Robust Best Point Selection under Unreliable User Feedback

Qixu Chen
The Hong Kong University of Science and Technology
Kowloon, Hong Kong
qchenax@connect.ust.hk

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
Kowloon, Hong Kong
raywong@cse.ust.hk

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 more generalized 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) an algorithm that asks an even smaller number of questions empirically, with provable performance guarantee. 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.

PVLDB Reference Format:
Qixu Chen and Raymond Chi-Wing Wong. Robust Best Point Selection under Unreliable User Feedback. PVLDB, 17(11): 2681 - 2693, 2024.
doi:10.14778/3681954.3681955

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/qixuchen/PersistErr.

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 [25, 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, 43, 46] 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, 34, 37, 41, 43, 44, 46] 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 [2]. 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

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3681955

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 [45].

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 questions. We focus on a type of question that is widely adopted [31, 34, 37, 41, 43, 44, 46], which is to display two points from D and asks the user to select the preferred point. Due to the simplicity of this question type, the techniques developed for this question type can be easily extended to other types of questions (e.g., displaying more than two points in each question and asking for the favorite one). In our technical report [8], we show how the proposed algorithms can be applied to other question types as well.

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, 20] 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 [17, 20, 24] are caused by cognitive biases or other sources of interference, and the answer to the same question may be *consistently* wrong. For example, when comparing a car priced at \$5000 and another car originally priced at \$10000 but currently offered at a 50% discount, despite the former having slightly better specifications and offering higher utility, Alice may consistently choose the latter. This decision is influenced by the psychological phenomenon known as the "anchoring bias", wherein individuals are swayed by the allure of discounts [48]. 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, algorithms in this line of research, such as *Active-Ranking* [20] and *Pref-Learn* [34], ask more than 60 questions when the input dataset is large (i.e., 1 million points), which is undesirable. Users will lose the plot and get frustrated if they need to answer excessive questions [26, 35].

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. Compared to [7], our algorithms are more generalized and practical since they effectively handle both types of errors in a unified way. This advancement is attributed to a novel and previously unexplored geometrical concept proposed in this paper called the *confidence region*. The confidence region exhibits several interesting properties when dealing with user errors, and we leverage these properties in the design of our algorithms.

Contributions. We summarize our contributions as follows. Firstly, we study the *interactive best point retrieval* problem considering both persistent errors and random errors. Secondly, we show a lower bound on the number of questions needed (also called *round complexity*) for our problem. Thirdly, we study a novel geometrical concept called the *confidence region*, which is instrumental in effectively handling both types of errors. Fourthly, we propose (1) an algorithm with asymptotically optimal round complexity; and (2) an algorithm with even better empirical performance. Both algorithms return the best point with provable guarantee. 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 describe the general framework of our algorithms. We present the details of proposed algorithms in Section 5 and show 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, various other preference-based queries have been proposed. The k-nearest neighbors query (kNN) [39] returns k points closest to a user-provided example point based on Euclidean distances. Similarly, the similarity query [38] uses a complex distance function to find the k closest points. The problem with these two queries is that an example point must be provided by the user in advance, which can be demanding. Some recent studies aim to combine the ideas of top-k and skyline. [10, 29, 30] consider returning a fixed number of points to the user whose preference is estimated to lie in a region of the function space. However, if this region is large, the output size must also increase to guarantee the retrieval of high-quality points. In the worst case, it degenerates to the original skyline query. [1, 9, 32] propose k-regret minimizing query, which computes a set such that for any utility function, there exists a point in the set whose *regret ratio* $\leq \epsilon$, a user parameter. However, to guarantee a small ϵ, the output size could be large for some datasets (e.g., > 1000 points when $\epsilon = 0.5\%$ [1]).

Interactive queries learn the user's preferences through interaction and return points based on these preferences. [28] introduces the interactive skyline query, which reduces skyline size by learning the user's relative skyline importance. The user is asked to partition points into *superior* and *inferior* groups. However, the output size remains uncontrolled even with complete attribute preference. The interactive similarity query [3, 4, 37] learns a distance function from user interactions to find the k points closest to a query point. However, it requires users to assign *relevant scores* to hundreds of points, which 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 [31, 34, 37, 41, 43, 44, 46]. [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, [46] introduces *UH-Simplex* and *UH-Random*, which only display points inside the database. [43] 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], which proposes the *Verify-Point* and *Verify-Space* algorithms to find the best point considering *random user errors.* However, these methods have several drawbacks. Firstly, they handle random errors through majority votes on repetitive questions, which fail against *persistent* errors, as the feedback to the same question will be the same. Our experiments show their accuracy drops by over 10% when addressing persistent errors compared with addressing random errors. Additionally, repeating the same question is not suitable for many applications. For example, Alice would be confused if asked to compare the same pair of cars three times. They also display artificial points during interactions. Our algorithms overcome these limitations.

