and (4) *candidate size*, which is the number of points in $C$ 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 generated distance functions (i.e., expected points), and the average performance was reported.

In the following, the parameter setting of our algorithms is studied in Section 6.1. The performance of algorithms on the synthetic and real datasets is presented in Section 6.2 and 6.3, respectively. In Section 6.4, a user study under a purchasing car scenario is demonstrated. Finally, the experiments are summarized in Section 6.5.

## 6.1 Performance Study of Our Algorithms

In Figure 9, we studied the parameter $\beta$, which is used in the *priority* in algorithm *BS* (shown in Definition 5.4), through evaluating the number of questions asked and the interaction time. The result shows that the two measurements are nearly the same with different $\beta$. When $\beta = 4$, the two measurements reach the smallest. Thus, we stick to $\beta = 4$ in the rest of our experiments.

Figure 10 shows our exploration on parameter $\gamma$, which decides the update of $\mathcal{P}$ in *EDI*. We varied $\gamma$ from 1 to 6 and evaluated the number of questions asked and the interaction time. Both measurements fluctuate slightly. The number of questions asked is around 18 and the interaction time is comparably shorter when $\gamma = 3$ and $\gamma = 4$. Thus, we set $\gamma = 4$ in the rest of our experiments.

## 6.2 Results on Synthetic Datasets

We studied our algorithm *DI* on the synthetic dataset with 1 ordered dimension and 1 unordered dimension. For completeness, our algorithms *BS* and *EDI*, and existing ones were also involved. In Figure 12, the shaded rectangles represent the preprocessing time. All the algorithms preprocess nearly the same time (*UtilityApprox* will be discussed later). Figure 11 shows the interaction time and the candidate size by varying the number of questions we can ask. It can be seen that all the algorithms are fast. They can finish within 0.1 seconds. Our algorithm *DI* performs the best among all the algorithms. It only takes $10^{-3}$ seconds. *BS* and *EDI* are also efficiently. Their interaction times are within $2 \times 10^{-3}$ seconds. Besides, our algorithms reduce the candidate size the most effectively. They reduce the candidate size to less than 3% after asking 3 questions.

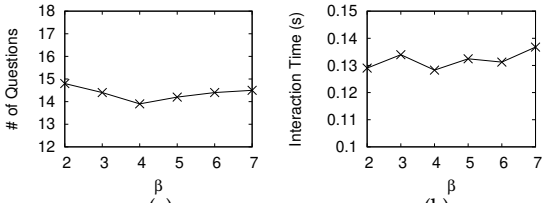
We compared our algorithms *BS* and *EDI* against existing algorithms on the synthetic datasets with 4 ordered dimensions and 4 unordered dimensions. Figure 12 and Figure 13 demonstrate the performance of all the algorithms. The preprocessing time of all the algorithms are short and close. Note that algorithm *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. Since the preprocessing time of different algorithms are close and it affects little to the user interaction, we do not show the preprocessing time later. The candidate size is shown in Figure 13(b). 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* does not reduce the candidate size effectively. After 4 questions were asked, there are still more than 21% of points left in $C$. In comparison, our algorithms only keep less than 4% of points. The interaction time of all the algorithms are shown in Figure 13(a). All the algorithms only take a few seconds. Our algorithms are the fastest except for *ActiveRanking*. As for *ActiveRanking*, it does not prune points that cannot be the user's favorite point during the interaction. This neglects the connection between points and results in asking more questions. Although our algorithms spend slightly more time, their interaction time 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.

We studied the scalability of *BS* and *EDI* on the number of dimensions with existing ones by varying the number of ordered dimensions $d_o$ and the number of unordered dimensions $d_u$, respectively. Each algorithm is measured by the interaction time and the number of questions asked. In Figure 14, we fixed $d_u = 4$ and increased $d_o$ from 3 to 6. The results show that all the algorithms are fast. They can finish within 3.4 seconds. Besides, our algorithms ask the least number of questions. When $d_o = 4$, they ask about 16 questions. *UH-Random* asks 19.4 questions, which is the smallest number among existing algorithms. In Figure 15, we fixed $d_o = 4$ and varied $d_u$ from 3 to 6. Our algorithms also perform the best in terms of the number of questions asked. When $d_u = 6$, *BS* and *EDI* ask 25 and 28.4 questions, respectively. In contrast, *UH-Random* that asks the least number of questions among existing algorithms needs 32 questions. Besides, the interaction time of *UH-Random* and *ActiveRanking* increase largely when $d_u$ increases. When $d_u = 6$, they run one order of magnitude longer than our algorithms.

