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 college 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 the user's preference, it would greatly alleviate this burden.

The interactive regret query [5], [6], one of the most representative *multi-criteria decision-making* query, is designed for this scenario. It finds a tuple that is *close to* the user's favorite one with the help of user interaction. Specifically, it models each 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 interactive regret query works by engaging a user in a series of interactive rounds. In each round, the user answers a question. The question consists of a pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of tuples, and asks the user to select the one s/he prefers. Based on the user's answer, the user's utility function is learned implicitly. Once the utility function has been sufficiently learned, the desired tuple w.r.t. the learned utility function is returned.

Table I: The Car Database

Car	Price	Horsepower	Fuel Efficiency	$f_{\boldsymbol{u}_1}(\cdot)$
$\boldsymbol{p}_1$	\$5000	450 hp	25 mpg	5.05
$\boldsymbol{p}_2$	\$4000	400 hp	30 mpg	5.60
$\boldsymbol{p}_3$	\$6000	500 hp	22 mpg	4.50
$\boldsymbol{p}_4$	\$3500	350 hp	28 mpg	5.65
$\boldsymbol{p}_5$	\$4500	420 hp	27 mpg	5.28

Figure 1 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 represents a candidate 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 orange rectangle in the root node N_1 represents question $\langle \boldsymbol{p}_1, \boldsymbol{p}_2 \rangle$. It is connected to two child nodes, where the left child node N_2 indicates that the user prefers $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$, denoted by $\boldsymbol{p}_1 \succ \boldsymbol{p}_2$; 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 node is returned.

There are various interactive algorithms designed for this query: UH-Random [5], UH-Simplex [5], and SinglePass [6]. These algorithms define designated data structures to store the information collected during the interaction. In each interactive round, based on the information stored in these data structures, a question is selected from the candidate ones for interaction. As could be noticed, these existing algorithms are *short-term* focused, i.e., they select a question from candidate ones 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 beneficial to individual rounds, can negatively affect subsequent rounds and hinder the entire interaction process.

To see this, Figure 1 highlights two parts in the tree when a user is presented with two different initial questions at the root node, namely $\mathcal{Q}_1 = \langle \boldsymbol{p}_1, \boldsymbol{p}_2 \rangle$ and $\mathcal{Q}_2 = \langle \boldsymbol{p}_8, \boldsymbol{p}_9 \rangle$. One can observe that both questions are equally potent for interaction if we only consider their short-term effects in this individual round, since they both lead to two other internal nodes. However, the sub-tree (i.e., the long-term effects) resulting from $\mathcal{Q}_2$ is generally deeper than that resulting from $\mathcal{Q}_1$. This

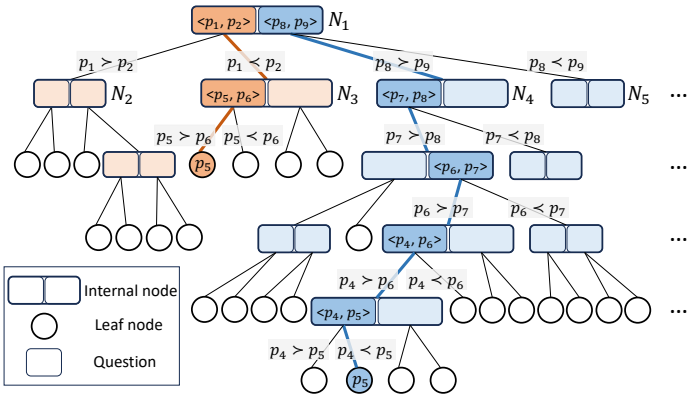


Figure 1: Interaction Process

indicates that if $\mathcal{Q}_2$ is selected for interaction (like all existing algorithms do), there will be more questions asked to the user in the subsequent (and thus, the entire) interaction.

This issue has been verified in our experiments. Existing algorithms, e.g., SinglePass [6], require asking hundreds of questions under typical settings (e.g., 727 questions on the *player* dataset [7]). This long interaction process may make users feel frustrated and struggle to provide high-quality feedback. In the literature of marketing research [8], [9], it is suggested that the maximum number of questions should be around 10. This substantial gap in the number of questions prevents existing algorithms from being viable in real-world scenarios.

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 asked with the reward so that if a large number of questions are asked, the accumulated reward is small. This design enables us to minimize the number of questions asked by maximizing the accumulated reward, 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 maps the user's utility function to a geometric space and maintains a polyhedron in this space to describe the information collected during the interaction. The use of polyhedrons has been proven effective by many existing works [3], [5], [10] for precisely describing the user's utility function. By leveraging this formulation, algorithm EA optimizes the decision making and is able to identify the tuple to be returned by a small number of interactive rounds.

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 based on RL. It reduces the number of interactive rounds needed by the existing algorithms by more than 70% under typical settings.
- We show an approximate algorithm AA based on RL. It is scalable in terms of the number of attributes. It can handle datasets with 4-5 times more attributes than the SOTA.
- 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.

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. It is designed to interact with a user to learn his/her preference, and then return a tuple whose regret ratio is smaller than a given threshold. There are a few interactive algorithms for this query [5], [6], [11]. The first algorithm UtilityApprox is proposed by [11]. 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 [5].

To overcome this deficiency, [5] 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 several 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, these two algorithms involve complicated computation to derive users' preferences, limiting their applicability to datasets with only a few attributes. To accelerate, [6] proposes algorithm SinglePass, at the cost of collecting less information during the interaction. Consequently, it needs to ask many questions (e.g., hundreds of questions). Besides, as remarked previously, all these three algorithms design each question by only focusing on the current interactive round without considering the overall interaction process. This leads to more questions and longer execution times than our algorithms under typical settings.

