Strongly Truthful Interactive Regret Minimization

Min Xie, Raymond Chi-Wing Wong
Hong Kong University of Science and Technology
mxieaa@cse.ust.hk,raywong@cse.ust.hk

Ashwin Lall
Denison University
lalla@denison.edu

ABSTRACT

When faced with a database containing millions of tuples, an end user might be only interested in finding his/her (close to) favorite tuple in the database. Recently, a regret minimization query was proposed to obtain a small subset from the database that fits the user's needs, which are expressed through an unknown *utility function*. Specifically, it minimizes the "regret" level of a user, which we quantify as the *regret ratio* if s/he gets the best tuple in the selected subset but not the best tuple among all tuples in the database.

We study how to enhance the regret minimization query with *user interactions*: when presented with a small number of tuples (which can be artificial tuples or true tuples inside the database), a user is asked to indicate the tuple s/he favors the most among them. In particular, we are also interested in the special case of determining the *favorite* tuple for a user in the entire database with a small amount of interaction, measured by the number of questions we ask the user.

Different from the previous work which displays artificial tuples to users, we achieve a stronger result in this paper by always displaying true tuples in the database. Specifically, we present a generic framework for interactive regret minimization, under which we propose algorithms that ask an asymptotically optimal number of questions in 2-dimensional spaces and algorithms with provable performance guarantees in d-dimensional spaces ($d \geq 2$) where each dimension corresponds to a description of a tuple. Experiments on real and synthetic datasets showed that our algorithms outperform the existing one by locating the favorite tuple and guaranteeing a small regret ratio with much fewer questions.

CCS CONCEPTS

• **Information systems → Data analytics**.

KEYWORDS

regret minimization; user interaction; data analytics

ACM Reference Format:
Min Xie, Raymond Chi-Wing Wong and Ashwin Lall. 2019. Strongly Truthful Interactive Regret Minimization. In *2019 International Conference on Management of Data (SIGMOD '19), June 30 - July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3300068

1 INTRODUCTION

In order to assist the user in finding the tuple s/he is interested in, a database system provides some operators to return a representative subset to the user to fit the user's need. Such operators can be regarded as *multi-criteria decision-making* tools and it can be applied in various domains, including house buying, car purchase and job search. For example, in a house database where each house is described by some attributes, Alice wants to find an inexpensive house with a large size, which is as new as possible (i.e., price, size and house age are some attributes/criteria that Alice would consider when she buys a house). In the literature [7, 22, 23], Alice's preference could be represented by a monotonic preference function, called a *utility function*, in her mind. Based on this function, each house in the database has a *utility*. A high utility indicates that this house is favored by Alice and the house with the highest utility is the *favorite* house of Alice. Depending on whether the utility function is provided to the database, various approaches were researched towards multi-criteria decision-making, including the *top-k query*, the *skyline query* and the *k-regret query*.

In the traditional top-k query [9, 16, 17, 25, 29], a user has to provide his/her utility function explicitly. Then, the k tuples with the highest utilities are returned. Unfortunately, it is hard for most users to specify their utility functions explicitly. If the utility function is not known in advance, the *skyline* query [5, 6, 18, 21, 24] can be applied. In particular, a *"domination"* concept is used. A tuple p is said to *dominate* another tuple q if p is not worse than q on each attribute and p is better than q on at least one attribute. Then, the utility of p is always higher than that of q and, p is more desirable regardless of the utility function. Tuples which are not dominated by any other tuples are returned in the skyline query. Unfortunately, the skyline query could have a large output size (at worst the whole database), making it difficult to provide a small representative subset of the whole database.

Recently, a *k-regret* query [23, 26, 30] was studied, which solves multi-criteria decision-making from a novel perspective. In particular, it overcomes the deficiencies of the top-k (which requires the user to provide the exact utility function) and skyline query (which does not have a controllable output size). Specifically, a k-regret query finds k tuples from the database such that the utility of any user's favorite among these k tuples is guaranteed to be a small fraction (quantified as the *regret ratio*) less than the utility of his/her favorite in the whole database, regardless of the utility function. For example, a k-regret query on the house database returns k houses so that Alice can find a house in the returned set that she is interested in (since this house makes her regret ratio small) without providing her utility function.

In this paper, we study how *user interactions* would help in reducing the user's regret ratio, ranging from 0% to 100%, in the k-regret query. In particular, we are also interested in the special case of achieving a 0% regret ratio, i.e., we want to find the *favorite* tuple for a user in the whole database with the help of user interactions. Intuitively, instead of asking the user for the exact utility function directly (which is difficult for a user to answer), we ask the user to provide "hints" on what his/her utility function might look like (which is easier for a user to provide). Based on the user feedback, we implicitly learn the utility function and determine the tuple that s/he is interested in. Note that we do not want to ask the user a question which is too difficult to answer (e.g., asking the user for the exact utility function in the top-k query). Thus, we stick to the assumption made in [22] in this paper, which requires little user effort: *when presented with a short list of tuples, the user is able to tell his/her favorite tuple (i.e., the tuple with the maximum utility) among them.* This kind of user interaction naturally appears in our daily life. For example, a realtor might show the customers around a few candidate houses and ask which one the customers favor the most. A shoe seller might let the customers try a few different shoes, followed by a question: which one do you feel the most comfortable with?

To motivate the problem, we consider an application scenario where a realtor wants to help Alice to find her (close to) favorite house in the market by interactively learning Alice's preference and making recommendations (another car purchase application scenario is described in our user study in Appendix D.1). Note that buying a house is one of the big purchases in our life. Thus, it is important for Alice to find a house which is as close to her favorite one as possible (i.e., to achieve a small regret ratio). Otherwise, Alice might feel regretful for not buying a better one for a long time.

