Interactive Search with Reinforcement Learning

Weicheng Wang[1], Victor Junqiu Wei[1], Min Xie[2], Di Jiang[3], Lixin Fan[3], Haijun Yang[3]
[1]*The Hong Kong University of Science and Technology* [2]*Shenzhen Institute of Computing Sciences* [3]*Webank*
{wwangby@connect., victorwei@}ust.hk xiemin@sics.ac.cn {dijiang@, lixinfan@, navyyang@}webank.com

***Abstract*—The interactive regret query is one of the most representative multi-criteria decision-making queries. It identifies tuples that satisfy users' preferences via iterative user interaction. In each interactive round, it asks users a question to learn about their preferences. Once the users' preferences are sufficiently learned, it returns tuples based on the learned preferences. Nevertheless, existing algorithms for this query are typically short-term focused, i.e., they ask questions by only considering each individual interactive round, without taking the overall interaction process as a whole. This may harm the long-term benefit, leading to a large number of rounds in the overall process. To address this, we propose two algorithms based on reinforcement learning, aiming to effectively improve the overall interaction process.**

We first formalize the interactive regret query as a Markov Decision Process. Then, we propose two interactive algorithms, namely EA and AA, which utilize reinforcement learning to learn a good policy for selecting questions during the interaction. Both algorithms are optimized not only for the current interactive round but also for the overall interaction process, with the goal of minimizing the total number of questions asked (i.e., the total number of interactive rounds). Extensive experiments were conducted on synthetic and real datasets, showing that our algorithms reduce the number of questions asked by approximately 50% compared to existing ones under typical settings.

***Index Terms*—query optimization, reinforcement learning**

I. Introduction

One crucial role of database systems is to assist users in searching for tuples in the database that align with their preferences, such as finding a car, admitting students, or renting an apartment [1]–[4]. Consider the scenario where a user Alice wants a new car in the market. The car database, as illustrated in Table I, contains numerous cars described by several attributes, e.g., price, horsepower, and fuel efficiency. Without assistance, Alice would face the daunting task of manually sifting through these cars, making it difficult for her to locate the desired ones. If a database system could help to find cars that fit Alice's preference, it would greatly alleviate her burden.

In the literature, there are two traditional queries: the top-k query [5], [6] and the skyline query [7]. Both model the user's preference by a *utility function* $f_{\boldsymbol{u}}$. Then, each tuple $\boldsymbol{p}$ is assigned a *utility* $f_{\boldsymbol{u}}(\boldsymbol{p})$ (i.e., a function score), indicating how well this tuple aligns with the user's preference. The top-k query returns the k tuples with the highest utilities, e.g., if $k = 1$, car $\boldsymbol{p_4}$ is returned from Table I. However, this query assumes that users can explicitly state their utility functions, which is often unrealistic in practice [8], [9]. The skyline query avoids this by adopting the notion of "domination". A tuple $\boldsymbol{p}$ is said to *dominate* another tuple $\boldsymbol{q}$ if $\boldsymbol{p}$ is not worse than $\boldsymbol{q}$ on each attribute and $\boldsymbol{p}$ is better than $\boldsymbol{q}$ on at least one attribute. Tuples that are not dominated by any other tuples are returned. Intuitively, the skyline query considers all possible utility functions and returns those tuples that have the highest utilities w.r.t. at least a utility function. Nevertheless, its output size is uncontrollable [8], [10], e.g., in Table I, all cars are not dominated by any other cars, and thus, they are returned.

Table I: The Car Database

Car	Price	Horsepower	Fuel Efficiency	$f_{\boldsymbol{u}}(\cdot)$
$\boldsymbol{p}_1$	\$5000	450 hp	20 mpg	5.0
$\boldsymbol{p}_2$	\$4000	400 hp	30 mpg	6.0
$\boldsymbol{p}_3$	\$6000	500 hp	22 mpg	4.0
$\boldsymbol{p}_4$	\$3500	350 hp	28 mpg	6.5
$\boldsymbol{p}_5$	\$4500	440 hp	29 mpg	5.5
$\boldsymbol{p}_6$	\$5500	480 hp	25 mpg	4.5

The interactive regret query [9] addresses these limitations. Given a database, it interacts with a user to learn the user's utility function (without explicitly stating it) and returns one tuple from the database based on the learned utility function (a fixed output size). Specifically, it engages a user in a series of interactive rounds. In each round, the user answers a question by selecting the one s/he prefers from a pair of presented tuples $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$, where $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are from the database. For example, in Figure 1, Alice is presented with cars $\boldsymbol{p}_5$ and $\boldsymbol{p}_6$ from Table I. Based on her answer (car $\boldsymbol{p}_5$), Alice's utility function is learned implicitly. If Alice consistently selects cars with lower prices, it can be inferred that price plays a dominant role in her utility function. Once the utility function is sufficiently learned, the desired tuple w.r.t. the learned utility function is returned, e.g., the cheapest car $\boldsymbol{p}_4$ may be returned to Alice.

Figure 2 visualizes this process in a tree structure (see more in Section IV-A). Each internal node, denoted by N, represents an interactive round. It is associated with multiple rectangles, each of which denotes a question that can be asked in this interactive round. Note that all pairs of tuples in the database can be the candidate questions. We only show two here for better visualization. Each rectangle is connected to two child nodes, which correspond to the user's answers to the question. For instance, the dashed-grid rectangle in the root node N_1 denotes question $\langle \boldsymbol{p}_5, \boldsymbol{p}_6 \rangle$. It is connected to two child nodes, where the left child node N_2 indicates that the user prefers $\boldsymbol{p}_5$ to $\boldsymbol{p}_6$, denoted by $\boldsymbol{p}_5 \succ \boldsymbol{p}_6$; similarly for the right child node N_3. At each internal node, users can be presented with different questions or give different answers, ultimately reaching different child nodes. The interaction ends when it reaches a leaf node, and the tuple in the leaf is returned, e.g., the interaction in Figure 1 corresponds to the bold path in Figure 2.

The importance of optimizing this process is evident. In

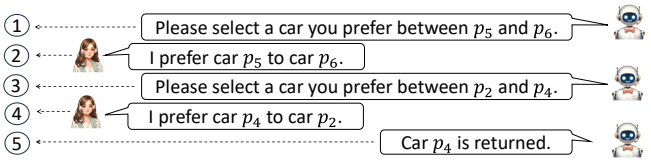


Figure 1: Interaction Process

marketing research [11]–[13], it is crucial to keep the number of questions asked to the user on a manageable scale. If users are burdened with excessive questions, they may lose focus and become frustrated, which can negatively affect the interaction. For instance, recall the scenario of purchasing a car. It would likely lead to dissatisfaction and disengagement if Alice needs to answer hundreds or even thousands of questions.

There are various algorithms for the interactive regret query: UH-Random [9], UH-Simplex [9], and SinglePass [14]. They define several designated data structures to store the information collected during the interaction. In each interactive round, based on the information stored, a question is selected from the candidate ones for interaction. As could be noticed, these algorithms are *short-term* focused, i.e., they select a question only for the current individual round rather than considering the *long-term* implications, i.e., the impact of the selected question on subsequent interaction. This short-term focus, while may be useful to individual rounds, can negatively affect subsequent rounds and hinder the entire interaction process.

To see this, suppose there are two different initial questions at the root node in Figure 2, namely $\mathcal{Q}_1 = \langle \boldsymbol{p}_5, \boldsymbol{p}_6 \rangle$ and $\mathcal{Q}_2 = \langle \boldsymbol{p}_3, \boldsymbol{p}_6 \rangle$ that can be presented to a user. One may observe that both questions seem equally potent for interaction if we only consider their short-term effects in this individual round, since they both lead to other internal nodes. However, these two questions result in different subsequent interactions (i.e., the long-term effects), where the sub-tree with dashed-grid (resp. white) rectangles corresponds to the subsequent interactions resulting from $\mathcal{Q}_1$ (resp. $\mathcal{Q}_2$). As can be noticed, the sub-tree with white rectangles is generally deeper. This indicates that if $\mathcal{Q}_2$ is selected for interaction (as all existing algorithms do), there will be more questions asked to the user in the future.

Motivated by this limitation, we develop two algorithms, an exact algorithm (EA) and an approximate algorithm (AA), to incorporate the long-term needs for the interactive regret query. We first formalize the interaction process using a notion of *interaction tree*, and model it as a sequential decision process: in each interactive round, the algorithm makes a decision by selecting a question from the candidate ones for interaction. Then, we model this process as a Markov Decision Process (MDP) and use reinforcement learning (RL) to optimize it.

RL is an effective method for long-term optimization. In its framework, each decision made is evaluated by a reward, and the objective is to maximize the accumulated reward over the entire process. We can correlate the number of questions with the reward so that if a large number of questions are asked, the accumulated reward is small, thereby focusing on the entire interaction process rather than just individual interactive rounds.

Our algorithms EA and AA have different priorities. Algorithm EA is an exact algorithm that returns a tuple strictly close to the user's favorite one. It gathers comprehensive information from a geometric perspective in each interactive round to make a decision. Intuitively, it maintains a polyhedron to describe the information collected during the interaction. The use of polyhedrons has been proven effective by existing studies [3], [9], [15]. By leveraging this formulation, algorithm EA identifies the desired tuple in a small number of interactive rounds.

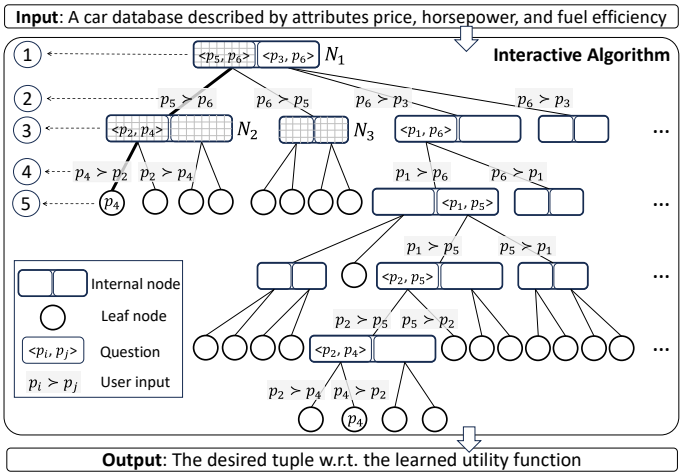


Figure 2: Interaction Tree

Although a polyhedron can precisely describe the information collected from the interaction, it can be costly to compute, especially when there are many attributes. Motivated by this, we propose an approximate algorithm AA, which avoids the computation of polyhedrons. It only gathers essential information in each interactive round in exchange for a low computational cost. This design allows algorithm AA to be more scalable and capable of managing datasets with many attributes effectively. As shown in Section V, it is able to handle datasets with 4-5 times more attributes than the SOTA algorithm.