There are other relevant interactive algorithms designed for other types of queries. For instance, [12]–[14] proposes algorithms for the interactive similarity query. These algorithms aim to learn the user's desired tuple and the corresponding distance function through user interaction, ultimately returning tuples with the smallest distance to the desired tuple. However, these algorithms ask users to assign *relevance scores* to thousands of tuples, which is impractical and overly demanding in real-world scenarios. Another algorithm Adaptive proposed by [15] 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 [15]. Moreover, [3], [10] propose algorithms HDPI and GE-Graph, that return a tuple whose utility is the highest or among the top-k, respectively. However, these algorithms concentrate on the ranking of tuples, which is a secondary source of 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 the field of machine learning, the interactive regret query is related to the problem of *learning to rank* [16]–[19], which learns the ranking of tuples by interacting with the user. However, most of the existing algorithms [16]–[18] often overlook the inter-relations between attributes (e.g., understanding that a price of \$200 is more desirable than \$500 by being cheaper). Thus, it necessitates a large amount of questions for interaction [5] to learn such inter-relations. Algorithm *ActiveRanking* [19] considers the inter-relation between tuples to learn the ranking by interacting with the user. However, it aims to derive the complete ranking of all tuples, which can result in unnecessary questions due to the similar reason stated for [15].

B. Reinforcement Learning

Reinforcement learning (RL) is proposed to guide agents to decide what actions to take in specific environments [20], [21], 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 [22], [23] and trajectory simplification [24], [25] in the database community.

In the area of index construction, [22] 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, [23] 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.

As for trajectory simplification, RL has been effectively utilized to reduce the number of points in a trajectory while preserving its essential features. [24] 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 sliding window mechanism allows for an online simplification process, ensuring that only the relevant points are retained. [25] also employs two RL agents. The first agent is to identify a set of candidate points for deletion, and the second agent is to make the final decision on which points to delete.

In this paper, we model the process of the interactive regret query as a Markov Decision Process and use RL to optimize the process. To the best of our knowledge, we are the first to utilize RL for the interactive regret query.

III. Problem Definition

The input of our problem is a set $\mathcal{D}$ of n tuples. Each tuple is described by d attributes, and thus, 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 the tuple. Without loss of generality, following [5], [26], we assume that each dimension is normalized to $(0, 1]$ and a large value is preferred. Table III shows an example, where $\mathcal{D}$ contains five 2-dimensional points. In the rest of the paper, we use the words "tuple/point" and "attribute/dimension" interchangeably. The frequently used notations are summarized in Table II.

Utility Function. Following [5], [10], [27], we model the user preference by a linear function $f_{\boldsymbol{u}}$, called the *utility function*, which is one of the most proliferate and effective representation since the inception of utility modeling [15], [28], [29].

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

- $\boldsymbol{u} = (u[1], ..., u[d])$, 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 [11], [30], 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 [5], [30]. 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*. Given a dataset $\mathcal{D}$ and a threshold ϵ, the goal is to interact with a user to find a point $\boldsymbol{q} \in \mathcal{D}$ whose regret ratio over $\mathcal{D}$ w.r.t. the user utility vector is smaller than threshold ϵ. Following the framework in [5], [10], the interaction occurs in *rounds*. Each round consists of three steps.

- *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 decision of each interactive round to minimize the number of questions asked to the user, i.e., the total number of interactive rounds. Formally, we are interested in the following problem.

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* [31] in space $\mathbb{R}^d$. For example, in the 3-dimensional space $\mathbb{R}^3$, as shown in Figure 3, the utility space $\mathcal{U}$ is a triangle.

Table II: Frequently Used Notations

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

Hyper-plane. In a d-dimensional space $\mathbb{R}^d$, we can build a *hyper-plane* [31] for each pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points in $\mathcal{D}$.

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

This hyper-plane passes through the origin with its unit normal in the same direction as $\boldsymbol{p}_i - \boldsymbol{p}_j$. It divides space $\mathbb{R}^d$ into two halves, called *half-spaces* [31]. The half-space above (resp. below) hyper-plane $h_{i,j}$, 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 that of $\boldsymbol{p}_j$ w.r.t. the user's utility vector [5].

Combining these definitions, the lemma below shows our foundation for learning the user's preference. For lack of space, some proofs of lemmas/theorems can be found in the Appendix.

Lemma 1 ([5], [10]). 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. Assume that there are two points $\boldsymbol{p}_1 = (0, 0.6, 0)$ 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 $\boldsymbol{p}_1 - \boldsymbol{p}_2 = (-0.4, 0.6, 0)$) shown in Figure 2. 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 can 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 $\mathcal{R}$. The utility range is the intersection of a set of half-spaces and utility space $\mathcal{U}$, i.e., it is a polyhedron [31] 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 (see Sections IV-B and IV-C).

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

p	$p[1]$	$p[2]$	$f_u(p)$
$\boldsymbol{p}_1$	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

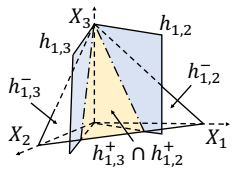


Figure 2: Utility Range

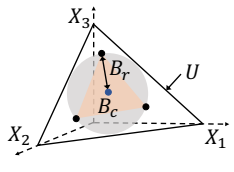


Figure 3: Outer Sphere

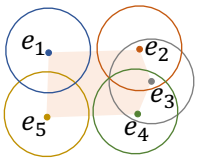


Figure 4: Construct E

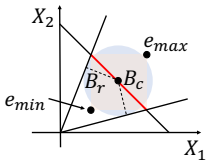


Figure 5: State in AA

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 2, 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 1).

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.

Each internal node N denotes an intermediate interaction status. Except for the utility range, it 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_i and N_j of N, where node N_i (resp. N_j) 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 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 1. If the question $\langle \boldsymbol{p}_1, \boldsymbol{p}_2 \rangle$ is selected and the user prefers $\boldsymbol{p}_2$ to $\boldsymbol{p}_1$, the interaction status moves to node N_3, where $\mathcal{R}_{N_3} = \mathcal{R}_{N_1} \cap h^+_{2,1}$. Note that in Figure 1, 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 presented with different questions or they give different answers to the questions. The length of each path is equal to 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 traverse 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, it is a question (i.e., a pair of points) selected from the candidate pool. The user is asked which one s/he prefers between these 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. prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$), we move from node N to its child node N_i (resp. N_j) and *transit* from state $\mathcal{R}_N$ to state $\mathcal{R}_{N_i} = \mathcal{R}_N \cap h^+_{i,j}$ (resp. $\mathcal{R}_{N_j} = \mathcal{R}_N \cap h^+_{j,i}$). 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.