In Figure 16, we evaluated the scalability of our algorithms on the data size $n$. *BS* and *EDI* scale the best in terms of the number of questions asked. For example, when $n = 10,000$, *BS* and *EDI* ask 13.2 and 14.6 questions, while the best existing algorithm *UH-Random* asks 17.3 questions. In particular, *ActiveRanking* asks 125.5 questions. Our algorithms also perform well on the interaction time. Their spend less than 0.6 seconds even if $n = 1,000,000$. In contrast, the interaction time of *UH-Random*, *UtilitApprox* and *ActiveRanking* increase dramatically with the increasing $n$. When $n \geq 100,000$, they take one order of magnitude longer than our algorithms. Note that *RH* is slightly faster than our algorithms, since it does not prune points during the interaction. However, this leads it to ask more questions. It asks around 3 times more questions than our algorithms. Although our algorithms spend slightly more time, their interaction time are still short (within 0.6 seconds) and they can obtain the user's favorite points by asking a few questions.

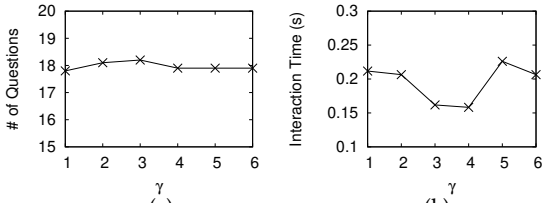
## 6.3 Results on Real Datasets

We studied the performance of our algorithms against existing algorithms, namely *ActiveRanking*, *UtilityApprox*, *UH-Random*, and *RH*, on 4 real datasets: *Car*, *House*, *NBA* and *Wine*. All the algorithms are fast and they take less than 0.1 seconds to execute. This is because that after the preprocessing, the number of points left in a real dataset is usually smaller than that in a synthetic dataset. Due to



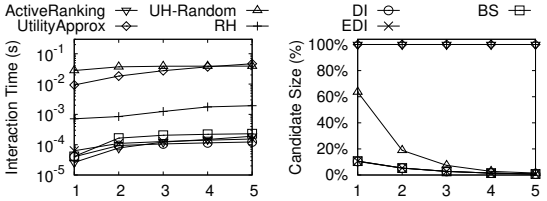

(a) (b)

**Figure 9: Parameter $\beta$**




(a) (b)

**Figure 10: Parameter $\gamma$**




(a) (b)

**Figure 11: 1 ordered & 1 unordered**

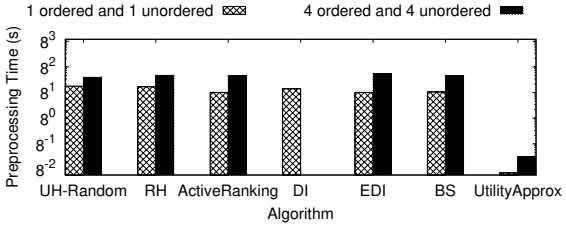
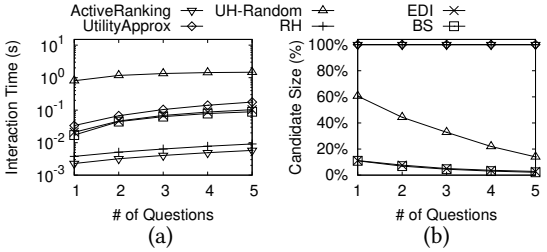
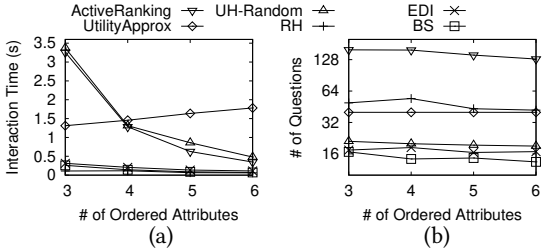