To the best of our knowledge, we are the first to propose interactive algorithms based on RL for the interactive regret query. Our contributions are summarized below:

- We propose a notion of interaction tree to formalize the process of the interactive regret query as a Markov Decision Process. This establishes the foundation for designing interactive algorithms based on reinforcement learning (RL).
- We present an exact algorithm EA and an approximate algorithm AA based on RL. EA significantly reduces the number of interactive rounds. AA scales well with many attributes.
- Extensive experiments were conducted on both synthetic and real datasets. The results show that compared with the best-known existing algorithms, our algorithms not only reduce the number of interactive rounds significantly, but also run faster by an order of magnitude under *typical* settings (i.e., the settings commonly adopted by existing work [9], [14]).

The rest of this paper is organized as follows. Section II discusses the related work. Section III formally defines our problem. Algorithms EA and AA are presented in Section IV. Section V demonstrates our experimental results, and finally, Section VI concludes the paper with possible future work.

II. Related Work

A. Interactive Algorithm

The interactive regret query defines a criterion called *regret ratio*, which evaluates how regretful a user is when s/he sees the returned tuple instead of the whole database. Its goal is to interact with a user to learn the user's preference, and then return a tuple whose regret ratio is smaller than a given threshold. There are a few interactive algorithms for this query [8], [9], [14]. The first algorithm UtilityApprox is proposed by [8]. It constructs several artificial tuples in each interactive round and asks the user to tell which one s/he prefers. In this way, it can design tuples specialized to learn the user's preference. However, since the tuples used for interaction are *fake* (i.e., tuples not selected from the database), they might be unrealistic (e.g., a car with 10 dollars and 50000 horsepower). The user would be disappointed if the tuples displayed in the interaction with which s/he is satisfied do not exist [9].

To overcome this, [9] proposes algorithms UH-Random and UH-Simplex that utilize *real* tuples (i.e., tuples selected from the database) for interaction. Intuitively, in each interactive round, UH-Random randomly selects tuples from the database to generate a question, while UH-Simplex selects tuples from the database that are likely to be the best according to some criteria. However, both algorithms involve complicated computation, limiting their applicability to datasets with a few attributes. To accelerate, [14] proposes algorithm SinglePass, at the cost of collecting less information in each interactive round. Consequently, it needs to ask many questions (e.g., hundreds of questions). Besides, as remarked early, all these algorithms design each question by only focusing on the current interactive round without considering the overall interaction process. This leads to many questions and long execution times.

There are relevant interactive algorithms designed for other queries. For instance, [16] proposes an algorithm that focuses on learning user preferences through user interaction. Nevertheless, it prioritizes the derivation of the user's preference rather than returning desired tuples, which may lead to unnecessary questions. For example, if Alice prefers car $\boldsymbol{p}_1$ to both $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$, her preference between $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ is less interesting in our problem, but this additional comparison might be useful in [16]. Moreover, [3], [15] propose algorithms that return a tuple whose utility is the highest or among the top-k. However, these algorithms focus on the ranking of tuples, which is the secondary information derived from their utilities. In contrast, our algorithms directly focus on the utility difference between the returned tuple and the best tuple in the database.

In machine learning, the interactive regret query is closely related to the problem of *learning to rank* [17]–[20], which learns the ranking of tuples by interacting with the user. However, most of the existing algorithms [17]–[19] often overlook the inter-relations in attributes (e.g., recognizing that a price of \$200 is more desirable than \$500 because it is cheaper), thus necessitating a large number of questions. [20] considers the inter-relation to learn the ranking of tuples, but it still asks unnecessary questions due to the same reason as stated for [16].

B. Reinforcement Learning

Reinforcement learning (RL) is proposed to guide agents to decide what actions to take in specific environments [21], [22], where the environment is generally modeled as a Markov Decision Process (MDP). The objective is to maximize the cumulative reward, where each reward is obtained after an action is taken. In recent years, RL has been successfully applied to various problems, such as index construction [23], [24] and trajectory simplification [25], [26] in the database community.

In the area of index construction, [23] leverages RL to learn a good policy for the QD-Tree to make partitioning decisions. The objective is to maximize the data-skipping ratio for a given query workload. Moreover, [24] utilizes RL to improve the traditional spatial index R-Tree. It proposes two RL agents. One is to decide which sub-tree to insert when a new tuple is added, and the other one is to determine how to split the node.

For trajectory simplification, [25] uses a fixed-size sliding window over the trajectory and trains an RL agent to decide which point within the window should be deleted. This mechanism allows for an online simplification process, ensuring that only relevant points are retained. [26] also employs two RL agents, which identify a set of candidate points for deletion and make the final decision on which points to delete, respectively.

To the best of our knowledge, we are the first to model the interactive regret query as a MDP and use RL to optimize it.

III. Problem Definition

The input is a set $\mathcal{D}$ of n tuples. Each tuple is described by d attributes and can be regarded as a d-dimensional point $\boldsymbol{p} = (p[1], p[2], ..., p[d])$, where $p[i]$ ($i \in [1, d]$) represents the i-th attribute value of $\boldsymbol{p}$. Without loss of generality, we assume that each dimension is normalized to $(0, 1]$ and a large value is preferred [9], [27]. Table III shows an example, where $\mathcal{D}$ has five 2-dimensional points. In the following, we use the words "tuple/point" and "attribute/dimension" interchangeably. The frequently used notations are summarized in Table II.

Utility Function. Following [9], [15], [28], we model the user preference by a linear function $f_{\boldsymbol{u}}$, called the *utility function*.

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

- $\boldsymbol{u} = (u[1], ..., u[d])$, called *utility vector*, is a d-dimensional non-negative vector. Each $u[i]$ represents the importance of the i-th attribute to the user ($i \in [1, d]$), and a large value means the attribute is important. The domain of $\boldsymbol{u}$ is called the *utility space* and denoted by $\mathcal{U}$. Without loss of generality, following [8], [29], we assume that $\sum_{i=1}^{d} u[i] = 1$.
- Function value $f_{\boldsymbol{u}}(\boldsymbol{p})$ is called the *utility* of $\boldsymbol{p}$ w.r.t. $\boldsymbol{u}$. It represents to what extent a user prefers $\boldsymbol{p}$. A high utility means that $\boldsymbol{p}$ is preferred by the user, and the point with the highest utility is regarded as the user's favorite point.

Example 1. Consider Table III where the utility function is $f_{\boldsymbol{u}}(\boldsymbol{p}) = 0.3p[1] + 0.7p[2]$, i.e., utility vector $\boldsymbol{u} = (0.3, 0.7)$. The utility of $\boldsymbol{p}_3$ w.r.t. $\boldsymbol{u}$ is $f_{\boldsymbol{u}}(\boldsymbol{p}_3) = 0.3 \times 0.5 + 0.7 \times 0.8 = 0.71$. The utilities of other points are computed similarly. Point $\boldsymbol{p}_3$ with the highest utility is the user's favorite point.

Regret Ratio [9], [29]. Given a dataset $\mathcal{D}$ and a utility function $f_{\boldsymbol{u}}$, the regret ratio of a point $\boldsymbol{q} \in \mathcal{D}$ over $\mathcal{D}$ w.r.t. $f_{\boldsymbol{u}}$ is:

$$regratio(\boldsymbol{q}, \boldsymbol{u}) = \frac{max_{\boldsymbol{p}\in\mathcal{D}} f_{\boldsymbol{u}}(\boldsymbol{p}) - f_{\boldsymbol{u}}(\boldsymbol{q})}{max_{\boldsymbol{p}\in\mathcal{D}} f_{\boldsymbol{u}}(\boldsymbol{p})}$$

Intuitively, the regret ratio of $\boldsymbol{q}$ measures the proportional difference between the highest utility in set $\mathcal{D}$ and the utility of point $\boldsymbol{q}$. If $regratio(\boldsymbol{q}, \boldsymbol{u})$ is below a small threshold, the utility of $\boldsymbol{q}$ is close to the highest utility in the dataset. In this case, we regard point $\boldsymbol{q}$ to be comparable to the user's favorite point.

Example 2. Continue Example 1, where $\boldsymbol{u} = (0.3, 0.7)$. The regret ratio of $\boldsymbol{p}_2$ is $regratio(\boldsymbol{p}_2, \boldsymbol{u}) = \frac{0.71-0.58}{0.71} = 0.18$.

Problem. We now present our problem *Interactive Search with Reinforcement Learning*. The goal is to interact with a user to find a point $\boldsymbol{q} \in \mathcal{D}$ whose regret ratio w.r.t. the user is smaller than a threshold ϵ. Following the framework in [9], [15], the interaction works in *rounds*. Each round has three components.

- *Question Selection.* The interactive agent adaptively selects one question from the candidate questions for interaction.
- *Information Maintenance.* Based on the user's answer, we learn the user's preference implicitly and update the information maintained to find the point with a small regret ratio.
- *Stopping Condition.* If the stopping condition is satisfied, we stop the interaction and return a point based on the information maintained. Otherwise, we start another round.

Our objective is to develop an *interactive agent* with the help of reinforcement learning. This agent guides the question selection in each interactive round to minimize the total number of questions asked, i.e., the total number of interactive rounds.

Problem 1. (**Interactive Search with Reinforcement Learning, ISRL**) Given a dataset $\mathcal{D}$ and a threshold ϵ, we aim to develop an interactive agent with reinforcement learning. The goal of the agent is to interact with a user by as few interactive rounds as possible to find a point in $\mathcal{D}$ whose regret ratio w.r.t. the user's utility vector is smaller than the given threshold ϵ.

IV. Algorithms

In this section, we first give an overview with some preliminaries in Section IV-A. Then, we present two algorithms for problem ISRL in Sections IV-B and IV-C, respectively. The first algorithm is an exact algorithm, while the second one is an approximate algorithm that achieves better scalability.

A. Overview

We explore problem ISRL from a geometric perspective.

Utility Space. Each utility vector $\boldsymbol{u}$ can be seen as a point in a d-dimensional geometric space $\mathbb{R}^d$. Recall that we assume (1) $u[i] \geq 0$ for each dimension and (2) $\sum_{i=1}^{d} u[i] = 1$. The utility space $\mathcal{U}$, i.e., the domain of $\boldsymbol{u}$, can be seen as a *polyhedron* [30] in space $\mathbb{R}^d$. For example, in the 3-dimensional space $\mathbb{R}^3$, as shown in Figure 5, the utility space $\mathcal{U}$ is a triangle.