We present two algorithms that substantiate this MDP in different ways. Both algorithms utilize reinforcement learning to develop an interactive agent, which strategically determines the best action (i.e., a pair of points) for each state (i.e., utility range) to maximize the accumulated reward (i.e., minimize the path length). The first algorithm provides an accurate output, while the second one achieves better scalability.

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 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 [10], [30], [31], 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 3. 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. We say that $\boldsymbol{e}$ *covers* the extreme utility vectors in $S_{\boldsymbol{e}}$. Our objective is to select a set E of m_e extreme utility vectors that 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, identifying the optimal set E is an NP-hard problem.

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.

Proof Sketch. We prove it based on the *Maximum Coverage Problem* [33], which 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 4. 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 $\mathcal{B}_r$ (see Figure 3). Mathematically, the construction of the outer sphere can be modeled as an optimization problem based on the set $\mathcal{E}$ of $\mathcal{R}$. The objective is to minimize 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 smaller than or equal to the radius $\mathcal{B}_r$, i.e.,

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

where $||\cdot||$ denotes the Euclidean distance between two points.

Since this is a non-convex optimization problem, 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 extreme utility 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 shows that this iterative algorithm converges to a local optimum.

Lemma 3. In each successive iteration, the outer sphere's radius $\mathcal{B}_r$ 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 the utility range $\mathcal{R}$ 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*. Let us 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 $\mathcal{R}$ to a terminal polyhedron effectively.

To achieve this, we construct several terminal polyhedrons inside $\mathcal{R}$. Each of these terminal polyhedrons can be obtained by narrowing down the current utility range $\mathcal{R}$ 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 positive 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 $\mathcal{R} \cap h^+_{i,j}$ and those terminal polyhedrons in $\mathcal{R} \cap h^-_{i,j}$ 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 how 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.

One may notice that set $\mathcal{V}$ consists of two parts: randomly sampled utility vectors and extreme utility vectors. The former part 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 $\mathcal{R}$.

Proof Sketch. We prove it based on the well-known Chernoff-Hoeffding Inequality [35]. □

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 in set $\mathcal{V}$ 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., a utility range $\mathcal{R}$) 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. Note that, as shown below, our method of constructing the action space enables $\mathcal{R}$ to be strictly narrowed. When $\mathcal{R}$ itself is a terminal polyhedron (by 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

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

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 [36] 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 [36] 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.

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 $\mathcal{R} \leftarrow \mathcal{U}$ (line 5) and compute the current state s (line 6). 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 8). 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 9-12). 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 13-17), 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 7). After that, we randomly draw a batch of transitions from the replay memory to update the main network $Q(s,a;\Theta)$ (line 19). The target network $\hat{Q}$ is periodically synchronized with the main network $Q(s,a;\Theta)$ to stabilize the learning process (line 20).

Inference. Given the learned Q-function, we present the interaction process in Algorithm 2. Initially, we set the utility range $\mathcal{R} \leftarrow \mathcal{U}$ (line 3) and compute the current state s (line 4). 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 6). 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 7-10). Then, we compute the new state s' based on the updated $\mathcal{R}$ (line 11). The interaction stops when $\mathcal{R}$ becomes a terminal polyhedron (line 5). 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).

Algorithm 1: Algorithm EA (Training)

Input: A training set of utility vectors, ϵ and a point set $\mathcal{D}$
Output: Learned Q-function $Q(s, a; \Theta)$
Initialize $Q(s, a; \Theta)$, $\hat{Q}(s, a; \Theta')$ and replay memory $\mathcal{M}$;
for *each* $\boldsymbol{u}$ *in the training set* **do**
  $\mathcal{R} \leftarrow \mathcal{U}$;
  $s \leftarrow$ a state based on the outer sphere and selected extreme utility vectors of $\mathcal{R}$;
  while $\mathcal{R}$ *is not a terminal polyhedron by Lemma 6* **do**
    $a \leftarrow$ an action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ by ε-greedy on Q-values;
    if $\boldsymbol{p}_i \cdot \boldsymbol{u} \geq \boldsymbol{p}_j \cdot \boldsymbol{u}$ **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{i,j}$;
    else
      $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{j,i}$;
    $s' \leftarrow$ the new state based on the updated $\mathcal{R}$;
    if *the updated* $\mathcal{R}$ *is a terminal polyhedron* **then**
      Add (s, a, c, s') into $\mathcal{M}$;
    else
      Add $(s, a, 0, s')$ into $\mathcal{M}$;
    $s \leftarrow s'$;
  Draw samples from $\mathcal{M}$ to update $Q(s, a; \Theta)$;
  Periodically synchronize $\hat{Q}(s, a; \Theta')$;

Algorithm 2: Algorithm EA (Inference)

Input: Learned Q-function $Q(s, a; \Theta)$, ϵ and a point set $\mathcal{D}$
Output: A tuple $\boldsymbol{p}$ whose regret ratio is below ϵ
$\mathcal{R} \leftarrow \mathcal{U}$;
$s \leftarrow$ a state based on the outer sphere and selected extreme utility vectors of $\mathcal{R}$;
while $\mathcal{R}$ *is not a terminal polyhedron by Lemma 6* **do**
  $a \leftarrow$ an action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ with the largest Q-value;
  if *the user prefers* $\boldsymbol{p}_i$ *to* $\boldsymbol{p}_j$ **then**
    $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{i,j}$;
  else
    $\mathcal{R} \leftarrow \mathcal{R} \cap h^+_{j,i}$;
  $s \leftarrow$ the new state based on the updated $\mathcal{R}$;
return The point $\boldsymbol{p}_{\mathcal{R}}$ s.t. $\forall \boldsymbol{u} \in \mathcal{R}$, $regratio(\boldsymbol{p}_{\mathcal{R}}, \boldsymbol{u}) \leq \epsilon$;

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. Figure 5 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. Let $\mathcal{B}_r$ (resp. $\mathcal{B}_c$) be the radius (resp. center) of $\mathcal{B}$. By the definition of an inner sphere, for each $h^+_{i,j} \in \mathcal{H}$ 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 $\mathcal{B}_r$. Formally, we formulate the LP as follows.

