Robust Best Point Selection under Unreliable User Feedback

ABSTRACT

The task of finding a user's utility function (representing the user's preference) by asking them to compare pairs of points through a series of questions, each requiring him/her to compare 2 points for choosing a more preferred one, to find the best point in the database is a common problem in the database community. However, in real-world scenarios, users may provide unreliable answers due to two major types of errors, namely *persistent* errors and *random* errors. Existing interaction algorithms either simply assume that all answers provided by the user are reliable, or are capable of handling *random* errors only, which can lead to finding *undesirable* points, ignoring persistent errors. To address this challenge, we propose novel algorithms that are robust to both persistent and random errors made by the user. Specifically, we propose (1) an algorithm that asks an asymptotically optimal number of questions, and (2) two algorithms that ask a small number of questions empirically, all of which have provable performance guarantees. Our experiments on both real and synthetic datasets demonstrate that our algorithms outperform existing methods in terms of accuracy, even with a small number of questions asked.

ACM Reference Format:

. 2024. Robust Best Point Selection under Unreliable User Feedback. In *Proceedings of SIGMOD 2024 (Conference acronym '24)*. ACM, New York, NY, USA, 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

When faced with a database containing millions of points, an end user may be only interested in finding his/her favorite point in the database. For example, a user may want to buy a car with a cheap price and a high horsepower from a car database, and s/he might only investigate the cars that excel in these aspects. Here, price and horsepower are some *attributes* that a user will consider when buying a car. To help the user efficiently find the most interesting point, *multi-criteria decision-making queries* are proposed to return a representative set from the database which consists of potentially interesting points. Two popular queries are the top-k query [23, 33] and the skyline query [5]. The top-k query returns k points from the dataset with the highest *utility* w.r.t a utility function. However, in practice, this utility function may not be known in advance. The second query, namely the skyline query, is based on a concept called *dominance*. Specifically, a point p *dominates* another point q if p is not worst than q in any attribute and is better than q in at least one attribute. The skyline query returns all points that are not dominated by any other points in the database. Although the knowledge of the user's utility function is not required, the skyline query does not guarantee a controllable return size, which might be as large as the entire database in the worst case [29, 32].

Recently, a novel interactive framework [31, 42, 44] was proposed to combine the advantage of both the top-k query (which has a fixed return size) and the skyline query (which does not need an exact utility function). Without any required knowledge of the user in advance, it asks the user a number of *rounds* of simple questions and learns the user's preference progressively, and recommends points based on the learned preference. A widely applied form of questions [31, 41, 42, 44] is to present 2 points in each round, and asks the user to select the preferred one. Consider the car purchasing scenario. The interactive framework simulates a sales assistant that asks Alice to indicate her preference among several pairs of cars and make recommendations based on her answers.

Although with good experimental performance, one key factor that prevents the existing interactive algorithms from being more practical is that they take no consideration of the *reliability* of the user feedback and implicitly assume that the user is *always* correct. However, in practice, the user's feedback can be *unreliable* due to various reasons. For example, during the interaction, even though Alice wants a car with good horsepower and costs as low as possible, she may indicate that she prefers a more expensive car to a cheap car due to a *careless mistake* (e.g., mis-clicking) or some *cognitive bias* (e.g., she mistakenly thinks that a car with a higher price must have better horsepower). Although a sales assistant can observe Alice's real intent by considering her overall choices, making this error when interacting with the existing algorithms will make them believe that Alice is willing to spend more. Consequently, these algorithms prune a set of cars from further consideration, possibly including the real best car for Alice.

It is worth mentioning that making small errors in the decision-making process can cause unforgettable and unchangeable consequences. For example, a common type of error that occurs in the trading market is known as a "fat finger error", which refers to a mistake made by a trader in the trading system by clicking or pressing the wrong key. In 2018, Samsung Security made a wrong transaction worth 100 billion dollars due to a fat finger error, which could have resulted in a loss of 428 million dollars, equivalent to 12.17% of the company's market capitalization [1]. Another example is about one of the critical milestones of one's life: the selection of the tertiary school. In 2020, a student in Mainland China achieved a top-tier score in the National College Entrance Examination in China (also called gaokao). However, he was admitted to a low-ranked college with a similar name to his target university since he confused the name of the two schools in the tertiary school selection system, which may cause a huge impact on his future life [43].

Motivated by the deficiency of existing algorithms, in this paper, we study the problem called the *interactive best point retrieval* problem under unreliable user input, which is more realistic. Roughly speaking, our problem is to find in a dataset D the best point (i.e., the point with the highest *utility*) for a user w.r.t. his/her personalized utility function f, which we do not know in advance and needs to be learned by asking the user to answer several rounds of

Conference acronym '24, June 18–23, 2024, Seattle, WA, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

questions. We focus on a type of question that is widely adopted [31, 34, 37, 41, 42, 44], which is to display two points from D, namely p_i and p_j, and ask the user to select the preferred point. In each round, the user has a probability at most θ to select the less preferred points. We also studied other types of questions, such as displaying more than two points in each question and asking for the favorite one. The results could be found in Section B.4 of the Appendix.

There are two major types of user errors that cause the unreliability in the user's answers, namely *random errors* and *persistent errors*. Random errors [7, 18] occur due to unintentional reasons such as mis-clicking and pressing the wrong key, which means that the answer to the same question may be different when asked again due to this careless mistake. On the other hand, persistent errors [15, 18, 22] are caused by cognitive biases or other sources of interference, and the answer to the same question may be *consistently* wrong. Compared to random errors, persistent errors are harder to be handled, since in the case of a random error, the user's real preference can be revealed by repeating the same question several times. The techniques developed in this paper can handle both types of errors, or even a combination of them.

Unfortunately, most existing interactive algorithms which do not consider user errors will return *undesirable* points based on *wrongly learned* utility functions. Their accuracy in returning the best point under the setting of unreliable user feedback is unsatisfactory (e.g., smaller than 70% on a 4-d dataset with 1 million points). Moreover, direct adaptations of existing algorithms considering user errors from the field of "learning to rank" and machine learning turn out to be inefficient since they ask too many questions. In our experiment, they ask more than 60 questions when the input dataset is large (i.e., 1 million points), which is undesirable. In the literature of marketing research [26, 35], users will lose the plot and get frustrated if they are asked to answer excessive questions. Therefore, minimizing the number of questions is also crucial. The most closely related work is [7] which aims at finding the best point considering random user errors. However, the techniques developed in [7] can only handle random errors by asking the same pair of points multiple times and taking the majority vote, which cannot be adapted to address persistent errors because the majority vote also results in incorrect preferences in this scenario.

Contributions. We summarize our contributions as follows. Firstly, we study the *interactive best point retrieval* problem considering not only random errors but also persistent errors. Secondly, we show a lower bound on the number of questions needed (also called *round complexity*) for our problem. Thirdly, we propose (1) an algorithm with asymptotically optimal round complexity; and (2) two algorithms with even better empirical performance. All algorithms return the best point with provable guarantees. Lastly, we conducted comprehensive experiments to demonstrate the superiority of our algorithms. They maintain high accuracy (e.g., nearly to 100% in most experiments) with only a small number of questions, but existing approaches either ask too many questions (e.g., twice as many as ours) or are much more inaccurate (e.g., more than 10% less accuracy than ours).

Organizations. The rest of the paper is organized as follows. Section 2 shows the related work. The formal definition of the studied problem is given in Section 3. In Section 4, we show the lower bound on the number of rounds required by this problem and give an algorithm with asymptotically optimal round complexity. In Section 5, we present two algorithms with even better empirical performance. We present the experimental results in Section 6. Section 7 concludes the paper.

2 RELATED WORK

Various queries are proposed to assist the multi-criteria decision-making. The *preference-based queries* return points based on the preference or the expected point of the user, and *interactive queries* involve user interaction to find points that may interest the user.

Besides the top-k and skyline queries mentioned in Section 1, many other types of preference-based queries are proposed. The k-nearest neighbors query (kNN) [39] returns k points that are closest to an example point given by the user based on their Euclidean distances. The similarity query [38] defines a more complicated distance function and finds k closest points to a given point using this function. The problem with these two queries is that an example point must be provided by the user in advance, which may be too demanding in many situations. Some recent studies [8, 29, 30] aim at combining the idea of top-k and skyline by returning points that are not dominated in a region of the entire function space. However, these approaches only partially solve the problem of the top-k and skyline queries since it still requires an estimation of the user's preference function. If the estimation is poor and the most interesting point must be returned, the output size will increase. In the worst case, it degenerates to the original skyline query.

The interactive queries learn the user's preference by interacting with the user and return points based on the learned preference. [28] proposes the interactive skyline query, which reduces the skyline size by learning the relative skyline importance of the user. During the interaction, the user is asked to partition points into *superior* and *inferior* groups. However, the output size is still uncontrolled even if the preference for all attributes is obtained. The interactive similarity query [3, 4, 37] interacts with the user to learn a distance function, which is then used to find k points that are closest to a query point. However, during the interaction, the user is asked to assign *relevant scores* to hundreds of points, this quantitative question is too demanding for most users.

Although the type of interaction varies, one widely adopted interaction is to display a pair (or set) of points in each round and ask the user to select the preferred one [3, 18, 31, 34, 37, 42, 44]. [31] proposes an interactive regret minimization query, which aims at minimizing the *regret ratio* of the return set with a fixed size by learning the user's utility function using artificial points. To overcome the deficiency of using unreal points, [44] introduces *UH-Simplex* and *UH-Random*, which only display points inside the database. [42] proposes *HD-PI* and *RH* that return one of the top-k points of the user. However, these algorithms all assume that the user makes no error, and make decisions based on user's answers without questioning their reliability. It is hard to directly adapt these algorithms to handle user errors, and their performance degenerates when the user makes mistakes.

The most closely related work is [7] in which they propose algorithms *Verify-Point* and *Verify-Space* that return the best point

considering *random user errors*. However, their methods have several drawbacks. Firstly, they address random user errors by taking majority votes on repetitive questions. This technique, however, cannot handle *persistent* errors since the feedback to the same question will be the same. In our experiments, their accuracy decreases by more than 10% when addressing persistent errors compared with addressing random errors. Also, repeating the same question is not suitable for many applications. For example, Alice will be confused if the seller asks her to compare the same pair of cars three times. Secondly, they still display artificial points during the interaction. The algorithms proposed by us overcome all these limitations.

In the field of machine learning (ML) and information retrieval (IR), the robustness to erroneous user input is considered in some algorithms [10, 11, 18, 19, 34, 36]. However, since their focuses (e.g., finding the exact ranking) are typically different from ours, they tend to be extravagant in the number of pairwise comparisons used, and thus, are inefficient for the user to interact with. [34] proposes *Preference-Learning* to learn user's preference and addresses user errors by introducing a slack variant in a linear SVM. It tends to ask extra questions since the major focus of this work is to approximate a preference vector. [18] aims at computing the ranking of all points and uses the majority vote to resolve conflicts in comparison results. This line of work needs more questions to find the ordering of non-best points, which is not a concern in our problem. [10] finds top-k points with initial partial ordering information and handle errors using majority votes. They require more questions than ours since they do not consider the geometric relation between points.

Compared with the existing work, our work has the following advantages. Firstly, we do not require any knowledge of the user's utility function in advance and we guarantee a small return size, which overcomes the limitation of traditional top-k and skyline queries. Secondly, we reduce the user's effort by only requiring the user to answer a small number of simple questions, while some existing algorithms ask a large number of questions (e.g., [11, 18, 34]) and others ask difficult questions (e.g., [4, 28]). Finally, our algorithms allow the user to be possibly unreliable without any significant impact on the quality of the points returned, which entails more practical value compared to some existing work that assumes perfect user feedback (e.g., [31, 42, 44]).

3 PROBLEM DEFINITION

In this section, we provide the formal definitions for the *interactive best point retrieval* problem, the *random errors*, and the *persistent errors*. The commonly-used symbols are summarized in Table 2 in the Appendix. The input to our problem is a d-dimensional dataset D. It may contain more than d attributes, but we assume that the user is interested in exactly d of them. The i-th dimensional value of a point p is denoted by $p[i]$ for $i \in [1, d]$, and the range of value for each attribute is normalized to $[0, 1]$. For each attribute, we assume that a higher value is preferred, but our techniques can be easily extended to the case where a small value is preferred.

Following [29, 31, 34, 42, 44, 45], we focus on the family of linear utility functions. That is, the family of functions that express the utility of a point p as $f(p) = u \cdot p$, where u is a d-dimensional non-negative vector called the *utility vector*. The utility vector u encodes the user's preference, where a higher value at the i-th dimension (i.e., $u[i]$) implies that the user is more concerned about the i-th attribute. We are interested in returning a set of points with a fixed size l containing the best point, i.e., the point with the highest utility w.r.t u, which we denote by $p_{max} = \arg\max_{p \in D} u \cdot p$. Note that scaling u has no influence on the rank of points as well as the best point. Therefore, we assume that $\sum_{i=1}^{d} u[i] = 1$.

The utility vector u of a user is initially unknown and will be learned by interacting with the user. The interaction continues for several rounds and the user will answer one question in each round. In the rest of the paper, we use the term "round" and "question" interchangeably. We adopt a popular type of questions [18, 24, 25, 34, 42] which is to display two points, namely p_i and p_j, from D, and the user is asked to choose the preferred point. Let $>$ and $<$ denote the *ground truth* relation between two points' utilities, and $\succ$ and $\prec$ denote the preference *indicated* by the user. That is, $p_i > p_j$ if $u \cdot p_i > u \cdot p_j$, and $p_i \succ p_j$ if the user indicates that s/he prefers p_i to p_j. The existing algorithms assume that the user *always* selects the point with a higher utility (i.e., if $p_i \succ p_j$, then $p_i > p_j$ with 100% certainty). In practice, however, the user occasionally chooses the point with a lower utility (e.g., due to the reasons described in Section 1). We say that the user makes an *error* if $p_i > p_j$ but the indicated preference is $p_j \succ p_i$. An error is called a *persistent error* if the answer to the same question is consistent; otherwise, it is a *random error*. From now on, we assume that all the errors made are persistent since if we can handle persistent errors, the case for random errors could be handled automatically. We assume that the user makes an error in each distinct question with an error rate at most θ. θ can be naturally assumed to be less than 0.5 for reasonable users according to [7, 21]. In their studies, θ is at most 5%. Following [12–14, 16, 17, 20, 36], we assume that the user errors are independent across different pairs of points.

In this paper, we are interested in the following problem:

Problem 1. *Given a dataset D with size n, an error rate upper bound θ and a return size l, how to interact with the user to find a set with size at most l such that the probability that this set contains the best point is maximized?*

4 THE *FINDCYCLES* ALGORITHM

In this section, we introduce our first algorithm, *FC* (*FindCycles*), which returns a set containing the best point with asymptotically optimal round complexity. In Section 4.1, we first introduce some basic concepts that will be used for *FC* and for later sections. Then, we present the general framework of all our proposed algorithms and prove the lower bound of the number of rounds required for our problem in Section 4.2. In Section 4.3, we introduce *FC* and show that it is asymptotically optimal, given that $l = O(\sqrt{n})$.

4.1 Preliminaries

In geometry, the *convex hull* of a dataset D, denoted by $conv(D)$, is the smallest convex set containing all points in D [27]. A point $p \in D$ is a *vertex* of $conv(D)$ if $p \notin conv(D/\{p\})$. Let b_i be the d-dimensional point with its i-th coordinate being 1 and all other coordinates being 0. Furthermore, let $B = \{b_i | 1 \le i \le d\}$ and O be the origin. Consider the set of points that are both in D and in $conv(D \cup B \cup \{O\})$. We call these points the *upper hull vertices* and denote this set by $uhull(D)$. For example, in Figure 1, we visualize

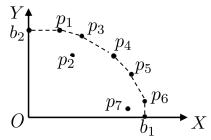


Figure 1: The upper hull

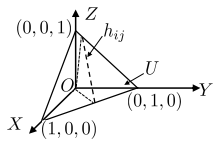


Figure 2: Utility space in 3d

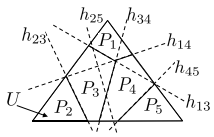


Figure 3: Example on partitions

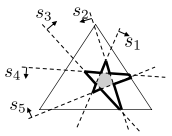


Figure 4: Example on confidence regions

a 2-d dataset $D = \{p_i | i \in [1, 7]\}$. Its upper hull vertices are p_1, p_3, p_4, p_5 and p_6. One important conclusion is that the best point must be in $uhull(D)$ [29, 44]. Therefore, the set of upper hull vertices constitutes possible candidates of the best point. Let n denote the size of $uhull(D)$. For the ease of illustration, in the rest of this paper, when we say D, we mean $uhull(D)$ unless otherwise specified.

Recall that all possible utility vectors form a set $U = \{u | u[i] \geq 0$ and $\sum_{i=1}^{d} u[i] = 1\}$. U is called the *utility space* and is a $(d-1)$-dimensional convex polytope. Consider a 3-dimensional example in Figure 2. The utility space U is a planar triangle with vertices $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. For any two points p_i and p_j in D where $i < j$, we can construct a hyperplane h_{ij} that has its normal $p_i - p_j$ and passes through the origin. h_{ij} intersects with U and divides the utility space into two *halfspaces* [27]. The halfspace above (resp. below) h_{ij} is denoted by h_{ij}^{+} (resp. h_{ij}^{-}), which contains all utility vectors u that satisfy $u \cdot p_i > u \cdot p_j$ (resp. $u \cdot p_i < u \cdot p_j$).