Hyper-plane. In a d-dimensional space $\mathbb{R}^d$, we can build a *hyper-plane* [30] for each pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points in $\mathcal{D}$.

$$h_{i,j} = \{\boldsymbol{r} \in \mathbb{R}^d \mid \boldsymbol{r} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j) = 0\}$$

Table II: Frequently Used Notations

Notation	Definition
$\mathcal{D}$ and $\boldsymbol{p}$	A dataset of n tuples and a d-dimensional point
$f_{\boldsymbol{u}}$ / $\boldsymbol{u}$ and $\mathcal{U}$ / $\mathcal{R}$	The utility function / vector and the utility space / range
$regratio(\boldsymbol{p}, \boldsymbol{u})$, ϵ	The regret ratio of point $\boldsymbol{p}$ w.r.t. $\boldsymbol{u}$ and the regret threshold
$h_{i,j}$, $h^+_{i,j}$, $h^-_{i,j}$	The hyper-plane of $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, positive / negative half-spaces
$\mathcal{H}$	The set of intersecting half-spaces of $\mathcal{R}$
N and $\mathcal{R}_N$	A node in the I-tree and its associated utility range
s, r, a	The state, reward and action in the MDP
$\boldsymbol{e}$ / $\mathcal{E}$ / E / $S_{\boldsymbol{e}}$	An extreme vector / the set of all extreme vectors / the selected extreme vectors of $\mathcal{R}$ / the neighborhood set of $\boldsymbol{e}$
m_e / m_h	The number of selected extreme vectors / actions
$\mathcal{T}$ and $p_{\mathcal{T}}$	A terminal polyhedron and its associated point to return
$Q(;\Theta)$ / $\hat{Q}(;\Theta')$	The main network / the target network in DQN

This hyper-plane passes through the origin with its unit normal in the same direction as $\boldsymbol{p}_i - \boldsymbol{p}_j$. It divides $\mathbb{R}^d$ into two halves, called *half-spaces* [30]. The half-space above (resp. below) $h_{i,j}$, denoted by $h^+_{i,j}$ (resp. $h^-_{i,j}$), contains all utility vectors $\boldsymbol{u} \in \mathbb{R}^d$ such that $\boldsymbol{u} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j) > 0$ (resp. $\boldsymbol{u} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j) < 0$). This said, if the user's utility vector $\boldsymbol{u}$ is in $h^+_{i,j}$ (resp. $h^-_{i,j}$), the utility of $\boldsymbol{p}_i$ must be higher (resp. lower) than $\boldsymbol{p}_j$ w.r.t. $\boldsymbol{u}$ [9].

Combining these definitions, the lemma below shows our foundation for learning the user's preference. Due to the lack of space, proofs can be found in our technical report [31].

Lemma 1 ([9], [15]). Given $\mathcal{U}$ and two points $\boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{D}$ that are presented to a user in an interactive round, the user's utility vector is in $h^+_{i,j} \cap \mathcal{U}$ if and only if the user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$.

Example 3. Given $\boldsymbol{p}_1 = (0, 0.6, 0)$ and $\boldsymbol{p}_2 = (0.4, 0, 0)$, we can build a hyper-plane $h_{1,2} = \{\boldsymbol{r} \in \mathbb{R}^d \mid \boldsymbol{r} \cdot (-0.4, 0.6, 0) = 0\}$ (since $\boldsymbol{p}_1 - \boldsymbol{p}_2 = (-0.4, 0.6, 0)$) in Figure 4. Its unit norm is in the same direction as vector $(-0.4, 0.6, 0)$. If a user prefers $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$, we learn that the user's utility vector is in $h^+_{1,2} \cap \mathcal{U}$.

Based on Lemma 1, we can interact with a user and build hyper-planes to progressively narrow the utility space. We refer to the narrowed utility space as the *utility range* and denote it by $\mathcal{R}$, which is the intersection of $\mathcal{U}$ and a set of half-spaces (i.e., it is a polyhedron [30] that contains the user's utility vector). When the interaction proceeds, $\mathcal{R}$ gradually becomes smaller. When $\mathcal{R}$ becomes sufficiently small, it meets the stopping condition so that we can find a point whose regret ratio w.r.t. the user's utility vector is smaller than threshold ϵ.

Example 4. Suppose that a user prefers $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$ and prefers $\boldsymbol{p}_1$ to $\boldsymbol{p}_3$. We can build two hyper-planes $h_{1,2}$ and $h_{1,3}$ as shown in Figure 4, resulting in $\mathcal{R} = h^+_{1,2} \cap h^+_{1,3} \cap \mathcal{U}$ (yellow triangle).

Interaction Tree. The interaction process can be represented as a tree, called *interaction tree* or *I-Tree* for short (Figure 2).

Each node N in the I-tree represents an interaction status. It is associated with a utility range, denoted by $\mathcal{R}_N$, e.g., if N is the root node, $\mathcal{R}_N = \mathcal{U}$. When the context is clear, we simply denote the utility range $\mathcal{R}_N$ of node N by $\mathcal{R}$.

Each leaf node denotes a terminal interaction status, whose $\mathcal{R}$ is small enough to meet the *stopping condition*, i.e., there is a point $\boldsymbol{p} \in \mathcal{D}$ whose regret ratio is smaller than threshold ϵ w.r.t. any utility vectors in $\mathcal{R}$ (the checking of this condition is deferred to Sections IV-B and IV-C). Since the user's utility vector is in $\mathcal{R}$, this stopping condition ensures the regret ratio of $\boldsymbol{p}$ is smaller than threshold ϵ w.r.t. the user's utility vector.

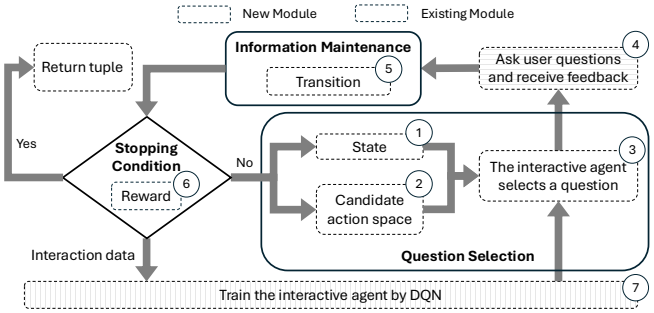


Figure 3: Our Workflow

Each internal node N denotes an intermediate interaction status. Except for the utility range, each internal node is also associated with a pool of candidate questions that can be used to interact with the user. Each candidate question consists of a pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points in $\mathcal{D}$ and it leads to two child nodes N_{pos} and N_{neg} of N, where N_{pos} (resp. N_{neg}) corresponds to the interaction status when the user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$ (resp. prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$), whose associated utility range is $\mathcal{R}_N \cap h^+_{i,j}$ (resp. $\mathcal{R}_N \cap h^+_{j,i}$). For instance, assume that the current interaction status is node N_1 in Figure 2. If the question $\langle \boldsymbol{p}_5, \boldsymbol{p}_6 \rangle$ is selected and the user prefers $\boldsymbol{p}_5$ to $\boldsymbol{p}_6$, the interaction status moves to node N_2, where $\mathcal{R}_{N_2} = \mathcal{R}_{N_1} \cap h^+_{5,6}$. Note that in Figure 2, we only show two candidate questions in each internal node for simplicity. The complete set of candidate questions should contain all pairs of points in $\mathcal{D}$.

The interaction process of a user can be regarded as a path from the root node to a leaf node in the I-Tree. Users traverse different paths if they are asked different questions or give different answers. The length of each path is the number of questions asked to the user, i.e., the number of interactive rounds.

The formulation of I-Tree motivates us to develop an *interactive agent* that guides the traversal of the I-Tree: whenever an internal node N is reached, it decides which question (i.e., a pair of points) is selected for interaction, until we reach a leaf node that meets the stopping condition. Below we model this process as a *Markov Decision Process (MDP)*, which consists of four components: state, action, transition, and reward.

- *State.* A state captures the information of the interaction status for the interactive agent to make decisions. In the I-Tree, the utility range $\mathcal{R}_N$ in each node N represents a state.
- *Action.* An action is a decision made by the interactive agent that affects the state. In the I-Tree, an action is a question (i.e., a pair of points) selected from the candidate pool. The user is asked which one s/he prefers between the two points.
- *Transition.* A transition describes the change from one state to another by taking an action. Consider a state $\mathcal{R}_N$ in node N and an action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$. If the user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$ (resp. $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$), we move from node N to its child node N_{pos} (resp. N_{neg}) and *transit* from state $\mathcal{R}_N$ to state $\mathcal{R}_{N_{pos}} = \mathcal{R}_N \cap h^+_{i,j}$ (resp. $\mathcal{R}_{N_{neg}} = \mathcal{R}_N \cap h^+_{j,i}$). Note that since the update only depends on the current utility range $\mathcal{R}_N$, it ensures the *Markov property* (i.e., the future state only depends on the current state instead of all the previous states).
- *Reward.* A reward associated with a transition indicates the effect of the action taken at a given state. In the I-Tree, the reward is inversely proportional to the path length. This said, the longer the path, the smaller the accumulated reward.

Our proposed algorithms substantiate this MDP within the interactive framework of [9], [15], as outlined in Figure 3. There are seven modules. In each interactive round, the interactive agent strategically selects the best question (module 3) based on the current state (module 1) and the candidate action space (module 2). Upon receiving the user feedback (module 4), the algorithms derive the transition (module 5) and generate a reward (module 6). Using the interaction data, the algorithms train the interactive agent via reinforcement learning (module 7), with the goal of maximizing the accumulated reward. Note that modules 1, 2, 5, and 6 are new in this paper.

B. Exact Algorithm EA

We present an exact algorithm EA that returns a point whose regret ratio is below threshold ϵ. We show its MDP formulation and the way to learn a policy for the MDP.

1) MDP Formulation: We formulate the state, action, transition, and reward of the MDP of algorithm EA as follows.

MDP: State. We first show how we represent the utility range $\mathcal{R}$ in each node using a fixed-length state vector.

Recall that $\mathcal{R}$ is a polyhedron. In literature [15], [29], [30], a polyhedron is commonly described by the set of *extreme utility vectors*, denoted by $\mathcal{E}$, where each extreme utility vector $\boldsymbol{e}$ in $\mathcal{E}$ is a corner point of $\mathcal{R}$. For example, suppose that $\mathcal{R}$ is a triangle shown in orange in Figure 5. The three black points are the extreme utility vectors of $\mathcal{R}$. Therefore, a straightforward idea to represent $\mathcal{R}$ is to concatenate all extreme utility vectors in $\mathcal{E}$. However, this method encounters an issue of the varying number of extreme utility vectors in different polyhedrons, making it hard to maintain a fixed-length vector representation.

To address this, we concisely represent $\mathcal{R}$ in two parts. (1) *Selected extreme utility vectors.* Given a positive number m_e, we select a fixed number (i.e., m_e) of representative extreme utility vectors in $\mathcal{E}$, expecting them to provide detailed insights about the shape and characteristics of $\mathcal{R}$. Since only part of $\mathcal{E}$ are selected, we inevitably neglect some vectors in $\mathcal{E}$. To compensate, we introduce the second part. (2) *Outer sphere.* We approximate $\mathcal{R}$ with the smallest sphere $\mathcal{B}$ that encloses all utility vectors in $\mathcal{R}$ (thus encloses all extreme utility vectors in $\mathcal{E}$), expecting this sphere to provide a broad overview of $\mathcal{R}$.

(1) Extreme utility vectors. Consider the set $\mathcal{E}$ of all extreme utility vectors of $\mathcal{R}$. If some vectors in $\mathcal{E}$ are clustered together (e.g., the Euclidean distance between any two in the cluster is less than a threshold d_ϵ), it typically suffices to select one from them to capture the essential information of its neighborhood.