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

where 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 defined to be the smallest axis-aligned rectangle that contains $\mathcal{R}$ (see Figure 5 for an example). 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., $\forall i \in [1, d], \boldsymbol{e}_{min}[i] = \min\{u[i] | \boldsymbol{u} \in \mathcal{R}\}$ and $\boldsymbol{e}_{max}$ contains the largest values of utility vectors in $\mathcal{R}$ along each dimension, i.e., $\forall i \in [1, d], \boldsymbol{e}_{max}[i] = \max\{u[i] | \boldsymbol{u} \in \mathcal{R}\}$. 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 the i-th dimension value of $\boldsymbol{e}_{max}$.

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

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. Specifically, we adopt an effective heuristic method to select point pairs that can narrow $\mathcal{R}$ as much as possible, without incurring the costly computation of exact 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 inner sphere's center $\mathcal{B}_c$. 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.,

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

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 problem, we utilize the two vectors $\boldsymbol{e}_{min}$ and $\boldsymbol{e}_{max}$ of the outer rectangle of $\mathcal{R}$. We stop the interaction when the Euclidean distance between $\boldsymbol{e}_{min}$ and $\boldsymbol{e}_{max}$ is smaller than $2\sqrt{d}\epsilon$, i.e., $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$, and return the point $\boldsymbol{p}$ that has the highest utility w.r.t. $\boldsymbol{u} = (\boldsymbol{e}_{min} + \boldsymbol{e}_{max})/2$ (stopping condition). As shown in the lemma below, 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 setting of rewards 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 the Interactive Agent.: Following a similar process in Section IV-B2, we also adopt the deep Q-learning with experience replay for algorithm AA.

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 $\hat{Q}(s, a; \Theta')$;

Algorithm 4: Algorithm AA (Inference)

1 **Input:** Learned Q-function $Q(s, a; \Theta)$, ϵ and a point set $\mathcal{D}$
2 **Output:** A point $\boldsymbol{p}$ with regret ratio w.r.t. the user below $d^2\epsilon$
3 $\mathcal{H} \leftarrow \emptyset$; $s \leftarrow$ the state based on $\mathcal{H}$;
4 **while** $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$ **do**
5 $\quad a \leftarrow$ an action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ with the largest Q-value;
6 $\quad$ **if** *the user prefers* $\boldsymbol{p}_i$ *to* $\boldsymbol{p}_j$ **then**
7 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{h^+_{i,j}\}$;
8 $\quad$ **else**
9 $\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{h^+_{j,i}\}$;
10 $\quad s \leftarrow$ the state based on the updated $\mathcal{H}$;
11 **return** $\boldsymbol{q} = \arg\max_{\boldsymbol{p} \in \mathcal{D}} f_{\boldsymbol{u}}(\boldsymbol{p})$, where $\boldsymbol{u} = \frac{\boldsymbol{e}_{min} + \boldsymbol{e}_{max}}{2}$;

Training. We present the training process in Algorithm 3. Since it employs a similar framework as Algorithm 1, its line-by-line explanation is omitted. 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 is detailed in Algorithm 4. This process is similar to Algorithm 2, with the key differences in the maintenance of set $\mathcal{H}$ (lines 6-9), the state representation (lines 3 and 10), and the stopping condition (line 4). This process finally returns the point that has the highest utility w.r.t. utility vector $\boldsymbol{u} = (\boldsymbol{e}_{min} + \boldsymbol{e}_{max})/2$ (line 11).

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.

V. Experiment

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

Datasets. The experiments were conducted on both synthetic and real datasets that were commonly used in existing studies [5], [37]–[39]. For the synthetic datasets, we used *anti-correlated* datasets that were produced by the generator designed for *skyline* operators [37], [40]. For the real datasets, we used datasets *Car* [41] and *Player* [7]. Dataset *Car* contains 10,668 cars described by price, mileage, and miles per gallon (mpg). Dataset *Player* consists of $17,386$ basketball players, each described by twenty attributes, including age, points, offensive rebounds, defensive rebounds, assists, etc.

For all these datasets, each attribute was normalized to $(0, 1]$. Note that the existing work [5] preprocessed datasets to contain skyline points only (which are all possible top-1 points w.r.t. at least a utility function). Consistent with the setting, 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* [5], 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* [5], 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* [6], 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 compute and maintain some polyhedrons during the interaction, which can be costly in high-dimensional space. Therefore, we follow the setting of existing studies [5] and did not compare them when the dimensionality 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 dimensionality d. Following [5], [6], the default setting 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 [42] 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 to minimize the gap between the Q-value predicted by $Q(s, a; \Theta)$ and the optimal Q-value derived from $\hat{Q}(s, a; \Theta')$. The discount factor γ was set to be 0.8. The synchronization of target network $\hat{Q}(s, a; \Theta')$ with $Q(s, a; \Theta)$ was done once every 20 updates of the main network.

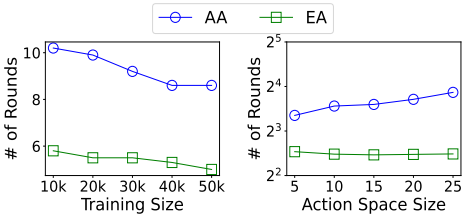
Performance measurement. We evaluated the performance of each algorithm using three measurements. (1) *Execution time.* The total time taken by the entire interaction process. (2) *Regret ratio.* The actual regret ratio of the returned point. Note that in problem ISRL, we did not minimize the regret ratio. Instead, as long as the actual regret ratio is smaller than the given threshold, the point returned is valid. (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 6(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 6(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.