**Figure 12: Preprocessing Time**



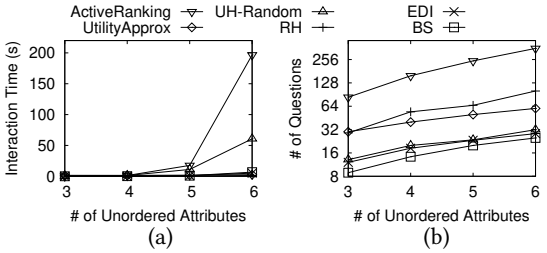

(a) (b)

**Figure 13: 4 ordered & 4 unordered**



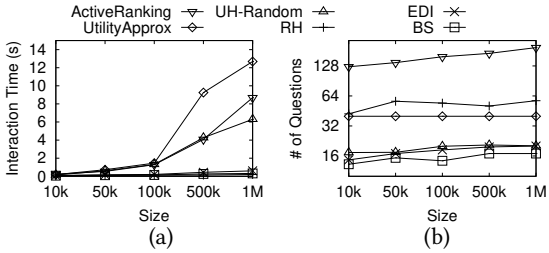

(a) (b)

**Figure 14: Vary $d_o$**




(a) (b)

**Figure 15: Vary $d_u$**




(a) (b)

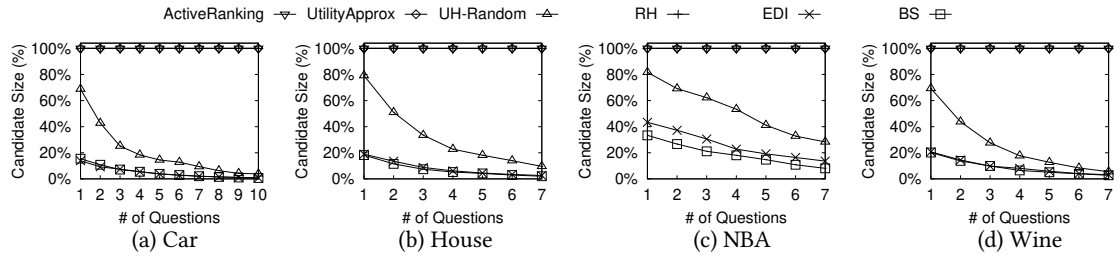
**Figure 16: Vary $n$**

the lack of space, we only show the candidate size of each algorithm. The interaction time of each algorithm can be found in the technical report [26]. Figure 17 presents the candidate size when we varied the number of questions we can ask. Our algorithms *EDI* and *BS* reduce the candidate size the most effectively. For example, on datasets *Car*, *House* and *Wine*, after asking 2 questions, our algorithms only contain less than 15% points in $C$, while the existing algorithms contain at least 42%. For dataset *NBA*, after 7 questions, the best existing algorithms contains 28% points in $C$. In comparison, our algorithms only contain less than 13%.

## 6.4 User Study

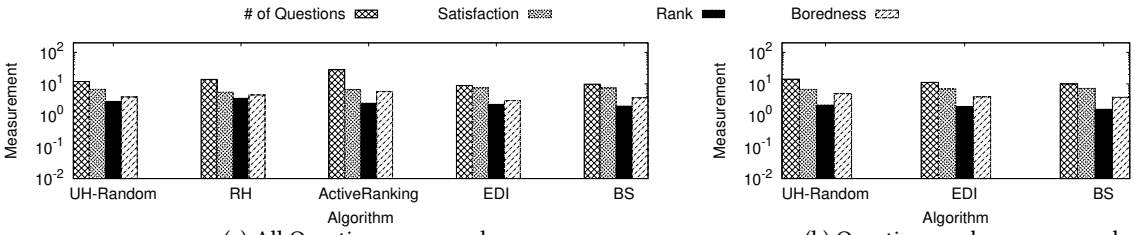
We conducted a user study on dataset *Car* to see (1) the impact of user mistakes on the final results, since users might make mistakes or provide inconsistent feedback during the interaction and (2) the impact of difficulty questions since there might be questions that users cannot answer. Following the setting in [20, 27, 29], 30 users were recruited and their average result was reported. We compared our algorithms *BS* and *EDI* against 3 existing algorithms, namely *Active-Ranking*, *UH-Random* and *RH*. We did not involve *UtilitApprox* since there are problems to apply it in real life scenarios [27, 29] as mentioned in Section 2. The adaptation of the existing algorithms shown previously was maintained. Each algorithm aims at finding the user's favorite car.