Motivated by the widely adopted DBSCAN algorithm [32], we build a neighborhood set $S_{\boldsymbol{e}}$ for each extreme utility vector $\boldsymbol{e} \in \mathcal{E}$, to record the extreme utility vectors whose distance to $\boldsymbol{e}$ is smaller than a given threshold d_ϵ, i.e., $S_{\boldsymbol{e}} = \{\boldsymbol{e}' \in \mathcal{E} \mid ||\boldsymbol{e}' - \boldsymbol{e}|| \leq d_\epsilon\}$, where $||\cdot||$ denotes the Euclidean distance between $\boldsymbol{e}'$ and $\boldsymbol{e}$. We say that $\boldsymbol{e}$ *covers* the vectors in $S_{\boldsymbol{e}}$. Our objective is to select a set E of m_e extreme utility vectors that

Table III: Database ($\boldsymbol{u} = (0.3, 0.7)$)

$\boldsymbol{p}$	$p[1]$	$p[2]$	$f_{\boldsymbol{u}}(\boldsymbol{p})$
$\boldsymbol{p}_1$	0	1.0	0.70
$\boldsymbol{p}_2$	0.3	0.7	0.58
$\boldsymbol{p}_3$	0.5	0.8	0.71
$\boldsymbol{p}_4$	0.7	0.4	0.49
$\boldsymbol{p}_5$	1.0	0	0.30

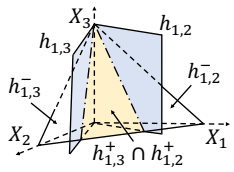


Figure 4: Utility Range

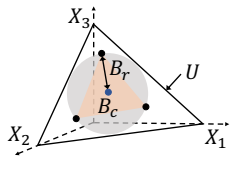


Figure 5: Outer Sphere

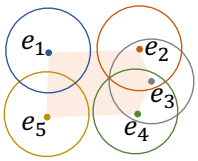


Figure 6: Construct E

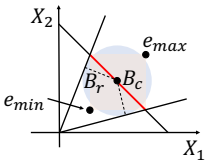


Figure 7: State in AA

collectively covers the most extreme utility vectors in $\mathcal{E}$, i.e., $\max_{E \subseteq \mathcal{E}, |E|=m_e} |\bigcup_{\boldsymbol{e} \in E} S_{\boldsymbol{e}}|$. However, this is non-trivial.

Lemma 2. Identifying a set E of m_e extreme utility vectors such that $|\bigcup_{\boldsymbol{e} \in E} S_{\boldsymbol{e}}|$ is maximized is an NP-hard problem.

Due to the intractability, we compute E by a greedy algorithm, which is known to have an $(1 - \frac{1}{e})$-approximation [33], [34]. Specifically, we initialize (1) $E = \emptyset$ and (2) $\mathcal{E}$ to be a set of all extreme utility vectors of $\mathcal{R}$. In each iteration, we construct E by adding into E the vector $\boldsymbol{e}$ that covers the most uncovered extreme utility vectors in $\mathcal{E}$. This process repeats until $|E| = m_e$ or all extreme utility vectors in $\mathcal{E}$ are covered.

Example 5. Consider Figure 6. There are five extreme utility vectors of $\mathcal{R}$. Since the set $S_{\boldsymbol{e}_3}$ of $\boldsymbol{e}_3$ contains the most extreme utility vectors ($|S_{\boldsymbol{e}_3}| = 3$), it is added to E in the first iteration.

Note that although other clustering methods, in addition to DBSCAN, can be adapted to select utility vectors, we adopt our setting for its good empirical performance (see Section V).

(2) Outer sphere. The outer sphere $\mathcal{B}$ can be defined by its center $\mathcal{B}_c$ and radius $\mathcal{B}_r$ (Figure 5). To construct it, we solve an optimization problem, which minimizes the radius $\mathcal{B}_r$, while ensuring that the Euclidean distance from the center $\mathcal{B}_c$ to any extreme utility vector $\boldsymbol{e} \in \mathcal{E}$ is at most the radius $\mathcal{B}_r$, i.e.,

$$\begin{aligned} &\text{minimize} && \mathcal{B}_r \\ &\text{subject to} && ||\mathcal{B}_c - \boldsymbol{e}|| \leq \mathcal{B}_r \text{ for each } \boldsymbol{e} \in \mathcal{E}. \end{aligned}$$

Since this is non-convex optimization, we adopt an iterative algorithm to find the local optimum. Initially, we randomly generate a vector as the center $\mathcal{B}_c$. In each iteration, we calculate the distance from each extreme utility vector to $\mathcal{B}_c$. Let $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ be the two vectors in $\mathcal{E}$ with the largest and second-largest distances to $\mathcal{B}_c$, respectively. We then move the center $\mathcal{B}_c$ towards $\boldsymbol{e}_1$ by an offset $\frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||)$. The iterative process stops when the offset is smaller than a predefined threshold, or it reaches the maximum number of iterations. The following lemma gives the convergence.

Lemma 3. In each successive iteration, $\mathcal{B}_r$ becomes smaller.

Taken together, we concatenate (1) the m_e selected extreme utility vectors in $\mathcal{E}$ and (2) the outer sphere's center and radius to get a $(dm_e + d + 1)$-dimensional vector to define a state.

MDP: Action. Given a state (i.e., $\mathcal{R}$), each pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points in $\mathcal{D}$ can be an action. Thus, the action space consists of $O(n^2)$ actions. With such a large action space, the agent may struggle to explore all possible actions. To solve this, we propose to use a subset of pairs as the action space for each state.

Recall that $\mathcal{R}$ in each leaf node satisfies the stopping condition: there is a point $\boldsymbol{p} \in \mathcal{D}$ whose regret ratio is smaller than ϵ w.r.t. any utility vectors in $\mathcal{R}$. We refer to a polyhedron that meets this condition as a *terminal polyhedron*. Denote a terminal polyhedron by $\mathcal{T}$ and the corresponding point with regret ratio below ϵ by $\boldsymbol{p}_{\mathcal{T}}$. Suppose that the current $\mathcal{R}$ is not a terminal polyhedron. We need to interact with a user for more rounds, deriving more half-spaces to narrow down $\mathcal{R}$ until it becomes a terminal polyhedron. In light of this, we expect to select those pairs $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ as the actions such that the hyper-planes $h_{i,j}$ can help to narrow utility range $\mathcal{R}$ effectively.

To achieve this, we construct several terminal polyhedrons inside $\mathcal{R}$, each of which can be obtained by narrowing down the current $\mathcal{R}$ using some half-spaces. Let $P_{\mathcal{R}}$ be the set of all $\boldsymbol{p}_{\mathcal{T}}$ corresponding to these terminal polyhedrons, i.e., $P_{\mathcal{R}} = \{\boldsymbol{p}_{\mathcal{T}} | \ \mathcal{T}$ is a terminal polyhedron constructed within $\mathcal{R}\}$. Given a number m_h, we randomly select m_h pairs of points from $P_{\mathcal{R}}$ to form the action space. Intuitively, these pairs provide directive hints to locate the terminal polyhedron that contains the user's utility vector. To illustrate, consider a pair $\langle \boldsymbol{p}_{\mathcal{T}_i}, \boldsymbol{p}_{\mathcal{T}_j} \rangle$, which defines a hyper-plane h. If the user prefers $\boldsymbol{p}_{\mathcal{T}_i}$ to $\boldsymbol{p}_{\mathcal{T}_j}$, $\mathcal{T}_i$ is more likely to contain the user's utility vector than $\mathcal{T}_j$. Better still, $\mathcal{R}$ will be narrowed to be $\mathcal{R} \cap h^+$ and those terminal polyhedrons in $\mathcal{R} \cap h^-$ are no longer eligible. Therefore, by restricting the action space in this way, we can quickly reduce ineligible terminal polyhedrons and narrow $\mathcal{R}$ towards the terminal polyhedron that contains the user's utility vector.

The following discusses the method to construct terminal polyhedrons in a utility range $\mathcal{R}$. As a by-product, this method also helps to decide whether $\mathcal{R}$ itself is a terminal polyhedron.

Specifically, we build a set $\mathcal{V}$ of utility vectors that contains two parts: (1) the utility vectors randomly sampled from $\mathcal{R}$, and (2) the set $\mathcal{E}$ of extreme utility vectors of $\mathcal{R}$. For each $\boldsymbol{u} \in \mathcal{V}$, we construct a terminal polyhedron in two steps (if $\boldsymbol{u}$ is not in a polyhedron already constructed): (1) we find a point $\boldsymbol{p}_i \in \mathcal{D}$ with the highest utility w.r.t. $\boldsymbol{u}$; and (2) for each point $\boldsymbol{p}_j \in \mathcal{D} \setminus \{\boldsymbol{p}_i\}$, we build a hyper-plane, denoted by $\epsilon h_{i,j}$:

$$\epsilon h_{i,j} = \{\boldsymbol{r} \in \mathbb{R}^d \mid \boldsymbol{r} \cdot (\boldsymbol{p}_i - (1 - \epsilon)\boldsymbol{p}_j) = 0\}.$$

Then $\mathcal{R}_{\boldsymbol{u}} = \mathcal{R} \cap \bigcap_{\boldsymbol{p}_j \in \mathcal{D} \setminus \{\boldsymbol{p}_i\}} \epsilon h^+_{i,j}$ is a terminal polyhedron.

Lemma 4. The regret ratio of $\boldsymbol{p}_i$ over $\mathcal{D}$ w.r.t. any utility vector in $\mathcal{R}_{\boldsymbol{u}} = \mathcal{R} \cap \bigcap_{\boldsymbol{p}_j \in \mathcal{D} \setminus \{\boldsymbol{p}_i\}} \epsilon h^+_{i,j}$ is smaller than threshold ϵ.

The former part of $\mathcal{V}$ (i.e., randomly sampled utility vectors) enables terminal polyhedrons with large volumes to have a high probability of being constructed. This is crucial because these polyhedrons contain more utility vectors, thereby being more likely to include the user's utility vector. The lemma below suggests a proper number $\mathcal{N}$ of sampled utility vectors.

Lemma 5. Given two terminal polyhedrons $\mathcal{T}_1$ and $\mathcal{T}_2$ within $\mathcal{R}$, a confidence parameter δ, a small error parameter τ, and a

number $\mathcal{N} = O(\frac{d+ln(1/\delta)}{\tau^2})$, if $vol(\mathcal{T}_1) - vol(\mathcal{T}_2) \geq 2\tau vol(\mathcal{R})$, then $sample(\mathcal{T}_1) - sample(\mathcal{T}_2) \geq 0$ with probability at least $1-\delta$, where $sample(\mathcal{T})$ is the number of sampled vectors in $\mathcal{T}$, $vol(\mathcal{T})$ is the volume of $\mathcal{T}$, and $vol(\mathcal{R})$ is the volume of $\mathcal{R}$.

Lemma 5 indicates that a polyhedron with a larger volume is likely to include more sampled utility vectors, and thus, this polyhedron is more likely to be constructed based on $\mathcal{V}$.

The latter part of set $\mathcal{V}$ (i.e., $\mathcal{E}$) is to provide the side information used to decide whether $\mathcal{R}$ is a terminal polyhedron.