In each round, two points from D, denoted by p_i and p_j, are displayed to the user. When the user indicates the preference between them, we could learn that the utility vector of the user is in h_{ij}^{+} or h_{ij}^{-}. Let s denote a halfspace indicated by the user and S the set of all halfspaces indicated by the user so far. Besides, let s_{ij} denote the halfspace selected by the user when p_i and p_j are displayed (i.e., s_{ij} could be either h_{ij}^{+} or h_{ij}^{-} based on the answer provided by the user). We say that a region (or a point) is supported by a halfspace s if this region (or this point) lies completely in s. Given a set S' of halfspaces, we say that a region (or a point) is supported by set S' if this region (or this point) is supported by each halfspace in S'. When the user makes no error, the utility vector u must lie in the intersection of all halfspaces in S (i.e., $u \in \cap_{s \in S} s$). Many existing algorithms [41, 42, 44, 45] utilize this property to approximate the utility vector, which explains why they are prone to user errors: When the wrong halfspace is chosen, the real u will not locate in the intersection of all halfspaces. Consequently, the resulting estimation of u is less accurate and the best point may be missed.

For each point $p_i \in D$, its corresponding *best partition* P_i, or *partition* for short, is the set of utility vectors in U that give p_i the highest utility score. This means that for any $u \in P_i$ and for any $p_j \in D/\{p_i\}$, $u \cdot p_i > u \cdot p_j$. Therefore, P_i is the intersection of all h_{ij}^{+} for $p_j \in D/\{p_i\}$ and the utility space U, i.e., $P_i = (\bigcap_{p_j \in D/\{p_i\}} h_{ij}^{+}) \cap U$, which is also a $(d-1)$-dimensional convex polytope. As an example, consider Figure 3 where we show the partitions corresponding to a 3-d dataset containing 5 points in its upper hull. The utility space U is the outer triangle. The partitions P_1 to P_5, which are 2-d polygons, are bounded by *solid* lines and the intersection of (some) h_{ij} and U are shown as *dashed* lines.

Next, we introduce another important concept, called *confidence region*, in Definition 1.

Definition 1. *The i-**th confidence region**, denoted by R^i, is the maximal area in the utility space U that is supported by at least one set of $|S| - i$ halfspaces in S. Mathematically, $R^i = \bigcup_{S' \in S_{|S|-i}} (\cap_{s \in S'} s)$ where $S_{|S|-i}$ is the set of all $(|S| - i)$-subsets of S.*

If the user makes at most i errors during the interaction, the utility vector is not supported by at most i halfspaces in S. In this case, we immediately have $u \in R^i$ by its definition. Consider the example shown in Figure 4. The dashed lines represent the hyperplanes asking the user, and the arrow on each hyperplane shows the halfspace indicated by the user. Assume that the user has indicated 5 halfspaces so far, i.e., $S = \{s_1, s_2, s_3, s_4, s_5\}$. R^0 is the area supported by all 5 halfspaces, which is the gray pentagon. R^1 is the area supported by at least one set of all halfspaces except 1 halfspace (i.e., 4 out of 5 halfspaces), which corresponds to the star-shaped area bounded by bold lines. If the user makes at most 1 error when indicating s_1, s_2, s_3, s_4 and s_5, his/her utility vector must lie in R^1. Although R^0 must be convex, when $i \geq 1$, R^i could be non-convex and is in fact made of the union of several disjoint convex polytopes. In Figure 4, R^1 is the union of R^0 and 5 small convex polytopes (triangles in this example) adjacent to R^0.

We have the following observation.

Observation 1. *For any $i' \geq i$, $R^i \subseteq R^{i'}$.*

We say that a partition P_j *overlaps* with a confidence region R^i if $P_j \cap R^i \neq \emptyset$. Based on Observation 1, if P_j overlaps with R^i, P_j also overlaps with all $R^{i'}$s where $i' \geq i$. On the other hand, if P_j does not overlap with R^i, it also does not overlap with all $R^{i'}$s where $i' \leq i$. Since when the user makes at most i errors, the real u falls in R^i, it follows that only points whose partitions overlap with R^i can be the best point, while those whose partitions do not overlap with R^i cannot be. For example, if the user makes no errors, then only points whose partitions overlap with R^0 can be the best point. Therefore, when determining which points should be in the returned set, we adopt a strategy that first includes points whose partitions overlap with R^0, followed by points whose partitions overlap with R^1, R^2 and so on.

To make sure that the return size is at most l, we only maintain confidence regions that guarantee to overlap with at most l partitions when questions related to all pairs of points (i.e., all possible questions) are asked. The relation between R^i and the number of partitions it overlaps is shown in Lemma 1.

Lemma 1. *If questions related to all pairs of points are asked, the confidence region R^i overlaps with at most $2i + 1$ partitions.*

Proof sketch. Denote the set of points whose partitions overlap with R^i by RS. Since each pair of points in RS has been compared, there are in total $|RS|(|RS| - 1)/2$ comparisons. Each comparison yields one loser. Any points in RS cannot lose more than i times, otherwise, it will not be in R^i. Thus, we have $|RS|(|RS| - 1)/2 \leq i|RS|$, so $|RS| \leq 2i + 1$. The complete proofs of theorems and lemmas in this paper can be found in Section C of the Appendix. □

Corollary 1. *Let $k \le \left\lfloor \frac{l-1}{2} \right\rfloor$. If questions related to all pairs of points are asked, R^k overlaps with at most l partitions.*

Corollary 1 suggests that, whenever $k \le \left\lfloor \frac{l-1}{2} \right\rfloor$, if we maintain R^k in the algorithm, we guarantee that R^k overlaps with at most l partitions and thus, at most l points could be returned.

4.2 Algorithm Framework

In this section, we introduce a general framework of all our proposed algorithms. We maintain $k+1$ confidence regions, namely $R^0, R^1, ..., R^k$, where k is chosen as an integer at most $\left\lfloor \frac{l-1}{2} \right\rfloor$. Note that R^i in Lemma 1 (and R^k in Corollary 1) is the final R^i (R^k) after questions related to all pairs of points are asked. In our algorithms, since we have not asked any questions at the beginning, R^is are initialized to the entire utility space and will be updated when more questions are asked. We use a subscript t, i.e., R^i_t, to denote R^i just after t halfspaces are indicated. Lemma 2 shows the rule of updating R^i_t where $i \in [0,k]$, when the $(t+1)$-th halfspace is indicated.

Lemma 2. *For each $i \in [0,k]$ and each $t \ge 0$, let s be the halfspace indicated in round $t+1$. The relation between R^i_t and R^i_{t+1} is as follows:*

$$R^i_{t+1} = R^i_t \cap s \qquad \text{if } i = 0$$
$$R^i_{t+1} = (R^i_t \cap s) \cup (R^{i-1}_t \cap s^-) \qquad \text{if } i \in [1,k]$$

where s^- is the complement of s.

Proof sketch. We prove this lemma using induction. The special case where $i = 0$ is trivially correct since R^0_t is the regions supported by all t halfspaces. We then show that if at round t', all $R^i_{t'}$s are correct, then at round $t'+1$, the resulting $R^i_{t'+1}$ is indeed the maximal region supported by at least $t'+1-i$ halfspaces. □

As a running example, consider Figure 5 where $k = 1$ and $t = 2$. Assume that the user indicated h^+_{34} and h^+_{14} in round 1 and round 2, respectively (Recall that h^+_{ij} is the halfspace containing P_i). Then, R^0_2 is the gray region and R^1_2 is the area bounded by bold lines. If the user indicates h^+_{13} the next round, we know from Lemma 2 that $R^0_3 = R^0_2 \cap h^+_{13}$, and $R^1_3 = (R^1_2 \cap h^+_{13}) \cup (R^0_2 \cap h^-_{13})$. The resulting R^0_3 and R^1_3 are shown in Figure 7, where R^0_3 is the gray region and R^1_3 is the area bounded by bold lines.

Lemma 3. *For each $i \in [0,k]$ and each $t \ge 0$, we have (1) $R^i_{t+1} \subseteq R^i_t$ and (2) $R^i_t \subseteq R^{i+1}_{t+1}$.*

Proof sketch. (1) can be proved using Lemma 2 and the fact that $R^{i-1}_t \subseteq R^i_t$. To prove (2), observe that for any point $v \in R^i_t$, v is supported by a set of $(t-i)$ halfspaces. The same set of $(t-i)$ halfspaces makes v also in R^{i+1}_{t+1}, which implies that $R^i_t \subseteq R^{i+1}_{t+1}$. □

In round t, a partition P_j is said to *belong to* a confidence region R^i_t if either (1) R^i_t ($i \ge 1$) overlaps with P_j but R^{i-1}_t does not, or (2) R^i_t ($i = 0$) overlaps with P_j. That is, a partition belongs to the smallest confidence region that overlaps with it. Note that knowing the partitions overlapping with each R^i could help us easily derive the partitions belonging to each R^i, and vice versa. Similarly, we say that a point p_j belongs to R^i_t if its corresponding partition P_j belongs to R^i_t. From Lemma 3, confidence regions can only reduce by each round. Due to this shrinking behavior, a partition P_j that belongs to R^i_t in round t may no longer belong to R^i_{t+1} in round $t+1$. If this happens, we say that P_j is *detached from* R^i_{t+1} (or simply R^i if the context is clear). Note that after being detached from R^i_{t+1}, P_j will belong to R^{i+1}_{t+1} in round $t+1$ (by Lemma 3, P_j overlaps with R^{i+1}_{t+1} since $R^i_t \subseteq R^{i+1}_{t+1}$). When more and more new halfspaces are indicated, a partition is gradually detached from R^0, R^1, R^2 until R^k. Since we are not interested in partitions that do not overlap with R^k, if a partition P_i is detached from R^k, we say that P_i (and point p_i) is *discarded*. We use X^i to denote the set of partitions belonging to R^i, and use X^i_t to denote X^i just after round t.

Consider our running example in Figure 5. Assume that $k = 1$ and $t = 2$, and recall that R^0_2 is the gray region and R^1_2 is the area bounded by bold lines. Since both P_1 and P_3 overlaps with R^0_2, we knot that both P_1 and P_3 (and thus p_1 and p_3) belong to R^0_2. P_2 overlaps with R^1_2 but not R^0_2, so it belongs to R^1_2. In the next round ($t = 3$), consider the updated R^0_3 and R^1_3 in Figure 7, where R^0_3 is the gray region and R^1_3 is the area bounded by bold lines. Since P_3 no longer overlaps with R^0_3, it is detached from R^0_3 and belongs to R^1_3 instead. P_2 is detached from R^1_3 and is thus discarded.

Stopping Condition. The algorithm stops when the number of partitions overlapping with R^k is at most l. Then, it returns the points that correspond to these partitions.

We are now ready to present the lower bound on the number of rounds required.

Theorem 1. *Given an input size n, an output size l and an error rate upper bound θ. Let $k = \left\lfloor \frac{l-1}{2} \right\rfloor$. There exists a dataset such that any question-asking strategy must ask $\Omega(\frac{l}{\theta} + n)$ questions before the stopping condition is satisfied.*

Proof sketch. We first prove that there exists a dataset D such that for any $p_i \in D$ and its partition P_i, the following property holds: for any non-empty set of halfspaces $HS \subseteq \{s_{ab} | a, b \ne i\}$, if the intersection of halfspaces in HS forms a non-empty area, then P_i also overlaps with this area. Then, for any $p_i \in D$, if in some round P_i is detached from some confidence region R^m where $m \in [0,k]$, the question in this round must satisfy one of the following: (1) This question is related to p_i and some other point p_j, and p_i is less preferred to p_j, and (2) This question makes R^m an empty region. With $k = \left\lfloor \frac{l-1}{2} \right\rfloor$, we then show that the round complexity for these two cases are $\Omega(ln)$ and $\Omega(\frac{l}{\theta} + n)$, respectively. Since $\Omega(ln) > \Omega(\frac{l}{\theta} + n)$, the lower bound is $\Omega(\frac{l}{\theta} + n)$. □

4.3 The *FC* Algorithm

In this section, we detail the algorithm *FC*, which uses an asymptotically optimal number of rounds. We first introduce an important concept called *cycle*.

Definition 2. *A set of $m \ge 3$ points $\{p_1, p_2, \ldots, p_m\}$ form a **cycle** if the user indicates that $p_1 \succ p_2, \ldots, p_{m-1} \succ p_m$, and $p_m \succ p_1$. Two cycles are called **disjoint** if there do not exist two points, namely p_i and p_j, such that $p_i \succ p_j$ appears in both cycles.*

Due to possible user errors, we do not assume transitivity in the indicated preferences. That is, $p_i \succ p_j$ and $p_j \succ p_k$ (indicated by a user) do not imply $p_i \succ p_k$. Lemma 4 reveals the relation between the occurrences of cycles and confidence regions.

Lemma 4. *If there are $i+1$ disjoint cycles, then the i-th confidence region $R^i = \emptyset$.*

Proof sketch. We first show that the intersection of halfspaces corresponding to a cycle is an empty region. Since $R^i = \bigcup_{S' \in S_{|S|-i}} (\cap_{s \in S'} s)$, where S' has a cardinality of $|S|-i$, if there are $i+1$ disjoint cycles, no matter how we set S', $\cap_{s \in S'} s$ will be empty since there is at least one cycle in S'. Thus, R^i is also empty. □

Intuitively, *FC* has two stages, namely Stage 1 and Stage 2, and each stage consists of several rounds. After each round, it updates (1) the confidence regions R^is and (2) the sets of partitions belonging to each confidence region, i.e., X^is, and checks if the stopping condition described in Section 4.2 is met. We omit how R^is and X^is are maintained here and will explain them in detail in Section 5. In Stage 1, *FC* adopts a question selection strategy (to be described next) that attempts to quickly obtain disjoint cycles, thereby reducing some confidence regions to empty and detaching a large number of partitions from them. When Stage 1 ends, it is shown later (in Lemma 5) that at most k disjoint cycles are obtained. Let j be the number of disjoint cycles obtained just after Stage 1 (where $j \le k$). By Lemma 4, this means that $R^0, R^1, \ldots, R^{j-1}$ all become empty. Then, *FC* enters Stage 2. At this stage, a partition either belongs to one of the non-empty confidence regions $R^j, R^{j+1}, \ldots, R^k$, or is already discarded. The algorithm then continues to discard more partitions by displaying two points whose partitions are still not discarded, until the number of remaining partitions is at most l (i.e., the stopping condition described in Section 4.2 is satisfied).

Next, we describe the details of these 2 stages. In Stage 1, *FC* maintains a set *PS* of points, initialized to an empty set. Initially, the algorithm randomly selects two points from D, asks the user's preference on them, and inserts them into *PS*. Then, it randomly picks a point from D that has not been picked before, which is called the *focusing point*, denoted by p_f, and asks the user's preferences between p_f and each point in *PS*. After all points in *PS* are compared with p_f, p_f is inserted into *PS*. *FC* then selects a new point from D as the new focusing point and repeats the above process. Stage 1 stops when one of the following two conditions is met: (1) it already lasts for $\max(\frac{k(k-1)}{2}, \frac{3k}{\theta})$ rounds just after a focusing point is inserted into *PS*; or (2) $R^{k-1} = \emptyset$. Lemma 5 states several important properties when Stage 1 ends.

Lemma 5. *When Stage 1 ends, the following properties hold:*
1. The number of obtained disjoint cycles is at most k.
2. If Stage 1 ends on Condition (1), then the expected number of obtained disjoint cycles is at least $\max(0, k-13)$.

Proof sketch. Assume that Stage 1 ends after round t. Since $R^{k-1}_{t-1} \subseteq R^k_t \neq \emptyset$ (Lemma 3), thus the number of disjoint cycles must be smaller than k (Lemma 4), which proves property 1. To prove Property 2, we first show that if Stage 1 ends on Condition (1), the user makes at least k errors with probability at least $1 - \frac{1}{k}$. We then show that if at least k errors are made, the expected number of disjoint cycles is at least $k-12$. Combining them together yields the result. □

FC then enters Stage 2, in which it needs to further discard the remaining partitions. To do so, in each round, *FC* adopts a naive approach that randomly selects two points whose partitions are not discarded, and displays them to the user. Note that this naive approach is sufficient to obtain the theoretical result about asymptotically optimality (to be shown in Corollary 2). In practice, however, Stage 2 of *FC* adopts some selection strategies that will be introduced in Section 5.1.2 for better empirical performance. The algorithm terminates when at most l partitions remain (i.e., the stopping condition described in Section 4.2 is satisfied). Theorem 2 presents the major conclusion of *FC*.