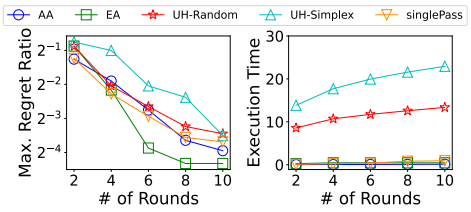
Interaction process. Figures 7 and 8 show the progress in the interaction process on the 4-dimensional and 20-dimensional synthetic datasets, respectively. We report the current *maximum regret ratio* and the accumulated execution time at the end of each interactive round. Note that the current maximum regret ratio is different from the actual regret ratio of the final returned point. Following [5], [11], we computed it as follows. At the end of any interactive round, we could obtain the inner sphere based on the set of intersecting half-spaces learned so far. 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 sample utility vector and 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 7(a)). After 8 interactive rounds, its maximum regret ratio dropped below 0.05. In comparison, the maximum regret ratio of UH-Simplex was 0.19, which was about 4 times higher than ours. Besides, the execution time of algorithm EA was also low. It took 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 and achieved a maximum regret ratio below 0.1; this was 31% faster than competitor algorithm SinglePass, whose maximum regret ratio



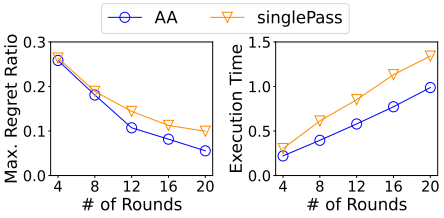
(a) (b)

Figure 6: Training



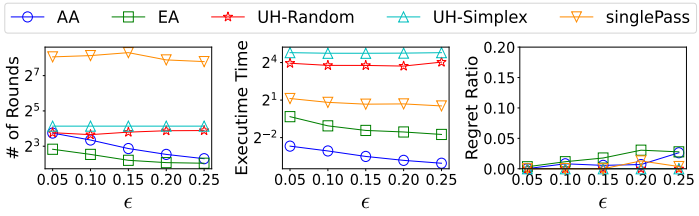
(a) (b)

Figure 7: Interaction process in 4D



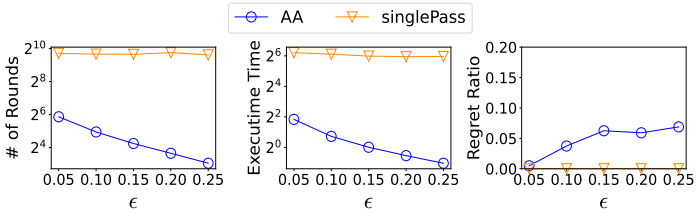
(a) (b)

Figure 8: Interaction process in 20D



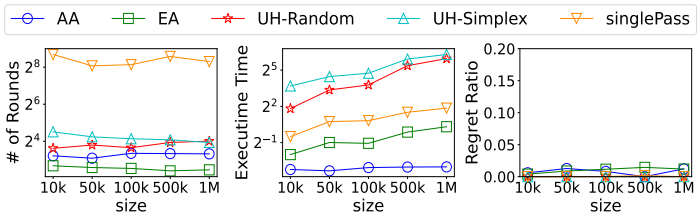
(a) (b) (c)

Figure 9: vary ϵ in 4D dataset



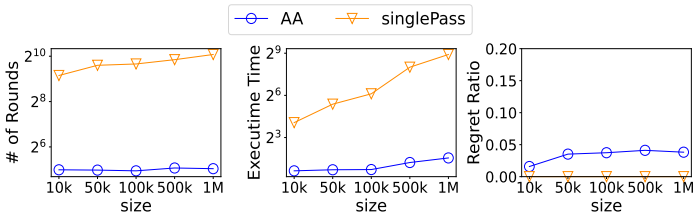
(a) (b) (c)

Figure 10: Vary ϵ in 20D dataset



(a) (b) (c)

Figure 11: Vary size in 4D dataset



(a) (b) (c)

Figure 12: Vary size in 20D dataset

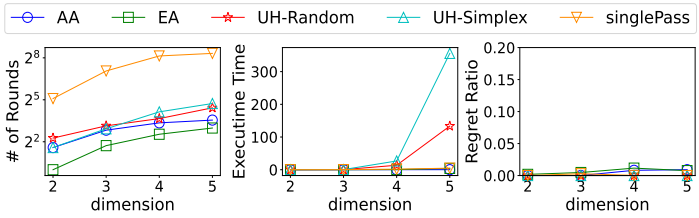
was 34% higher than ours. These results verified that our algorithms progressed well during the interaction process.

Varying threshold ϵ. We studied the impact of the threshold ϵ of the regret ratio on all algorithms by varying it from 0.05 to 0.25. The other parameters were set by default.

Figure 9 shows the performance on a 4-dimensional synthetic dataset. Figures 9(a) and 9(b) report the number of interactive rounds and execution time, respectively. As shown there, our algorithms consistently outperformed existing ones 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 7.1 to 4.1 when ϵ varied from 0.05 to 0.25. This is because a larger ϵ could be translated to an easier regret ratio requirement, and thus, the algorithms only needed to learn less information during the interaction with fewer interactive rounds. However, most existing algorithms did not effectively leverage this property. They needed almost the same number of interactive rounds, regardless of the value of ϵ. Figure 9(c) reports the actual regret ratio of the final returned point. The results show that all the algorithms achieved the regret ratio below the threshold. In particular, although the proposed algorithm AA is an approximate algorithm (whose regret ratio is bounded by up to a factor of d^2, see Section IV-C), its actual regret ratio was still below the given threshold.

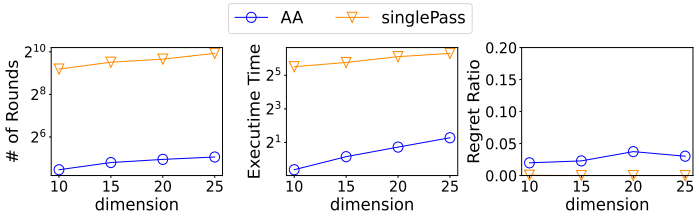
Figure 10 shows the performance on a 20-dimensional synthetic dataset. Consistent with the previous results, algorithm AA showed clear advantages in the number of interactive rounds and execution time (Figures 10(a) and (b)). It took at least an order of magnitude fewer interactive rounds and less execution time than algorithm SinglePass. For example, when $\epsilon = 0.15$, algorithm AA only required 19 interactive rounds within 1.1 seconds, while algorithm SinglePass needed 800.7 interactive rounds and 63.4 seconds. Figure 10(c) shows the actual regret ratio of the final returned point. It again verified that although algorithm AA was an approximate algorithm, its actual regret ratio was below the threshold empirically.