Each algorithm was measured via: (1) *The number of questions asked.* (2) *Degree of satisfaction* which is a score from 1 to 10 given by each user. It indicates how satisfied the user is with the returned car. (1 denotes the least satisfied and 10 means the most satisfied). (3) *Rank* which is the ranking given by each user. Since users sometimes gave the same degree of satisfaction for different algorithms, to distinguish them clearly, we asked each user to give a ranking of the algorithms. (4) *Degree of boredness* which is a score from 1 to 10




(a) Car (b) House (c) NBA (d) Wine

**Figure 17: Real Datasets**




(a) All Questions answered (b) Question can be unanswered

**Figure 18: User Study**

given by each user. It indicates 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 18(a) shows the results. The number of questions asked by *BS* and *EDI* are 9.8 and 8.9, 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 ranks the best. Their degree of satisfaction are 7.4 and 7.5, respectively. In comparison, the degree of satisfaction of existing algorithms are less than 6.7 and the least satisfying algorithm *RH* is 5.5, especially. Our algorithms also obtain the least boredness score. Their degree of boredness are 3.6 and 2.9, respectively, while the degree of boredness of existing algorithms are more than 3.8.

Figure 18(b) study the situation when the questions cannot be answered by users. Note that to reduce the workload of users, we only compare our algorithms *BS* and *EDI* with *UH-Random* which performs the best among existing algorithms as shown in Figure 18(a). We adapt the point selection of the 3 algorithms as follows when users cannot answer the question.

Algorithm *BS* selects two points whose hyper-plane has the highest priority in each interactive round. If the user cannot answer the question, we select the hyper-plane that has the second highest priority. Algorithm *EDI* maintains a candidate set $\mathcal{P}$ for point selection. If the user cannot answer the question, we initialize $\mathcal{P}$ again and select points as questions from the new $\mathcal{P}$. Algorithm *UH-Random* randomly selects two points as questions asked to a user in each interactive round. If the user cannot answer, we randomly select another two points.

The result shows that our algorithm *BS* and *EDI* perform better than *UH-Random* with respect to all the measurements. *BS* and *EDI* ask 10 and 11.2 questions, respectively, while *UH-Random* asks 14.2 questions. The degree of satisfaction and boredness of the 3 algorithms *BS*, *EDI* and *UH-Random* are 7.2, 6.9, 6.7 and 3.7, 3.9 4.9, respectively. Our algorithms also rank better than *UH-Random*. The results verify that our algorithms are affected little by the difficult questions compared with the existing algorithm.

## 6.5 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 large improvement on the interaction time and the number of questions asked (e.g., under typical settings, our algorithms ask up to 10 times fewer questions than existing algorithms). (2) Our algorithms scale well on the number of dimensions and the dataset size (e.g., our algorithm *BS* asks 25 questions in 6 seconds on the dataset with 3 ordered dimensions and 6 unordered dimensions, 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*, *House* and *Wine*, while the existing algorithms can only reduce to 42%). In summary, our algorithm *DI* asks the least number of questions in the shortest time for the special case of IOU. Our algorithms *BS* and *EDI* ask the least number of 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 attributes and unordered attributes. On the dataset with one ordered attribute and one unordered attribute, we propose algorithm *DI*, which is asymptotically optimal w.r.t. the number of questions asked. In general cases, two algorithms *BS* and *EDI* are presented, and they perform well w.r.t. the number of questions asked theoretically and empirically. Extensive experiments showed that our algorithms are both efficient and effective. They find the user's favorite tuple by asking a few questions within a short time. As for future work, we consider that users might make mistakes when answering questions.

# 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, 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 *Proceedings of the First International Conference on 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 System* 39, 2 (2014).

[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. https://doi.org/10.1109/TII.2017.2695583

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

[8] Evangelos Dellis and Bernhard Seeger. 2007. Efficient Computation of Reverse Skyline Queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria) *(VLDB '07)*. VLDB Endowment, 291–302.

[9] 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.

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

[11] 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.

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

[13] 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 (jan 2011), 729–765.

[14] 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.

[15] Alchemer LLC. 2021. https://www.alchemer.com/resources/blog/how-many-survey-questions/

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] QuestionPro. 2021. https://www.questionpro.com/blog/optimal-number-of-survey-questions/

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

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

[24] 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.