Theorem 2. *Given an input size n, an output size $l = O(\sqrt{n})$ and an error rate upper bound θ, when $k = \left\lfloor \frac{l-1}{2} \right\rfloor$, FC returns a set of points with size at most l using $O(\frac{l}{\theta} + n)$ rounds on expectation.*

Proof sketch. Firstly, note that when $l = O(\sqrt{n})$, Stage 1 finishes within $O(\frac{l}{\theta} + n)$ rounds. When entering Stage 2, there are 2 cases: (1) Condition 1 is satisfied, and (2) Condition 1 is not satisfied, but Condition 2 is satisfied. We then show that the expected round complexity combining these 2 cases is $O(n)$. Therefore, the total expected round complexity is $O(\frac{l}{\theta} + n)$. □

Corollary 2. *The round complexity of FC is asymptotically optimal.*

5 THE *SHAPE-EXACT* AND *SHAPE-SAMPLING* ALGORITHMS

In this section, we introduce two algorithms, namely *SE* (*Shape-Exact*) (Section 5.1) and *SS* (*Shape-Sampling*) (Section 5.2), adopting different question selection strategies from *FC* with better empirical performance. Both algorithms follow the framework described in Section 4.2, and their major distinction lies in how they maintain (1) confidence regions (i.e., R^is) and (2) the set of partitions belonging to each confidence region (i.e., X^is). Intuitively, *SE* maintains data structures that record the *exact* shapes of R^is and decide X^is. *SS* keeps R^is *conceptually* and determines X^is using some pre-computed results on a set of randomly sampled points. Consequently, *SS* is more time-efficient with a slight sacrifice on the accuracy of retrieving the best point.

5.1 The *SE* Algorithm

In this section, we introduce *SE* by solving two sub-problems, namely (1) how to maintain the data structures for deciding (a) confidence regions (i.e., R^is) and (b) the set of partitions belonging to each confidence region (i.e., X^is), and (2) how to design question selection strategies to ask a small number of questions.

5.1.1 Data Structure Maintenance. *SE* records in total $2k+2$ data structures to maintain the exact shapes of confidence regions and to decide the partitions overlapping with each confidence region. The first $k+1$ data structures, denoted by $C^0, C^1, ..., C^k$, each records the shape of the corresponding confidence region R^i. The remaining $k+1$ data structures, denoted by $O^0, O^1, ..., O^k$, each stores the

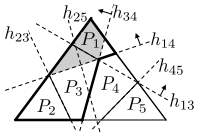


Figure 5: *SE* example before asking h_{13}

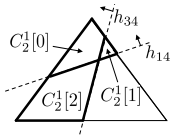


Figure 6: Polytopes in C_2^1

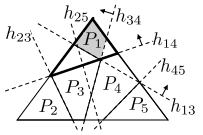


Figure 7: *SE* example after asking h_{13}

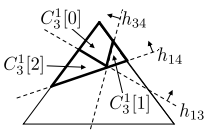


Figure 8: Polytopes in C_3^1

information to decide the partitions overlapping with the corresponding confidence regions $R^0, R^1, ..., R^k$, respectively. Similar to above, for each $i \in [0,k]$, we also use C_t^i and O_t^i to denote C^i and O^i just after round t, respectively.

Since the confidence regions R^is (except R^0) can be non-convex, we cannot simply represent them as the *intersection* of halfspaces. Therefore, we record each R_t^i as the *union* of several disjoint convex polytopes. We use a list, C_t^i, to store the set of disjoint convex polytopes whose union forms R_t^i. Let $C_t^i[j]$ denote the j-th convex polytope in C_t^i. Then, $R_t^i = \bigcup_{j=0}^{|C_t^i|-1} C_t^i[j]$. Initially, when $t=0$, all $R_0^0, R_0^1, ..., R_0^k$ are the entire utility space (a single convex polytope) and, for each $i \in [0,k]$, C_0^i contains one element (i.e., $C_0^i[0]$) which corresponds to this polytope. According to Lemma 2, at round t, R_t^i can be computed by R_{t-1}^i, R_{t-1}^{i-1} and s where s is the halfspace indicated at round t. Therefore, the convex polytopes to be stored in C_t^i can be computed using the polytopes in C_{t-1}^i and C_{t-1}^{i-1}, which are readily computed in round $t-1$. The updating rules are as follows: (1) for each polytope $c \in C_{t-1}^i$, $c \cap s$ is inserted into C_t^i if $c \cap s$ is non-empty; (2) for each polytope $c \in C_{t-1}^{i-1}$, $c \cap s^-$ is inserted into C_t^i if $c \cap s^-$ is non-empty.

Consider the running example in Figure 5. Let $k=1$ and $t=2$. Assume that the user indicated h_{34}^+ and h_{14}^+ in round 1 and round 2, respectively. Then, R_2^0 corresponds to the gray area and R_2^1 is the region bounded by bold lines. The convex polytopes forming R_2^1 is stored in the list $C_2^1 = \{C_2^1[0], C_2^1[1], C_2^1[2]\}$ and is shown in Figure 6. Assume that h_{13}^+ is indicated at round 3. R_3^0 (the gray area) and R_3^1 (the region bounded by bold lines) are shown in Figure 7. Using the above updating rules, the convex polytopes in $C_3^1 = \{C_3^1[0], C_3^1[1], C_3^1[2]\}$ that form R_3^1 is shown in Figure 8.

Next, we describe the methods to maintain O_t^is, which stores the information to decide the partitions overlapping with R_t^is. We first introduce a new concept called *sub-partition*. A sub-partition has the form $P_i \cap (\bigcap_{s \in S'} s)$, where P_i is a partition and S' is a set of halfspaces. As a special case, a partition P_i is also a sub-partition where S' is empty. For example, in Figure 5, h_{13} divides P_4 into two sub-partitions, namely $P_4 \cap h_{13}^+$ and $P_4 \cap h_{13}^-$. The point corresponding to the sub-partition is the same as the original partition (i.e., $P_i \cap (\bigcap_{s \in S} s)$ still corresponds to point p_i).

O_t^i is a list of sub-partition sets, where the list is of the same length as C_t^i. The j-th sub-partition set in O_t^i is denoted by $O_t^i[j]$, which stores the set of sub-partitions that lie in $C_t^i[j]$. For example, in Figure 9, the set of sub-partitions in $C_2^1[0]$ (the gray area) is $O_2^1[0] = \{P_1 \cap h_{34}^+, P_3 \cap h_{14}^+\}$. Initially, when $t=0$, since each C_0^i contains only 1 convex polytope (i.e., $C_0^i[0]$) which corresponds to the entire utility space that contains all partitions, each O_0^i has one element (i.e., $O_0^i[0]$) which is a set containing all partitions. At round t, the elements in the empty-initialized O_t^i can be computed using O_{t-1}^i and O_{t-1}^{i-1}, which are readily computed in round $t-1$. The update of O_t^i is closely related to the update of C_t^i. Specifically,let s be the halfspace indicated at round t. For each $j \in [0, |C_t^i| - 1]$: (1) if $C_t^i[j] \subseteq s$, find $j' \in [0, |C_{t-1}^i| - 1]$ such that $C_t^i[j] = C_{t-1}^i[j'] \cap s$. For each $o \in O_{t-1}^i[j']$, $o \cap s$ is inserted to $O_t^i[j]$ if $o \cap s$ is non-empty; (2) if $C_t^i[j] \subseteq s^-$, find $j' \in [0, |C_{t-1}^{i-1}| - 1]$ such that $C_t^i[j] = C_{t-1}^{i-1}[j'] \cap s^-$. For each $o \in O_{t-1}^{i-1}[j']$, $o \cap s^-$ is inserted to $O_t^i[j]$ if $o \cap s^-$ is non-empty. Note that j' must exist, since based on the updating rules of C_t^i, each $C_t^i[j]$ must be derived either from (1) $c \cap s$ where $c \in C_{t-1}^i$, or (2) from $c \cap s^-$ where $c \in C_{t-1}^{i-1}$.

Consider the running example in Figure 9 where $k=1$ and $t=2$, the set of sub-partitions in $C_2^1[0]$ (the gray region) is $O_2^1[0] = \{P_1 \cap h_{34}^+, P_3 \cap h_{14}^+\}$. Assume that h_{13}^+ is indicated in round 3. Then in Figure 10, since $C_3^1[0]$ (the gray region) equals to $C_2^1[0] \cap h_{13}^+$, its corresponding set of sub-partition $O_3^1[0] = \{P_1 \cap h_{34}^+ \cap h_{13}^+\} = \{P_1 \cap h_{34}^+\}$. Since $P_3 \cap h_{14}^+ \cap h_{13}^+$ is empty, it is not inserted to $O_3^1[0]$. The set of partitions that overlap with R_t^i can be easily derived using O_t^i. For example, in Figures 7 and 8, $R_3^1 = C_3^1[0] \cup C_3^1[1] \cup C_3^1[2]$. Since $O_3^1[0] = \{P_1 \cap h_{34}^+\}$, $O_3^1[1] = \{P_1 \cap h_{34}^-\}$ and $O_3^1[2] = \{P_3 \cap h_{14}^+\}$, the set of partitions overlapping with R_3^1 is $\{P_1, P_3\}$.

5.1.2 Question Selection Strategies. Next, we describe how to choose the question in each round so that the required number of questions can be reduced. Observe that to detach a partition P from a confidence region R^i, P must also be detached from all $R^{i'}$s where $i' \le i$. Thus, intuitively, a question is more preferred if (1) it can detach partitions from R^i with a smaller i and (2) it can detach a larger number of partitions. For example, questions that can detach a large number of partitions from R^0 are the most preferred. We develop two question selection strategies that comply with the above intuition, namely the *random-based selection* and the *score-based selection*. The random-based selection has a faster process time, and the score-based selection asks fewer questions empirically. In the following, we describe the two selection strategies in detail.

The Random-based Selection. In round t, let X_t^i be the set of partitions that belongs to R_t^i, and let $\mathcal{X}_t^i$ be the set of points whose partition is in X_t^i. The random-based selection runs for at most $k+1$ iterations. In iteration i ($i \in [1, k+1]$), it sets a *candidate point set*, denoted by CP, to be $\bigcup_{j=0}^{i-1} \mathcal{X}_t^j$, enumerates all (order-insensitive) pairs of points consisting of points in CP and randomly permutes them, and sequentially checks each pair that has not been checked in previous iterations. If it finds a pair that has not been asked before, it selects this pair and stops. If such a pair cannot be found after depleting all pairs in this iteration, it starts the next iteration. Note that this strategy favors questions that can detach partitions belonging to R_t^i with a small value of i. It also guarantees to find an unasked pair if one exists, since otherwise, the stopping condition described in Section 4.1 is already met.

As a running example, assume that $k=1$ and at round t we have $\mathcal{X}_t^0 = \{p_1, p_2, p_3\}$ and $\mathcal{X}_t^1 = \{p_4, p_5\}$. In the first iteration,

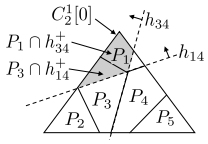


Figure 9: Sub-partitions in $C_2^1[0]$

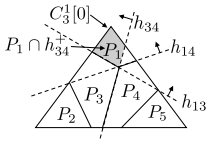


Figure 10: Sub-partitions in $C_3^1[0]$

	h_{ij}^+	h_{ij}^-	score ($\alpha = 0.2$)
h_{13}	$\{P_1\}$	$\{P_2, P_3, P_5\}$	1
h_{14}	$\{P_1\}$	$\{P_2, P_4, P_5\}$	0.2
h_{23}	$\{P_2\}$	$\{P_1, P_3, P_4, P_5\}$	0.2
h_{25}	$\{P_2\}$	$\{P_5\}$	0
h_{34}	$\{P_2, P_3\}$	$\{P_4, P_5\}$	0
h_{45}	$\{P_1, P_2, P_3, P_4\}$	$\{P_5\}$	0

Table 1: Table L

CP is set to $\{p_1, p_2, p_3\}$. We randomly permute all pairs of points consisting of points in CP (i.e., (p_1, p_2), (p_2, p_3) and (p_1, p_3)) and sequentially consider each of them. If we find a pair that is unasked before (e.g., (p_1, p_2)), we use this pair. If all 3 pairs are already asked, we enter the second iteration, where CP is set to $\mathcal{X}_t^0 \cup \mathcal{X}_t^1$ (i.e., $\{p_1, p_2, p_3, p_4, p_5\}$).

The Score-based Selection. Although the random-based selection has a fast processing time, it does not fully utilize the distribution of partitions to select the optimal hyperplane. Therefore, the score-based selection is designed to further reduce the number of questions required. We first introduce a data structure that will be used for the score-based selection. For each partition P_a and a hyperplane h_{ij}, there are 3 possible relationships between P_a and h_{ij}: (1) $P_a \in h_{ij}^+$, (2) $P_a \in h_{ij}^-$, and (3) P_a intersects with h_{ij}. Consider the example in Figure 5. For hyperplane h_{13}, P_1 is in h_{13}^+, P_2, P_3 and P_5 are in h_{13}^-, and P_4 intersects with h_{13}. We maintain a table L to store these relationships, where each row corresponds to a hyperplane h_{ij} and has 3 columns: (1) h_{ij}^+, which stores all partitions that lie in h_{ij}^+, (2)h_{ij}^-, which stores all partitions that lie in h_{ij}^-, and (3) *score* which will be explained next. The table L corresponding to Figure 5 is shown in Table 1.

Let $Num(s, X_t^i)$ denote the number of partitions in X_t^i that lies in a halfspace s. We define the *score* of a hyperplane h at round t to be $score_t(h) = \min(\sum_{i=0}^{k} \alpha^i Num(h^+, X_t^i), \sum_{i=0}^{k} \alpha^i Num(h^-, X_t^i))$, where α is a parameter smaller than 1 which captures the relative priority between different X_t^is. According to our empirical results in Section 6, we set $\alpha = 0.2$. Note that this definition gives higher scores to those hyperplanes intersecting R_t^i with a smaller i and with partitions in X_t^i more evenly distributed on each side, indicating a higher chance of detaching more partitions. In round t, the hyperplane with the highest score that has not been chosen before will be selected and its corresponding points will be displayed. Consider the example in Figure 5 where $t = 2$ and $k = 1$. After h_{34}^+ and h_{14}^+ are indicated by the user in the first 2 rounds, the resulting R_2^0 is shown in the gray region and R_2^1 is the region bounded by the bold lines. Then, $X_2^0 = \{P_1, P_3\}$ and $X_2^1 = \{P_2\}$. The scores of hyperplanes at this stage are listed in Table 1. p_1 and p_3 will be displayed to the user in the next round since h_{13} obtains the highest score.

When the input size is large, the score-based selection could be time-consuming due to a large number of question candidates. Therefore, we introduce a *lazy updating technique* to reduce the processing cost of the score-based selection. Details of the lazy updating technique can be found in Section A of the Appendix due to space constraints. In our experiments, depending on the input datasets, this technique accelerates the processing time by 5% - 60%.

Based on the question selection strategy applied, there are two variants of *SE*, which are called *SE-random* and *SE-score*, respectively. The time complexities for each round of these two variants are presented in Theorem 3.

Theorem 3. *Given an input size n, dimensionality d and target return size l. Let $\binom{a}{b} = \frac{a!}{a!(a-b)!}$ be the binomial coefficient. The time complexities of each round of SE-random and SE-score are $O(\binom{t+\lfloor (l-1)/2 \rfloor}{t} |V| ldn^2)$ and $O(\binom{t+\lfloor (l-1)/2 \rfloor}{t} |V| ldn^2 + n^3)$, respectively, where t is the current number of rounds and $|V|$ is the maximum number of vertices in all polytopes processed.*

Proof sketch. Given the parameter k, we first prove that at round t, at most $O(\binom{t+k}{t} kn)$ sub-partitions need to be updated. Updating a partition requires $O(|V| dn)$ time [2], where $|V|$ is the maximum number of vertices in all sub-partitions. Thus, the time complexity for updating all partitions is $O(\binom{t+k}{t} |V| kdn^2)$. The time for selecting the next question using the random-based selection is $O(\binom{t+k}{t} kn + kn^2)$, and for the score-based selection it is $O(\binom{t+k}{t} kn + n^3)$. Since $k \leq \left\lfloor \frac{l-1}{2} \right\rfloor$, the total time complexity then follows. □

It is worth mentioning that $|V|$ is not large in typical scenarios. From [40], $|V| = O(m^{\lfloor \frac{d}{2} \rfloor})$ where m is the maximum number of halfspaces bounding a polytope. Typically, $m << n$ although it can be $n - 1$ in the worst case. For our experiment where $d = 5$ and $n = 10,000$, the value of m is smaller than 100. Besides, the value of d is not large (at most 7 in most cases) due to the limited number of attributes considered by humans in decision-making [7, 30, 42, 44].

5.2 The *SS* Algorithm

In algorithm *SE*, we maintain data structures that record confidence regions (i.e., R^is) and the sets of partitions belonging to each confidence region (i.e., X^is). However, observe that the stopping condition only depends on X^is. If X^is can be obtained through some other ways, maintaining R^is will not be necessary. Therefore, in algorithm *SS*, R^is are only kept conceptually, and X^is are determined using some pre-computed results on a set of randomly sampled points. Compared with *SE*, *SS* is much more time-efficient by sacrificing the accuracy a little bit in finding the best point.

SS consists of two phases, namely the *pre-processing phase* and the *running phase*. The task of the pre-processing phase is to uniform-randomly sample a number Y of points from each partition P using techniques developed in existing studies (e.g., [6]). To distinguish between data points and sample points, we use q to denote a sampled point and $P(q)$ to denote the partition where q is sampled from. After the pre-processing phase is finished, the sampled points can be stored so that future runs can skip this phase and start directly with the running phase.