In the field of machine learning (ML) and information retrieval (IR), the robustness to erroneous user input is considered in some algorithms [12, 13, 20, 21, 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. [20] 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. [12] 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 to existing work, our approach has several advantages. Firstly, we do not require prior knowledge of the user's utility function and ensure a small return size, addressing the limitations of traditional top-k and skyline queries. Secondly, we minimize user effort by requiring only a few simple questions, unlike some algorithms that ask many (e.g., [13, 20, 34]) or difficult questions (e.g., [4, 28]). Finally, our algorithms accommodate potentially unreliable

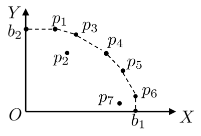


Figure 1: The upper hull

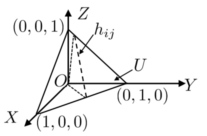


Figure 2: Utility space in 3d

user feedback without significantly impacting the quality of the results, offering more practical value than approaches that assume perfect user feedback (e.g., [31, 43, 46]).

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 1. 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. Since the number of attributes considered in human decision-making is typically limited, similar to [7, 41, 43], we focus on the case where d is not too large (e.g., $d \leq 7$), although the developed techniques can be applied to any d. 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, 43, 46, 47], 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 [31, 34, 37, 41, 43, 46] 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 $p_j \succ p_i$, and we assume that the user makes an error when comparing each pair of points with an error rate *at most* θ. An error is called a *persistent error* if the answer to the same pair of points is *consistently wrong*; otherwise, it is a *random error.* Notably, persistent errors are harder to be handled compared to random errors. This is because when a user has a random error, if the same question involving two points are asked

Table 1: Commonly-used Symbols

D	the input dataset
d	the dimensionality of D
n	size of $uhull(D)$
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 return size
t	number of rounds
R^i (R^i_t)	the i-th confidence region (just after round t)
X^i (X^i_t)	set of partitions belonging to R^i (R^i_t)

to him/her *multiple* times, it is possible that s/he could indicate the correct preference in one of the questions, giving a chance for the system to know the inconsistency from the user for the same question implying a possible error. But, when a user has a persistent error, if the same question is asked multiple times, it is not possible that s/he could indicate the correct preference, which means that this increases the difficulty of inferring his/her preference. 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. In practice, θ can be set to a reasonable number larger than the real error rate, and the exact value of the real error rate need not be known. According to [7, 23], θ can be naturally assumed to be less than 0.5 for reasonable users. In their studies, θ is at most 5%. Following [14–16, 18, 19, 22, 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 ALGORITHM FRAMEWORK

In this section, we present the general framework of our proposed algorithms. We first introduce some useful concepts in Section 4.1. Then, in Section 4.2, we present the algorithm framework and prove the lower bound on the required number of rounds for our problem.

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 $B = \{b_i | 1 \leq i \leq d\}$ 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 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, 46]. 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.

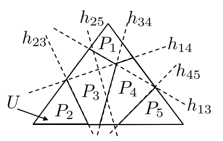


Figure 3: Example on partitions

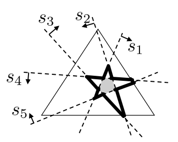


Figure 4: Example on confidence regions

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, 43, 46, 47] 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 an 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 our technical report [8]. □

Corollary 1. *Let t_l be the number of questions required such that R^k overlaps with at most l partitions. If $k \leq \left\lfloor \frac{l-1}{2} \right\rfloor$, then t_l is at most the total number of possible questions. In practice, t_l is much smaller than the total number of possible questions.*

If $k \leq \left\lfloor \frac{l-1}{2} \right\rfloor$, Corollary 1 suggests that t_l is at most the number of all possible questions. In practice, t_l is typically much smaller than the number of all possible questions (often less than 30 in our experiments). In the rest of this paper, we set $k = \left\lfloor \frac{l-1}{2} \right\rfloor$ unless otherwise specified.

4.2 The General Framework

In this section, we introduce the general framework of our proposed algorithms. We maintain $k + 1$ confidence regions, namely $R^0, R^1, ..., R^k$ (Later in Section 5, we will see that we need not maintain the real R^is. Instead, they are only maintained *conceptually*). Note that R^i in Lemma 1 (and R^k in Corollary 1) is the final R^i (R^k) after some questions 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 $t \geq 0$, 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 region 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 6, 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 $t \geq 0$, 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 ($i \geq 1$) 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 shrink 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_t^i 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_2^0 is the gray region and R_2^1 is the area bounded by bold lines. Since both P_1 and P_3 overlaps with R_2^0, we knot that both P_1 and P_3 (and thus p_1 and p_3) belong to R_2^0. P_2 overlaps with R_2^1 but not R_2^0, so it belongs to R_2^1. In the next round ($t = 3$), consider the updated R_3^0 and R_3^1 in Figure 6, where R_3^0 is the gray region and R_3^1 is the area bounded by bold lines. Since P_3 no longer overlaps with R_3^0, it is detached from R_3^0 and belongs to R_3^1 instead. P_2 is detached from R_3^1 and is thus discarded.

Stopping Condition. The algorithm stops when the number of partitions overlapping with R^k is at most l (i.e., $\left|\bigcup_{i=0}^{k} X^i\right| \leq l$). 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 to reach the stopping condition.