There can be many candidate houses in the market and thus, Alice might have to trade-off between different attributes. For example, Alice might be willing to pay more on buying a new house than buying an old one. This trade-off is often individualistic and might not be known by the realtor in a complete and explicit way. To recommend houses effectively, the realtor can show Alice around a short list of houses and Alice can indicate the house she favors the most. The house favored by Alice might differ from other non-favorite houses in some ways, which reflects the trade-off in Alice's mind. With this information, the realtor can filter out those houses in the market definitely uninteresting to Alice. For those houses where Alice's preference is still unknown, another short list of recommendation could be made. By this interactive procedure, Alice can be provided with more and more accurate recommendations and her regret ratio is smaller and smaller since her preference is learnt implicitly. In this paper, we automate this interactive process, which has many practical applications in our daily life such as personalized recommendation system and self-guided shopping. According to our experimental evaluation, user interactions are very useful: they reduce both the user regret and the number of tuples displayed *significantly*. For example, to identify the favorite tuple for a user on a dataset with 4 attributes, a database system returns more than 1000 tuples if no user interaction is involved, while it needs to display as few as 25 tuples if user interactions are allowed.

In this paper, we want to help a user to find his/her (close to) favorite tuple (i.e., achieve a small regret ratio) with as little effort as possible (i.e., by examining as few options as possible). For example, we do not want to ask Alice to visit dozens of houses every weekend. Specifically, we measure the *user effort* by (1) the number of questions asked (i.e., the number of rounds of user interactions) and (2) the number of tuples displayed in each question (i.e., the question size, denoted by s). In particular, we want to answer the question: *how many questions do we need to ask to determine the favorite tuple or to guarantee a certain regret for a user?*

In the existing method [22] for interactive regret minimization, the database system creates artificial tuples (i.e., fake tuples not inside the database) and presents those tuples to the user during the interaction, which, however, is not desirable. For example, Alice might be shown a fake house during her interaction with the realtor and she is attracted by this house. Alice may wish to inspect this house in person to know more details about it. However, this is impossible since Alice cannot visit such a fake house that does not exist. If a database system relies on fake tuples for learning the user preference, it can be difficult to be applied in such real scenarios. Moreover, since Alice is attracted by this fake house, Alice might be encouraged to spend more time on interacting with the realtor, hoping to obtain an even better house at the end. However, if Alice finally finds that the house she favors is fake, Alice can be disappointed and think that the realtor is a fraud. Cheating customers with fake products can cause huge losses in reputation and profits (e.g., Hyundai offered 85 million to settle a lawsuit due to the overstatement of its cars [1]). In comparison, we always display true tuples inside the database during the interaction. However, this makes the problem more difficult and challenging: instead of being allowed to present any arbitrary tuple to the user, we are restricted to the tuples in the database. In this paper, we propose *strongly truthful* algorithms which tackle this difficulty. Specifically, if an interactive algorithm A always displays true tuples inside the database during the user interaction, A is *strongly truthful.* The existing interactive algorithm in [22] is *not* strongly truthful. There are some other preference learning algorithms in the literature, but they do not focus on interactive regret minimization. We postpone their detailed discussion to Section 2.

Contributions. We propose the first strongly truthful algorithms for interactive regret minimization. Specifically,

- We present a generic framework for interactive regret minimization and favorite tuple determination.
- We prove a lower bound on the number of questions needed to determine the favorite tuple in the database.
- We model the user preference by *utility hyperplanes* and present two effective pruning strategies to obtain the candidate set of favorite tuples for a user.
- We propose two solutions with an asymptotically optimal number of interactions in a 2-dimensional space and two solutions with provable guarantees in a d-dimensional space under the framework where each dimension corresponds to an attribute of a tuple.
- We conducted experiments to demonstrate the superiority of the proposed methods. Under typical settings, our solutions guarantee the same user regret as existing methods while asking half as many questions.

Organization. We discuss the related work in Section 2. In Section 3, we define the regret minimization problem and the generic interactive framework. The asymptotically optimal solutions in 2-dimensional spaces appear in Section 4 and the solutions with provable guarantees in d-dimensional spaces are described in Section 5. Experiments are presented in Section 6 and finally, Section 7 concludes this paper.

2 RELATED WORK

The k-regret query was first introduced in [23]. Intuitively, given a set of k tuples, a user is $x\%$ happy with the set if the highest utility of tuples in the set is at least $x\%$ of the highest utility of all tuples in the whole database. In this case, we say that the user is $(100 - x)\%$ regretful and the *regret ratio* of the user is $(100 - x)\%$. A k-regret query returns a set of k tuples such that the *maximum regret ratio* over all users is *minimized.* [22] extended the traditional k-regret query to *interactive regret minimization* by considering user interactions, which are shown to be useful in reducing the regret ratio from both theoretical and practical points of view. In this paper, we mainly follow the setting in [22]. However, their solution has some disadvantages. Firstly, they display artificial tuples to approximate the utility function, which is not desirable. Secondly, their algorithm has a very poor performance when the user wants the favorite tuple (i.e. with a 0% regret ratio) in the whole database. In comparison, our algorithms are *different* by always displaying true tuples in the database and providing provable performance guarantees even when the user requires a 0% regret ratio.

Apart from interactive regret minimization studied in this paper, there are alternative methods [13, 16, 21, 27] which learn users' preference implicitly based on their feedback.