Similar to a data point, a sample point q is said to *belong to* a confidence region R^i if either (1) $q \in R^i$ and $i = 0$, or (2) $q \in R^i$ and $q \notin R^{i-1}$ where $i \geq 1$. q is said to be *discarded* if $q \notin R^k$. Different

from *SE*, we do not compute R^is in *SS*. Instead, we maintain $k+1$ sets, namely $Q^0, Q^1, ..., Q^k$, where Q^i stores the set of sample points that belong to R^i where $i \in [0,k]$. Similar to above, we use Q_t^i to denote Q^i just after round t.

In the running phase, initially, all the sample points belong to Q_0^0, and other sets $Q_0^1, Q_0^2, \ldots, Q_0^k$ are empty. In round t, the empty-initialized Q_t^i is constructed as follows: (1) for each point $q \in Q_{t-1}^i$, q is inserted to Q_t^i if $q \in s$; (2) for each point $q \in Q_{t-1}^{i-1}$, q is inserted to Q_t^i if $q \notin s$; With the above updating rules, each Q_t^i stores the samples belonging to the corresponding R_t^i. Assuming that the number of sampled points is adequately large (to be discussed next), given $Q_t^0, Q_t^1, \ldots, Q_t^k$, X_t^is can be easily determined (i.e., $X_t^0 = \{P(q)|q \in Q_t^0\}$, and $X_t^i = \{P(q)|q \in Q_t^i\}/\{P(q)|q \in Q_t^{i-1}\}$). Since the question selection strategies we developed for *SE* in Section 5.1.2 and the stopping condition stated in Section 4.2 only require X_t^is, they can be easily applied to *SS*. Based on the question selection strategy, *SS* also has two variants, namely *SS-random* and *SS-score*.

Since *SS* uses sample points to decide which partition is still under consideration, even if the partition corresponding to the best point still overlaps with R^k, the best point will not be returned if there is no sample point in the intersection of this partition and R^k. To make this probability small, the sample size should be sufficiently large. Lemma 6 decides a sufficient number of samples which guarantees that this case happens with a probability at most a user parameter ε.

Lemma 6. *The intersection of hyperplanes in set $\{h_{ij}|p_i, p_j \in D\}$ divides the utility space into a number of cells, where each cell corresponds to a unique ranking of points in D. Given a parameter ε, if the number of cells in any partition is at most ρ, by sampling $Y = \frac{\rho}{\varepsilon e}$ points from each partition, SS returns the best point with probability at least $1-\varepsilon$ if the cell where the real utility vector lies overlaps with R^k.*

Proof sketch. When the algorithm stops, if the cell where the real utility vector lies overlaps with R^k, then the best point will not be returned only if there is no sample point in this cell. We show that the upper bound of this event's probability is $\frac{\rho}{Y}e^{-1}$. Solving $\frac{\rho}{Y}e^{-1} \leq \varepsilon$ yields the lemma. □

Although in the worst case $\rho = O(n^d)$ [27], in practice, the sample size Y can be set to a much smaller number. Our experimental results in Section 6 show that by sampling 1000 points from each partition, the accuracy gap between *SE* and *SS* is smaller than 1%. Theorem 4 presents the time complexity analysis on *SS*.

Theorem 4. *Given an input size n, dimensionality d and the return size l. Let Y be the number of samples from each partition. The time complexity for the pre-processing phase is $O((|V|+Y)dn^2)$, where $|V|$ is the maximum number of vertices in all partitions. In the running phase, the time complexities in each round of SS-random and SS-score are $O(Ydn + ln^2)$ and $O(Ydn + n^3)$, respectively.*

Proof sketch. The time complexity for the pre-processing phase is $O((|V| + Y)dn^2)$ since computing all partitions takes $O(|V|\,dn^2)$ time and sampling all Yn points takes $O(Ydn^2)$ time [6]. Given the parameter k, in each round of the running phase, the time required to update $Q^0, \ldots, Q^k$ is $O(Ydn)$, and the time required to decide the next question in *SS-random* and *SS-score* are $O(kn^2)$ and $O(n^3)$, respectively. Since $k \leq \left\lfloor \frac{l-1}{2} \right\rfloor$, the total time complexity then follows. □

As a final remark, note that Section 4.3 did not include details on two configurations for *FC*, namely (1) how to maintain confidence regions (i.e., R^is) and the sets of partitions belonging to each confidence region (i.e., X^is) and (2) what question selection strategy should be used for Stage 2. To address the first configuration, we can choose to follow the method used by *SE* (i.e., maintaining data structures for R^is and X^is) or *SS* (i.e., keeping R^is conceptual and deciding X^is using sampled points). As for the second configuration, we can opt for either the random-based selection or the score-based selection. In Section 6, we will show that we use the method used by *SE* to maintain R^is and X^is, and use the score-based selection for Stage 2, since it yields both a satisfactory accuracy and fast processing time.

Lastly, Theorem 5 bound the probability that the best point is returned by our algorithms.

Theorem 5. *Given an error rate upper bound θ, a return size l, and a parameter ε as defined in Lemma 6. if our proposed algorithms terminate in T rounds, then the probability that the best point is returned by SE and FC is at least $1 - \frac{e^{\lfloor (l-1)/2 \rfloor - \theta T}}{(\frac{\lfloor (l-1)/2 \rfloor}{\theta T})^{\lfloor (l-1)/2 \rfloor}}$; the probability that the best point is returned by SS is at least $1 - \varepsilon - \frac{e^{\lfloor (l-1)/2 \rfloor - \theta T}}{(\frac{\lfloor (l-1)/2 \rfloor}{\theta T})^{\lfloor (l-1)/2 \rfloor}}$.*

Proof sketch. *SE* and *FC* fail to return the best point only when the user makes more than k errors out of T questions, which can be bound directly using Chernoff inequality. The failure probability of *SS* can also be proved similarly. □

6 EXPERIMENT

Our experiments were conducted on a computer with 3.10 GHz CPU and 64GB RAM. All programs were implemented in C/C++.

Datasets. We conducted experiments on synthetic and real datasets. Statistics of the datasets are summarized in Table 3 in the Appendix. For synthetic datasets, we generated *anti-correlated* datasets using a dataset generator developed for skyline operators [5]. For real datasets, we used 3 real datasets: *AirQuality*, *Weather*, and *HTRU*. *AirQuality* has 420,478 tuples with 4 attributes, *Weather* includes 96,483 weather records with 6 attributes, and *HTRU* has 17,898 points with 7 attributes. Each dimension is normalized into the range of $[0, 1]$. We preprocessed all the datasets to contain only the skyline points, which are the possible best points for any utility function. As can be found in Section B.1.5 of the Appendix, this preprocessing step does not unfairly favor our algorithms.

Algorithms. We compare our proposed algorithms, namely *FC*, *SE* and *SS*, against the competitor algorithms, including (a) algorithms that do not consider user errors, namely *HD-PI* [42], *UH-Simplex* [44] and *UtilApprox* [31], and (b) algorithms that consider user errors, namely *Verify-Point* [7], *Active-Ranking* [18] and *Pref-Learn* [34]. Note that [7] also proposes another algorithm *Verify-Space*. Since its performance is similar to *Verify-Point* in [7], we do not include it here. To make the comparison fair, we adapt each of

them to return at most l points that are most likely to be the best point. The adaptions are summarized below.

(1) Algorithms *HD-PI*, *Verify-Point* and *UH-Simplex* all maintain a set of candidate points during their interaction processes. We return all points in the candidate set when its size is no more than l. Specifically, since the candidate set maintained by *HD-PI* stores the possible top-k points, we set k to 1. The set maintained in *Verify-Point* stores the possible best points, which need no further adaption. In *UH-Simplex*, the candidate set contains points with regret ratios possibly lower than a parameter ϵ. Following [7, 42], we set ϵ to $1 - f(p_2)/f(p_1)$, where p_1 and p_2 are the best and second best points according to the utility vector, which is equivalent to finding the best point. (2) Algorithm *Active-Ranking* aims at learning the entire ranking of all points by interacting with the user. We return the top-l points after the entire ranking is obtained. (3) Algorithm *Pref-Learn* interacts with the user to learn the user's utility vector. Algorithm *UtilApprox* returns points with regret ratios smaller than a parameter ϵ by estimating the user's utility vector. For these two algorithms, we return the top-l points w.r.t to the learned utility vector after the learning processes finish. Specifically, for *Pref-Learn*, we set its error threshold to 10^{-6} since according to [34], the learnt vector is very close to the theoretical optimum if the error threshold is less than 10^{-5}. For *UtilApprox*, we set ϵ in the same way as *UH-Simplex* ($\epsilon = 1 - f(p_2)/f(p_1)$) to find the best point.

Parameter Setting. We evaluate the performance of each algorithm by varying different parameters: (1) the dataset size N, (2) the dimensionality d, (3) the user error rate upper bound θ, (4) the return size l, (5) the parameter α in the score-based selection (Section 5.1.2), and (6) the parameter Y in Theorem 4 controlling the number of samples from each partition. The default setting for each synthetic dataset is $N = 100,000$ and $d = 4$. The default value of θ is 0.05, which, according to the human reliability assessment data in [21], is a reasonable upper bound for the human error rate. According to the results in Section 6.1, we set the default value $l = 5$, $\alpha = 0.2$, and $Y = 1000$.

Performance Measurement. The performance of each algorithm is evaluated by the following measurements: (1) *Accuracy* which is the probability that the best point is returned. Formally, accuracy is defined as $\frac{T_{ret}}{T_{tot}}$ where T_{tot} is the total number of trails and T_{ret} is the number of times the best point is returned. (2) *Number of questions* required to return the points. (3) *Processing time* which is the average processing time to decide the next question. Each setting is repeated 100 times and the average value is reported.

The following sections are organized as follows. In Section 6.1, we analyze the impact of various parameters on the performance of our algorithms. We then present the experimental results on synthetic datasets (Section 6.2) and real datasets (Section 6.3). The findings from a user study are discussed in Section 6.4. Finally, we provide a summary of the experiments in Section 6.5.

6.1 Experiments on Parameter Setting

In this section, we study the effect of different configurations on algorithms *SE*, *SS* and *FC*. We also evaluate the impact of different settings of parameters l, α and Y.

We studied the effect of the score-based selection and random-based selection on *SE* and *SS*. Details can be found in Section B.1.1 of the Appendix due to the lack of space. We observe no significant difference in terms of accuracy between the two strategies. The score-based selection requires fewer questions, while the random-based selection has a faster processing time. Since asking fewer questions indicates less user effort, which is considered more important than a small gain in response time, we choose the score-based selection as the default strategy. In the rest of this section, when we describe *SE* (*SS*), we mean *SE-score* (*SS-score*).

We compared the processing time of *SE* and *SS* with and without the lazy updating technique on synthetic and real datasets. Details can be found in Section B.1.2 of the Appendix. The results consistently demonstrate that both algorithms have faster processing times when they use the lazy updating technique. The degree of speedup varies across datasets, with *SE* accelerated by 5-30% and *SS* accelerated by 10-60%. Due to these improvements, in subsequent sections, we present the performance of *SE* and *SS* with this technique enabled.

We tested the impact of different configurations on the performance of *FC*, including (1) which method (i.e., the method used by *SE* and the one by *SS*) should be used to maintain R^is and X^is and (2) which selection (i.e., the random-based selection and the score-based selection) should be used as the question selection strategy for Stage 2 of *FC*. The results can be found in Section B.1.3 of the Appendix. We eventually decided to use (1) the method of *SE* (i.e., maintaining data structures for both R^is and X^is) to maintain R^is and X^is and (2) the score-based selection as the question selection strategy for Stage 2, since this combination yields the highest accuracy with a relatively small number of questions.

Figure 11 analyzes the impact of varying the value of l from 1 to 9 on our algorithms. According to Figure 11 (a) and (b), when l increases, the accuracies and the number of questions of all our algorithms also increase. This is because with a larger value of l, the best point is less likely to be discarded, but more questions are required to discard other points. Based on these results, we select $l = 5$ as our default setting since it yields a high level of accuracy while asking a small number of questions.

We varied α from 0.1 to 1.0 to study its impact on *FC*, *SE* and *SS*. Changing α does not significantly affect the accuracies of the algorithms. We chose $\alpha = 0.2$ since it minimized the number of questions needed for all 3 algorithms. We also examined the impact of increasing parameter Y from 50 to 5000 on the performance of *SS*. When Y increases, *SS* demonstrates higher accuracy, an increased number of questions, a longer processing time to decide the next question, and a longer time for the pre-processing phase. We chose $Y = 1000$ since it offers satisfactory accuracy and a fast processing time. Figures related to the experiments on α and Y are put in Section B.1.4 of the Appendix.

6.2 Experiments on Synthetic Datasets

Figure 12 summarizes our study on the algorithms' performance in handling different types of user errors: random errors, persistent errors and a combination of both. In this figure, "random" means that all errors are random errors, "persist" means that all errors are persistent errors, and "combined" means that 50% of the errors are persistent errors and the rest are random errors. Except for *Verify-Point*, all algorithms exhibit consistent performance across the three

error types, since they avoid asking repeated questions, making persistent and random errors equivalent. *Verify-Point* achieves slightly lower accuracy than our algorithms when dealing with random errors. However, since its technique (i.e., asking the same question several times) fails to address persistent errors, its accuracy decreases when the number of persistent errors increases. Since persistent errors are harder to be handled, we present algorithm performance assuming all user errors to be persistent for the rest of this section.

In Figure 13, we compare the performance of our algorithms (*FC*, *SE*, and *SS*) with existing methods on 4-d synthetic datasets of varying sizes (from 100 to 1 million). As shown in Figure 13 (a), our algorithms consistently outperform existing methods in terms of accuracy, with a widening performance gap when the dataset size increases. They achieve over 10% higher accuracy than the closest competitor (*UtilApprox*) for large input sizes (1 million). Among our algorithms, *SE* and *SS* are the most round-efficient, asking at most 10 more questions compared to the most round-efficient one (*HD-PI*). Even on large datasets (1 million points), our algorithms run at an interactive speed as they determine the next question within 3 seconds.

Figure 14 shows the effect of varying θ from 0.01 to 0.15. According to Figure 14 (a), our algorithms achieve the highest accuracies, decrease at the slowest rates and remain above 75% even with high error rates (e.g., 0.15), but all other methods fall below 65%. Increasing θ does not have a significant impact on the number of questions required by our algorithms, except for *FC*, whose number of questions decreases when θ increases since cycles can be obtained more quickly when more inconsistencies are involved.

Figure 15 shows our algorithms' scalability with increasing dimensionality d. Our algorithms consistently achieve higher accuracies for all dimensional settings compared to existing methods, and the difference in accuracy grows even larger when d increases. Besides, they require only 5-8 additional questions per dimensionality increase. Although *SE* and *SS* take around 20 seconds for $d = 5$ (since finding the best hyperplane in the large table L is time-consuming), *FC*'s processing time is still around 1 second.

Besides average processing time, we also measured the P95, median, and accumulated processing time of each algorithm. We observe that *SE* and *SS* decide 95% of their questions within 3 seconds, and *FC* take less than 1 second, demonstrating that they run at an interactive speed. The pre-processing phase of *SS* finishes within 2 minutes for even the largest (1 million points) or highest-dimensionality ($d = 5$) datasets, which is efficient given that the results of the pre-processing phase can be reused. Detailed results can be found in Section B.2.1 and B.2.2 of the Appendix for the sake of space.

In addition, we conducted experiments to analyze how utility vectors in different ranges of the utility space affect the performance of algorithms. Our finding shows that our algorithms consistently achieve the highest accuracy in all tested ranges of utility vectors. Details can be found in Section B.2.3 of the Appendix.

6.3 Experiments on Real Datasets

We compared the performance of our algorithms against the existing methods on 3 real datasets, namely *AirQuality*, *Weather* and *HTRU*. Due to the space constraint, we only present the results on the largest real dataset *AirQuality* in Figure 16. The results on *Weather* and *HTRU* can be found in Section B.3 of the Appendix. Our methods obtain higher accuracies than existing algorithms on all 3 datasets. In particular, the accuracies of *SE* and *SS* are near 100%, and *FC* also outperforms all other competitors by at least 5%. Although existing algorithms *HD-PI* and *Verify-Point* require 5-7 fewer rounds than our most round-efficient algorithm (*SS*), their accuracies are 10%-20% lower than ours, which is not satisfactory. Our algorithm runs at an interactive speed, since even the most time-consuming algorithm *SE* determines the next question within seconds on all 3 datasets.

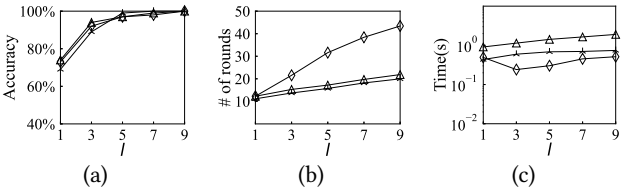
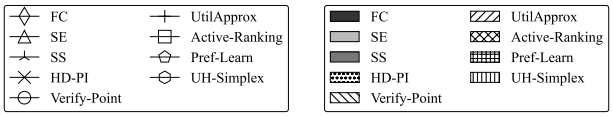


Figure 11: Effect of l (and k)



6.4 User Study

We conducted a user study on a real dataset *Airbnb* [9] to study user errors' impact on algorithm performance and our algorithms' effectiveness. *Airbnb* consists of 6809 Airbnb rentals in Amsterdam with 4 attributes: daily price, cleanliness rating, location rating and the number of reviews. Following [7, 42], we randomly sampled 1000 Airbnb rentals and recruited 25 participants. We compared our algorithms, namely *SE* and *SS*, against existing algorithms, namely *Verify-Point*, *HD-PI*, *Active-Ranking* and *Pref-Learn*. For *Pref-Learn*, since it is hard to obtain the user's real utility vector, it is re-adapted following [7, 42]: It maintains an estimated utility vector u. If 75% [34] of some randomly selected questions answered by the user can be correctly predicted using u, it stops and returns the top-l points w.r.t. u. We set $l = 5$ in our study, following our experimental settings.