Theorem 1. *Given an input size n, a parameter l and an error rate upper bound θ. There exists a dataset such that any question-asking strategy must ask* $\Omega(\frac{l}{\theta} + n)$ *questions to discard at least* $n - l$ *points.*

Proof sketch. We first prove that there exists a dataset D such that the following property holds for any $p_i \in D$ and its partition P_i: For any non-empty set of halfspaces $HS \subseteq \{h_{ab}^{*} | a, b \neq i, * \in \{+, -\}\}$, if the intersection of halfspaces in HS forms a non-empty region, then P_i also overlaps with this region. This property implies that, 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 halfspace indicated in this round must satisfy one of the following: (1) This halfspace is related to p_i and some other point p_j, and p_i is less preferred to p_j; or (2) This halfspace makes R^m an empty region. We then show that to discard at least $n - l$ partitions, 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 $\Omega(\frac{l}{\theta} + n)$. □

5 THE *SS* AND *FC* ALGORITHMS

In this section, we introduce two algorithms, namely *SS* (*Shape-Sampling*) (Section 5.1), and *FC* (*FindCycles*) (Section 5.2), which are developed based on the framework described in Section 4.2. These algorithms differ primarily in the strategies they use to choose questions in each round. For *SS*, we propose two (sub-)strategies that are empirically demonstrated to require only a small number of questions. On the other hand, *FC* employs a strategy that is shown to have an asymptotically optimal round complexity.

5.1 The *SS* Algorithm

In this section, we introduce *SS* by addressing two sub-problems, namely (1) what data structures are required and how to update them in each round, and (2) how to design question selection strategies such that a small number of questions is required.

5.1.1 Data Structure Maintenance. While it is possible to directly compute the exact shapes of confidence regions (i.e., R^is), updating these non-convex regions can be computationally expensive when d increases. However, it is worth noting, as discussed in Section 4.2, that the stopping condition relies solely on the sets of partitions belonging to each confidence region (i.e., X^is). As long as X^is can be obtained, maintaining the exact shapes of 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 (and thus the name *Shape-Sampling*). By sacrificing a small degree of accuracy in finding the best point, *SS* achieves an efficient processing time, as demonstrated in Section 6.

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. In the running phase, *SS* maintains $k + 1$ sets, namely $Q^0, Q^1, ..., Q^k$, where Q^i stores the set of sample points that falls in R^i where $i \in [0, k]$. We use Q_t^i to denote Q^i just after round t.

Next, we describe how to maintain Q_t^is and X_t^is in the running phase. Initially, when $t = 0$, since all partitions overlaps with $R_0^0, R_0^1, \ldots, R_0^k$, each of $Q_0^0, Q_0^1, \ldots, Q_0^k$ stores the entire set of sample points. In round t, let s be the halfspace indicated at this round, the empty-initialized Q_t^i is constructed using Q_{t-1}^i and Q_{t-1}^{i-1} as follows: (1) For each point $q \in Q_{t-1}^i$, q is inserted to Q_t^i if $q \in s$; (2) If $i > 0$, for each point $q \in Q_{t-1}^{i-1}$, q is inserted to Q_t^i if $q \notin s$. It is easy to verify using Lemma 3 that with the above updating rules, each Q_t^i stores the sample points in R_t^i. After Q_t^is are obtained, with the assumption that the number of sampled points is adequately large (to be discussed next), X_t^is can be determined as follows: (1) When $i = 0$, $X_t^i = \{P(q) | q \in Q_t^i\}$; (2) When $i > 0$, $X_t^i = \{P(q) | q \in Q_t^i\} / \{P(q) | q \in Q_t^{i-1}\}$. *SS* stops when the size of $\bigcup_{i=0}^{k} X_t^i$ is at most l, and it returns the points that correspond to the partitions in $\bigcup_{i=0}^{k} X_t^i$.

To illustrate *SS*, we use Figure 5 as a running example. Assume that $l = 3$ and $k = 1$, and the number of sample points is sufficiently large. Initially, Q_0^0 and Q_0^1 each contains a set of all sample points. We have $X_0^0 = \{P_1, P_2, P_3, P_4, P_5\}$ and $X_0^1 = \emptyset$. After the user indicated h_{34}^+ and h_{14}^+ in the first 2 rounds, Q_2^0 contains all sample points in R_2^0 (i.e., the gray region) and Q_2^1 contains all sample points in R_2^1 (i.e., the area bounded by bold lines). Then, $X_2^0 = \{P_1, P_3\}$ and $X_2^1 = \{P_2\}$. Since $\left|\bigcup_{i=0}^{k} X_2^i\right| = |\{P_1, P_2, P_3\}| = 3 \leq l$, *SS* stops and returns $\{p_1, p_2, p_3\}$.

Since *SS* uses sample points to decide X^is, when it terminates, 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 4 decides a sufficient number of samples which guarantees that this case happens with a probability at most a user parameter ε.