Varying dataset size n. In Figures 11 and 12, 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 5.5 and 10.0 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



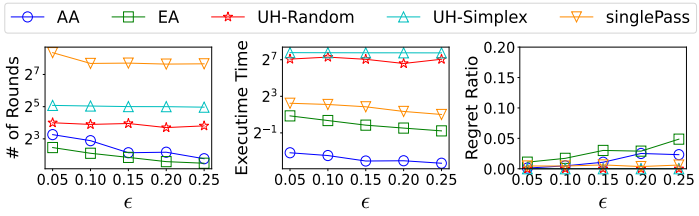
(a) (b) (c)

Figure 13: Vary d in low dimensions



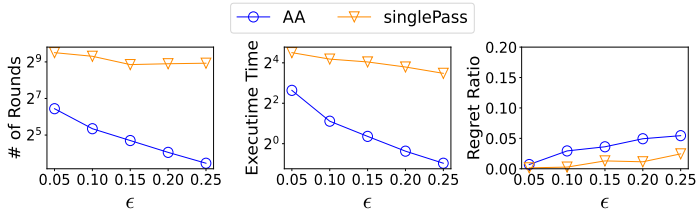
(a) (b) (c)

Figure 14: Vary d in high dimensions



(a) (b) (c)

Figure 15: Vary ϵ in Car



(a) (b) (c)

Figure 16: Vary ϵ in Player

instance, on the 20-dimensional dataset, the execution time of algorithm SinglePass increased from 16.7 seconds to 480.6 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 the question 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. These results demonstrate the good scalability of our algorithms in handling large datasets.

Varying dimensionality d. In Figures 13 and 14, 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 7.9 and 11.7 interactive rounds in 4.0 and 0.17 seconds, respectively. In comparison, the SOTA algorithm UH-Random took 21.5 interactive rounds in 133.5 seconds.

B. Performance on Real Datasets

We evaluated the performance of our algorithms against existing ones on two real datasets by varying the threshold ϵ from 0.05 to 0.25. Figures 15 and 16 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, when $\epsilon = 0.2$, it only needed 3.0 interactive rounds, while the SOTA algorithm UH-Random required 13 interactive rounds. In other words, we reduced the number of interactive rounds by 77% 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 11 interactive rounds while algorithm SinglePasss took 487.2 interactive rounds, leading to a 97.7% reduction in rounds. These findings verify the effectiveness of our algorithms in real-life scenarios.

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 ones (e.g., on the 4-dimensional dataset with $\epsilon = 0.2$, while existing algorithms needed at least 14.7 interactive rounds, EA only required 4.2 interactive rounds). (2) Our algorithms scaled well on the dataset size and the dimensionality (e.g., our algorithms needed 11.7 interactive rounds when $d = 5$, while all existing algorithms required at least 21.5 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 97.7% reduction in the number of interactive rounds compared to existing ones). In summary, both our algorithms had demonstrated their advantages. (1) Algorithm EA is an exact algorithm and it needed the fewest interactive rounds with a small execution time, due to its accurate modeling. (2) Although algorithm AA is an approximate algorithm, its actual regret ratio was still 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 using the interaction tree. This establishes the foundation for designing interactive algorithms based on reinforcement learning (RL). We present an exact algorithm EA and an approximate algorithm AA,

to strike a balance between accuracy and scalability. 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. As for future work, we consider the case where users make mistakes when answering questions.

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] 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.

[6] 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.

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

[8] QuestionPro, 2021. [Online]. Available: https://www.questionpro.com/blog/optimal-number-of-survey-questions/

[9] 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.

[10] 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.

[11] 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.

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

[13] I. Bartolini, P. Ciaccia, and M. Patella, "Domination in the probabilistic world: Computing skylines for arbitrary correlations and ranking semantics," *ACM Transactions on Database System*, vol. 39, no. 2, 2014.

[14] I. Bartolini, P. Ciaccia, and F. Waas, "Feedbackbypass: A new approach to interactive similarity query processing," in *Proceedings of the 27th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 201–210.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

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

[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] J. M. Phillips, "Chernoff-hoeffding inequality and applications," 2013. [Online]. Available: https://arxiv.org/abs/1209.6396

[36] 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.

[37] 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.

[38] 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.

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

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

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

[42] 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.

APPENDIX A
PROOF

Proof of Lemma 1. Denote the user's utility vector by $\boldsymbol{u}$.

If a user prefers point $\boldsymbol{p}_i$ to point $\boldsymbol{p}_j$, then $f_{\boldsymbol{u}}(\boldsymbol{p}_i) > f_{\boldsymbol{u}}(\boldsymbol{p}_j)$, i.e., $\boldsymbol{u} \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j) > 0$. This implies that $\boldsymbol{u} \in h^+_{i,j}$ (or $\boldsymbol{u} \in h^-_{j,i}$), and thus, $\boldsymbol{u}$ must be in $h^+_{i,j} \cap \mathcal{U}$.

If $\boldsymbol{u}$ is in $h^+_{i,j} \cap \mathcal{U}$, then $\boldsymbol{u} \in h^+_{i,j}$ (or $\boldsymbol{u} \in h^-_{j,i}$). We have $f_{\boldsymbol{u}}(\boldsymbol{p}_i) > f_{\boldsymbol{u}}(\boldsymbol{p}_j)$, and thus, the user must prefer $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$. □

Proof of Lemma 2. We prove it based on the *Maximum Coverage Problem* [33], [34], which is an NP-hard problem. Given a number k and a collection of sets $\mathcal{C} = \{C_1, C_2, C_3...\}$, the Maximum Coverage Problem is to find a subset $\mathcal{C}' \subseteq \mathcal{C}$ such that $|\mathcal{C}| \leq k$ and $|\bigcup_{C_i \in \mathcal{C}'} C_i|$ is maximized.