The user study contains two parts. In Part 1, the user was interacted with each algorithm for several rounds until the algorithm returns a list of at most l items. During each round, two Airbnb rental options were shown, and the user was asked to select the preferred option. Once the list was returned, the user was required to select one item from the list as the *selected favorite rental option* of the algorithm. After the selected favorite rental options of all algorithms are chosen by the user, the user was asked to select one option among all of these selected favorite rental options as his/her *tentative* best point. Then, a set of additional questions were asked to confirm whether this *tentative* best point was indeed preferred to each of the other selected favorite rental options. Whenever each of these selected favorite rental options, says p, was more preferred to the tentative best point p_{best}, an additional question was asked to compare these two points (i.e., p and p_{best}) and the point preferred by the user with more questions became the new tentative best point. After we finish the process, the current tentative best point is regarded as the best point of the user.

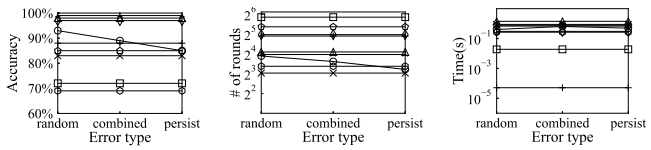


Figure 12: Effect of different types of errors

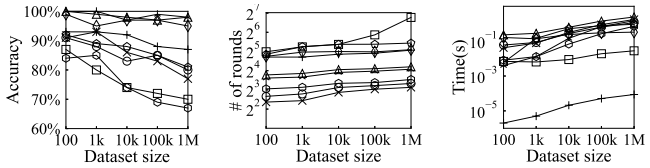


Figure 13: Effect of input size on 4d datasets

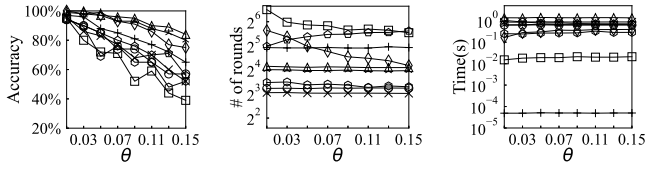


Figure 14: Effect of θ

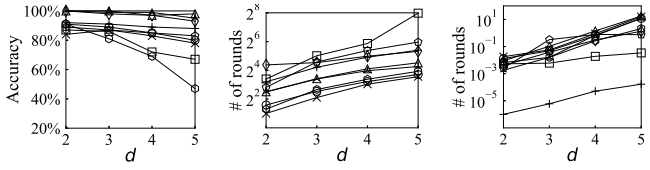


Figure 15: Effect of d

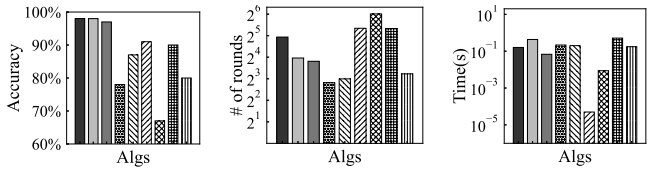


Figure 16: Results on dataset *AirQuality*

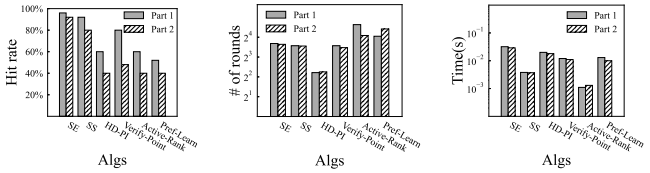


Figure 17: Results on user study

Part 2 is similar to Part 1 but in Part 2, we want to introduce a *known* persistent error. In Part 2, in the first round (for each algorithm), the user was explicitly required to select the *less preferred* one, and the remaining rounds are kept the same as Part 1 (by asking the user to select the *preferred* one). This (real) less preferred one in the first round is recorded as the more preferred one in the system/algorithm, which is considered as a *known* persistent error "artifically" introduced by Part 2. If in later rounds of the algorithm, the same question was selected again to ask this user, the recorded answer would be used automatically. The reason for this design is to introduce some persistent errors in each algorithm. We evaluated algorithms' performance with the following metrics: (1) the *hit rate* which is the probability that the best point is included in the returned list, (2) the number of rounds required to return the list, and (3) the average processing time to decide the next question. The average scores of all participants are reported.

Figure 17 displays the results. *SE* and *SS* achieve about a 90% hit rate in both Part 1 and Part 2, significantly outperforming other competitors. The hit rates of all algorithms in Part 2 are generally lower than those in Part 1 due to the introduced persistent error. However, our algorithms' hit rates remain relatively stable, demonstrating their ability to handle persistent errors. In contrast, *Verify-Point* experiences a hit rate drop of over 30% in Part 2. While *HD-PI* asks 5-6 fewer questions than ours, its hit rate is 30% and 40% lower in Parts 1 and 2, respectively. The processing time of our algorithms is fast since they determine the next question within 0.1 seconds.

Since $l = 5$, we derive that $k = 2$. We measured how frequently users make errors using algorithm *SE*, using the properties that the best point belongs to the i-th confidence region only if at most i errors are made, and the best point is not in the returned set only if at least $k + 1$ errors are made. We regard the best point found in Part 1 by our user study as the real best point of the user (since we used additional checking questions to make sure that this point is the real best point with high probability) and record necessary information that decide the confidence region this point belongs to. The lower bound of the empirical user error rate can be calculated as $\frac{\sum_x e_x}{\sum_x t_x}$, where e_x is the number of errors made by user x (calculated based on the above properties) and t_x is the number of rounds used by user x. Among 633 questions asked by *SE*, users made 23 errors (excluding the artificial errors from Part 2), resulting in a 3.6% empirical error rate.

6.5 Summary

The experiments demonstrated the superiority of our proposed algorithms, namely *FC*, *SE* and *SS*, over existing approaches. (1) We are efficient and effective. We achieve nearly 100% accuracy in most of the experiments using a small number of rounds, which consistently outperforms existing algorithms. (2) We are scalable to the input size and dimensionality. For example, on 7-d dataset *HTRU*, *SE* and *SS* finish with around 30 questions and achieve over 98% accuracy, but algorithms *UtilApprox* and *Active-Ranking* obtain lower accuracies with even more rounds. (3) We are capable of handling many persistent errors. Even when the error rate is high (e.g., 0.15), all our algorithms still achieve more than 75% accuracy, which is at least 10% higher than other existing approaches.

7 CONCLUSION

In this paper, we propose interactive algorithms that return the user's best point with high confidence even when the user makes persistent errors and random errors during the interaction, which makes them more robust than existing approaches. Specifically, we proposed algorithm *FC*, requiring an asymptotically optimal number of rounds, and two algorithms *SE* and *SS*, requiring a small number of questions empirically. All 3 algorithms return the best

point with a provable guarantee. We conducted extensive experiments to demonstrate that our algorithms are efficient and effective when addressing both types of errors. In the future, we study how to handle user errors when categorical attributes are involved.

REFERENCES

[1] Yongkil Ahn. 2019. The economic cost of a fat finger mistake: a comparative case study from Samsung Securities's ghost stock blunder. *Journal of Operational Risk* 16, 2 (2019).

[2] David Avis and Komei Fukuda. 1991. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proceedings of the seventh annual symposium on Computational geometry*. 98–104.

[3] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. 2014. Domination in the probabilistic world: Computing skylines for arbitrary correlations and ranking semantics. *ACM Transactions on Database Systems (TODS)* 39, 2 (2014), 1–45.

[4] Ilaria Bartolini, Paolo Ciaccia, and Florian Waas. 2001. FeedbackBypass: A new approach to interactive similarity query processing. In *VLDB*. 201–210.

[5] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings 17th international conference on data engineering*. IEEE, 421–430.

[6] Apostolos Chalkis and Vissarion Fisikopoulos. 2020. volesti: Volume approximation and sampling for convex polytopes in r. *arXiv preprint arXiv:2007.01578* (2020).

[7] Qixu Chen and Raymond Chi-Wing Wong. 2023. Finding Best Tuple via Error-prone User Interaction. In *Proceedings of the 39th IEEE International Conference on Data Engineering*.

[8] Paolo Ciaccia and Davide Martinenghi. 2017. Reconciling skyline and ranking queries. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1454–1465.

[9] Airbnb dataset. [n. d.]. http://insideairbnb.com/get-the-data/.

[10] Eyal Dushkin and Tova Milo. 2018. Top-k sorting under partial order information. In *Proceedings of the 2018 International Conference on Management of Data*. 1007–1019.

[11] Brian Eriksson. 2013. Learning to top-k search using pairwise comparisons. In *Artificial Intelligence and Statistics*. PMLR, 265–273.

[12] Moein Falahatgar, Yi Hao, Alon Orlitsky, Venkatadheeraj Pichapati, and Vaishakh Ravindrakumar. 2017. Maxing and ranking with few assumptions. *Advances in Neural Information Processing Systems* 30 (2017).

[13] Moein Falahatgar, Ayush Jain, Alon Orlitsky, Venkatadheeraj Pichapati, and Vaishakh Ravindrakumar. 2018. The limits of maxing, ranking, and preference learning. In *International conference on machine learning*. PMLR, 1427–1436.

[14] Moein Falahatgar, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. 2017. Maximum selection and ranking under noisy comparisons. In *International Conference on Machine Learning*. PMLR, 1088–1096.

[15] Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. 2018. Optimal sorting with persistent comparison errors. *arXiv preprint arXiv:1804.07575* (2018).

[16] Reinhard Heckel, Nihar B Shah, Kannan Ramchandran, and Martin J Wainwright. 2019. Active ranking from pairwise comparisons and when parametric assumptions do not help. *The Annals of Statistics* 47, 6 (2019), 3099–3126.

[17] Reinhard Heckel, Max Simchowitz, Kannan Ramchandran, and Martin Wainwright. 2018. Approximate ranking from pairwise comparisons. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1057–1066.

[18] Kevin G Jamieson and Robert Nowak. 2011. Active ranking using pairwise comparisons. *Advances in neural information processing systems* 24 (2011).

[19] Yiling Jia, Huazheng Wang, Stephen Guo, and Hongning Wang. 2021. Pairrank: Online pairwise learning to rank by divide-and-conquer. In *Proceedings of the Web Conference 2021*. 146–157.

[20] Sumeet Katariya, Lalit Jain, Nandana Sengupta, James Evans, and Robert Nowak. 2018. Adaptive sampling for coarse ranking. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1839–1848.

[21] Barry Kirwan. 2017. *A guide to practical human reliability assessment*. CRC press.

[22] Rolf Klein, Rainer Penninger, Christian Sohler, and David P Woodruff. 2011. Tolerant algorithms. In *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19*. Springer, 736–747.

[23] Jongwuk Lee, Gae-won You, and Seung-won Hwang. 2009. Personalized top-k skyline queries in high-dimensional space. *Information Systems* 34, 1 (2009), 45–61.

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

[25] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.

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

[27] De Berg Mark, Cheong Otfried, van Kreveld Marc, and Overmars Mark. 2008. *Computational geometry algorithms and applications*. Spinger.

[28] Denis Mindolin and Jan Chomicki. 2009. Discovering relative importance of skyline attributes. *Proceedings of the VLDB Endowment* 2, 1 (2009), 610–621.

[29] Kyriakos Mouratidis, Keming Li, and Bo Tang. 2021. Marrying top-k with skyline queries: Relaxing the preference input while producing output of controllable size. In *Proceedings of the 2021 International Conference on Management of Data*. 1317–1330.

[30] Kyriakos Mouratidis and Bo Tang. 2018. Exact processing of uncertain top-k queries in multi-criteria settings. *Proceedings of the VLDB Endowment* 11, 8 (2018), 866–879.

[31] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino. 2012. Interactive regret minimization. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 109–120.

[32] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J Lipton, and Jun Xu. 2010. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1114–1124.

[33] Peng Peng and Raymong Chi-Wing Wong. 2015. k-hit query: Top-k query with probabilistic utility function. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 577–592.

[34] Li Qian, Jinyang Gao, and HV Jagadish. 2015. Learning user preferences by adaptive pairwise comparison. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1322–1333.

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

[36] Wenbo Ren, Jia Kevin Liu, and Ness Shroff. 2019. On sample complexity upper and lower bounds for exact ranking from noisy comparisons. *Advances in Neural Information Processing Systems* 32 (2019).

[37] Gerard Salton. 1989. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley* 169 (1989).

[38] Thomas Seidl and Hans-Peter Kriegel. 1997. Efficient user-adaptable similarity search in large multimedia databases. In *VLDB*, Vol. 97. 506–515.

[39] Zhexuan Song and Nick Roussopoulos. 2001. K-nearest neighbor search for moving query point. In *International Symposium on Spatial and Temporal Databases*. Springer, 79–96.

[40] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. 2017. *Handbook of discrete and computational geometry*. CRC press.

[41] Weicheng Wang and Raymond Chi-Wing Wong. 2022. Interactive mining with ordered and unordered attributes. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2504–2516.

[42] Weicheng Wang, Raymond Chi-Wing Wong, and Min Xie. 2021. Interactive Search for One of the Top-k. In *Proceedings of the 2021 International Conference on Management of Data*. 1920–1932.

[43] A Student with Top-tier Score Admitted by mediocre University (Chinese version only). 2020. https://news.southcn.com/node_6854f1135c/4357641930.shtml.

[44] Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. 2019. Strongly truthful interactive regret minimization. In *Proceedings of the 2019 International Conference on Management of Data*. 281–298.

[45] Jiping Zheng and Chen Chen. 2020. Sorting-based interactive regret minimization. (2020), 473–490.

D	the input dataset
d	the dimensionality of D
N	size of raw dataset
n	size of D's upper hull
u	the utility vector
θ	the upper bound of user error rate
p_i	data point in D
P_i	partition of p_i
$h_{i,j}$	the hyperplane related to p_i and p_j
l	the target return size
t	number of round
R^i	i-th confidence region
X^i	set of partitions belonging to R^i

Table 2: Commonly-used Symbols

This Appendix provides supplementary material that is not included in the main paper due to space constraints. Section A introduces a lazy updating technique that accelerates the score-based selection. Section B presents the additional experiments. Section C provides detailed proofs of lemmas and theorems in the paper. We summarized the commonly-used notations in Table 2.

A THE LAZY UPDATING TECHNIQUE

In this section, we introduce the lazy updating technique mentioned in Section 5.1.2 that can accelerate the score-based selection. Let $score_t(h) = \min(\sum_{i=0}^{k} \alpha^i Num(h^+, X_t^i), \sum_{i=0}^{k} \alpha^i Num(h^-, X_t^i))$ be the score of hyperplane h at round t. The technique is based on the following property of $score_t(h)$.

Lemma 7. *For any hyperplane h and any t, $score_t(h) \geq score_{t'}(h)$ for any $t' \geq t$. That is, the score of a hyperplane is non-increasing.*

Proof. Note that the relative position between a partition P and a hyperplane h never changes, i.e., P always lies in h^+, or h^-, or intersects with h. This implies that $\sum_{i=0}^{k} Num(h^+, X_t^i)$ (resp. $\sum_{i=0}^{k} Num(h^-, X_t^i)$) is a constant for any k. For a partition P that belongs to X_t^i in round t, P can only belong to $X_{t+1}^{i'}$ where $i' \geq i$ in round $t+1$. Therefore, we have $\sum_{i=0}^{k} \alpha^i Num(h^+, X_t^i) \geq \sum_{i=0}^{k} \alpha^i Num(h^+, X_{t+1}^i)$ for any t and any $\alpha \in [0,1]$, and similarly, $\sum_{i=0}^{k} \alpha^i Num(h^-, X_t^i) \geq \sum_{i=0}^{k} \alpha^i Num(h^-, X_{t+1}^i)$. Thus, we have

$$\begin{aligned} score_t(h) &= \min(\sum_{i=0}^{k} \alpha^i Num(h^+, X_t^i), \sum_{i=0}^{k} \alpha^i Num(h^-, X_t^i)) \\ &\geq \min(\sum_{i=0}^{k} \alpha^i Num(h^+, X_{t+1}^i), \sum_{i=0}^{k} \alpha^i Num(h^-, X_{t+1}^i)) \\ &= score_{t+1}(h) \end{aligned}$$

for ant t. Thus, we have $score_t(h) \geq score_{t'}(h)$ for any $t' \geq t$. □

Recall that in the original score-based selection, in each round, we need to compute the scores of all hyperplane candidates and select the one with the highest score. The lazy updating technique accelerates this process as follows. For each hyperplane h, we record its most recently computed score, which is called the *upper bound* of h since by Lemma 7, this score is an upper bound of the score h can obtain in any future rounds. In each round, to find the hyperplane with the highest score, we sequentially process all hyperplanes and record a current max score (initialized to 0) which is the highest among the hyperplanes processed so far. For each hyperplane to be processed, we first check if its upper bound is larger than the current max. If yes, we compute the score of this hyperplane and update its upper bound. Otherwise, this hyperplane cannot have the highest score and the computation of its score can be skipped.