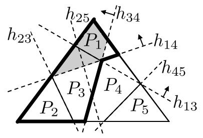


Figure 5: Confidence regions before asking h_{13}

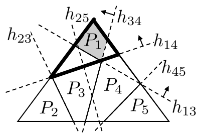


Figure 6: Confidence regions after asking h_{13}

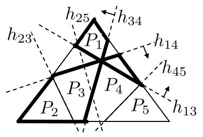


Figure 7: *FC* example just after round 3

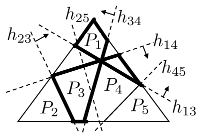


Figure 8: *FC* example just after round 4

Lemma 4. *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 $Y = O(\frac{n^d}{\varepsilon})$ [27], in practice, Y can be set by gradually increasing it until the accuracy of returning the best point converges, and thus, Y could be set to a much smaller number. In our experiments, on a 4-d dataset with 100000 points, by sampling 1000 points from each partition, the accuracy already converges, indicating a small accuracy loss caused by sampling.

5.1.2 Question Selection Strategies.

Next, we describe how *SS* chooses 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 $i' \leq i$. 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 in previous questions, 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 4.2 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, 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, we design the score-based selection to further reduce the number of questions required. We first introduce a data structure that will be used for this strategy. 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}. L 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 2.

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 between 0 and 1 capturing the relative priority between different X_t^is. In practical applications, α can be determined by conducting a grid search between 0 and 1 and finding the value that minimizes the required number of rounds. We set $\alpha = 0.2$ in this paper based on the empirical results in Section 6.2. 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 table L corresponding to this stage is shown in Table 2. Based on this table, p_1 and p_3 will be displayed to the user in the next round since h_{13} obtains the highest score.

Based on the question selection strategy applied, there are two variants of *SS*, which are called *SS-random* and *SS-score*, respectively. The time complexities for each round of these two variants are presented in Theorem 2.

Theorem 2. *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 $O(Yldn)$ and $O(Yldn + n^3)$, respectively.*

Table 2: Table L

	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

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]. In each round of the running phase, the time required to update Q^is and compute X^is is $O(Yldn)$, and the time required to decide the next question for *SS-random* and *SS-score* are $O(t)$ and $O(n^3)$, respectively, where t is the current number of rounds. Since typically $t \ll Ydn$, 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, $m \ll n$ 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, 43, 46].

Lastly, Theorem 3 bounds the probability that the best point is returned by *SS*. Intuitively, when $l = 10$, $\theta = 0.05$, $T = 20$ and $\varepsilon = 0.01$, Theorem 3 guarantees that *SS* find the best point with probability at least 82%. Note that this is a loose bound. In our experiments, if we set $l = 10$, our algorithms return the best point with probability at least 99%.

Theorem 3. *Given an error rate upper bound θ, a return size l, and a parameter ε as defined in Lemma 4. If SS terminates in T rounds, then the probability that the best point is returned by SS is at least* $1 - \varepsilon - e^{-\frac{(\lfloor (l-1)/2 \rfloor - \theta T)^2}{\lfloor (l-1)/2 \rfloor + \theta T}}$.

Proof sketch. *SS* does not return the best point only if either (1) there is no sample point in the cell containing the real utility vector; or (2) the cell containing the real utility vector does not overlap with R^k. By Lemma 4, (1) happens with probability at most ε. By Chernoff inequality, (2) happens with probability at most $e^{-\frac{(\lfloor (l-1)/2 \rfloor - \theta T)^2}{\lfloor (l-1)/2 \rfloor + \theta T}}$. Summing them together completes the proof. □

5.2 The *FC* Algorithm

Although *SS* typically requires only a small number of questions in practice, it does not have a guarantee on the number of questions needed before termination. In this section, we introduce our second algorithm *FC*, which has an asymptotically optimal number of rounds. We first introduce an important concept called *cycle*.

Definition 2. *A set of $m \geq 3$ 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 are no two points, 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 5 reveals the relation between the occurrences of cycles and confidence regions.

Lemma 5. *If there are $i + 1$ disjoint cycles, then the i-th confidence region $R^i = \varnothing$.*

Proof sketch. We first show that the intersection of halfspaces corresponding to a cycle is an empty region. Since $R^i = \bigcup_{S' \in S_{|S|-i}} (\bigcap_{s \in S'} s)$, where S' has a cardinality of $|S| - i$, if there are $i + 1$ disjoint cycles, no matter how we set S', $\bigcap_{s \in S'} s$ 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. Similar to *SS*, *FC* only keeps confidence regions conceptual and utilizes the same sampling technique described in Section 5.1.1 to maintain X^is, i.e., the sets of partitions belonging to each confidence region. After each round, it checks if it stops by checking if $\left|\bigcup_{i=0}^{k} X^i\right| \leq l$. In Stage 1, *FC* adopts a question selection strategy that is different from the two strategies described in Section 5.1.2. This strategy attempts to quickly obtain disjoint cycles (if cycles appear during interaction), 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 6 that at most k disjoint cycles are obtained. Let j be the number of disjoint cycles obtained just after Stage 1 (where $j \leq k$). By Lemma 5, 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. In this stage, *FC* chooses questions following either the random-based selection or the score-based selection in Section 5.1.2, until the stopping condition is met.