[25] Zhexuan Song and Nick Roussopoulos. 2001. K-Nearest Neighbor Search for Moving Query Point. In *Advances in Spatial and Temporal Databases*, Christian S. Jensen, Markus Schneider, Bernhard Seeger, and Vassilis J. Tsotras (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–96.

[26] Weicheng Wang and Raymond Chi-Wing Wong. 2021. *Interactive Mining with Ordered and Unordered Attributes*. Technical Report.

[27] 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.

[28] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. 194–205.

[29] 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.

[30] 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.

[31] 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.

## Appendix A EXPERIMENTS

Figure 19 shows the interaction time of all the algorithms when we varied the number of questions we can ask. All the algorithms run fast. Note that there are existing algorithms (e.g., *ActiveRanking*) spend less time than our algorithms. However, this only means that they process faster than our algorithms after each single question. As we can see in Figure 17, the existing algorithms ask more questions than our algorithms. Thus, the total interaction time of the existing algorithms are much longer than our algorithms.

## Appendix B TECHNIQUES

**Linear programming for Lemma 3.4** We define a variable $x$ and set the objective function to be $\max x$. For each point $q \in D \setminus \{p\}$, we build a constraint $(e - \frac{p+q}{2}) \cdot (p - q) > x$ (which is equal to $||p - e|| + x < ||q - e||$), where $e \in \mathcal{R}$. Formally, we construct a LP as follows.

$$\begin{aligned} &\text{maximize } x \\ &\text{subject to } (e - \frac{p+q}{2}) \cdot (p - q) > x \ for\ each\ q \in D \\ &\qquad e \in \mathcal{R} \end{aligned}$$

If the result $x < 0$, it implies that $\forall e \in \mathcal{R}$, $\exists q \in D \setminus \{p\}$ such that $||p - e|| > ||q - e||$. Point $p$ cannot be the nearest point of any $e \in \mathcal{R}$ and thus, it should be put out of consideration.

## Appendix C PROOF

**Proof of Theorem 3.2**. It is easy to see that there is a dataset $D$ such that each point $p \in D$ could be the user's favorite point, i.e., each point $p \in D$ could be the nearest point of some $e \in \mathcal{E}$. Any algorithm which utilizes pairwise comparison must identify all such 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 in the worst case. □

**Proof of Lemma 3.4**. If $\forall e \in \mathcal{R}$, $\exists q \in D$ such that $||p - e|| > ||q - e||$, i.e., $f(p) > f(q)$, point $p$ cannot be the nearest point of any expected point $e \in \mathcal{R}$. Thus, $p$ cannot be the user's favorite point and should be put out of consideration. □

**Proof of Lemma 3.5**. If $\exists q \in D$ such that $\forall e \in \mathcal{R}$, $||p - e|| > ||q - e||$, i.e., $f(p) > f(q)$, point $p$ cannot be the nearest point of any expected point $e \in \mathcal{R}$. Thus, $p$ cannot be the user's favorite point and should be put out of consideration. □

**Proof of Theorem 4.3**. We first show that our algorithm can obtain the fewest partitions and then analyze the time complexity.

Suppose there are $m$ partitions in the optimal case (i.e., the fewest partition 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, where $i \in [1, m]$. With a slight abuse of notation, 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 $\wedge'_{q_i,q_{i+1}}$. Assume that $\forall i \in [1, m]$, $\wedge_{q_i,q_{i+1}} \geq \wedge'_{q_i,q_{i+1}}$. Since $\wedge'_{q_m,q_{m+1}} = (1, 1)$, we have $\wedge_{q_m,q_{m+1}} \geq \wedge'_{q_m,q_{m+1}} = (1, 1)$. Thus, the number of partitions obtained by our algorithm will not be more than that of the optimal case. In the following, we are to show 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.