Lemma 6. The utility range $\mathcal{R}$ must be a terminal polyhedron if there is only one terminal polyhedron constructed based on the set $\mathcal{E}$ of extreme utility vectors of $\mathcal{R}$.

MDP: Transition. Given a state s (i.e., $\mathcal{R}$) and an action a (i.e., a pair of points $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ in $P_{\mathcal{R}}$), the interactive agent transits to one of two new states s' ($\mathcal{R} \cap h^+_{i,j}$ or $\mathcal{R} \cap h^+_{j,i}$) based on the user feedback. Our method of constructing the action space enables $\mathcal{R}$ to be strictly narrowed. When $\mathcal{R}$ itself is a terminal polyhedron (Lemma 6), the agent reaches a terminal state.

Lemma 7. The utility range $\mathcal{R}$ will be strictly narrowed, no matter which pair from $P_{\mathcal{R}}$ is selected as the action.

MDP: Reward. Consider a state s (i.e., $\mathcal{R}$) and an action a (i.e., a pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points in $P_{\mathcal{R}}$). If we transit to a terminal state after taking the action, the reward r is set to a positive constant c. Otherwise, it is 0. With this definition, the accumulated reward is inversely proportional to the number of interactive rounds. To see this, suppose that we go through a sequence of states $s_1, s_2, ...s_x$ and correspondingly, we receive a sequence of rewards $r_1, r_2, ..., r_{x-1}$. The accumulated reward is

$$\sum_{i=1}^{x-1} r_i = 0 + 0\gamma + ... + 0\gamma^{x-2} + c\gamma^{x-1} = c\gamma^{x-1},$$

where $\gamma < 1$ is the discount factor. Thus, if we go through a long sequence of states, the accumulated reward will be small.

2) Training and Inference of the Interactive Agent: We adopt the popular Deep Q-Learning [35] for the agent.

Deep-Q-Network (DQN) Learning. Deep Q-learning is a widely used model-free RL method. It employs a Q-function $Q^*(s,a)$ to represent the expected accumulated reward that the agent can obtain by taking action a in state s. In any given state, the agent selects the action with the highest Q-value. To approximate the Q-function $Q^*(s,a)$, Deep-Q-Network Learning [35] uses a deep neural network $Q(s,a;\Theta)$ with parameters Θ. We adopt deep Q-learning with experience replay for learning the Q-function. Given a batch of transitions (s,a,r,s') (i.e., we transit from state s to state s' by taking action a with reward r), parameters in $Q(s,a;\Theta)$ are updated with a gradient descent step by minimizing the mean square error (MSE) loss function, as shown in the following equation.

$$L(\Theta) = \sum_{(s,a,r,s')} (r + \gamma max_{a'} \hat{Q}(s',a';\Theta') - Q(s,a;\Theta))^2$$

where γ is the discount factor, and $\hat{Q}(s,a;\Theta')$ is the target network. Here, the target network $\hat{Q}(s,a;\Theta')$ is a separate neural network used to stabilize the learning process. It has the same architecture as the $Q(s,a;\Theta)$ network but with parameters Θ' that are copied from the Q-network at regular intervals and kept constant between updates. This target network helps to reduce oscillations and divergence in the training process by providing a stable target for the updates of the Q-function.

Algorithm 1: Algorithm EA (Training)

```
1  Input: A training set of utility vectors, ε and a point set D
2  Output: Learned Q-function Q(s,a;Θ)
3  Initialize Q(s,a;Θ), Q̂(s,a;Θ') and replay memory M;
4  for each u in the training set do
5      R ← U; s ← a state based on the outer sphere and
         selected extreme utility vectors of R;
6      while R is not a terminal polyhedron by Lemma 6 do
7          a ← an action ⟨p_i, p_j⟩ by ε-greedy on Q-values;
8          if p_i · u ≥ p_j · u then
9              R ← R ∩ h^+_{i,j};
10         else
11             R ← R ∩ h^+_{j,i};
12         s' ← the new state based on the updated R;
13         if the updated R is a terminal polyhedron then
14             Add (s,a,c,s') into M;
15         else
16             Add (s,a,0,s') into M;
17         s ← s';
18     Draw samples from M to update Q(s,a;Θ);
19     Periodically synchronize Q̂(s,a;Θ');
```

Algorithm 2: Algorithm EA (Inference)

```
1  Input: Learned Q-function Q(s,a;Θ), ε and a point set D
2  Output: A tuple p whose regret ratio is below ε
3  R ← U; s ← a state based on the outer sphere and selected
     extreme utility vectors of R;
4  while R is not a terminal polyhedron by Lemma 6 do
5      a ← an action ⟨p_i, p_j⟩ with the largest Q-value;
6      if the user prefers p_i to p_j then
7          R ← R ∩ h^+_{i,j};
8      else
9          R ← R ∩ h^+_{j,i};
10     s ← the new state based on the updated R;
11 return The point p_R s.t. ∀u ∈ R, regratio(p_R, u) ≤ ε;
```

Training. We present the training process in Algorithm 1. In the beginning, we initialize the main network Q, target network $\hat{Q}$, and the replay memory $\mathcal{M}$ (line 3). The replay memory records past experiences (states, actions, and rewards) to enable the agent to learn from a diverse set of experiences, improving stability and efficiency in training. In each epoch, we use a utility vector $\boldsymbol{u}$ from the training set for interaction. We start with the utility range $\mathcal{R} \leftarrow \mathcal{U}$ and compute the current state s (line 5). In each interactive round, we use ε-greedy to select the action (a pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points) according to Q-values (line 7). Based on the utility vector $\boldsymbol{u}$, we compute the utilities of $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, and update $\mathcal{R}$ accordingly (i.e., either $\mathcal{R} \cap h^+_{i,j}$ or $\mathcal{R} \cap h^+_{j,i}$) (lines 8-11). Next, we compute the new state s' based on the updated $\mathcal{R}$. With the current and new

states, we add the transition (s, a, r, s') into the replay memory (lines 12-16), where the reward is set to a constant c if s' is a terminal state; otherwise, the reward is set to 0. The interaction stops when $\mathcal{R}$ becomes a terminal polyhedron (line 6). After that, we randomly draw a batch of transitions from the replay memory to update the main network $Q(s, a; \Theta)$ (line 18). The target network $\hat{Q}$ is periodically synchronized with the main network $Q(s, a; \Theta)$ to stabilize the learning process (line 19).

Inference. Given the learned Q-function, we present the inference process in Algorithm 2. We initialize $\mathcal{R} \leftarrow \mathcal{U}$ and compute the current state s (line 3). In each interactive round, we select the action (a pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points) with the highest Q-value (line 5). Based on the user's feedback, $\mathcal{R}$ is updated accordingly (i.e., either $\mathcal{R} \cap h^+_{i,j}$ or $\mathcal{R} \cap h^+_{j,i}$) (lines 6-9). Then, we compute the new state s' based on the updated $\mathcal{R}$ (line 10). The interaction stops when $\mathcal{R}$ becomes a terminal polyhedron (line 4). We return the point $\boldsymbol{p}_{\mathcal{R}}$ associated with $\mathcal{R}$ such that $regratio(\boldsymbol{p}_{\mathcal{R}}, \boldsymbol{u}) \leq \epsilon$ for any utility vector $\boldsymbol{u} \in \mathcal{R}$ (Lemma 4).

3) Theoretical Analysis: Based on the restricted action space, we can guarantee that the interactive agent takes $O(n)$ interactive rounds for each user in the worst case.

Theorem 1. Algorithm EA interacts a user by $O(n)$ rounds.

C. Approximate Algorithm AA

Although algorithm EA extracts comprehensive information from $\mathcal{R}$ and returns an exact solution, it can be computationally expensive, primarily due to the polyhedron computations. In light of this, we present an approximate algorithm AA, which avoids the exact computation of polyhedrons, striking a balance between scalability and accuracy. Below, we outline its MDP formulation, and present its training and inference.

1) MDP Formulation.: The design of the state, action, transition, and reward for algorithm AA is shown as follows.

MDP: State. Recall that $\mathcal{R}$ is the intersection of a set of half-spaces and utility space $\mathcal{U}$. Instead of precisely computing the exact intersection as before, we only record the set of intersecting half-spaces, denoted by $\mathcal{H}$, i.e., $\mathcal{R} = \cap_{h^+_{i,j} \in \mathcal{H}} h^+_{i,j} \bigcap \mathcal{U}$.

Based on set $\mathcal{H}$, we represent a state, i.e., $\mathcal{R}$, by a fixed-length vector using two components: an *inner sphere* and an *outer rectangle*. The inner sphere approximates $\mathcal{R}$ from a central perspective, capturing the core region of $\mathcal{R}$. The outer rectangle, on the other hand, provides an overall approximation, covering the entire $\mathcal{R}$. Note that we do not use the outer sphere discussed in Section IV-B, since it involves the computation of the extreme utility vectors of $\mathcal{R}$ and computing the extreme utility vectors of $\mathcal{R}$ is as hard as computing $\mathcal{R}$ itself.

The inner sphere of $\mathcal{R}$ is defined to be a sphere $\mathcal{B}$ such that (1) its center is in $\mathcal{R}$; (2) it is inside each half-space $h^+_{i,j} \in \mathcal{H}$; and (3) it has the largest radius. Let $\mathcal{B}_r$ (resp. $\mathcal{B}_c$) denote the radius (resp. center) of $\mathcal{B}$. Figure 7 shows an example of an inner sphere, where the red line segment represents $\mathcal{R}$ in a 2-dimensional space. Given $\mathcal{H}$, we utilize Linear Programming (LP) to compute the desired inner sphere. By definition, for each $h^+_{i,j} \in \mathcal{H}$ that is built based on $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, the distance from $\mathcal{B}_c$ to any point in the hyper-plane $h_{i,j}$ is larger than $\mathcal{B}_r$.

$$
\begin{aligned}
\text{maximize} \quad & \mathcal{B}_r \\
\text{subject to} \quad & \textstyle\sum_{i'=1}^{d} \mathcal{B}_c[i'] = 1 \text{ and } \mathcal{B}_c[i'] \geq 0 \text{ for } i' \in [1, d], \\
& (\boldsymbol{p}_i - \boldsymbol{p}_j) \cdot \mathcal{B}_c > 0 \text{ for each } h^+_{i,j} \text{ in } \mathcal{H}, \\
& \frac{(\boldsymbol{p}_i - \boldsymbol{p}_j) \cdot \mathcal{B}_c}{||\boldsymbol{p}_i - \boldsymbol{p}_j||} \geq \mathcal{B}_r \text{ for each } h^+_{i,j} \text{ in } \mathcal{H}.
\end{aligned}
$$

The first two constraints guarantee that center $\mathcal{B}_c$ is in $\mathcal{R}$ and the last constraint restricts the distance from $\mathcal{B}_c$ to $h_{i,j}$.