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) $\bigcup_{i=0}^{k-1} X^i = \varnothing$. Lemma 6 states several important properties when Stage 1 ends.

Lemma 6. *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 $\max(0, k - 17)$.

Proof sketch. To prove Property 1, assume that Stage 1 ends after round t. Since it does not end after round $t - 1$, $\bigcup_{i=0}^{k-1} X^i_{t-1} \neq \varnothing$, it follows that $R^{k-1}_{t-1} \neq \varnothing$. By Lemma 3, $R^{k-1}_{t-1} \subseteq R^k_t \neq \varnothing$. Thus, the number of disjoint cycles must not exceed k (Lemma 5). To prove Property 2, we 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 $k - 16$. The expected number of obtained disjoint cycles is thus at least $(1 - \frac{1}{k})(k - 16) \geq k - 17$. □

FC then enters Stage 2, in which it needs to further discard the remaining partitions. To do so, *FC* follows either the random-based selection or the score-based selection in Section 5.1.2 to select a pair of points and display them to the user in each round. The algorithm terminates when at most l partitions remain.

To illustrate *FC*, assume that $l = 3$ and $k = 1$, and in the first 3 rounds of Stage 1 the user indicated $p_1 \succ p_3$, $p_3 \succ p_4$ and $p_4 \succ p_1$, which forms a cycle. The related halfspaces h_{13}^+, h_{34}^+ and h_{14}^- and the resulting confidence regions are shown in Figure 7, where R_3^0 is an empty region and R_3^1 is the union of the three regions bounded by bold lines. Since $X_3^0 = \emptyset$ and $X_3^1 = \{P_1, P_2, P_3, P_4\}$, Stage 1 ends because Condition (2) is satisfied. After the user indicated h_{23}^- in round 4, the resulting R_4^1 is the union of the three regions bounded by bold lines shown in Figure 8. Since $X_4^0 = \emptyset$ and $X_4^1 = \{P_1, P_3, P_4\}$, $\left|\bigcup_{i=0}^{k} X_4^i\right| = |\{P_1, P_3, P_4\}| = 3 \leq l$. *FC* stops and returns $\{p_1, p_3, p_4\}$.

Theorem 4 presents the major conclusion of *FC*.

Theorem 4. *Given an input size n, an output size l and an error rate upper bound θ, 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 Stage 1 finishes within $O(\frac{l}{\theta}+n)$ rounds as long as $l = O(\sqrt{n})$. 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.*

Lastly, Theorem 5 bounds the probability of *FC* returning the best point.

Theorem 5. *Given an error rate upper bound θ, a return size l, and a parameter ε as defined in Lemma 4. If FC terminate in T rounds, then the probability that the best point is returned by FC is at least* $1 - \varepsilon - e^{-\frac{(\lfloor (l-1)/2 \rfloor - \theta T)^2}{\lfloor (l-1)/2 \rfloor + \theta T}}$.

Proof sketch. The proof is nearly identical to the proof of Theorem 3. □

6 EXPERIMENT

6.1 Experimental Setup

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 the technical report [8]. 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.

Algorithms. We compare our proposed algorithms, namely *FC*, *SS-score* and *SS-random*, against the competitor algorithms, including (a) algorithms that do not consider user errors, namely *HD-PI* [43], *UH-Simplex* [46] and *UtilApprox* [31], and (b) algorithms that consider user errors, namely *Verify-Point* [7], *Active-Ranking* [20] 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, 43], 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 2 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 [23], is a reasonable upper bound for the human error rate. According to the results in Section 6.2, 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. We report the processing time per question since compared to the total processing time, it is a more informative metric for evaluating the algorithm's responsiveness during user interaction. Each setting is repeated 100 times and the average value is reported. In each

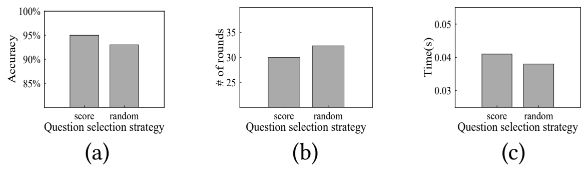


Figure 9: Effect of question selection strategies on *FC*

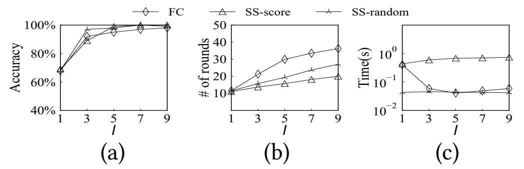


Figure 10: Effect of l (and k)

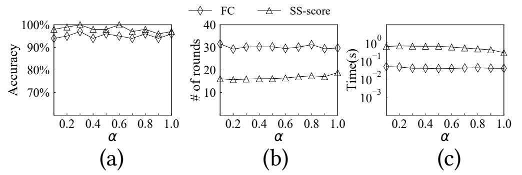


Figure 11: Effect of α

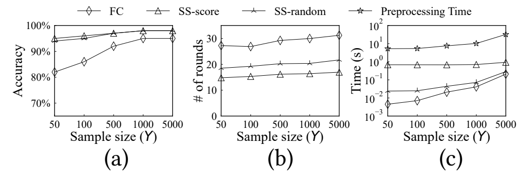


Figure 12: Effect of sample size

repetition, we randomly sample a vector u from the utility space as the underlying utility vector, and the point with the highest utility with respect to u in the dataset is regarded as the real best point.