[13, 21] learned the user preference by asking the user to partition a given tuple set into a *desirable* group and an *undesirable* group. Moreover, [13] focused on the preference learning on the *undetermined* attributes. Specifically, an attribute is *undetermined* if there is no universal preference defined on that attribute for all users. For example, on some attributes such as car brands, the preferences can vary dramatically from one user to another user. Different from them, all attributes in our problem are determined. Besides, instead of asking the users for both desirable and undesirable tuples, we ask them to pick their favorite tuple only, which requires less user effort than partitioning tuples.

[21] assumed that all attributes are of different *importance.* Specifically, given two attributes A and B, if A is more important than B, a tuple with a better value on A is *unconditionally* preferred to tuples with worse values on A, regardless of their values on B. This is a strong assumption. For example, if a user indicates that size is more important than price, the user prefers a house of 1,000 square feet and 1 million dollars to a house of 999 square feet and 0.5 million dollars. Different from them, we assign weights to indicate the attribute importance, which is more reasonable in real cases.

The personalized skyline query of identifying the "truly interesting" tuples was supported in [16] where the user preference is modeled as a strict partial ordering on attributes. For example, Alice prefers skyline houses considering price only to skyline houses considering both price and size.

[27] studied the problem of approximating the user's preference function by pairwise comparisons. However, they focus on deriving the hidden preference accurately representing the *entire* ordering of *all* tuples rather than applying the preference for finding a *single* tuple which is close to or equal to the user's favorite tuple while we focus on applying the learned preference to find this tuple effectively. Nevertheless, based on the learned preference, they can compute the top ranked tuple and return it for interactive regret minimization. However, they require the user to answer more questions than needed since the core of [27] did not exploit to find the (close to) favourite tuple for a user. For example,

if Alice prefers house p_1 to p_2, and prefers house p_3 to p_4, it is less interesting to ask Alice for her preference between p_2 and p_4, which are definitely not her favorite house, but this additional comparison might be unavoidable in their case.

Interactive regret minimization is also related to the *learning to rank* problem in machine learning[11, 19]. In particular, [11] studied active ranking using pairwise comparisons where they defined the ranking of tuples according to their distances to a common reference point in an embedding space. However, they cannot guarantee the quality of the selected comparisons; that is, some of their comparisons can be less informative while we select comparisons that guide us to reduce the regret. Nevertheless, the top ranked tuple they learned can be returned for interactive regret minimization, but they also have the disadvantage of asking more questions than needed due to a similar reason stated for [27].

Bandit approaches [2, 12, 14] and metric learning approaches [20] were also considered in preference learning. However, all these methods suffer from the drawback of not exploiting the relation between tuples (where a relation means that a tuple is preferable to another tuple) and thus, require more feedback from the user. In comparison, we consider the inter-relation between tuples and thus, even some tuples have never been seen by the user, we can accurately filter out those tuples if they are definitely uninteresting to the user.

Compared with the existing methods, we have the following advantages. Firstly, we require less user effort. Specifically, we ask the user a few questions and, at each question, a user only need to pick his/her favorite tuple among a few tuples, while some existing methods ask significantly more questions [11, 27] and some other methods ask more difficult questions [13, 21]. Secondly, we ensure the strong truthfulness property by showing true tuples only while some existing methods rely on artificial tuples [22] to approximate the preference function. Thirdly, our assumption made on the user preference is more reasonable while the assumptions made in some existing studies [16, 21] are stronger and less intuitive. Finally, we exploit the additional structure in the problem and utilize the inter-relation between tuples to effectively filter out unqualified tuples. As a result, we can return a "good" tuple for a user very effectively.

3 PROBLEM DEFINITION

The input to our problem is a tuple set D with n tuples (i.e., $|D| = n$) in a d-dimensional space (we assume d to be a fixed constant in this paper). Note that each tuple in D could be described by many more than d attributes, but the user will select precisely d of them that s/he is interested in.

3.1 Terminologies

We use the word "tuple" and "point" interchangeably in the rest of this paper. We denote the i-th dimensional value of a point $p \in D$ by $p[i]$ where $i \in [1, d]$. Without loss of generality, we assume that each dimension is normalized to (0,1] and for each $i \in [1, d]$, there exists at least one point $p \in D$ such that $p[i] = 1$. We denote the L1-norm and L2-norm of p by $\|p\|_1$ and $\|p\|_2$, respectively. We assume that a larger value in each dimension is preferable to all users. If a smaller value is preferable (e.g., price), we can modify the dimension by subtracting each value from 1 so that it satisfies the above assumption. Consider the example in Table 1. We are given a database $D = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ containing 8 points where each point is associated with two attributes, namely X_1 and X_2, (i.e., $d = 2$) with normalized attribute values.

p	X_1	X_2	$f(p)$ $(u = (0.3, 0.7))$	p	X_1	X_2	$f(p)$ $(u = (0.3, 0.7))$
p_1	0	1	0.7	p_5	1	0.2	0.44
p_2	0.2	1	0.76	p_6	1	0	0.3
p_3	0.6	0.9	0.81	p_7	0.35	0.2	0.245
p_4	0.9	0.6	0.69	p_8	0.3	0.6	0.51

Table 1: Database and Utilities

Same as [7, 17, 22, 23, 26], the user preference is modeled by an unknown *linear utility function*, denoted by f, which is a mapping $f: \mathbb{R}^d_+ \rightarrow \mathbb{R}_+$. A utility function f is *linear* if $f(p) = u \cdot p$ where $f(p)$ is the *utility* of p w.r.t. f and u is a *utility vector*. Note that u is a d-dimensional non-negative real vector where $u[i]$ measures the importance of the i-th dimensional value in the user preference. In the rest of the paper, we also refer f by its utility vector u. For each user, we define a *regret ratio* [23] based on his/her utility vector u.