B ADDITIONAL EXPERIMENTS

In this section, we present the experiments that are not included in the main body due to space constraints. Section B.1, B.2, B.3 and B.4 show the supplementary results on parameter settings, experiments on synthetic datasets, experiments on real-world datasets and experiments on question type flexibility, respectively. Statistics of synthetic and real datasets used in our experiments can be found in Table 3. For each synthetic dataset, we name it using its dimensionality and size. For example, "4d100" means the synthetic dataset with dimensionality 4 ($d = 4$) and size 100 ($N = 100$).

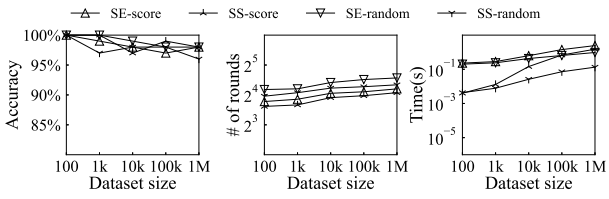


Figure 18: Comparison between the score-based selection and the random-based selection

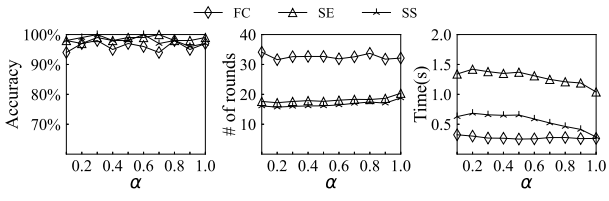


Figure 19: Effect of α

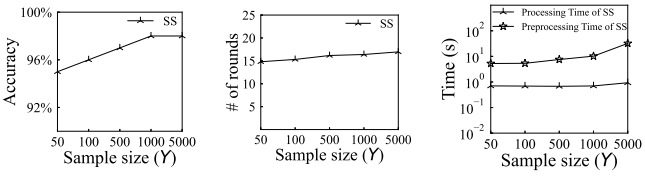


Figure 20: Effect of sample size

Dataset	size	dimensionality	skyline size
4d100	100	4	87
4d1k	1000	4	403
4d10k	10000	4	1484
4d100k	100000	4	3780
4d1m	1000000	4	6699
2d100k	100000	2	40
3d100k	100000	3	538
5d100k	100000	5	11352
AirQuality	420478	4	3948
Weather	96483	6	2110
HTRU	19898	7	11720

Table 3: Dataset Statistics

B.1 Additional Experiments on Parameter Setting

B.1.1 Experiments Comparing two Question Selection Strategies. We studied the effect of the score-based selection and random-based selection on *SE* and *SS*. Figure 18 shows the results. We observe no significant difference in terms of accuracy between the two strategies. The score-based selection requires fewer questions, while the random-based selection has a faster processing time. Since asking fewer questions indicates less user effort, which is considered more important than a small gain in response time, we choose the score-based selection as the default strategy. In the rest of this section, when we describe *SE* (*SS*), we mean *SE-score* (*SS-score*).

B.1.2 Experiments Showing the Effectiveness of the Lazy Updating Technique. We compared the processing time of *SE* and *SS* with and without the lazy updating technique on synthetic and real datasets. The results are listed in Table 4. All synthetic datasets are named using the convention stated at the beginning of Section B. The results consistently demonstrate faster processing times with the lazy updating technique for both algorithms. The degree of speedup varies across datasets, with *SE* accelerated by 5-30% and *SS* accelerated by 10-60%. In particular, we found that the acceleration is more significant when N or d is large (i.e., $N = 1,000,000$ or $d = 5$), which demonstrates the effectiveness of the lazy updating technique when a large number of hyperplanes needs to be considered.

B.1.3 Experiments Showing the Effect of Different Configurations on FC. We tested the impact of different configurations on the performance of *FC*, including (1) whether the method of *SE* or *SS* should be used to maintain R^i and X^i; and (2) whether the random-based selection or the score-based selection should be used as the questions selection strategy for Stage 2. The results of different combinations are shown in Figure 21. The bars labeled with "SE, score" means that the results are obtained when using the method of *SE* to maintain R^i and X^i and using the score-based selection as the questions selection strategy for Stage 2. The other labels are interpreted similarly. We found that the accuracies of variants using *SE* is higher than the variants using *SS*, and the score-based selection uses fewer questions than the random-based selection. We eventually decided to use (1) the method of *SE* and (2) the score-based selection as the default configuration of *FC* since it yields the highest accuracy with a relatively small number of questions.

B.1.4 Experiments Showing the Effect of α and Y. We varied α from 0.1 to 1.0 to study its impact on *FC*, *SE* and *SS*. Figure 19 (a) shows that changing α does not significantly affect the accuracies of the algorithms. However, increasing α reduces processing time due to detaching partitions from higher-level confidence regions more quickly, leading to fewer data structures needed to be maintained. We chose $\alpha = 0.2$ since it minimized the number of questions needed for all 3 algorithms.

We also examined the impact of increasing parameter Y from 50 to 5000 on the performance of *SS*. Figure 20 shows that when Y increases, *SS* demonstrates higher accuracy, an increased number of questions, a longer processing time to decide the next question, and a longer time for the pre-processing phase. We chose $Y = 1000$ since it offers satisfactory accuracy and a fast processing time.

B.1.5 Experiments Showing the Advantages of Preprocessed Datasets Containing Skyline Points Only. To show that preprocessing the dataset to contain only the skyline points does not make the comparison among algorithms unfair, we also record the performance of our competitors which do not need preprocessing on the raw 4-d synthetic dataset with 100,000 points. In particular, since *HD-PI, Verify-Point* and *UH-Simplex* all require this preprocessing step [7, 42, 44], we run the remaining competitors, namely *UtilApprox*, *Active-ranking* and *Pref-Learn* on the raw input. Table 5 presents a comparison of their performance on both the processed and raw datasets, with "proc" indicating results on the processed dataset (i.e., the dataset with skyline points only) and "raw" indicating results on the raw dataset. We observe that when running on the raw input, all three algorithms tend to ask more questions and obtain lower accuracies. This is expected since with the raw input they also need to consider a large number of non-skyline points. Particularly, the processing time of *UtilApprox* increases by two orders of magnitude, and *Active-ranking* asks 180 times more questions due to its objective of ranking all points before reporting the best tuple.

B.2 Additional Experiments on Synthetic Datasets

B.2.1 Experiments Showing P95, Median and Accumulated Processing Times. In addition to the average processing time, we also report other time statistics in Table 6, including the processing time at the 95th percentile (P95), median processing time, and accumulated processing time of all rounds. These statistics were collected from the 4-dimensional synthetic dataset with a size of 100,000. It is worth noting that at least 95% of questions of our algorithms can be determined in 3 seconds, which proves that our proposed algorithms run at an interactive speed.

B.2.2 Experiments on Time Required for the Pre-processing phase of SS. We studied the time required for the pre-processing phase of *SS* on synthetic datasets and real datasets. The results are summarized in Table 7. All synthetic datasets are named using the convention stated at the beginning of Section B. We found that even on the synthetic dataset with the highest dimensionality (i.e., 5d100k) and the dataset with the largest size (i.e., 4d1m), the pre-processing

Dataset	*SE*		*SS*	
	w. lazy update	w/o. lazy update	w. lazy update	w/o. lazy update
4d100	2.31×10^{-1}	2.40×10^{-1}	4.14×10^{-3}	5.96×10^{-3}
4d1k	2.79×10^{-1}	2.82×10^{-1}	1.31×10^{-2}	2.12×10^{-2}
4d10k	6.26×10^{-1}	7.33×10^{-1}	1.45×10^{-1}	2.48×10^{-1}
4d100k	1.42	1.86	6.84×10^{-1}	1.06
4d1m	2.44	3.49	1.50	2.43
2d100k	9.16×10^{-3}	1.38×10^{-2}	1.79×10^{-3}	1.88×10^{-3}
3d100k	1.05×10^{-1}	1.20×10^{-1}	1.60×10^{-2}	2.27×10^{-2}
5d100k	1.78×10^{1}	2.49×10^{1}	1.23×10^{1}	2.08×10^{1}
AirQuality	4.33×10^{-1}	4.92×10^{-1}	6.80×10^{-2}	1.27×10^{-1}
Weather	2.58	3.33	6.13×10^{-1}	1.38
HTRU	2.92	3.21	5.01×10^{-2}	1.11×10^{-1}

Table 4: Processing time of *SE* and *SS* with/without the lazy updating technique

	UtilApprox (proc)	UtilApprox (raw)	Pref-Learn (proc)	Pref-Learn (raw)	Active-ranking (proc)	Active-ranking (raw)
Accuracy	88%	87%	85%	84%	72%	70%
# of rounds	30.08	30.35	41.57	43.13	58.32	10671.50
Time (s)	5.22×10^{-5}	2.01×10^{-3}	7.78×10^{-1}	1.26	1.90×10^{-2}	1.12×10^{-3}

Table 5: Effect of skyline preprocessing

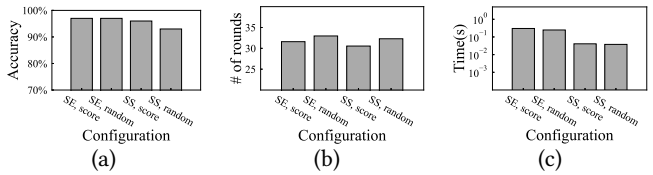


Figure 21: Different configurations of *FC*

Algorithm	average (s)	P95 (s)	median (s)	accumulated (s)
FC	3.02×10^{-1}	7.91×10^{-1}	2.68×10^{-1}	10.09
SE	1.42	2.87	6.45×10^{-1}	2.44×10^{1}
SS	6.84×10^{-1}	2.51	2.72×10^{-1}	1.08×10^{1}
HD-PI	8.85×10^{-1}	1.46	2.15×10^{-1}	7.26
Verify-Point	6.42×10^{-1}	7.05×10^{-1}	1.12×10^{-1}	1.17×10^{1}
UtilApprox	5.22×10^{-5}	8.34×10^{-5}	5.39×10^{-5}	1.57×10^{-3}
Active-Ranking	1.90×10^{-2}	7.93×10^{-2}	6.78×10^{-3}	1.11
Pref-Learn	7.78×10^{-1}	1.41×10^{-1}	5.74×10^{-2}	3.23×10^{1}
UH-Simplex	2.66×10^{-1}	3.13	6.61×10^{-2}	2.77

Table 6: Time statistics

phase requires less than 1 minute. Moreover, it also finishes within 2 minutes on all real datasets, including the 7-d dataset *HTRU*. We conclude that the pre-processing phase is efficient given that this phase only needs to be run once and its results can be stored and reused for future runs.

B.2.3 Experiments on the Effect of Different User Preferences. Since different users have distinct utility vectors, we also studied the algorithms' performance when utility vectors are sampled from different regions in the utility space U. To sample u from different regions, in each setting, we select a non-empty set $I \subsetneq \{1, 2, \ldots, d\}$ where d is the dimensionality, randomly set $u[i]$ in the range $[0, 0.5]$ if $i \in I$, and randomly set $u[i]$ in the range $[0.5, 1]$ if $i \notin I$. The utility vectors are then normalized such that $\sum_{1 \le i \le d} u[i] = 1$. The performance of our algorithms on 4-d synthetic datasets with size 100,000 are reported in Table 8, 9 and 10. For completeness, the performance of existing algorithms is also reported in Table 11, 12 and 13. In these tables, each row corresponds to a different choice of I, and the average of all choices of I is reported as well. We observe

Dataset	Time (s)
4d100	3.58
4d1k	4.83
4d10k	6.75
4d100k	9.24
4d1m	10.64
2d100k	0.18
3d100k	0.90
5d100k	46.73
AirQuality	5.35
Weather	35.02
HTRU	85.19

Table 7: Time for *SS* pre-processing phase

	FC	SE	SS
{1}	94%	100%	98%
{2}	93%	97%	99%
{3}	97%	100%	100%
{4}	94%	98%	97%
{1,2}	94%	99%	96%
{1,3}	97%	99%	98%
{1,4}	98%	98%	99%
{2,3}	98%	98%	100%
{2,4}	99%	99%	99%
{3,4}	98%	99%	100%
{1,2,3}	94%	100%	99%
{1,2,4}	99%	100%	100%
{1,3,4}	97%	100%	99%
{2,3,4}	97%	99%	100%
average	96%	99%	99%

Table 8: Effect of different u on our algorithms (Accuracy)

	FC	SE	SS
{1}	25.56	17.60	16.38
{2}	31.54	17.77	16.43
{3}	32.94	17.54	16.21
{4}	30.77	17.55	16.28
{1,2}	27.98	17.92	16.34
{1,3}	28.67	17.43	16.74
{1,4}	35.66	17.70	16.79
{2,3}	33.67	17.58	16.14
{2,4}	26.09	17.44	16.18
{3,4}	29.50	17.29	16.07
{1,2,3}	34.60	17.85	16.45
{1,2,4}	34.97	17.07	16.35
{1,3,4}	33.96	17.60	16.64
{2,3,4}	36.39	17.44	16.13
average	31.59	17.56	16.37

Table 9: Effect of different u on our algorithms (# of rounds)

	FC	SE	SS
{1}	5.73×10^{-1}	1.31	6.83×10^{-1}
{2}	4.56×10^{-1}	1.41	7.12×10^{-1}
{3}	4.28×10^{-1}	1.38	7.21×10^{-1}
{4}	4.54×10^{-1}	1.30	6.63×10^{-1}
{1, 2}	4.98×10^{-1}	1.34	7.04×10^{-1}
{1, 3}	4.86×10^{-1}	1.31	6.54×10^{-1}
{1, 4}	4.09×10^{-1}	1.25	6.35×10^{-1}
{2, 3}	4.31×10^{-1}	1.38	7.28×10^{-1}
{2, 4}	5.29×10^{-1}	1.32	6.77×10^{-1}
{3, 4}	4.67×10^{-1}	1.33	6.92×10^{-1}
{1, 2, 3}	4.20×10^{-1}	1.31	6.72×10^{-1}
{1, 2, 4}	4.12×10^{-1}	1.35	6.79×10^{-1}
{1, 3, 4}	4.20×10^{-1}	1.26	6.23×10^{-1}
{2, 3, 4}	4.04×10^{-1}	1.33	6.94×10^{-1}
average	4.56×10^{-1}	1.33	6.81×10^{-1}

Table 10: Effect of different u on our algorithms (Time (s))

that our algorithms consistently outperform existing algorithms on almost every setting of I in terms of accuracy. Furthermore, our algorithms exhibit stability with minimal performance variations across different I values. In contrast, some existing algorithms (e.g., Active-Ranking) show sensitivity to changes in I, resulting in accuracy fluctuations exceeding 20%.

B.3 Additional Experiments on Real Datasets

This section presents the experimental results on real datasets that are not included in the main body due to space constraints. Figure 22 shows the performance of our algorithms and the competitors on dataset *Weather*, and Figure 23 shows the results on dataset *HTRU*. Our algorithms' performance on these two datasets is consistent with our findings in Section 6.3. They outperform all existing methods by at least 5% accuracies with only a small number of questions. For those existing algorithms that finish with fewer questions than ours, namely *HD-PI*, *Verify-Point* and *UH-Simplex*, their accuracies are at least 10% lower than ours, which is not satisfactory.

B.4 Experiments on Question Type Flexibility

To test the flexibility of our algorithms on different types of questions, in this section, instead of asking questions in the form of pairwise comparison, we adapted our algorithms *FC*, *SE* and *SS* to ask two new types of questions, namely in each round (a) displaying $s > 2$ points and asking the user to select the favorite one, and (b) displaying $s > 2$ points and asking the user to divide them into two groups called the *superior group* and the *inferior group*. We describe the results of these variants in Section B.4.1 and B.4.2, respectively.