The rest of the paper is organized as follows. In Section 6.2, we analyze the impact of various parameters on the performance of our algorithms. We then present the experimental results on synthetic datasets (Section 6.3) and real datasets (Section 6.4). The findings from a user study are discussed in Section 6.5. Finally, we provide a summary of the experiments in Section 6.6.

6.2 Experiments on Parameter Setting

In this section, we evaluate the impact of different parameter settings, including l, α and Y, on our algorithms.

We studied the effect of using the score-based selection and the random-based selection in Stage 2 of *FC*. Figure 9 shows the results. We observe that the score-based selection obtains a higher accuracy and requires fewer questions, while the processing time of the random-based selection is slightly faster. Since the increase in accuracy and round efficiency is considered more important than a small gain in processing time, we chose the score-based selection as the default question selection strategy for Stage 2 of *FC*.

Figure 10 analyzes the impact of varying the value of l from 1 to 9 on our algorithms. According to Figure 10 (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.

In Figure 11, we varied α from 0.1 to 1.0 to study its impact on *FC* and *SS-score* (note that α is a parameter in the score-based selection so it does not affect *SS-random*, which uses the random-based selection). 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 both algorithms. In Figure 12, we studied the influence of increasing parameter Y from 50 to 5000 on the performance of *FC*, *SS-score* and *SS-random*. As Y increases, all algorithms exhibit improved accuracies, a higher number of questions, and a longer processing time. The time required by the pre-processing phase also increases. We chose $Y = 1000$ as it provides a satisfactory level of accuracy while maintaining a fast processing time.

6.3 Experiments on Synthetic Datasets

Figure 13 summarizes our study on the algorithms' performance when 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 similar 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 fraction of persistent errors increases. Since persistent errors are harder to be handled, we assume all user errors are persistent for the rest of this section.

In Figure 14, we compare the performance of our algorithms (*FC*, *SS-score*, and *SS-random*) with existing methods on 4-d synthetic datasets of varying sizes (from 100 to 100 million). As shown in Figure 14 (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 (100 million). Among our algorithms, *SS-score* is the most round-efficient, asking at most 10 more questions compared to the most round-efficient one (*HD-PI*). *SS-random* asks slightly more questions than *SS-score*. However, it scales well on large datasets (100 million points) since it determines the next question within 0.7 seconds.

Figure 15 shows the effect of varying θ from 0 to 0.15. According to Figure 15 (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 16 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 to 8 additional questions per dimensionality increase. Although *SS-score* takes around 10 seconds for $d = 5$ (since finding the best hyperplane in the large table L is time-consuming), the processing time of *SS-random* and *FC* is still within 1 second.

6.4 Experiments on Real Datasets

We compared the performance of our algorithms against the existing methods on 3 real datasets, namely *AirQuality*, *Weather* and

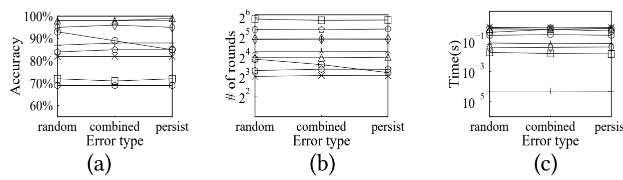


Figure 13: Effect of different types of errors

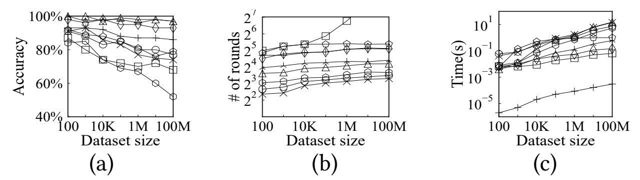


Figure 14: Effect of input size on 4d datasets

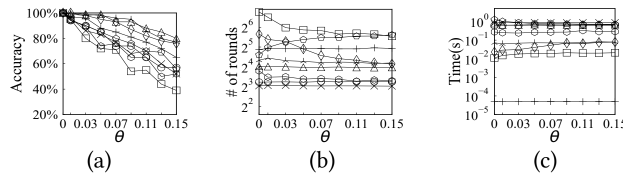


Figure 15: Effect of θ

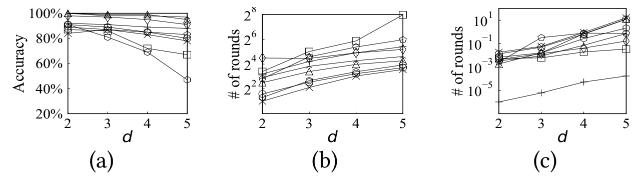
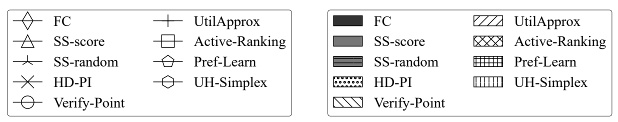