The outer rectangle of $\mathcal{R}$ is the smallest axis-aligned rectangle containing $\mathcal{R}$ (see Figure 7). It can be concisely represented by two vectors, namely $\boldsymbol{e}_{min}$ and $\boldsymbol{e}_{max}$, where $\boldsymbol{e}_{min}$ is composed of the smallest values of utility vectors in $\mathcal{R}$ along each dimension, i.e., $\forall i' \in [1, d], \boldsymbol{e}_{min}[i'] = \min\{u[i'] | \boldsymbol{u} \in \mathcal{R}\}$; similarly for $\boldsymbol{e}_{max}$ that uses the largest values instead. To obtain these two vectors, we can also utilize LP by restricting $\boldsymbol{u}$ in $\mathcal{R}$ and minimizing/maximizing its i'-th dimension value $u[i']$ for $i' \in [1, d]$, e.g., the LP below computes $\boldsymbol{e}_{max}[i']$.

$$
\begin{aligned}
\text{maximize} \quad & u[i'] \\
\text{subject to} \quad & \textstyle\sum_{j'=1}^{d} u[j'] = 1 \text{ and } u[j'] \geq 0 \text{ for } j' \in [1, d], \\
& (\boldsymbol{p}_i - \boldsymbol{p}_j) \cdot \boldsymbol{u} > 0 \text{ for each } h^+_{i,j} \text{ in } \mathcal{H}.
\end{aligned}
$$

The state $\mathcal{R}$ is then represented as the concatenation of the two components, i.e., (1) the center and radius of the inner sphere and (2) the two utility vectors of the outer rectangle.

MDP: Action. As remarked before, the entire action space consists of $O(n^2)$ actions, posing a significant impede to the efficiency of RL exploration. To mitigate this issue, we also restrict the action space by selecting a subset of point pairs. We adopt an effective heuristic method to select point pairs that can narrow $\mathcal{R}$ as much as possible, without incurring the costly exact computation of polyhedrons.

Suppose there is an ideal pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ in $\mathcal{D}$ such that hyper-plane $h_{i,j}$ divides $\mathcal{R}$ into two equal halves. Then no matter which point ($\boldsymbol{p}_i$ or $\boldsymbol{p}_j$) is preferred by the user, $\mathcal{R}$ can be further narrowed by half (i.e., $\mathcal{R} \cap h^+_{i,j}$ or $\mathcal{R} \cap h^+_{j,i}$). However, it is hard to evaluate whether a hyper-plane $h_{i,j}$ can divide $\mathcal{R}$ into two equal halves without the exact information of $\mathcal{R}$. Thus, we utilize the inner sphere and identify the hyper-planes that are close to the center of the inner sphere of $\mathcal{R}$, hoping that these hyper-planes are more likely to divide $\mathcal{R}$ into two equal halves.

Consider a hyper-plane $h_{i,j}$ and the center $\mathcal{B}_c$ of the inner sphere. We use $dist(\mathcal{B}_c, h_{i,j})$ to denote the minimum distance from $\mathcal{B}_c$ to any point on $h_{i,j}$. Given a positive number m_h, we find m_h pairs $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ in $\mathcal{D}$ such that (1) $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$ and $\mathcal{R} \cap h^+_{j,i} \neq \emptyset$ and (2) $dist(\mathcal{B}_c, h_{i,j})$ is among the top-m_h smallest one. Here, the first condition guarantees that $\mathcal{R}$ can be strictly smaller after each interactive round. We check it by LP. For example, to check $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$, we define a variable x and set $(\boldsymbol{p}_i - \boldsymbol{p}_j) \cdot \boldsymbol{u} > x$ for each $h^+_{i,j} \in \mathcal{H}$. If there is $x \geq 0$, then there is a vector $\boldsymbol{u}$ inside $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$, i.e.,

$$
\begin{aligned}
\text{maximize} \quad & x \\
\text{subject to} \quad & \textstyle\sum_{i'=1}^{d} u[i'] = 1 \text{ and } u[i'] \geq 0 \text{ for } i' \in [1, d], \\
& (\boldsymbol{p}_i - \boldsymbol{p}_j) \cdot \boldsymbol{u} > x \text{ for each } h^+_{i,j} \text{ in } \mathcal{H}.
\end{aligned}
$$

Lemma 8. For each pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ in the restricted action space, $\mathcal{R}$ can be strictly narrowed after an interactive round.

MDP: Transition. Similarly as before, given a state s (i.e., a utility range $\mathcal{R}$) and an action a (i.e., a pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points), the interactive agent transits to one of two new states s' (i.e., $\mathcal{R} \cap h^+_{i,j}$ or $\mathcal{R} \cap h^+_{j,i}$) based on the user feedback. However, compared to its counterpart in algorithm EA that decides if we reach a terminal state by constructing terminal polyhedrons in $\mathcal{R}$ (Lemma 6), it is hard to decide whether we have reached a terminal state, given the set $\mathcal{H}$ of intersecting half-spaces only.

To tackle this, we utilize the two vectors $\boldsymbol{e}_{min}$ and $\boldsymbol{e}_{max}$ of the outer rectangle of $\mathcal{R}$. We stop the interaction when $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$, and return the point $\boldsymbol{p}$ that has the highest utility w.r.t. $\boldsymbol{u} = \frac{1}{2}(\boldsymbol{e}_{min} + \boldsymbol{e}_{max})$. As shown in Lemma 9, the regret ratio of $\boldsymbol{p}$ w.r.t. the user's utility vector $\boldsymbol{u}^*$ is bounded.

Lemma 9. If $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$, $regratio(\boldsymbol{p}, \boldsymbol{u}^*) \leq d^2\epsilon$.

Note that since we do not compute the exact utility range $\mathcal{R}$, the bound on the regret ratio of $\boldsymbol{p}$ is not strictly smaller than the required ϵ; instead, it is bounded by up to a factor of d^2. Nonetheless, as will be shown in Section V, the actual regret ratio of the returned point is typically smaller than ϵ.

MDP: Reward. We adopt the same reward setting as algorithm EA, i.e., if we transit to a terminal state after taking the action, the reward r of taking this action is set to a positive constant c. Otherwise, the reward r is set to 0.

2) Training and Inference of the Interactive Agent.: Following a similar process in Section IV-B2, we also adopt the deep Q-learning with experience replay for algorithm AA.

Training. The training process is outlined in Algorithm 3. Since it employs a similar framework as Algorithm 1, we omit a detailed line-by-line explanation. Nonetheless, it differs from Algorithm 1 in the following three aspects:

(1) Instead of directly maintaining polyhedron $\mathcal{R}$, it maintains the set $\mathcal{H}$ of intersecting half-spaces (lines 10-13).
(2) We represent a state s by the inner sphere $(\mathcal{B}_c, \mathcal{B}_r)$ and the outer rectangle $(\boldsymbol{e}_{min}, \boldsymbol{e}_{max})$ computed based on $\mathcal{H}$, without computing any exact polyhedrons (lines 5-7, 14).
(3) We stop interaction if $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$ (line 8).

Inference. Given the learned Q-function, the inference process of algorithm AA is similar to Algorithm 2 (see pseudocode in [31]). The key differences are the state representation (which is computed based on $\mathcal{H}$) and the stopping condition (i.e., $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$). It finally returns the point that has the highest utility w.r.t. utility vector $\boldsymbol{u} = \frac{1}{2}(\boldsymbol{e}_{min} + \boldsymbol{e}_{max})$.

3) Theoretical Analysis: We guarantee that the interactive agent takes $O(n^2)$ interactive rounds for each user.

Lemma 10. Algorithm AA interacts a user by $O(n^2)$ rounds.

Corollary 1. Let Nr^* denote the optimal number of interactive rounds and Nr_{AA} denote the number of interactive rounds by algorithm AA. We have $Nr_{AA} \leq \frac{n(n-1)}{(d-1)(\log_2(1/\epsilon)-3)-4} Nr^*$.

V. Experiment

We conducted experiments on a Mac with a M3 chip. All programs were implemented in Python.

Algorithm 3: Algorithm AA (Training)

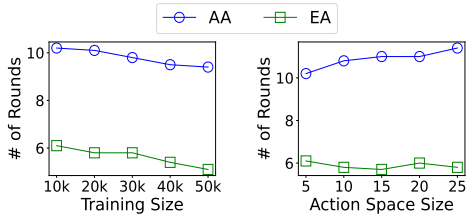
1 **Input:** A training set of utility vectors, ϵ and a point set $\mathcal{D}$
2 **Output:** Learned Q-function $Q(s, a; \Theta)$
3 Initialize $Q(s, a; \Theta)$, $\hat{Q}(s, a; \Theta')$ and replay memory $\mathcal{M}$;
4 **for** *each* $\boldsymbol{u}$ *in the training set* **do**
5 $\quad \mathcal{H} \leftarrow \emptyset$; $(\mathcal{B}_c, \mathcal{B}_r) \leftarrow$ the inner sphere computed via $\mathcal{H}$;
6 $\quad (\boldsymbol{e}_{min}, \boldsymbol{e}_{max}) \leftarrow$ the outer rectangle computed via $\mathcal{H}$;
7 $\quad s \leftarrow$ the state based on $(\mathcal{B}_c, \mathcal{B}_r)$ and $(\boldsymbol{e}_{min}, \boldsymbol{e}_{max})$;
8 $\quad$ **while** $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$ **do**
9 $\quad\quad a \leftarrow$ an action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ by ε-greedy on Q-values;
10 $\quad\quad$ **if** $\boldsymbol{p}_i \cdot \boldsymbol{u} \geq \boldsymbol{p}_j \cdot \boldsymbol{u}$ **then**
11 $\quad\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{h^+_{i,j}\}$;
12 $\quad\quad$ **else**
13 $\quad\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{h^+_{j,i}\}$;
14 $\quad\quad s' \leftarrow$ the state based on the updated $\mathcal{H}$;
15 $\quad\quad$ **if** s' *is the terminal state* **then**
16 $\quad\quad\quad$ Add (s, a, c, s') into $\mathcal{M}$;
17 $\quad\quad$ **else**
18 $\quad\quad\quad$ Add $(s, a, 0, s')$ into $\mathcal{M}$;
19 $\quad\quad s \leftarrow s'$ and update $(\mathcal{B}_c, \mathcal{B}_r)$ and $(\boldsymbol{e}_{min}, \boldsymbol{e}_{max})$;
20 $\quad$ Draw samples from $\mathcal{M}$ to update $Q(s, a; \Theta)$;
21 $\quad$ Periodically synchronize $\hat{Q}(s, a; \Theta')$;

Datasets. We adopted both synthetic and real datasets commonly used in existing studies [9], [36]–[38]. For the synthetic datasets, we used *anti-correlated* datasets produced by the generator designed for *skyline* operators [7], [36]. For the real datasets, we used datasets *Car* [39] and *Player* [40]. Dataset *Car* contains 10,668 cars described by six attributes, including price, mileage, etc. Dataset *Player* consists of $17,386$ basketball players, each described by twenty attributes, including age, points, offensive and defensive rebounds, etc.

For all these datasets, each attribute was normalized to $(0, 1]$. Note that the existing work [9] preprocessed datasets to contain skyline points only. Consistent with their settings, we also preprocessed the datasets to enable a fair comparison.