Since the algorithms need to display s (instead of 2) points in each round, we adapted the score-based selection as follows: instead of returning the pair with the highest score whose preference is unknown, we scan through all pairs with unknown preferences and find $\frac{(s-1)(s-2)}{2} + 1$ pairs with the highest score, sort them in decrease scores and select distinct points in these pairs until s points are selected, which are displayed to the user (the reason why

	HD-PI	Verify-Point	UtilApprox	Active-Ranking	Pref-Learn	UH-Simplex
{1}	79%	76%	92%	76%	88%	66%
{2}	84%	85%	91%	78%	92%	75%
{3}	80%	80%	84%	63%	87%	72%
{4}	73%	77%	87%	75%	91%	66%
{1, 2}	83%	77%	92%	76%	83%	85%
{1, 3}	80%	87%	93%	77%	92%	60%
{1, 4}	84%	84%	89%	53%	75%	57%
{2, 3}	72%	79%	93%	72%	82%	81%
{2, 4}	77%	79%	90%	78%	86%	55%
{3, 4}	73%	77%	89%	73%	90%	64%
{1, 2, 3}	75%	76%	87%	66%	82%	57%
{1, 2, 3}	81%	80%	92%	62%	83%	59%
{1, 2, 3}	76%	85%	88%	61%	87%	52%
{1, 2, 3}	79%	84%	85%	79%	89%	61%
average	78%	80%	89%	71%	86%	65%

Table 11: Effect of different u on existing algorithms (Accuracy)

	HD-PI	Verify-Point	UtilApprox	Active-Ranking	Pref-Learn	UH-Simplex
{1}	7.93	9.65	29.09	60.92	44.16	12.15
{2}	8.10	9.49	29.78	60.94	42.91	10.34
{3}	8.43	9.27	29.17	97.21	41.80	10.62
{4}	8.35	9.57	27.85	63.14	44.63	12.26
{1, 2}	8.11	9.48	29.53	59.48	43.55	8.69
{1, 3}	8.24	9.29	29.39	56.00	40.96	11.89
{1, 4}	8.18	9.45	30.56	59.84	44.45	12.46
{2, 3}	8.02	9.64	30.67	60.01	42.47	10.25
{2, 4}	7.99	9.13	28.76	58.33	42.93	12.29
{3, 4}	8.46	9.40	29.75	56.91	41.25	12.44
{1, 2, 3}	8.17	9.66	33.14	57.93	42.80	10.78
{1, 2, 4}	8.09	9.21	31.15	56.40	41.48	10.23
{1, 3, 4}	8.10	9.36	30.34	55.26	42.42	11.58
{2, 3, 4}	8.28	9.42	29.40	60.12	41.53	13.30
average	8.18	9.43	29.90	61.61	42.67	11.38

Table 12: Effect of different u on existing algorithms (# of rounds)

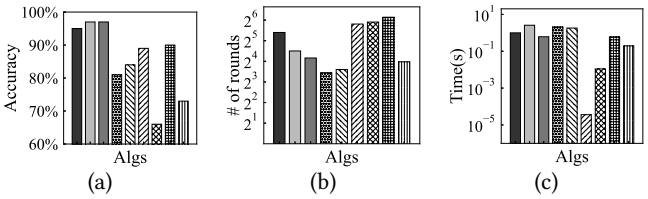


Figure 22: Results on dataset *Weather*

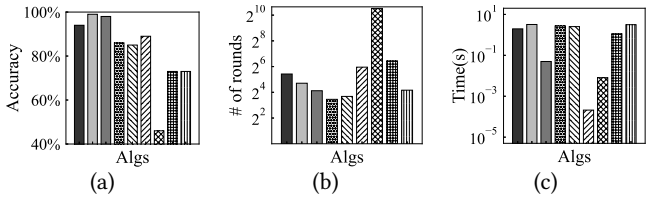
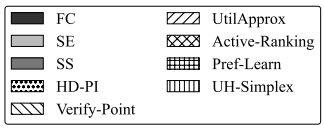


Figure 23: Results on dataset *HTRU*



$\frac{(s-1)(s-2)}{2} + 1$ pairs are maintained is to ensure that there are at least s distinct points in these pairs).

As a special treatment, since Stage 1 of *FC* has its own point selection strategy, we adapted it by first selecting two points using its own strategy, then selecting the remaining $s-2$ points using the adapted version of the score-based selection.

B.4.1 Selecting favorite point among s displayed points. In this set of experiments, we adapted the algorithms to display more than two points in each round. Specifically, $s > 2$ points are displayed

	HD-PI	Verify-Point	UtilApprox	Active-Ranking	Pref-Learn	UH-Simplex
{1}	3.82×10^{-1}	6.24×10^{-1}	5.07×10^{-5}	1.82×10^{-2}	7.49×10^{-1}	2.27×10^{-1}
{2}	3.04×10^{-1}	7.09×10^{-1}	5.38×10^{-5}	1.91×10^{-2}	7.54×10^{-1}	2.63×10^{-1}
{3}	2.85×10^{-1}	6.49×10^{-1}	5.05×10^{-5}	1.18×10^{-2}	7.65×10^{-1}	4.19×10^{-1}
{4}	3.03×10^{-1}	6.26×10^{-1}	5.12×10^{-5}	1.84×10^{-2}	7.39×10^{-1}	4.33×10^{-1}
{1, 2}	3.32×10^{-1}	6.34×10^{-1}	5.13×10^{-5}	1.94×10^{-2}	7.3×10^{-1}	1.94×10^{-1}
{1, 3}	3.24×10^{-1}	6.48×10^{-1}	5.12×10^{-5}	2.06×10^{-2}	7.63×10^{-1}	1.76×10^{-1}
{1, 4}	2.73×10^{-1}	6.14×10^{-1}	5.26×10^{-5}	1.92×10^{-2}	7.06×10^{-1}	3.77×10^{-1}
{2, 3}	2.87×10^{-1}	6.21×10^{-1}	5.21×10^{-5}	1.94×10^{-2}	7.42×10^{-1}	2.7×10^{-1}
{2, 4}	3.52×10^{-1}	6.47×10^{-1}	5.14×10^{-5}	1.98×10^{-2}	7.25×10^{-1}	3.98×10^{-1}
{3, 4}	3.11×10^{-1}	6.37×10^{-1}	5.23×10^{-5}	1.97×10^{-2}	7.44×10^{-1}	4.12×10^{-1}
{1, 2, 3}	2.80×10^{-1}	6.29×10^{-1}	5.18×10^{-5}	2.10×10^{-2}	7.38×10^{-1}	1.79×10^{-1}
{1, 2, 4}	2.74×10^{-1}	6.42×10^{-1}	5.96×10^{-5}	2.02×10^{-2}	7.36×10^{-1}	3.48×10^{-1}
{1, 3, 4}	2.80×10^{-1}	6.42×10^{-1}	5.20×10^{-5}	2.05×10^{-2}	7.44×10^{-1}	3.85×10^{-1}
{2, 3, 4}	2.69×10^{-1}	6.25×10^{-1}	5.03×10^{-5}	1.90×10^{-2}	7.86×10^{-1}	3.86×10^{-1}
average	3.04×10^{-1}	6.39×10^{-1}	5.22×10^{-5}	1.90×10^{-2}	7.44×10^{-1}	3.19×10^{-1}

Table 13: Effect of different u on existing algorithms (Time (s))

and the user is asked to select their favorite one among them. We simulate how the user decides which point to select among the s displayed points with a persistent error rate θ as follows: For each pair of displayed points, we compare them with a persistent error rate θ. The point that wins in the most number of comparisons is selected. If multiple points share the highest number of wins, we arbitrarily select one of them. For each pair of selected/non-selected points, we can use it to construct a halfspace using the technique in Section 4.1, and update the related data structures.

We summarize the results in Figure 24. From Figure 24 (a), when s is increased from 2 to 10, the accuracies of our algorithms slightly increases. This is because the selected point is likely to be ranked highly in the displayed points, thus when s grows, the percentage of incorrect halfspaces becomes smaller. In other words, the "effective" error rate is smaller than θ. When s increases, their number of questions decreases since more hyperplanes can be acquired in each round. Similarly, the processing time of these algorithms increases along with s because of the increasing amount of updates on data structures in each round.

B.4.2 Dividing s points into two groups. We adapted our algorithms to display $s > 2$ points in each round and ask the user to divide them into two groups called the *superior group* and the *inferior group*, where the superior (resp. inferior) group contains points with higher (resp. lower) utility scores. To simulate the dividing process with a persistent error rate θ, we first make pairwise comparisons on all pairs of s displayed points, each with a persistent error rate θ, and sort the points in decrease order of the number of comparisons in which they win. Then, we uniform-randomly pick a number i in range $[1, s-1]$, put the first i points into the superior group, and the rest $s-i$ points into the inferior group. Note that we assume each group has at least one point, otherwise, all points will belong to the same group and there is little information from this round. For each distinct pair of superior/inferior points, we can create a halfspace using the technique described in Section 4.1.

Figure 25 shows the algorithm performance when s varies from 2 to 10. When s grows, since more halfspaces can be obtained in each round, the number of rounds required by all algorithms tends to decrease. Meanwhile, obtaining more halfspaces means more data structures need to be updated in each round, and thus the processing time increases. Notably, the accuracies of all algorithms increases along with s, which is reasonable since with more points to be considered in each round, our simulation makes it harder to obtain wrong halfspaces.

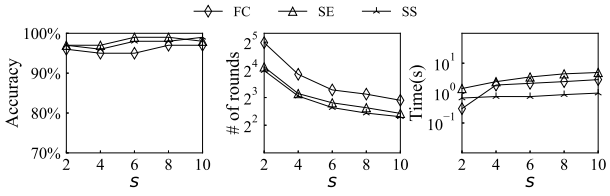


Figure 24: Varying question type: selecting favorite among s displayed points

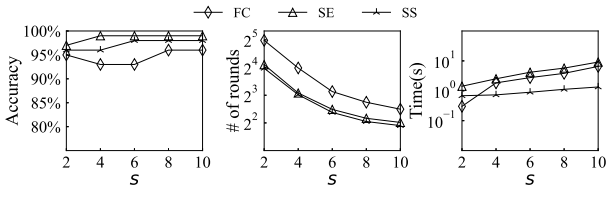


Figure 25: Varying question type: dividing s points into two groups

C RELATED PROOFS

This section details the proofs of lemmas and theorems presented in the main body of the paper.

C.1 Proof of Lemma 1

When the results of all pairwise comparisons are acquired, denote the set of points whose partitions overlap with R^i by RS. Since each pair of points in RS has been compared, there are in total $|RS|\,(|RS|-1)/2$ comparisons. Each comparison yields one loser, so there are $|RS|\,(|RS|-1)/2$ losers. Further, each point in RS cannot lose in more than i comparisons, otherwise, it is not in R^i since its partition is not supported by at least i halfspaces. Therefore, we have $|RS|\,(|RS|-1)/2 \le i\,|RS|$, so $|RS| \le 2i+1$.

C.2 Proof of Lemma 2

In this section, we prove the recurrence relation in Lemma 2 by induction. Firstly, notice that when $t = 0$, confidence regions $R_0^0, R_0^1, \ldots, R_0^k$ all corresponds to the entire utility space, which is trivially correct. Assume that at round t', all confidence regions are correct. Then, at round $t'+1$, when a new halfspace s arrives, $R_{t'+1}^0 = R_{t'}^0 \cap s$, since R^0 is the region supported by all halfspaces.

As for $R_{t'+1}^i$ where $i > 0$, firstly, for any points in $R_{t'+1}^i \cap s$, it is supported by $t'-i$ halfspaces in the first t' halfspaces and also supported by s. Therefore, it is supported by $t'+1-i$ halfspaces and should lie in $R_{t'+1}^i$. With a similar argument, we can also prove that any points in $R_t^{i-1} \cap s^-$ should also lie in $R_{t'+1}^i$. Expect for the above cases, no other points lie in $R_{t'+1}^i$. Therefore, all confidence regions are correct at round $t'+1$, which proves the correctness of the recurrence.

C.3 Proof of Lemma 3

We now show the correctness of the two relations stated in Lemma 3. We first prove that $R_{t+1}^i \subseteq R_t^i$. Firstly, note that by the definition of confidence region, $R_t^{i-1} \subseteq R_t^i$. By Lemma 2, $R_{t+1}^i = (R_t^i \cap s) \cup (R_t^{i-1} \cap s^-)$. Since $R_t^i \cap s \subseteq R_t^i$ and $R_t^{i-1} \cap s^- \subseteq R_t^{i-1} \subseteq R_t^i$, their union $R_{t+1}^i \subseteq R_t^i$ as desired.

Next, we prove that $R_t^i \subseteq R_{t+1}^{i+1}$. By the definition of confidence region, for any point $v \in R_t^i$, there is a set S' of at least $t-i$ halfspaces indicated by the user such that $v \in \bigcap_{s\in S'} s$. Notice that $\bigcap_{s\in S'} s \subseteq R_{t+1}^{i+1}$ by the definition of R_{t+1}^{i+1}. Therefore, we have that for any point $v \in R_t^i$, $v \in R_{t+1}^{i+1}$, thus, $R_t^i \subseteq R_{t+1}^{i+1}$.

C.4 Proof of Theorem 1

In this section, We show the proof of Theorem 1. We first show that there exists a dataset D with the following property:

Property 1. For any $p_i \in D$ and its partition P_i, for any non-empty set of halfspaces $HS \subseteq \{h_{ab}^* | a, b \ne i, * \in \{+,-\}\}$, if the intersection of halfspaces in HS forms a non-empty area, then P_i also overlaps with this area.

Consider a n-dimensional dataset $D_n = \{p_1, p_2, \ldots, p_n\}$ such that for each $p_i, i \in [1,n]$, its i-th dimensional value is 1 and all other dimensions have value 0. Recall that for any two points p_a and p_b, their associated hyperplane is denoted by h_{ab} with its normal $v_{ab} = p_a - p_b$. Further, the halfspace indicated by the user, denoted by s_{ab}, could be either h_{ab}^+ or h_{ab}^-. For ease of illustration, from now on, we assume without loss of generality that $s_{ab} = h_{ab}^+$. Mathematically, $s_{ab} = \{u \in U | v_{ab} \cdot u > 0\}$ where U is the utility space.

Consider any non-empty set of halfspaces HS. Assume that the intersection of halfspaces in HS forms a non-empty region, which we denote by $\mathcal{R}$ (i.e., $\mathcal{R} = \cap_{s\in HS} s$). We show that there is a non-empty intersection between P_i and $\mathcal{R}$. Specifically, let e_i be the vector with $e[i] = 1$ and all other entries $e[j], j \ne i$ being 0. Since the utility of p_i w.r.t. e_i is 1 and the utilities of all other points $p_j, j \ne i$ w.r.t e_j is 0, $e_i \in P_i$ by the definition of P_i. Further, for any $s_{ab} \in HS$, we have $e_j \cdot v_{ab} = e_j \cdot (p_a - p_b) = 0$. Now consider a vector $r \in \mathcal{R}$ and a small positive number λ. Construct $e' = e_i + \lambda r$. Given a small enough λ, we have $e' \in P_i$ since $e_i \in P_i$. Further, $e' \in \mathcal{R}$ since for any $s_{ab} \in HS$, we have $e' \cdot v_{ab} = (e_i + \lambda r) \cdot v_{ab} = \lambda r \cdot v_{ab} > 0$. Since e' is both in P_i and $\mathcal{R}$, we conclude that P_i and $\mathcal{R}$ have a non-empty intersection.

Given a dataset D with Property 1, and given a partition P_i that currently belongs to a confidence region R^j, there are only two ways to detach P_i from R^j, which we call them the *direct way* and the *indirect way*. One can verify these are indeed the only way to detach P_i from R^j given Property 1.

The direct way. P_i is detached from R^j if the user prefers another point p_h to p_i.

The indirect way. P_i is detached from R^j if R^j becomes empty.

By Lemma 3, if a partition is detached from R_t^j in round t, it will belong to R_{t+1}^{j+1} in round $t+1$. Since each partition initially belongs to R^0 and needs to be detached from R^k (where $k = \left\lfloor \frac{l-1}{2} \right\rfloor$) before it is discarded, we conclude that to discard a partition, it needs to be detached for $k+1$ times in total. The algorithm stops only when at most l points are not discarded. Since the direct way detaches one partition from some confidence region at each time, if partitions are only detached using the direct way, then $\Omega(kn)$ rounds are needed. It remains to bound the number of rounds required if the indirect way is used. To bound this number, we first present the following lemma.

Lemma 8. *There exists a dataset such that for any question-asking strategy, the i-th confidence region $R^i = \emptyset$ if and only if there are $i+1$ disjoint cycles.*

Proof. The "if" part is already proved in Lemma 4. For the "only if" part, we prove the following equivalent statement: There exists a dataset such that if there are no $i+1$ disjoint cycles, then the i-th confidence region $R^i \ne \emptyset$. To see this, we use the same dataset D_n constructed above. Assume that the user has made t comparisons. Since there are at most i disjoint cycles in these comparisons, by selectively removing i of them, the remaining $t-i$ comparisons will have no cycle. In other words, the remaining $t-i$ comparisons conform with some ranking on the points involved in these comparisons. Assume without generality that they conform with the ranking $p_1 > p_2 > p_3 > \ldots > p_m$, where $p_1, \ldots, p_m$ are the points involved in these $t-i$ comparisons. We can then construct a utility vector u that yields this ranking. Specifically, let u be constructed such that $u[i] > u[i+1]$ for any $i \in [1, m-1]$. Then, $\forall i \in [1, m-1], u \cdot p_i > u \cdot p_{i+1}$ since $u \cdot p_i = u[i] > u[i+1] = u \cdot p_{i+1}$. This means the $t-i$ halfspaces corresponding to the $t-i$ preferences support a non-empty region, and thus by definition, R^i is also non-empty. □

If we $n-l$ partitions need to be discarded using the indirect way, Lemma 8 tells us that we need k disjoint cycles to make $R^0, R^1, \ldots, R^{k-1}$ empty (Note that R^k cannot be empty, otherwise, there is no point to return.). k user errors are required to form k disjoint cycles. Since the expected number of rounds for the user to make one error is $\frac{1}{\theta}$, we need at least $\Omega(\frac{k}{\theta})$ rounds to accumulate k errors. After obtaining k disjoint cycles, all partitions are either discarded or belong to R^k. Since each question detaches at most 1 partition from R^k, We still need an extra number of $\Omega(n)$ rounds to detach partitions from R^k until only l of them are left. Therefore, the total expected number of rounds using the indirect way is $\Omega(\frac{k}{\theta}+n)$. Since the number used by the direct way is $\Omega(kn)$ and $\Omega(\frac{k}{\theta}+n) < \Omega(kn)$, and $k = \left\lfloor \frac{l-1}{2} \right\rfloor$, the lower bound is thus $\Omega(\frac{l}{\theta}+n)$.