Consider $i = 1$. Trivially, $\wedge'_{q_0,q_1} = \wedge_{q_0,q_1} = (1, 0)$. For the first partition $\Theta'_{q_1}$ of the optimal case, its corresponding point $q'_1$ should be the nearest point of any $e \in \Theta'_{q_1} = [\wedge'_{q_0,q_1}, \wedge'_{q_1,q_2}] = [\wedge_{q_0,q_1}, \wedge'_{q_1,q_2}]$. Thus, the corresponding point $q_1$ of the first partition $\Theta_{q_1} = [\wedge_{q_0,q_1}, \wedge_{q_1,q_2}]$ obtained by our algorithm should be $q_1 = q'_1$. Note that our algorithm continually extends the first partition $\Theta_{q_1}$ as long as its corresponding point $q_1$ is the nearest point of any $e \in \Theta_{q_1}$. We have $\wedge_{q_1,q_2} \geq \wedge'_{q_1,q_2}$.

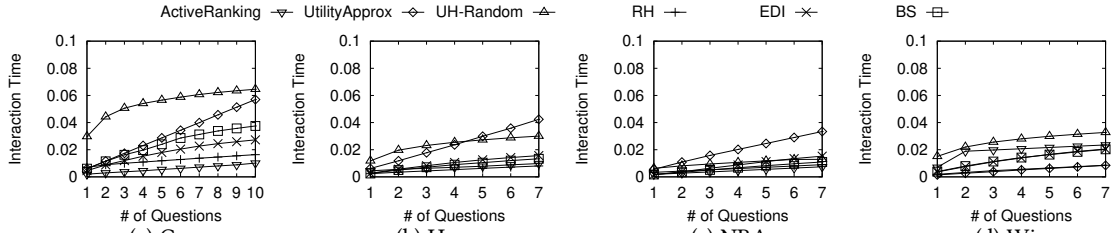
Consider $i \geq 2$. Suppose that $\wedge_{q_{i-1},q_i} \geq \wedge'_{q_{i-1},q_i}$ is true for the $(i-1)$-th partition. For the $i$-th partition $\Theta'_{q_i}$ of the optimal case, its corresponding point $q'_i$ should be the nearest point of any $e \in \Theta'_{q_i} = [\wedge'_{q_{i-1},q_i}, \wedge'_{q_i,q_{i+1}}]$. Since $\wedge_{q_{i-1},q_i} \geq \wedge'_{q_{i-1},q_i}$, $q'_i$ must be the nearest point of any $e \in [\wedge_{q_{i-1},q_i}, \wedge'_{q_i,q_{i+1}}]$. Thus, the corresponding point $q_i$ of the $i$-th partition $\Theta_i = [\wedge_{q_{i-1},q_i}, \wedge_{q_i,q_{i+1}}]$ obtained by our algorithm should be $q_i = q'_i$. Because $\Theta_{q_i}$ is continually extended as long as its corresponding point $q_i$ is the nearest point of any $e \in \Theta_{q_i}$, we have $\wedge_{q_i,q_{i+1}} \geq \wedge'_{q_i,q_{i+1}}$.

Therefore, $\wedge_{q_i,q_{i+1}} \geq \wedge'_{q_i,q_{i+1}}$ is true for each partition $\Theta_{q_i}$, where $i \in [1, m]$. Based on our previous discussion, our algorithm divides the expected space into the fewest partitions.

Now, we move to the analysis of the time complexity. Since there are $n$ points, we first need $O(n \log n)$ time to sort all the points. Then, consider the step that we add point $p_i$ to update the division of the expected space. We need $O(1)$ time to scan each of the points $< q_k, q_{k-1}, ..., q_{l+1} >$. This can be charged to the replacements of $< q_k, q_{k-1}, ..., q_{l+1} >$. We also spend $O(1)$ time at $q_l$. This can be charged to the addition of $p_i$. Since a point is added and replaced at most once in the process, we need $O(1)$ time for each point. Because there are $n$ points, we need $O(n)$ time. Thus, The expected space can be divided into partitions in $O(n \log n)$ time in total. □