Definition 3.1 ([23]). Given a set $S \subseteq D$ and a utility vector u, the *regret ratio* of S over D w.r.t. u , denoted by $\mathrm{rr}_D(S, u)$, is defined to be $\frac{\max_{p \in D} u \cdot p - \max_{p \in S} u \cdot p}{\max_{p \in D} u \cdot p} = 1 - \frac{\max_{p \in S} u \cdot p}{\max_{p \in D} u \cdot p}$.

Note that $\mathrm{rr}_D(S, u)$ is the same for different scaled vectors of u. Without loss of generality, we assume $\sum_{i=1}^{d} u[i] = 1$.

Given the utility vector u and a set $S \subseteq D$, $\max_{p \in S} u \cdot p \leq \max_{p \in D} u \cdot p$ since S is a subset of D and thus, the regret ratio ranges from 0% to 100%. The user whose utility vector is u will be happy if the regret ratio of S is close to 0% since the maximum utility of points in S is close to the maximum utility of points in D w.r.t. u. In particular, a user is interested in the point in D which maximizes the utility w.r.t. his/her utility function. Specifically, given a utility vector u, a point p is the *maximum utility point of D w.r.t. u* if $p = \arg\max_{q \in D} u \cdot q$. We also call such a maximum utility point the *favorite* point of the user in the whole database. We summarize the frequently used notations in Appendix A (Table 3).

Example 3.2. Let $u = (0.3, 0.7)$. Consider p_2 in Table 1. Its utility w.r.t. u is $f(p_2) = u \cdot p_2 = 0.3 \times 0.2 + 0.7 \times 1 = 0.76$. The utilities of other points in D are shown in Table 1. Note that the maximum utility point of D is p_3 and its utility is

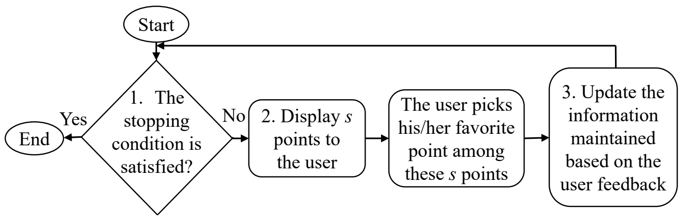

Start
1. The stopping condition is satisfied?
Yes
End
No
2. Display s points to the user
The user picks his/her favorite point among these s points
3. Update the information maintained based on the user feedback

Figure 1: The Interactive Framework

0.81. Consider a set $S = \{p_2\}$. The regret ratio of S over D w.r.t. u is $\text{rr}_D(S, u) = 1 - \frac{\max_{p \in S} u \cdot p}{\max_{p \in \mathbb{P}} u \cdot p} = 1 - \frac{0.76}{0.81} = 6.17\%$. □

3.2 Interactive Framework

We study how user interactions help in improving the regret ratio [22]. Our interactive framework is formalized in Figure 1. Specifically, the system interacts with a user with an *unknown* utility vector for *rounds* until certain stopping conditions are satisfied. At each round, the system asks the user a question by displaying s points. After the user picks the point s/he favors the most, we update the information maintained for learning the utility vector. Finally, the interaction stops and the system returns a point in D which makes the regret ratio small or it identifies the maximum utility point according to the information learned during the interaction. Formally, we are interested in the following problem:

- (**Interactive Regret Minimization (IRM)** [22]) How many questions do we need to ask the user to get the user's regret ratio below ϵ for some $0\% \leq \epsilon \leq 100\%$?

We are also interested in the special case where $\epsilon = 0\%$:

- (**Maximum Utility Point Determination (MUD)**) How many questions do we need to ask to get the user's maximum utility point (i.e., 0% regret ratio)?

We solve both IRM and MUD by considering the following:

1. **(Stopping Condition)** When can we stop interactions?
2. **(Point Selection)** How to select s points to display?
3. **(Information Maintenance)** What types of information should we maintain and how to update the information based on the user feedback?

The algorithm proposed in [22] asks $O(\log_s(1/\epsilon))$ questions and returns a point guaranteeing an ϵ regret ratio for IRM. However, the algorithm in [22] is not suitable for MUD where $\epsilon = 0\%$ and the number of questions it asks can be very large (e.g., it asks four times more questions than our methods according to our experiments). In comparison, we solve IRM/MUD with a stronger result. Specifically, we maintain more useful information for learning the utility vector by displaying true tuples inside the database, which supports novel stopping conditions and effective pruning strategies for non-maximum utility points, that do not appear in [22].

3.3 Lower Bound

A lower bound on the number of questions on IRM is proven in [22], which is $\Omega(\log_s(1/\epsilon))$. Note that this bound cannot be applied on MUD where $\epsilon = 0\%$. We first present a lower bound on MUD. Due to the lack of space, the proofs of Theorems/Lemmas in this paper can be found in Appendix E.

THEOREM 3.3. *For any dimensionality d, there is a dataset of n d-dimensional points such that any algorithm needs to ask $\Omega(\log_s n)$ questions to determine the user's favorite point.*

3.4 Truthfulness of Interactive Algorithms

In this section, we introduce the concept *"truthfulness"* [22], which is a desirable property for an interactive algorithm.