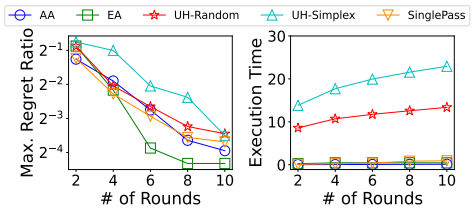
Baselines. We compared our algorithms EA and AA against all existing ones designed for the interactive regret query. (1) *UH-Random* [9], the state-of-the-art (SOTA) algorithm. It is a random-based algorithm that randomly selects a pair of points from a candidate set as the question in each interactive round. (2) *UH-Simplex* [9], a greedy algorithm that selects a pair of extreme points from the convex hull of the candidate set as the question in each interactive round. (3) *SinglePass* [14], a heuristic algorithm that selects pairs of points based on a predefined random sequence and rule-based filters. Note that algorithms EA, UH-Random, and UH-Simplex involve polyhedron computations during the interaction, which can be costly in high dimensions. Thus, we followed the setting of [9] and did not compare them when the dimension exceeds 10.

Parameter setting. We evaluated the performance of each algorithm by varying (1) the threshold of regret ratio ϵ; (2) the dataset size n; and (3) the number of dimensions d. Following the typical settings commonly adopted by existing work [9],



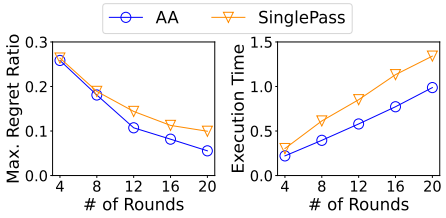
(a) (b)

Figure 8: Training



(a) (b)

Figure 9: Interaction process in 4D



(a) (b)

Figure 10: Interaction process in 20D

[14], the default values of parameters on synthetic datasets is $\epsilon = 0.1$, $n = 100,000$, and $d = 4$ unless stated explicitly.

We randomly sampled 10,000 utility vectors from the utility space for training. The DQN models contained 1 hidden layer of 64 neurons with SELU [41] as the activation function. In the training process, the size m_h of action space was set to 5 and the reward constant c was set to 100. The learning rate was set to 0.003. The parameter ε in ε-greedy exploration in model training was set to 0.9. The size of the replay memory was set to 5,000. The main network $Q(s, a; \Theta)$ was updated by sampling a batch of 64 transitions from the replay memory, using gradient descent of the MSE loss function. The discount factor γ was set to be 0.8. The synchronization of target network $\hat{Q}(s, a; \Theta')$ was done once every 20 updates of $Q(s, a; \Theta)$.

Measurement. We evaluated the performance of each algorithm using three measurements. (1) *Execution time.* The total time taken by the entire interaction process. (2) *Regret ratio.* The actual regret ratio of the returned point. (3) *The number of questions asked.* The number of interactive rounds needed to find a point with its regret ratio below the given threshold ϵ. We ran each experiment 10 times and reported the average.

A. Performance on Synthetic Datasets

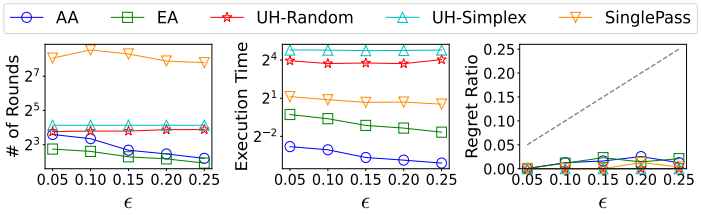
Training. We conducted experiments to study the impact of (1) the size of the training set and (2) the size of the action space on algorithms EA and AA, using a 4-dimensional synthetic dataset. We find the following. (1) With a larger training size in Figure 8(a), both algorithms EA and AA needed fewer interactive rounds, since they could train a better policy with more training data. (2) The results of varying the size m_h of action space are shown in Figure 8(b). With a larger action space, algorithm AA needed more interactive rounds to return the desired point. This validates the usefulness of our restricted action space, since a large action space increases the difficulty in RL exploration. In contrast, the performance of algorithm EA was less sensitive to the size of the action space. This is because it adopts a more accurate state representation, and thus, it is easier to optimize its interactive agent's decision-making.

Interaction process. Figures 9 and 10 show the progress in the interaction process on the 4-dimensional and 20-dimensional synthetic datasets, respectively. We report the current *maximum regret ratio* and the accumulated execution time at the end of each interactive round. Note that the current maximum regret ratio is different from the actual regret ratio of the final returned point. Following [8], [9], we computed it as follows. At the end of any interactive round, we could obtain the current inner sphere. The point $\boldsymbol{p} \in \mathcal{D}$ that had the highest utility w.r.t. the sphere's center was identified. We then randomly sampled 10,000 utility vectors in the intersection and computed the actual regret ratio of $\boldsymbol{p}$ w.r.t. each sampled utility vector. Finally, the maximum one was reported. Intuitively, the maximum regret ratio depicts the worst-case performance of an algorithm.

On the 4-dimensional dataset, algorithm EA was effective in reducing the regret ratio (see Figure 9(a)). After 8 interactive rounds, its maximum regret ratio dropped below 0.05. In contrast, the maximum regret ratio of UH-Simplex was 0.19, about 4 times higher than ours. Besides, the execution time of algorithm EA was also low (around 0.1 seconds in each interactive round). On the 20-dimensional dataset, our algorithm AA performed the best. For example, it took 0.58 seconds to complete 12 interactive rounds with a maximum regret ratio below 0.1; this was 31% faster than algorithm SinglePass, whose maximum regret ratio was 34% higher. These results verified that our algorithms progressed well during the interaction process.

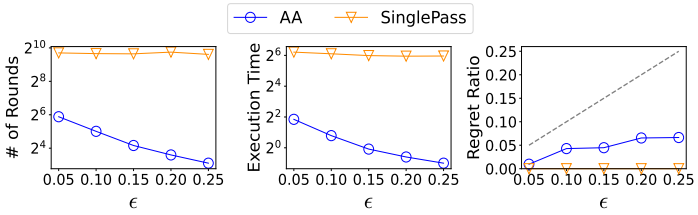
Varying threshold ϵ. We studied the impact of the threshold ϵ on all algorithms by varying it from 0.05 to 0.25.

Figure 11 shows the performance on a 4-dimensional synthetic dataset. Figures 11(a) and 11(b) report the number of interactive rounds and the execution time, respectively. As shown there, our algorithms consistently outperformed existing algorithms in all cases. For instance, when $\epsilon = 0.25$, our algorithms required only one-third of the interactive rounds compared to existing algorithms and were 1-2 orders of magnitude faster than them. In particular, each of our algorithms also had its own merits. Algorithm EA took fewer interactive rounds due to its accurate state representations, while algorithm AA ran faster since it did not need to compute any exact polyhedrons. Moreover, when ϵ increased, our algorithms required fewer interactive rounds and less execution time. For example, the number of interactive rounds required by algorithm EA decreased from 6.7 to 3.8 when ϵ varied from 0.05 to 0.25. This is because a larger ϵ could be translated to a more relaxed regret ratio requirement, allowing the algorithms to learn less information during the interaction within fewer interactive rounds. Figure 11(c) reports the actual regret ratio of the final returned point. The dotted line represents the given threshold ϵ of the regret ratio. Note that in problem ISRL, we did not aim to minimize the regret ratio. Instead, as long as the actual regret ratio is smaller than the given threshold (i.e., the dotted line), the returned point is *valid*. The results show that all algorithms returned valid points. In particular, although algorithm AA is an approximate algorithm (whose regret ratio



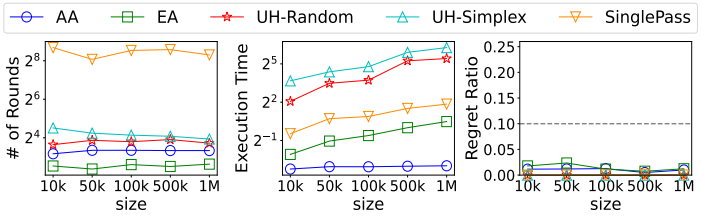
(a) (b) (c)

Figure 11: vary ϵ in 4D dataset



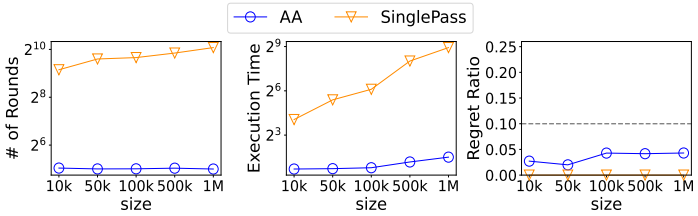
(a) (b) (c)

Figure 12: Vary ϵ in 20D dataset



(a) (b) (c)

Figure 13: Vary size in 4D dataset



(a) (b) (c)

Figure 14: Vary size in 20D dataset

is bounded by up to a factor of d^2, see Section IV-C), its actual regret ratio was still below the given threshold.

Figure 12 shows the performance on a 20-dimensional synthetic dataset. Consistent with the previous results, algorithm AA showed clear advantages in the number of interactive rounds and execution time (Figures 12(a) and (b)). It took at least an order of magnitude fewer interactive rounds and less execution time than SinglePass. For example, when $\epsilon = 0.15$, algorithm AA only required 17.9 interactive rounds within 0.95 seconds, while algorithm SinglePass needed 800.7 interactive rounds and 63.4 seconds. Figure 12(c) shows the actual regret ratio. The regret ratios of all algorithms were below the threshold (the dotted line). SinglePass had a lower actual regret ratio than AA. However, its interaction process is *overly aggressive* by asking too many questions, while the interaction process of AA asked *just enough* questions to get a valid point.

Varying dataset size n. In Figures 13 and 14, we studied the scalability of all algorithms w.r.t. the dataset size n, by varying n from 10k to 1M on the 4-dimensional and 20-dimensional datasets, respectively. Our algorithms consistently required the fewest interactive rounds in all cases. For instance, on the 4-dimensional dataset, when $n = 1$M, algorithms EA and AA required 6.2 and 10.1 interactive rounds, respectively, while the best existing algorithm UH-Simplex required 15.3 interactive rounds. Besides, the execution times of all algorithms increased with the larger dataset sizes, as expected. For instance, on the 20-dimensional dataset, the execution time of algorithm SinglePass increased from 16.7 seconds to 487.6 seconds when n varied from 10k to 1M. This is because, with a larger dataset size, an algorithm needs to handle more points to derive questions for interaction, leading to longer execution times. Nevertheless, the execution time of our algorithms only increased slightly when the dataset size increased. On the 20-dimensional dataset, the execution time of algorithm AA only increased from 1.6 to 2.9 seconds when n varied from 10k to 1M. As for the actual regret ratios, all algorithms satisfied the requirement (below the dotted line). SinglePass had a lower actual regret ratio than AA. However, it asked an excessive number of questions, leading to an unnecessarily low regret ratio.