**Proof of Theorem 4.5**. Let $< \Theta_{q_1}, \Theta_{q_2}, ..., \Theta_{q_m} >$ be the division of the expected space decided by $S_n$ from bottom to top with their corresponding points $< q_1, q_2, ..., q_m >$. For each partition $\Theta_{q_j} = [\wedge_{q_{j-1},q_j}, \wedge_{q_j,q_{j+1}}]$, where $j \in [1, m]$, $q_j$ is the nearest point of any $e \in \Theta_{q_j}$. Since $\wedge_{q_{j-1},q_j}$ is the intersection of hyperplane $h_{q_{j-1},q_j}$ and $\mathcal{E}$, $[\wedge_{q_0,q_1}, \wedge_{q_{j-1},q_j}] \subseteq h^+_{q_{j-1},q_j}$ and thus, $q_j$ will not be the nearest point of any $e \in [\wedge_{q_0,q_1}, \wedge_{q_{j-1},q_j}]$. Similarly, Since $\wedge_{q_j,q_{j+1}}$ is the intersection of hyper-plane $h_{q_j,q_{j+1}}$ and $\mathcal{E}$, $[\wedge_{q_j,q_{j+1}}, \wedge_{q_m,q_{m+1}}] \subseteq h^-_{q_j,q_{j+1}}$ and thus, $q_j$ will not be the nearest point of any $e \in [\wedge_{q_j,q_{j+1}}, \wedge_{q_m,q_{m+1}}]$. Therefore, each point can only correspond to one partition. Since $|D| = n$, there will be at most $n$ partitions and thus, there will be at most $n$ corresponding points. The candidate set $C$ could be initialized to contain at most $n$ points. The worst case is achieved when all the points in the dataset are the same in the first dimension, i.e., $\forall p, q \in D$, $p[1] = q[1]$. The expected space could be divided into $n$ partitions and each point in $D$ is the nearest point of the expected point in one of the partitions.

In each interactive round, our algorithm *DI* presents to the user with the middle points $q_j$ and $q_{j+1}$ to the user. Based on the user feedback, points $< q_1, q_2, ..., q_j >$ or $< q_{j+1}, q_{j+2}, ..., q_m >$ can be deleted. Thus, $C$ can be reduced by half. Since there are at most $n$ points in $C$ initially, $C$ can be reduced to 1 in $O(\log n)$ rounds. □



(a) Car (b) House (c) NBA (d) Wine

**Figure 19: Real Datasets (Interaction Time)**

**Proof of Corollary 4.6**. As shown in Theorem 3.2, the lower bound of the number of questions asked is $\Omega(\log n)$. Since algorithm *DI* determines the user's favorite point by asking $O(\log n)$ questions, it is asymptotically optimal in terms of the number of questions asked. □

**Proof of Lemma 5.1**. We are to find the expected points $e$ such that $\forall q \in D \setminus \{p\}$, $p$ is closer to $e$ than $q$, i.e., $e \in \bigcap_{q \in D \setminus \{p\}} h^+_{p,q}$. In geometry, the intersection of a set of half-spaces is a polyhedron or an empty space [7]. Thus, there is at most one maximal polyhedron $\mathcal{E}_p \subseteq \mathcal{E}$ such that $p$ is the nearest point of any $e \in \mathcal{R}_p$. □

**Proof of Lemma 5.8**. Based on our scanning strategy, for each point $q_j$, where $j \in [l+1, k]$, $\wedge_{p_i,q_j}$ is closer to $e_1$ than $\wedge_{q_{j-1},q_j}$, and thus, $\Theta_j \subseteq h^+_{p_i,q_j}$. Figure 7 is a such case when there are two unordered dimensions. Since $\Theta_j \subseteq h^+_{p_i,q_j}$, $p_i$ must be closer to any $e \in \Theta_j$ than $q_j$. Since $q_j$ is the nearest point of any $e \in \Theta_j$ among $S_{i-1}$, point $p_i$ must be the nearest point of any $e \in \Theta_j$ among $S_i$. Thus, $p_i$ is the nearest point of any $e \in [\wedge_{q_l,q_{l+1}}, \wedge_{q_k,e_2}]$ among $S_i$.

Since $p_i$ is the nearest point of any $e \in \Theta_{l+1}$ among $S_i$, $\wedge_{p_i,q_l}$ must be closer to $e_1$ than $\wedge_{q_l,q_{l+1}}$. Figure 8 shows such case when there are two unordered dimensions. Note that $\wedge_{p_i,q_l}$ is farther away from $e_1$ than $\wedge_{q_{l-1},q_l}$. This means that (1) $[\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}] \subseteq h^+_{p_i,q_l}$ and (2) $[e_1, \wedge_{p_i,q_l}] \subseteq h^-_{p_i,q_l}$. As for (1), since $q_l$ is the nearest point of any $e \in \Theta_l$ among $S_{i-1}$, $p_i$ must be the nearest point of any $e \in [\wedge_{p_i,q_l}, \wedge_{q_l,q_{l+1}}]$ among $S_i$. As for (2), since $q_l$ is closer to any $e \in [e_1, \wedge_{p_i,q_l}]$ than $p_i$, $p_i$ cannot be the nearest point of any $e \in [e_1, \wedge_{p_i,q_l}]$ among $S_i$. The nearest point of any $e \in [e_1, \wedge_{p_i,q_l}]$ among $S_i$ is the same as that among $S_{i-1}$. □