C.5 Proof of Lemma 4

In this section, we give proof to Lemma 4. First, we show that the intersection of halfspaces corresponding to a cycle is an empty region. Consider a cycle consists of points $p_1, p_2, \ldots, p_m$, where the user indicated $p_1 \succ p_2, p_2 \succ p_3, \ldots, p_{m-1} \succ p_m$ and $p_m \succ p_1$. A utility vector u that satisfies all these preferences does not exist, since we must have $u \cdot p_1 > u \cdot p_m$ and $u \cdot p_m > u \cdot p_1$. Therefore, the intersection of halfspaces corresponding to a cycle is an empty region.

Since $R^i = \bigcup_{\overline{S} \in \binom{S}{i}} (\cap_{s \in S/\overline{S}} s)$, where $\binom{S}{i}$ is all i-subset of S , if there are $i+1$ disjoint cycles, no matter how we set $\overline{S}$, there is at least one cycle in $S/\overline{S}$, and $\cap_{s \in S/\overline{S}} s$ is an empty region. The union of a set of empty regions is also empty.

C.6 Proof of Lemma 5

The proof comprises two parts. In the first part, we show that when Stage 1 ends, the number of obtained disjoint cycles is at most k. In the second part, we show when Stage 1 ends on Condition (1), the expected number of disjoint cycles obtained is at least $\max(0, k-13)$.

We first prove that when Stage 1 ends, the number of obtained disjoint cycles is at most k. Assume that Stage 1 ends just after round t. We know that $R_{t-1}^{k-1} \neq \emptyset$, otherwise, Stage 1 should ends after round $t-1$. Thus, by Lemma 3, $R_{t-1}^{k-1} \subseteq R_t^k \neq \emptyset$. Given that $R_t^k \neq \emptyset$, by applying Lemma 4, the number of obtained disjoint cycles is cannot exceed k, since otherwise R_t^k will become empty.

We then prove that when Stage 1 ends on Condition (1), the expected number of disjoint cycles obtained is at least $\max(0, k-13)$. To see this, we first show that when Stage 1 ends on Condition (1), the user makes at least k errors with probability at least $1-\frac{1}{k}$. We then show that if at least k errors are made, the expected number of disjoint cycles is at least $k-12$. Combining them together, the expected number of disjoint cycles is at least $(1-\frac{1}{k})\cdot(k-12) \geq k-13$.

To see that the user makes at least k errors with probability at least $1-\frac{1}{k}$, note that when Condition 1 is satisfied, Stage 1 lasts for at least $\max(\frac{k(k-1)}{2}, \frac{3k}{\theta})$ rounds. Let X be the random variable denoting the number of errors the user made after at least $\frac{3k}{\theta}$ rounds. The expected number of errors is $\mu_X = 3k$, and the variance is $\sigma_X^2 = 3k(1-\theta)$. Using Chebyshev inequality, we obtain that $P(X < k) \leq \frac{1}{k}$.

Next, we show that when if the user already makes k errors, the expected number of disjoint cycles is at least $k-12$. Note that when Stage 1 ends on Condition (1), there are at least $m \geq k$ points in PS (recall that PS is the set of points maintained in Stage 1 of FC), and every pair of these points has been compared. We first show that an error does not result in a cycle with probability at most $\frac{2}{m(1-\theta)^2}$ (note that the cycle need not to be disjoint for now). For ease of illustration, we sort these m points in decreasing order of their real utilities and label them from p_1 to p_m (i.e., $u \cdot p_1 > u \cdot p_2 > \ldots > u \cdot p_m$). Consider an error that occurs when comparing two points, p_i and p_j. Each pair of points are equally probable to be this pair. we consider different values of i and j and the situation in each case where this error does not cause a cycle.

- Case $j = i+1$. Since there are $\frac{m(m-1)}{2}$ comparisons in total and only $m-1$ of them satisfies $j = i+1$, this case happens with probability $\frac{2}{m}$. In this case, an error does not cause a cycle with a probability at most 1.
- Case $j = i+2$. This case happens with probability $\frac{2(m-2)}{m(m-1)}$. In this case, if the comparisons between pair (p_i, p_{i+1}) and pair (p_{i+1}, p_j) are both correct, an error in the comparison of (p_i, p_j) will cause a cycle (since this will result in $p_i \succ p_{i+1}$, $p_{i+1} \succ p_j$ and $p_j \succ p_i$). Therefore, in this case, an error does not cause a cycle with a probability at most $(1-(1-\theta)^2)$.
- Case $j = i+3$. This case happens with probability $\frac{2(m-3)}{m(m-1)}$. In this case, the error in the comparison of (p_i, p_j) will cause a cycle if either the comparisons between pair (p_i, p_{i+1}) and pair (p_{i+1}, p_j) are both correct, or the comparisons between pair (p_i, p_{i+2}) and pair (p_{i+2}, p_j) are both correct. Therefore, in this case, an error does not cause a cycle with a probability at most $(1-(1-\theta)^2)^2$.
- ...

Summing all cases up, we conclude that an error does not cause a cycle with a probability at most $\sum_{x=1}^{m-1}(1-(1-\theta)^2)^{x-1} \cdot \frac{2(m-x)}{m(m-1)}$. Since $\theta < \frac{1}{2}$, this quantity is upper bounded by $\frac{8}{m}$.

Next, we show that the expected number of disjoint cycle we can obtain is at least $k-12$. Note that in the above, we constrain ourselves on cycles that are formed by p_i, p_j, and a point p_a where $i < a < j$. With this construction, two cycles (p_i, p_j, p_a) and $(p_{i'}, p_{j'}, p_{a'})$ are possible not disjoint only if $p_i = p_{i'}$ or $p_j = p_{j'}$. To see this, observe that if $p_i \neq p_{i'}$ and $p_j \neq p_{j'}$, no matter what values a and a' take, the two cycles do not share a common preference. For a given pair p_i and p_j, the expected number of pairs in the remaining $k-1$ errors that contain p_i or p_j is at most 4 (among the remaining $\frac{m(m-1)}{2}-1$ pairs, $2m-1$ of them contain p_i or p_j). This means that on expectation, at most 4 among $(j-i-1)$ points ranked between p_i and p_j will be "occupied" by cycles formed by other errors. Thus, with an argument similar to the above (i.e., discuss different values of i and j), we can show that if y points between p_i and p_j are "occupied" by other cycles, the probability that an error when comparing p_i and p_j does not cause a disjoint cycle is upper bounded by $\sum_{x=1}^{y} \frac{2(m-x)}{m(m-1)} + \sum_{x=y+1}^{m-1}(1-(1-\theta)^2)^{x-y} \cdot \frac{2(m-x)}{m(m-1)} \leq \frac{y}{m} + \frac{2}{m(1-\theta)^2} \leq$

$\frac{y+8}{m}$. Given that $E[y] \leq 4, E[\frac{y+8}{m}] \leq \frac{12}{m}$. Thus, the expected number of disjoint cycles obtained is at least $k - \frac{12}{m}k \geq k - 12$.

C.7 Proof of Theorem 2

In this section, we show the proof of Theorem 2. Firstly, since $l < O(\sqrt{n})$ and $k = \left\lfloor \frac{l-1}{2} \right\rfloor$, Stage 1 of *FC* runs for at most $O(\max(\frac{k(k-1)}{2}, \frac{3k}{\theta})) = O(\frac{l}{\theta} + n)$ rounds. When *FC* enters stage 2, there are 2 cases: (1) Condition 1 is satisfied, and less than k errors are made in Stage 1, and (2) Condition 1 is not satisfied, but Condition 2 is satisfied. Next, we bound the number of rounds required for these 2 cases.

For case (1), by Lemma 5, since Stage 1 ends on Condition (1), the expected number of disjoint cycles is at least $k - 13$. Let k' be the number of disjoint cycles that appear. Then by Lemma 4, confidence regions $R^0, R^1, \ldots, R^{k'-1}$ are all empty, and all remaining partitions are either discarded or belong to $R^{k'}, \ldots, R^{k'}$. Since the expected value of $k - k'$ is a constant, we only need to detach partitions form confidence regions for $O(n)$ times in total. Since each round detaches at least one partition from one confidence region, the expected number of questions required to discard all except l remaining partitions is $O(n)$ (see the proof of Theorem 1 in Section C.4).

For case (2), since Condition (2) is satisfied, R^{k-1} is empty. All partitions are either discarded, or belong to R^k. Since each round detaches at least one partition from one confidence region, Stage 2 ends in $O(n)$ rounds.

Because both cases take $O(n)$ rounds, the expected number of rounds for both cases is $O(n)$. Since Stage 1 takes $O(\frac{l}{\theta} + n)$ rounds and Stage 2 takes $O(n)$ rounds on expectation, the total expected number of rounds for *FC* is $O(\frac{l}{\theta} + n)$.

C.8 Proof of Theorem 3

In this section, we show the proof of Theorem 3. Let $|C_t^i|$ denote the number of convex polytopes in C_t^i. Given the parameter k, we first prove by induction that for any $0 \leq i \leq k$ and any round $t \geq 0$, $|C_t^i| \leq \frac{(t+i)!}{t!\cdot i!}$. Firstly, when $t = 0$, $|C_0^i| \leq 1$ for any $i \in [0, k]$ since it has only 1 convex polytope, which is the entire utility space. Also, for any t, $|C_t^0| = 1$ since R_t^0 is the intersection of all halfspaces, which is a convex polytope. Now assume that at round t', $|C_{t'}^i| \leq \frac{(t'+i)!}{t'!\cdot i!}$ for all $i \in [0, k]$. Then at round $t'+1$, by the updating rules of $C_{t'+1}^i$ we have $|C_{t'+1}^i| \leq |C_{t'}^i| + |C_{t'}^{i-1}|$. Since $|C_{t'}^i| + |C_{t'}^{i-1}| \leq \frac{(t'+i)!}{t'!\cdot i!} + \frac{(t'+i-1)!}{(t')!\cdot(i-1)!} \leq \frac{(t'+1+i)!}{(t'+1)!\cdot i!}$, we conclude that $|C_{t'+1}^i| \leq \frac{(t'+1+i)!}{(t'+1)!\cdot i!}$, as desired.

For each partition, there is at most one sub-partition of it locate in each convex polytope of C_t^i. Since in round t there are $\sum_{i=0}^{k} \frac{(t+i)!}{t!\cdot i!} = O(\binom{t+k}{t}k)$ convex polytopes in total and each convex polytope contains at most n sub-partitions, we need to update $O(\binom{t+k}{t}kn)$ sub-partitions. By [2], given a set of m d-dimensional halfspaces bounding a polytope with v vertices, the vertices of this polytope can be computed in $O(mdv)$ time. Let $|V|$ be the maximum number of vertices in all sub-partitions. Since a sub-partition is bounded by at most $(n+t)$ halfspaces and typically $t \ll n$, updating a sub-partition takes $O(|V|\, dn)$ time. Since $k = \left\lfloor \frac{l-1}{2} \right\rfloor$, the time required to update all sub-partitions is thus $O(\binom{t+k}{t}kn \cdot |V|\, dn) = O(\binom{t+\lfloor (l-1)/2 \rfloor}{t} |V|\, ldn^2)$.

Next, we bound the time complexity for the random-based selection and the score-based selection. for the random-based selection, computing all X_t^is (i.e., the set of partitions belonging to R_t^is) given O_t^is takes $O(\binom{t+k}{t}kn)$ time. Each iteration of the random-based selection takes $O(n^2)$ time (since there are $O(n^2)$ pairs) and there are at most $(k+1)$ iterations, which takes $O(kn^2)$ time in total. Therefore, the total time complexity in each round of *SE-random* is $O(\binom{t+k}{t} |V|\, kdn^2 + \binom{t+k}{t}kn + kn^2) = O(\binom{t+\lfloor (l-1)/2 \rfloor}{t} |V|\, ldn^2)$.

For the score-based selection, computing all X_t^is takes $O(\binom{t+k}{t}kn)$ time. Computing the score for all $O(n^2)$ hyperplane candidates takes $O(n)$ time each. Therefore, each round of *SE-score* takes $O(\binom{t+k}{t} |V|\, kdn^2 + \binom{t+k}{t}kn + n^3) = O(\binom{t+\lfloor (l-1)/2 \rfloor}{t} |V|\, ldn^2 + n^3)$ time.

C.9 Proof of Lemma 6

In this section, we give proof to Lemma 6. When the algorithm ends, if the cell where the real utility vector lies overlaps with R^k, then the best point will not be returned only if there is no sample point inside this cell. Assume that there are ρ cells in the partition of the best point, and the total number of points sampled from this partition is m, further, assume that these ρ cells in this partition are denoted by $c_1, c_2, \ldots, c_\rho$ and have sizes $f_1, f_2, \ldots, f_\rho$ proportional to the total size of this partition, where $\sum_{i=1}^{\rho} f_i = 1$. With no prior knowledge of the distribution of the user's utility vector, the utility vector is assumed to be uniformly distributed. The probability that a cell c_i contains the user's utility vector and has no sample point in it is then $f_i(1-f_i)^m$. Thus, the probability that when *SS* ends, the cell containing the utility vector has no sample point in it is $\sum_{i=1}^{\rho} f_i(1-f_i)^m$, which is further upper bounded by $\sum_{i=1}^{\rho} f_i e^{-f_i m}$. $f_i e^{-f_i m}$ takes the maximum when $f_i = \frac{1}{m}$. Thus, $\sum_{i=1}^{\rho} f_i e^{-f_i m} \leq \frac{\rho}{m} e^{-1}$. Solving $\frac{\rho}{m} e^{-1} = \varepsilon$ yields $m \geq \frac{\rho}{\varepsilon e}$.

C.10 Proof of Theorem 4

In this section, we show the proof of Theorem 4. Let $|V|$ denote the maximum number of vertices in all partitions. Since a partition is bounded by at most n halfspaces, computing all partitions takes $O(|V|\, dn)$ time [2]. By [6], sampling one point from a d-dimensional polytope bounded by m halfspaces can be done in $O(md)$ time. Since each partition is bounded by at most n halfspaces, sampling all Yn points takes $O(Ydn^2)$ time. The time complexity for the preprocessing phase is thus $O((|V| + Y)dn^2)$.

Next, we bound the time complexity in each round of *SS-random* and *SS-score* given the parameter k. For *SS-random*, since the time for updating each sample point is $O(d)$, the time required to update $Q^0, \ldots, Q^k$ is $O(Ydn)$. X_t^is (i.e., the set of partitions belonging to R_t^is) can be determined by scanning through all $P(q)$ for $q \in Q_t^i$ and can be done in $O(Yn)$. The random-based selection takes $O(kn^2)$ time (see Section C.8). Summing them up, the time complexity for each round of *SS-random* is $O(Ydn + kn^2) = O(Ydn + ln^2)$ since $k \leq \left\lfloor \frac{l-1}{2} \right\rfloor$.

For *SS-score*, the time required to compute X_t^is is the same as above (i.e., $O(Ydn)$). The time for determining the next question

is $O(n^3)$ (see Section C.8). Therefore, the time complexity for each round of *SS-score* is $O(Ydn + n^3)$.

C.11 Proof of Theorem 5

We present the proof of Theorem 5 in this section. We first show the bound on algorithm *SE* and *FC*. Given the error rate θ, the return size l and $k = \left\lfloor \frac{l-1}{2} \right\rfloor$, if *SE* and *FC* terminates in T rounds, the expected number of error the user makes is θT. Since the best point is guaranteed to be returned if the user does not make more than k errors, the probability that the best point is not returned is upper-bounded by the probability that the user makes more than k errors. Using Chernoff inequality, this quantity is upper bounded by $\frac{e^{k-\theta T}}{(\frac{k}{\theta T})^k} = \frac{e^{\lfloor (l-1)/2 \rfloor - \theta T}}{(\frac{\lfloor (l-1)/2 \rfloor}{\theta T})^{\lfloor (l-1)/2 \rfloor}}$. The probability that the best point is returned is thus at least $1 - \frac{e^{\lfloor (l-1)/2 \rfloor - \theta T}}{(\frac{\lfloor (l-1)/2 \rfloor}{\theta T})^{\lfloor (l-1)/2 \rfloor}}$.

The bound on *SS* is almost identical, with the difference that even if the user makes no more than k errors, there is still a probability at most ε that *SS* cannot return the best point since there is no sample point in the intersection of R^k and the partition of the best point. Using union bound, the probability that *SS* cannot return the best point is at most $\varepsilon + \frac{e^{\lfloor (l-1)/2 \rfloor - \theta T}}{(\frac{\lfloor (l-1)/2 \rfloor}{\theta T})^{\lfloor (l-1)/2 \rfloor}}$, which completes the proof.