Varying dimensionality d. In Figures 15 and 16, we evaluated the scalability of algorithms w.r.t. the dimensionality by varying d from 2 to 5 on the low-dimensional datasets and from 5 to 25 on the high-dimensional datasets, respectively. For each algorithm, when d grew, the number of interactive rounds and the execution time increased as expected. This can be attributed to the increased complexity of learning the user's utility vector in high-dimensional spaces. Nonetheless, our algorithms consistently performed the best in all cases. For example, on the 5-dimensional dataset, our algorithms EA and AA took 8.0 and 12.2 interactive rounds in 7.4 and 0.17 seconds, respectively. In comparison, the SOTA algorithm UH-Random took 21.1 interactive rounds in 166.9 seconds.

Ablation Study for AA. In algorithm AA, the action space is restricted by selecting hyper-planes near the center of the inner sphere. To evaluate the effectiveness of this method, we compared a variant of AA, namely AA-Random, which selects hyper-planes at random to restrict the action space, on a 4-dimensional dataset. Figure 19 shows the results. Algorithm AA consistently required fewer interactive rounds than AA-Random, confirming the effectiveness of our heuristic. For instance, when $\epsilon = 0.05$, AA-Random asked 18.6 questions, while AA asked 12.1 questions, which is a 34.9% reduction.

B. Performance on Real Datasets

Varying threshold ϵ. We evaluated the performance of our algorithms against existing ones by varying the threshold ϵ from 0.05 to 0.25. Figures 17 and 18 show the results on datasets *Car* and *Player*, respectively. The results show that our algorithms performed well in terms of both the number of interactive rounds and execution time on real datasets.

On dataset *Car*, our algorithm EA consistently required the fewest interactive rounds under all settings of ϵ. For example,

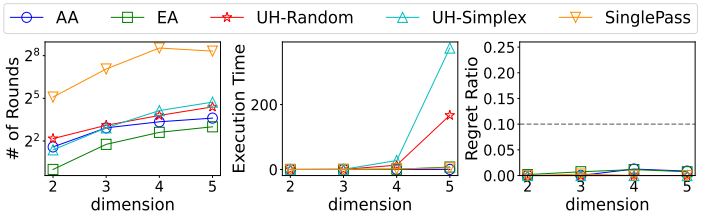


Figure 15: Vary d in low dimensions

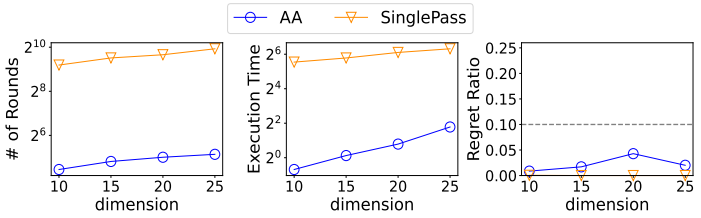


Figure 16: Vary d in high dimensions

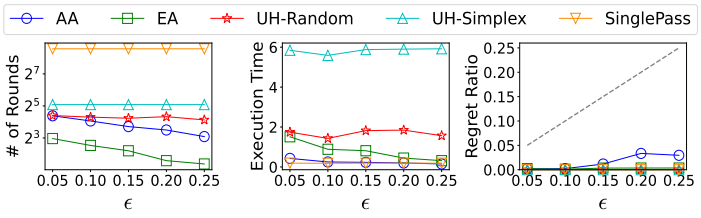


Figure 17: Vary ϵ in Car

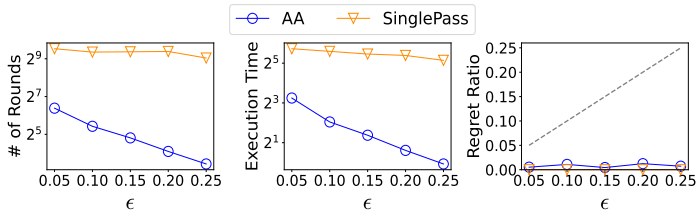


Figure 18: Vary ϵ in Player

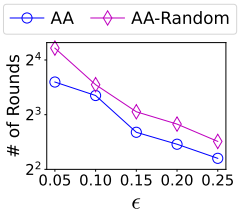


Figure 19: Ablation

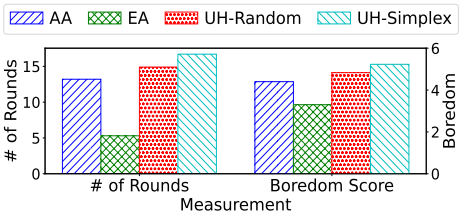


Figure 20: User Study

when $\epsilon = 0.2$, it only needed 3.0 interactive rounds, while the SOTA algorithm UH-Random required 20.1 interactive rounds. In other words, we reduced the number of interactive rounds by 85% compared to the best existing algorithms.

On dataset *Player*, our algorithm AA also beat the existing one in terms of the interactive rounds for any given ϵ. When $\epsilon = 0.25$, our algorithm AA took 10.8 interactive rounds while algorithm SinglePasss took 525.8 interactive rounds, leading to a 97.9% reduction in rounds. These findings verified the effectiveness of our algorithms in real-life scenarios.

For lack of space, we report examples of exact interactive questions asked by each algorithm in our technical report [31].

User study. We conducted a user study to verify the effectiveness of our algorithms on real users. Following [3], [9], [15], we randomly selected 1000 candidate cars from *Car*. We recruited 30 participants and reported their average results.

To reduce the workload, we excluded algorithm SinglePass, since it asked hundreds of questions (as shown in Figure 17). Each algorithm was evaluated using two metrics: (1) *the number of questions asked*, i.e., the number of interactive rounds and (2) *the boredom score*, i.e., a score between 1 and 10, with 1 being the least bored, reflecting how participants felt after answering multiple questions and reviewing the returned car.

Figure 20 shows the results. Our algorithms, EA and AA, outperformed the others across both metrics. For example, EA asked 5.3 questions, whereas existing algorithms required over 14.9 questions. Besides, participants reported lower boredom scores for our algorithms, with 3.3 for EA and 4.4 for AA, compared to boredom scores above 4.8 for the existing ones.

C. Summary

The experiments demonstrated the superiority of our algorithms over the best-known existing ones. (1) We are effective and efficient. Our algorithms EA and AA required fewer interactive rounds and less time than existing algorithms (e.g., on the 4-dimensional dataset with $\epsilon = 0.2$, while existing algorithms needed at least 14.7 interactive rounds, algorithm EA only required 4.5 interactive rounds). (2) Our algorithms scaled well on the dataset size and the dimensionality (e.g., our algorithms needed 12.2 interactive rounds when $d = 5$, while all existing algorithms required at least 21.1 interactive rounds). (3) Our algorithms showed great promise in real-world applications (e.g., on dataset *Player* with $\epsilon = 0.25$, algorithm AA achieved a 97.9% reduction in the number of interactive rounds compared to existing algorithms; in the user study, our algorithms also outperformed the other algorithms in terms of both the number of questions asked and the boredom score).

Besides, both our algorithms had demonstrated their merits. (1) Algorithm EA is an exact algorithm and it needed the fewest interactive rounds with a small execution time, due to accurate modeling. (2) Although algorithm AA is an approximate algorithm, its actual regret ratio was below the threshold. It also ran the fastest and scaled well with high dimensionality.

VI. Conclusion

In this paper, we formalize the process of the interactive regret query as a Markov Decision Process (MDP) using the interaction tree. We present an exact algorithm EA and an approximate algorithm AA based on reinforcement learning (RL). The experimental results show that compared to existing ones, our algorithms reduce the number of interactive rounds substantially and speed up the execution time by orders of magnitude under typical settings. For future work, we consider extending each interactive question to ask users to select a subset of desired tuples. While this increases user workload per round, it has the potential to learn more information in each round, thereby reducing the total number of interactive rounds.

REFERENCES

[1] M. Xie, T. Chen, and R. C.-W. Wong, "Findyourfavorite: An interactive system for finding the user's favorite tuple in the database," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2019, p. 2017–2020.

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

[3] W. Wang, R. C.-W. Wong, and M. Xie, "Interactive search with mixed attributes," in *In IEEE ICDE International Conference on Data Engineering*, 2023.

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

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

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

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

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

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

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

[11] A. LLC, 2024. [Online]. Available: https://www.alchemer.com/resources/blog/how-many-survey-questions/

[12] QuestionPro, 2024. [Online]. Available: https://www.questionpro.com/blog/optimal-number-of-survey-questions/

[13] M. Revilla and C. Ochoa, "Ideal and maximum length for a web survey," *International Journal of Market Research*, vol. 59, no. 5, pp. 557–565, 2017.

[14] G. Zhang, N. Tatti, and A. Gionis, "Finding favourite tuples on data streams with provably few comparisons," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 3229–3238.

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

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

[17] T.-Y. Liu, "Learning to rank for information retrieval," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 2010, p. 904.

[18] L. Maystre and M. Grossglauser, "Just sort it! a simple and effective approach to active preference learning," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, p. 2344–2353.

[19] B. Eriksson, "Learning to top-k search using pairwise comparisons," in *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, vol. 31. Scottsdale, Arizona, USA: PMLR, 2013, pp. 265–273.

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

[21] K. Li and J. Malik, "Learning to optimize," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=ry4Vrt5gl

[22] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," *Journal of Machine Learning Research*, vol. 23, no. 189, pp. 1–59, 2022.

[23] Z. Yang, B. Chandramouli, C. Wang, J. Gehrke, Y. Li, U. F. Minhas, P.-Å. Larson, D. Kossmann, and R. Acharya, "Qd-tree: Learning data layouts for big data analytics," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 193–208.

[24] T. Gu, K. Feng, G. Cong, C. Long, Z. Wang, and S. Wang, "The rlr-tree: A reinforcement learning based r-tree for spatial data," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–26, 2023.

[25] Z. Wang, C. Long, and G. Cong, "Trajectory simplification with reinforcement learning," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 684–695.

[26] Z. Wang, C. Long, G. Cong, and Q. Zhang, "Error-bounded online trajectory simplification with multi-agent reinforcement learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, New York, NY, USA, 2021, p. 1758–1768.

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

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

[29] W. Wang, R. C.-W. Wong, H. V. Jagadish, and M. Xie, "Reverse regret query," in *In IEEE ICDE International Conference on Data Engineering*, 2024.

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

[31] W. Wang, V. J. Wei, M. Xie, D. Jiang, L. Fan, and H. Yang, "Interactive search with deep reinforcement learning," Tech. Rep., 2025. [Online]. Available: https://github.com/anonymity848/ICDE2025/tree/main

[32] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[33] R. Cohen and L. Katzir, "The generalized maximum coverage problem," *Information Processing Letters*, vol. 108, no. 1, pp. 15–22, 2008.

[34] D. S. Hochba, "Approximation algorithms for np-hard problems," *SIGACT News*, vol. 28, no. 2, p. 40–52, jun 1997.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

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

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

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

[39] T. C. Dataset, 2024. [Online]. Available: https://www.kaggle.com/datasets/adityadesai13/used-car-dataset-ford-and-mercedes

[40] T. P. Dataset, 2024. [Online]. Available: https://www.kaggle.com/datasets/vivovinco/19912021-nba-stats?select=players.csv

[41] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 972–981.