Note that an interactive algorithm can display points outside the database in order to learn the user preference. However, the algorithm must be *truthful*. Specifically, an algorithm is truthful if the point favored the most by the user among all points displayed during the interaction is in the database [22]. If an algorithm displays only points in the database, it is said to be *strongly truthful* (which is a stronger form of truthfulness). If an algorithm is allowed to display points outside the database (e.g., artificial points) but it ensures that the favorite point of a user during the entire interaction is not an artificial point, it is said to be *weakly truthful*.

The existing algorithm in [22] is *weakly truthful*. Specifically, it constructs artificial points to approximate the user's utility function. The strategy of the algorithm [22] *scales down* each artificial point by a certain (potentially large) factor and displays those artificial points to the user so that the user's utility function could be learned *more effectively*. This operation is not reasonable. Firstly, it might present the user with some unrealistic and meaningless points. For example, the realtor can present a fake house of 0.1 square feet and 100 dollars to Alice in order to learn her utility function [22]. However, Alice might think that this is a ridiculous option since such a house does not exist in reality. Secondly, it is impossible for a user to truly evaluate a fake point that does not exist (e.g., inspect a recommended house in person), resulting in the difficulty in applying it in many real-world applications. In comparison, our solutions do not have such deficiencies since our algorithms are all *strongly truthful*: we always present the user with true points inside the database.

4 2-DIMENSIONAL ALGORITHMS

We begin with the 2-dimensional algorithms, MEDIAN and HULL, which find the favorite point with asymptotically optimal numbers of questions for $s = 2$ and $s \geq 2$, respectively.

In geometry, the convex hull of D, denoted by Conv(D), is the smallest convex set containing D. A point p in D is a *vertex* of Conv(D) if $p \notin \text{Conv}(D/\{p\})$. Let $b_1 = (1, 0)$ and $b_2 = (0, 1)$, which are the *boundary points* of a 2-dimensional dataset. Let $O = (0,0)$ be the origin. Consider the convex hull $\text{Conv}(D \cup \{b_1, b_2, O\})$. In this section, when we say the vertices in $\text{Conv}(D \cup \{b_1, b_2, O\})$, we mean only the vertices in D. Note that for each utility vector, its maximum utility point must be a vertex of $\text{Conv}(D \cup \{b_1, b_2, O\})$. We assume that there are m vertices in $\text{Conv}(D \cup \{b_1, b_2, O\})$ and they are sorted in the "clockwise" order, namely $p_1, p_2, \ldots, p_{m-1}, p_m$.

Our algorithms work as follows. We maintain a candidate set C of maximum utility points, which is initialized to be $\{p_1, \ldots, p_m\}$ (i.e., information maintenance) where $m \leq n = |D|$. We interact with the user (whose utility vector is u) until there is only one point p in C (i.e., stopping condition), which is definitely the maximum utility point. Then, we return p to the user, which is the solution for both IRM and MUD since $\text{rr}_D(\{p\}, u) = 0\% \leq \epsilon$. At each round, we display s vertices (i.e., point selection) which divide C into a number of equal partitions. According to the user feedback, we locate the partitions which contain the maximum utility point and we update C. The following lemma helps us to determine the location of the maximum utility point.

Lemma 4.1. *Given the m vertices $\{p_1, \ldots, p_m\}$ of $\text{Conv}(D \cup \{b_1, b_2, O\})$ and a linear utility function f, if p_{i^*} is the desired maximum utility point w.r.t. f where $i^* \in [1, m]$, we have*

$$f(p_1) \leq \ldots \leq f(p_{i^*-1}) \leq f(p_{i^*}) \geq f(p_{i^*+1}) \geq \ldots \geq f(p_m).$$

Corollary 4.2. *Given an integer i and a linear utility function f, if $f(p_i) \geq f(p_{i+1})$, $i^* \leq i$; if $f(p_i) < f(p_{i+1})$, $i^* > i$.*

Example 4.3. Consider our database in Table 1 and we visualize $D = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ in Figure 2. The convex hull $\text{Conv}(D \cup \{b_1, b_2, O\})$ is shown in Figure 2 where $b_1 = p_1$ and $b_2 = p_6$ in this example. The vertices in the "clockwise" order are $p_1, p_2, p_3, p_4, p_5, p_6$. Consider $u = (0.3, 0.7)$ in Table 1. The maximum utility point is p_3 (i.e., $i^* = 3$) and thus, $f(p_1) \leq f(p_2) \leq f(p_3) \geq f(p_4) \geq f(p_5) \geq f(p_6)$. □

4.1 The Median Algorithm

When $s = 2$, we utilize the median vertices of $\text{Conv}(D \cup \{b_1, b_2, O\})$ and locate the maximum utility point in C in a binary search manner with the help of Corollary 4.2. At each iteration, C is reduced by half. Specifically, we display two consecutive median vertices p_i and p_{i+1} in C, which divide C into two halves. If p_i is favored by the user, C is updated to be the first half of C. Otherwise, C is updated to be the remaining half of C. This process continues until C contains a single point. The pseudocode is shown in Algorithm 1.

To illustrate, consider Table 1 where $u = (0.3, 0.7)$. C is $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ initially and the user is presented with the median vertices $\{p_3, p_4\}$. Since $f(p_3) \geq f(p_4)$, C is updated to the first half of C, i.e., $\{p_1, p_2, p_3\}$. The process continues and finally, $C = \{p_3\}$ which is the favorite point.