We reduce our problem to the Maximum Coverage Problem. Concretely, (1) each set $S_{\boldsymbol{e}}$ corresponds to a set C_i and (2) the integer m_e corresponds to the number k in the Maximum Coverage Problem. The goal is to select m_e sets $S_{\boldsymbol{e}}$ such that the union of these sets contains the maximum number of extreme utility vectors. □

Proof of Lemma 3. Denote the center of the outer sphere before an iteration and after the iteration by $\mathcal{B}_c$ and $\mathcal{B}'_c$, respectively. Consider the extreme utility vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ with the largest and second-largest distances from $\mathcal{B}_c$, respectively. Since we move $\mathcal{B}_c$ towards $\boldsymbol{e}_1$ by an offset $\frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||)$, we have

$$\begin{aligned} ||\mathcal{B}'_c - \boldsymbol{e}_1|| &= ||\mathcal{B}_c - \boldsymbol{e}_1|| - \frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||) \\ &\leq ||\mathcal{B}_c - \boldsymbol{e}_1|| \end{aligned}$$

For any other extreme utility vector $\boldsymbol{e}$ of $\mathcal{R}$,

$$\begin{aligned} ||\mathcal{B}'_c - \boldsymbol{e}|| &= ||\mathcal{B}_c - \boldsymbol{e}|| + \frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||) \\ &\leq ||\mathcal{B}_c - \boldsymbol{e}_2|| + \frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||) \\ &\leq ||\mathcal{B}_c - \boldsymbol{e}_1|| \end{aligned}$$

Since polyhedron $\mathcal{R}$ is convex, any utility vector $\boldsymbol{u} \in \mathcal{R}$ can be represented by a linear combination of the extreme utility vectors. Thus, for any utility vector $\boldsymbol{u} \in \mathcal{R}$,

$$\begin{aligned} ||\mathcal{B}'_c - \boldsymbol{u}|| &\leq ||\mathcal{B}_c - \boldsymbol{u}|| + \frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||) \\ &\leq ||\mathcal{B}_c - \boldsymbol{e}_1|| + \frac{1}{2}(||\mathcal{B}_c - \boldsymbol{e}_1|| - ||\mathcal{B}_c - \boldsymbol{e}_2||) \\ &\leq ||\mathcal{B}_c - \boldsymbol{e}_1|| \end{aligned}$$

To sum up, after the iteration, the largest distance from a utility vector to the sphere's center will not be larger than that before the iteration, i.e., the radius of the sphere will not be larger than that before the iteration. □

Proof of Lemma 4. For each $\boldsymbol{p}_j$, based on the definition of $h^+_{i,j}$, we have $\forall \boldsymbol{u} \in \mathbb{R}^d$, $\boldsymbol{u} \cdot (\boldsymbol{p}_i - (1-\epsilon)\boldsymbol{p}_j) > 0$, i.e., $\boldsymbol{u} \cdot (\boldsymbol{p}_j - \boldsymbol{p}_i) < \boldsymbol{u} \cdot \epsilon \boldsymbol{p}_j$. Thus, for any $\boldsymbol{p_j} \in \mathcal{D} \setminus \{\boldsymbol{p_i}\}$ and any $\boldsymbol{u} \in \mathcal{R}_N \cap \bigcap_{\boldsymbol{p_j} \in \mathcal{D} \setminus \{\boldsymbol{p}_i\}} h^+_{i,j}$, we have $\boldsymbol{u} \cdot (\boldsymbol{p}_j - \boldsymbol{p}_i) < \boldsymbol{u} \cdot \epsilon \boldsymbol{p}_j$, i.e., $\frac{\boldsymbol{u} \cdot (\boldsymbol{p}_j - \boldsymbol{p}_i)}{\boldsymbol{u} \cdot \boldsymbol{p}_j} < \epsilon$. □

Proof of Lemma 5. Let us represent the volume of $\mathcal{R}$ by $vol(\mathcal{R})$. Denote by $sample(\mathcal{T})$ and $vol(\mathcal{T})$ the volume and the number of sampled utility vectors inside $\mathcal{T}$, respectively. Based on the well-known Chernoff-Hoerffding Inequality [2], [35], if sampling size $\mathcal{N} = O((1/\tau^2)(d + \ln(1/\delta))$, we can achieve $|\frac{sample(\mathcal{T})}{\mathcal{N}} - \frac{vol(\mathcal{T})}{vol(\mathcal{R})}| \leq \tau$, with probability at least $1 - \delta$.

Thus, for the two polyhedrons with volumes $vol(\mathcal{T}_1)$ and $vol(\mathcal{T}_2)$, we have

$$|\frac{sample(\mathcal{T}_1)}{\mathcal{N}} - \frac{vol(\mathcal{T}_1)}{vol(\mathcal{R})}| \leq \tau$$

$$|\frac{sample(\mathcal{T}_2)}{\mathcal{N}} - \frac{vol(\mathcal{T}_2)}{vol(\mathcal{R})}| \leq \tau$$

Combining these two inequalities, we have

$$|\frac{sample(\mathcal{T}_1)}{\mathcal{N}} - \frac{vol(\mathcal{T}_1)}{vol(\mathcal{R})} - \frac{sample(\mathcal{T}_2)}{\mathcal{N}} + \frac{vol(\mathcal{T}_2)}{vol(\mathcal{R})}| \leq 2\tau$$

, and then

$$|\frac{sample(\mathcal{T}_1) - sample(\mathcal{T}_2)}{\mathcal{N}} - \frac{vol(\mathcal{T}_1) - vol(\mathcal{T}_2)}{vol(\mathcal{R})}| \leq 2\tau$$

Thus, 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 $1 - \delta$. □

Proof of Lemma 6. If there is only one terminal polyhedron constructed, all the extreme utility vectors of $\mathcal{R}$ must lie within this polyhedron. Otherwise, we can construct another polyhedron based on the extreme utility vector that is not included. Since the polyhedron is convex, meaning any utility vector within it can be expressed as a linear combination of its extreme utility vectors, we can conclude that any utility vector in $\mathcal{R}$ must also be within the polyhedron $\mathcal{R}_{\boldsymbol{u}}$. This means $\mathcal{R}$ is contained in a terminal polyhedron. □