**Proof of Lemma 5.9**. Let $E = \{e_1, e_2, ..., e_m\}$ be the set of the extreme point of $\mathcal{R}$. Since $\mathcal{R}$ is convex, for any expected point $e \in \mathcal{R}$, there exist real numbers $a_1, a_2, ..., a_m$ such that $e = a_1e_1 + a_2e_2 + ... + a_me_m$ and $a_1 + a_2 + ... + a_m = 1$. Suppose $\forall e_i \in E$, point $p \in C$ is the nearest point of $e_i$, i.e., $\forall q \in C \setminus \{p\}$, $\sum_{j=1}^d (p[j] - e_i[j])^2 \geq \sum_{j=1}^d (q[j] - e_i[j])^2$. Thus, $\forall e \in \mathcal{R}$, we have

$$
\begin{aligned}
&\sum_{j=1}^{d}(p[j]-e[j])^2 - \sum_{j=1}^{d}(q[j]-e[j])^2 \\
&= \sum_{j=1}^{d}(p[j]-\sum_{i=1}^{m}a_ie_i[j])^2 - \sum_{j=1}^{d}(q[j]-\sum_{i=1}^{m}a_ie_i[j])^2 \\
&= \sum_{j=1}^{d}(p[j]^2 - 2p[j]\sum_{i=1}^{m}a_ie_i[j] + (\sum_{i=1}^{m}a_ie_i[j])^2) \\
&- \sum_{j=1}^{d}(q[j]^2 - 2q[j]\sum_{i=1}^{m}a_ie_i[j] + (\sum_{i=1}^{m}a_ie_i[j])^2) \\
&= \sum_{j=1}^{d}(p[j]^2 - 2p[j]\sum_{i=1}^{m}a_ie_i[j]) - \sum_{j=1}^{d}(q[j]^2 - 2q[j]\sum_{i=1}^{m}a_ie_i[j]) \\
&= \sum_{j=1}^{d}(\sum_{i=1}^{m}a_i(p[j]-e_i[j])^2 - \sum_{i=1}^{m}a_ie_i[j]^2) \\
&- \sum_{j=1}^{d}(\sum_{i=1}^{m}a_i(q[j]-e_i[j])^2 - \sum_{i=1}^{m}a_ie_i[j]^2) \\
&= \sum_{j=1}^{d}(\sum_{i=1}^{m}a_i(p[j]-e_i[j])^2) - \sum_{j=1}^{d}(\sum_{i=1}^{m}a_i(q[j]-e_i[j])^2) \\
&= \sum_{i=1}^{m}a_i(\sum_{j=1}^{d}(p[j]-e_i[j])^2 - \sum_{j=1}^{d}(q[j]-e_i[j])^2) \\
&\geq 0
\end{aligned}
$$

This means $\sqrt{\sum_{j=1}^d (p[j]-e[j])^2} - \sqrt{\sum_{j=1}^d (q[i]-e[j])^2} \geq 0$, i.e., point $p \in C$ is the nearest point of any expected point in $\mathcal{R}$. Based on our prune strategy, we could prune points in $C$ so that $|C| = 1$. This violates the claim that $|C| > 1$.

Therefore, if $|C| > 1$, there must exist at least two extreme points $e_1, e_2 \in \mathcal{R}$ such that their nearest points are different. Line segment $[e_1, e_2]$ will be divided into at least two partitions and thus, there are at least two corresponding points inserted into $\mathcal{P}$. □

**Proof of Corollary 5.11**. As shown in Theorem 3.2, the lower bound of the number of questions asked is $O(\log n)$. Since algorithm *EDI* determines the user's favorite point in $O(c \log n)$ rounds, where $c$ is a very small number, it is asymptotically optimal in terms of the number of questions asked. □