Algorithm 1 The Median algorithm

```
Input: A set $D$ and an unknown utility vector $u$
Output: The maximum utility point in $D$ with w.r.t. $u$
1: $C \leftarrow \{p_1, p_2, p_3, \ldots, p_m\}$, $start \leftarrow 1$, $end \leftarrow m$
2: while $|C| > 1$ do /* Stopping Condition */
3:     $i \leftarrow start - 1 + \lfloor \frac{end - start + 1}{2} \rfloor$ /* Point Selection */
4:     Ask the user whether s/he prefers $p_i$ to $p_{i+1}$
5:     if $p_i$ is preferable to $p_{i+1}$ then
6:         $end \leftarrow i$ /* Information Maintenance */
7:     else
8:         $start \leftarrow i + 1$ /* Information Maintenance */
9:     $C \leftarrow \{p_{start}, \ldots, p_{end}\}$
10: return the only point in $C$
```

Algorithm 2 The Hull algorithm

```
Input: A set $D$ and an unknown utility function $u$
Output: The maximum utility point in $D$ with w.r.t. $u$
1: $C \leftarrow \{p_1, p_2, p_3, \ldots, p_m\}$, $start \leftarrow 1$, $end \leftarrow m$
2: while $|C| > 1$ do /* Stopping Condition */
3:     if $|C| \leq s$ then
4:         Display $C$ to the user
5:         $C \leftarrow \{p\}$ where $p$ is the point picked by the user
6:     else
7:         $i_j \leftarrow start - 1 + \lfloor \frac{end - start + 1}{s+1} \rfloor * j, \forall j \in [0, s+1]$
8:         Divide $C$ into $s + 1$ partitions using $\{p_{i_j} | j \in [1, s]\}$, namely $C_0, \ldots, C_s$ where $C_j = \{p_{i_j+1}, \ldots, p_{i_{j+1}}\}$
9:         Display $S = \{p_{i_j} | j \in [1, s]\}$ /* Point Selection */
10:        $p_{i_j} \leftarrow$ the favorite point of the user
11:        $C \leftarrow C_j \cup C_{j-1}$ /* Information Maintenance */
12:        $start \leftarrow i_{j-1} + 1$, $end \leftarrow i_{j+1}$
13: return the only point in $C$
```

Theorem 4.4. *Median is strongly truthful and it determines the favorite point in $O(\log_2 n)$ rounds and in $O(n \log n)$ time.*

Combining Theorem 4.4 with Theorem 3.3, Median is asymptotically optimal in the number of questions asked.

4.2 The Hull Algorithm

Median can be applied when $s = 2$ only. In this section, we present the Hull algorithm for $s \geq 2$. Different from Median which partitions C into two halves in every round, Hull divides C into $s + 1$ partitions, namely $C_0, \ldots, C_s$, using a set of s vertices in C, denoted by S, which are then presented to the user. If the j-th point in S is the favorite point of the user, we update C to be $C_j \cup C_{j-1}$, which are the partitions where the maximum utility point is located according to Lemma 4.1. The pseudocode is shown in Algorithm 2.

To illustrate Hull, consider the example in Table 1 where $s = 2$ and $u = (0.3, 0.7)$. In the first round, we define $S =$

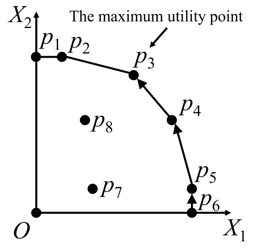


Figure 2: Convex Hull

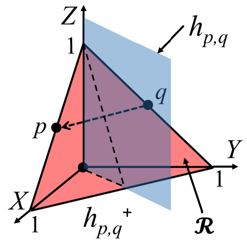


Figure 3: Utility Space

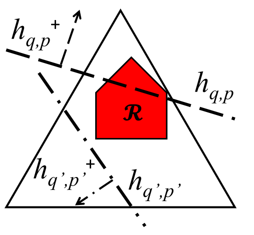


Figure 4: Hyperplane

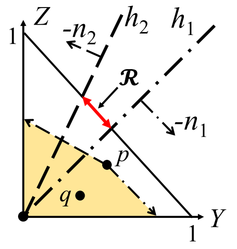


Figure 5: Conical Hull

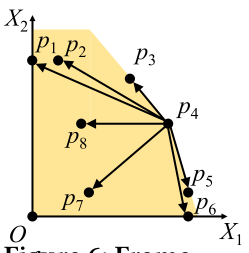


Figure 6: Frame

$\{p_2, p_4\}$, which is presented to the user. S divides $C = \{p_1, \ldots, p_6\}$ into 3 partitions, namely $C_0 = \{p_1, p_2\}, C_1 = \{p_3, p_4\}$ and $C_2 = \{p_5, p_6\}$. Since $f(p_2) \geq f(p_4)$, the maximum utility point must be in $C_0 \cup C_1 = \{p_1, p_2, p_3, p_4\}$ by Lemma 4.1, which C is updated to be. The process continues and finally, $C = \{p_3\}$ which is the desired maximum utility point.

THEOREM 4.5. *Hull is strongly truthful and it determines the favorite point in $O(\log_s n)$ rounds and in $O(n \log n)$ time.*

Combining Theorem 4.5 with Theorem 3.3, HULL is asymptotically optimal in the number of questions asked.

Remark. MEDIAN and HULL determine the favorite point (0% regret ratio) interactively. However, we can stop the interaction earlier (i.e., before the favorite point is determined) in IRM if $\epsilon > 0\%$. We postpone the detailed discussion to Section 5.2 where we show how to find a point p (not necessarily the favorite point) whose regret ratio is at most ϵ.

5 D-DIMENSIONAL ALGORITHMS

We are ready to describe our d-dimensional solutions. Specifically, we show how we address each of the three components (i.e., stopping condition, point selection and information maintenance) for IRM/MUD under the framework.