Figure 16: Effect of d



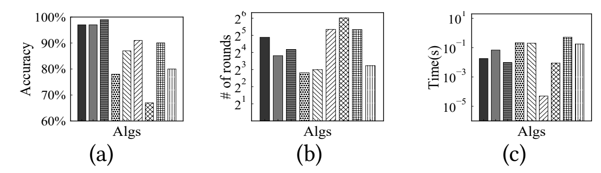


Figure 17: Results on dataset *AirQuality*

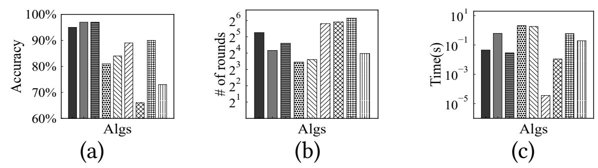


Figure 18: Results on dataset *Weather*

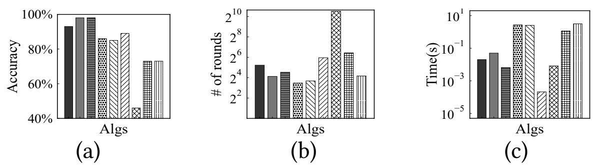


Figure 19: Results on dataset *HTRU*

HTRU. The results are summarized in Figure 17, 18 and 19, respectively. Our methods obtain higher accuracies than existing algorithms on all 3 datasets. In particular, the accuracies of *FC*, *SS-score*, and *SS-random* are consistently above 90%. Although existing algorithms *HD-PI* and *Verify-Point* require 5 to 7 fewer rounds than our most round-efficient algorithm (*SS-score*), their accuracies are 10% to 20% lower than ours, which is not satisfactory. Our algorithm runs at an interactive speed since they process each question within 0.6 seconds on all 3 datasets.

6.5 User Study

We conducted two user studies on a real dataset *Airbnb* [11]. *Airbnb* consists of 6809 Airbnb rentals in Amsterdam with 4 attributes: daily price, cleanliness rating, location rating and the number of reviews. Following [7, 43], we randomly sampled 1000 Airbnb rentals and recruited 25 participants.

(1) The first user study aims to study user errors' impact on algorithm performance and our algorithms' effectiveness. We compared *SS-score* against existing methods, 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, 43]: 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 $l = 5$ and derive that $k = 2$, following our experimental settings. This user study contains 3 parts. In Part 1, the user 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.

Part 2 and Part 3 follow the design of Part 1 with the only difference that in these two parts, each algorithm was required to stop and return a list of at most l options after t rounds. In Part 2, t is set to 10. In Part 3, t is set to the number of rounds required by *SS-score* to terminate (which is typically 12 to 13). Since some algorithms cannot guarantee the size of the returned list when forced to stop, they are re-adapted as follows: (1) For *SS-score*, we randomly return l points whose partition is in $\bigcup_{i=0}^{k} X^i$. (2) For *Verify-Point* and *HD-PI*, we randomly return l points from their candidate sets. (3) For *Active-Ranking*, we return the top-l points in the topological order resulting from the t comparisons. For each part, 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 20 displays the results. In Part 1, *SS-score* achieves over 90% hit rate, significantly outperforming other competitors. The processing time of our algorithm is also short since it determines the next question within 0.01 seconds. Given that *SS-score* obtains a hit rate exceeding 90%, we conclude that the linear utility function approximates the user's preference reasonably well. In Part 2, the hit rate of *SS-score* is lower than *HD-PI* and *Verify-Point*. This is explainable because there is a trade-off between round efficiency and error handling capacity. Since *SS-score* prioritizes error handling, it is expected that its round efficiency is lower than algorithms that either do not consider error handling or have a lower error handling capacity. Consequently, when forced to stop after round 10, *SS-score* has to randomly select l points from a relatively large number of candidates, resulting in a lower hit rate. However, as can be observed in Part 3, when allowed to use slightly more rounds (i.e., 2 to 3 more rounds), the hit rate of *SS-score* exceeds other baselines by a large margin, aligning with its performance in Part 1.

Based on the results obtained in Part 1, we estimated how frequently users make errors using algorithm *SS-score*, using the properties that if Y is large enough, the best point belongs to the i-th confidence region only if at least 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 decides the confidence region this point belongs to. The user error rate can be estimated 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 272 questions asked by *SS-score*, users made 16 errors, resulting in a 5.8% empirical error rate.