Proof of Lemma 7. Consider any pair $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ of points from $P_{\mathcal{R}}$. Based on our terminal polyhedron construction, points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ must have the highest utilities w.r.t. some utility vectors in $\mathcal{R}$, respectively. This means there exist some utility vectors $\boldsymbol{u}_1 \in \mathcal{R}$ (resp. $\boldsymbol{u}_2 \in \mathcal{R}$) such that $f_{\boldsymbol{u}_1}(\boldsymbol{p}_i) > f_{\boldsymbol{u}_1}(\boldsymbol{p}_j)$ (resp. $f_{\boldsymbol{u}_2}(\boldsymbol{p}_i) < f_{\boldsymbol{u}_2}(\boldsymbol{p}_j)$). Thus, $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$ and $\mathcal{R} \cap h^-_{i,j} \neq \emptyset$. If a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, $\mathcal{R}$ can be strictly narrowed by pruning the part $\mathcal{R} \cap h^-_{i,j}$. Similarly for the case where a user prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$. □

Proof of Theorem 1. Recall our action space construction. Since the pairs of points are from $P_{\mathcal{R}}$, they have the highest utilities w.r.t. some utility vectors in the utility range, respectively. No matter which action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ the interactive agent selects, $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$ and $\mathcal{R} \cap h^+_{j,i} \neq \emptyset$. If a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, $\mathcal{R}$ can be strictly narrowed by pruning the part $\mathcal{R} \cap h^+_{j,i}$. In this way, we will not build a terminal polyhedron that corresponds to $\boldsymbol{p}_i$ later. The number of polyhedrons constructed must be strictly reduced by at least 1. Similarly for the case where a user prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$. Since $|\mathcal{D}| = n$, there are $O(n)$ interactive rounds. □

Proof of Lemma 8. Since $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$ and $\mathcal{R} \cap h^+_{j,i} \neq \emptyset$, if a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, $\mathcal{R}$ can be strictly narrowed by pruning the part $\mathcal{R} \cap h^+_{j,i}$. Similarly for the case where a user prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$. □

Proof of Lemma 9. Denote the user's utility vector by $\boldsymbol{u}^*$ and the middle utility vector between $\boldsymbol{e}_{min}$ and $\boldsymbol{e}_{max}$ by $\boldsymbol{u}$, i.e., $\boldsymbol{u} = (\boldsymbol{e}_{min} + \boldsymbol{e}_{max})/2$. Let $\boldsymbol{p}^* = \arg\max_{\boldsymbol{q} \in \mathcal{D}} \boldsymbol{u}^* \cdot \boldsymbol{q}$ and $\boldsymbol{p} = \arg\max_{\boldsymbol{q} \in \mathcal{D}} \boldsymbol{u} \cdot \boldsymbol{q}$. We have the following derivation.

$$\begin{aligned}
\boldsymbol{u}^* \cdot \boldsymbol{p}^* - \boldsymbol{u}^* \cdot \boldsymbol{p} &= \boldsymbol{u}^* \cdot \boldsymbol{p}^* + \boldsymbol{u} \cdot \boldsymbol{p} - \boldsymbol{u} \cdot \boldsymbol{p} - \boldsymbol{u}^* \cdot \boldsymbol{p} \\
&\leq \boldsymbol{u}^* \cdot \boldsymbol{p}^* + \boldsymbol{u} \cdot \boldsymbol{p} - \boldsymbol{u} \cdot \boldsymbol{p}^* - \boldsymbol{u}^* \cdot \boldsymbol{p} \\
&= (\boldsymbol{u}^* - \boldsymbol{u}) \cdot \boldsymbol{p}^* - (\boldsymbol{u}^* - \boldsymbol{u}) \cdot \boldsymbol{p} \\
&= ||\boldsymbol{u}^* - \boldsymbol{u}|| \cdot ||\boldsymbol{p}^* - \boldsymbol{p}|| \cos\theta \\
&\leq \sqrt{d} ||\boldsymbol{u}^* - \boldsymbol{u}||
\end{aligned}$$

Secondly, there must exist a i^* such that $u[i^*] \geq 1/d$. Otherwise, $\sum_{i=1}^{d} u[i] < 1$, which contradicts the definition of $\boldsymbol{u}$. Let pi^* be the point in $\mathcal{D}$ with $pi^*[i^*] = 1$. Then, $\boldsymbol{u} \cdot \boldsymbol{p}^* \max_{\boldsymbol{q} \in \mathcal{D}} \boldsymbol{u} \cdot \boldsymbol{q} \geq \boldsymbol{u} \cdot pi^* \geq u[i^*] pi^*[i^*] \geq 1/d$.

To combine the two parts, we have

$$\frac{\boldsymbol{u}^* \cdot \boldsymbol{p}^* - \boldsymbol{u}^* \cdot \boldsymbol{p}}{\boldsymbol{u}^* \cdot \boldsymbol{p}^*} \leq d\sqrt{d} ||\boldsymbol{u}^* - \boldsymbol{u}||$$

Thus, if $||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq 2\sqrt{d}\epsilon$, then $||\boldsymbol{u}^* - \boldsymbol{u}|| = ||\boldsymbol{e}_{min} - \boldsymbol{e}_{max}|| \leq \sqrt{d}\epsilon$

$$\begin{aligned}
\frac{\boldsymbol{u}^* \cdot \boldsymbol{p}^* - \boldsymbol{u}^* \cdot \boldsymbol{p}}{\boldsymbol{u}^* \cdot \boldsymbol{p}^*} &\leq d\sqrt{d} ||\boldsymbol{u}^* - \boldsymbol{u}|| \\
&\leq \epsilon d^2
\end{aligned}$$

□

Proof of Lemma 10. Consider an action $\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle$ selected by the agent. If a user prefers $\boldsymbol{p}_i$ to $\boldsymbol{p}_j$, $\mathcal{R}$ can be strictly narrowed by pruning the part $\mathcal{R} \cap h^+_{j,i}$. This pair will not be used as an action subsequently since it cannot satisfy the first condition in our action space construction (i.e., $\mathcal{R} \cap h^+_{i,j} \neq \emptyset$ and $\mathcal{R} \cap h^+_{j,i} \neq \emptyset$). Similarly for the case where the user prefers $\boldsymbol{p}_j$ to $\boldsymbol{p}_i$. Since $|\mathcal{D}| = n$, there are $O(n^2)$ interactive rounds. □