Intuitively, we interact with the user for rounds until the stopping conditions (Section 5.2) are satisfied. At each round,

(1) (Point selection) We select s true tuples from the database and present them to the user (see Section 5.3).
(2) The user picks his/her favorite tuple among them.
(3) (Information maintenance) We update the data structures maintained based on the user feedback. The details are shown in Section 5.1. In particular, we introduce the "*utility hyperplane*", and two effective pruning strategies for removing non-maximum utility points.

Finally, we summarize our algorithms in Section 5.4.

5.1 Information Maintenance

We first define the data structures for learning the user preference. Specifically, we maintain two data structures in our solutions: a convex region $\mathcal{R}$ in the utility space which contains the user's true utility vector u and a candidate set $C \subseteq D$ which contains the user's maximum utility point.

Formally, recall that $\sum_{i=1}^{d} u[i] = 1$ and thus, u can be regarded as a (non-negative) point on a hyperplane $H = \{p \in \mathbb{R}^d | \sum_{i=1}^{d} p[i] = 1\}$. We define the *candidate utility range*, denoted by $\mathcal{R}$, to be the convex region on H which contains the user's true utility vector u. For example, in Figure 3 where $d = 3$, $\mathcal{R}$ is the triangular region $\{u \in \mathbb{R}^3_+ | u[1]+u[2]+u[3] = 1\}$ before the user provides any information on his/her utility vector u. Given the candidate utility range $\mathcal{R}$, we define the *candidate set* of maximum utility points, denoted by C, to be a subset of points in D such that for each u in $\mathcal{R}$, the maximum utility point of D w.r.t. u is in C. That is, if $p = \arg\max_{q \in D} u \cdot q$ where u is a vector in $\mathcal{R}$, the point p is in C. For example, C can be all skyline points in D initially.

Intuitively, if the user answers more questions, we learn more about his/her utility vector u. Then, $\mathcal{R}$ is smaller and the corresponding C is updated. For example, $\mathcal{R}$ can be updated to the convex region shown in Figure 4 after the user answers certain questions. In particular, when $\mathcal{R}$ is sufficiently small, we can determine a point in C whose regret ratio is bounded by ϵ (to be discussed in Section 5.2) and thus, we can stop the interaction and this point is the desired point which guarantees the required regret ratio.

We show how we update $\mathcal{R}$ using *utility hyperplanes* according to the user feedback (e.g., the user prefers p to q for some points p and q among the points we display) in Section 5.1.1. We show how we remove non-maximum utility points in C using *pruning strategies* in Section 5.1.2.

5.1.1 Maintenance on $\mathcal{R}$. Given two points p and q, we define a *utility hyperplane*, denoted by $h_{p,q}$, to be the hyperplane passing through the origin O with its unit normal in the same direction as $p - q$. The hyperplane $h_{p,q}$ partitions the space $\mathbb{R}^d$ into two halves. The half space above $h_{p,q}$ is denoted by $h^+_{p,q}$. The following lemma shows how we can update $\mathcal{R}$ to be a smaller region based on $h_{p,q}$.

LEMMA 5.1. *Given $\mathcal{R}$ and two points p and q, if a user prefers p to q, the user's utility vector u must be in $h^+_{p,q} \cap \mathcal{R}$.*

Example 5.2. We draw the hyperplane $h_{p,q}$ passing through the origin with its unit normal in the same direction as $p - q$ where $p = (\frac{1}{2}, 0, \frac{1}{2})$ and $q = (0, \frac{1}{2}, \frac{1}{2})$ in Figure 3. If the user prefers p to q (i.e., $u \cdot p > u \cdot q$), u is in the half space above $h_{p,q}$. $\mathcal{R}$ is then updated to be $h^+_{p,q} \cap \mathcal{R}$ (left sub-triangle). $\square$

Utility hyperplanes are very useful. Firstly, according to the user feedback during the interaction, we can construct a number of utility hyperplanes and use them to update $\mathcal{R}$ continually. Secondly, based on utility hyperplanes, we can develop some punning strategies to remove non-maximum utility points in C (to be discussed in Section 5.1.2).

Note that $\mathcal{R}$ is formed by half space intersections on the hyperplane H and thus, $\mathcal{R}$ can be regarded as a $(d-1)$-dimensional *convex hull* on H. In the rest of this section, we focus on $\mathcal{R}$ on H (e.g., the $(d-1)$-dimensional space in Figure 4). In practice, the number of intersecting half spaces is small since it is proportional to the number of questions a user answers, which is usually small empirically.

In geometry, the intersection between $\mathcal{R}$ and a utility hyperplane is called a *$(d-2)$-flat*, which is a subspace of dimensionality $d-2$. For example, $(d-2)$-flats in 2-dimensional spaces are points and $(d-2)$-flats in 3-dimensional spaces are lines. Note that each $(d-2)$-flat divides $\mathcal{R}$ into two partitions (if the intersection is not empty). For example, consider the 2-dimensional example in Figure 5 where $\mathcal{R}$ is a line segment (i.e., a 1-dimensional convex hull). The intersection (if not empty) between $\mathcal{R}$ and a utility hyperplane (i.e., a line in a 2-dimensional space) is a point (i.e., a 0-flat), which divides the "line segment" $\mathcal{R}$ into two smaller partitions. Depending on the user feedback, we locate the partition containing u by Lemma 5.1. When the context is clear, we represent each utility hyperplane by its corresponding $(d-2)$-flat on H.