(2) Our second user study aims to show the presence of persistent user errors in the real-world scenarios. In this user study, each Airbnb has 5 attributes: labeled price, discount, cleanliness rating, location rating and the number of reviews. For brevity, we will refer to cleanliness rating, location rating, and the number of reviews as the "other attributes" throughout this section. The term *labeled price* represents the original price before the discount, while the *final price* refers to the price after the discount has been applied (i.e., final price = labeled price $\times$(100%$-$ discount)). For example, given an option with a labeled price of \$300 and a 20% discount, its final price is \$300 $\times$ (100% $-$ 20%) = \$240. We have a hypothesis that some people in the world have the (wrong) impression that the final price *could* be low when they see a high discount rate. We could regard this impression as persistent errors in our user study. The user study consists of 3 settings. In Setting 1, we display all 5 attributes for each Airbnb, and the utility function is assumed to be linear w.r.t. all 5 attributes. In Setting 2, we still display all 5 attributes for each option, but the utility is assumed to be linear w.r.t. the final price (instead of the labeled price and the discount) and other attributes. Then, in Setting 3, only the final price and other attributes are displayed, and the utility is assumed to be linear w.r.t. the final price and other attributes. In each setting, the user interacts with algorithm *SS-score* until a list of at most l points are returned, and is then asked to select one option from the list as the *selected favorite rental option* of this setting. We refer to the selected favorite rental option of Setting 1, 2 and 3 as p_1, p_2 and p_3, respectively. After obtaining p_1, p_2 and p_3, the user is required to choose one

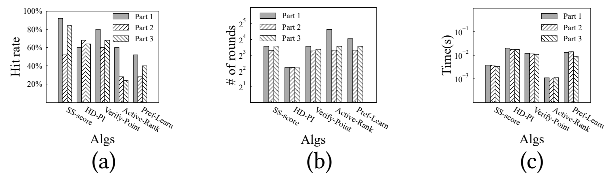


Figure 20: Results on user study

options among them as the best point. Notably, if p_3 is chosen as the best point, it indicates that the user considers the final price, rather than the labeled price and discount, when making their decision. For users who selected p_3 as the best point, an additional question is asked: The user is presented with p_2, including its labeled price, discount, and computed final price, and is asked if they think the final price is higher than their expectation. Note that when the user selected p_2 in Setting 2, only the labeled price and discount of p_2 are presented to the user. Given that the user considers final price when making the decision, if the computed final price of p_2 is higher than the user's expectation, it suggests that the user is prone to persistent errors, possibly due to the false impression that the final price could be low because of the high discount rate. Among the 21 users who selected p_3 as the best point, 33% (7) of them found that the final price of p_2 was higher than expected, indicating that they have a tendency to make persistent errors. Moreover, 24% (5) users found that their best point was not in the list of options recommended by Setting 2, indicating that 24% users missed their best point due to persistent errors.

6.6 Summary

The experiments demonstrated the superiority of our proposed algorithms, namely *FC*, *SS-score* and *SS-random*, 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, outperforming existing algorithms. (2) We are scalable to the input size and dimensionality. On the 7-d dataset *HTRU*, *SS-score* and *SS-random* finish with around 20 questions and achieve over 98% accuracy, but algorithms *UtilApprox* and *Active-Ranking* obtain lower accuracies with more rounds. (3) We are capable of handling many persistent errors. Even with a high error rate (e.g., 0.15), 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 robustly return the user's best point with high confidence, even when the user makes persistent and random errors. We introduce algorithm *FC*, which requires an asymptotically optimal number of rounds, and algorithm *SS*, which empirically requires fewer questions. Both have provable guarantees of returning the best point. Extensive experiments show our algorithms are efficient and effective in handling errors. In the future, we aim to extend our solutions to return the user's top-k points.

ACKNOWLEDGEMENT

We are grateful to the anonymous reviewers for their constructive comments on this paper. This work was supported in part by WEB24EG01-H.

REFERENCES

[1] Pankaj K Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. 2017. Efficient algorithms for k-regret minimizing sets. *arXiv preprint arXiv:1702.01446* (2017).

[2] 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).

[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] Qixu Chen and Raymond Chi-Wing Wong. 2023. Robust Best Point Selection under Unreliable User Feedback (Technical Report). https://github.com/qixuchen/PersistErr/blob/main/PersistError_Techreport.pdf.

[9] Sean Chester, Alex Thomo, S Venkatesh, and Sue Whitesides. 2014. Computing k-regret minimizing sets. *Proceedings of the VLDB Endowment* 7, 5 (2014), 389–400.

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

[11] Airbnb dataset. 2023. http://insideairbnb.com/get-the-data/.

[12] 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.

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

[14] 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).

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] Alchemer LLC. 2023. 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. 2023. 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, H Jagadish, and Min Xie. 2024. Reverse Regret Query. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE.

[43] 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.

[44] Weicheng Wang, Raymond Chi-Wing Wong, and Min Xie. 2023. Interactive search with mixed attributes. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2276–2288.

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

[46] 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.

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

[48] Yi Zong and Xiaojie Guo. 2022. An experimental study on anchoring effect of consumers' price judgment based on consumers' experiencing scenes. *Frontiers in Psychology* 13 (2022), 794135.