5.1.2 *Maintenance on C.*

In this section, we first present two pruning strategies, namely *hyperplane pruning* and *conical hull pruning*, to remove non-maximum utility points in C. Then, we present the detailed procedure of maintaining C based on the proposed pruning strategies.

Hyperplane Pruning. We model the relationship between two points p and q in C by their utility hyperplane. Intuitively, we can safely prune q if there is a p in C such that the user prefers p to q no matter which utility function s/he uses in $\mathcal{R}$. We summarize the result as follows.

Lemma 5.3. *Given $\mathcal{R}$, a point q can be pruned from C if there exists a point p in C such that $h^+_{q,p} \cap \mathcal{R} = \emptyset$.*

Example 5.4. Consider Figure 4 where we draw $\mathcal{R}$ and two utility hyperplanes, namely $h_{q,p}$ and $h_{q',p'}$. Since $h^+_{q',p'} \cap \mathcal{R} = \emptyset$, we prune q' from C by Lemma 5.3. Moreover, $h^+_{q,p} \cap \mathcal{R} \neq \emptyset$. We cannot prune either p or q since there could be a u in $\mathcal{R}$ such that p (q) is the maximum utility point. □

Complexity Analysis. Assume that $\mathcal{R}$ is formed by t half space intersections and it has m vertices. To perform hyperplane pruning for two points p and q, it takes $O(m)$ time to check if $h^+_{q,p} \cap \mathcal{R} = \emptyset$ by checking if there is a vertex of $\mathcal{R}$ in $h^+_{q,p}$.

Conical Hull Pruning. Similar to hyperplane pruning, we model the relationship between each pair of points in C by *conical hulls* (to be defined shortly) in conical hull pruning.

Denote the set of utility hyperplanes forming $\mathcal{R}$ by $\mathcal{H}$. Then, for each utility hyperplane h_i in $\mathcal{H}$, we denote its normal by n_i. Given a u in $\mathcal{R}$, $u \cdot n_i \geq 0$ for each h_i in $\mathcal{H}$. We define the *extreme vector set* of $\mathcal{H}$, denoted by $\mathcal{V}_{\mathcal{H}}$, to be $\mathcal{V}_{\mathcal{H}} = \{-n_i | \forall h_i \in \mathcal{H}\}$. When the context is clear, we simply denote $\mathcal{V}_{\mathcal{H}}$ by $\mathcal{V}$. The *conical hull* of a point p w.r.t. $\mathcal{V}$ [8] is defined to be $C_{p,\mathcal{V}} = \{q \in \mathbb{R}^d | (q-p) = \sum_{v_i \in \mathcal{V}} w_i v_i \text{ where } w_i \geq 0\}$ (which is a *convex cone* with apex p [28]). Note that the boundaries of $C_{p,\mathcal{V}}$ are *unbounded facets*, each of which is enclosed by some vectors in $\mathcal{V}$ and is a flat surface that forms a part of the boundary of $C_{p,\mathcal{V}}$.

Example 5.5. In the 3-dimensional example in Figure 4, $\mathcal{R}$ is a 2-dimensional convex hull and it is formed by 5 utility hyperplanes (i.e., $|\mathcal{H}| = 5$). To simplify the illustration, we instead consider a 2-dimensional example in Figure 5 where $\mathcal{R}$ is a line segment formed by two utility hyperplanes, namely h_1 and h_2. We denote their normals by n_1 and n_2, and thus, we have $\mathcal{V} = \{-n_1, -n_2\}$. The conical hull $C_{p,\mathcal{V}} = \{q \in \mathbb{R}^d | (q-p) = -w_1n_1 - w_2n_2 \text{ where } w_1, w_2 \geq 0\}$ is shown in the shaded region and it can be regarded as a convex cone with apex p. Besides, $C_{p,\mathcal{V}}$ has two unbounded boundary facets (i.e., the rays shooting from p, shown in dashed). □

Lemma 5.6. *Given two points p and q, and the vector set $\mathcal{V}$, if $q \in C_{p,\mathcal{V}}$, we have $u \cdot p \geq u \cdot q$ for each $u \in \mathcal{R}$.*

According to Lemma 5.6, given two points p and q in C, if $q \in C_{p,\mathcal{V}}$, we can prune q from C since the utility of p is at least the utility of q no matter which utility function a user uses in $\mathcal{R}$. For example, in Figure 4, the point q is in $C_{p,\mathcal{V}}$ and thus, it is pruned from C.

Next, we show how to determine whether $q \in C_{p,\mathcal{V}}$ efficiently for two points p and q given the vector set $\mathcal{V}$.

Consider a boundary facet F of $C_{p,\mathcal{V}}$. F is said to lie on a hyperplane if (1) for each point p' on F, p' lies on the hyperplane and (2) for each point p' in $C_{p,\mathcal{V}}$ but not on F, p' is below the hyperplane. Given a point q and a facet F, we say that q is on or below F if q is on or below the hyperplane that F lies on. A straight-forward way of checking whether $q \in C_{p,\mathcal{V}}$, followed directly from the definition of conical hull, is to determine whether q is on or below *all* boundary facets of $C_{p,\mathcal{V}}$. Note that the number of boundary facets is small in practice since $|\mathcal{V}|$ is proportional to the number of questions a user answers, which is usually small empirically.

Example 5.7. In Figure 5 (a 2-dimensional example), there are two boundary facets in $C_{p,\mathcal{V}}$, namely the rays shooting from p in the direction of $-n_1$ and $-n_2$, denoted by F_1 and F_2. Point q, which is in $C_{p,\mathcal{V}}$, is on or below F_1 and F_2 (i.e., q is on or below the hyperplanes that F_1 and F_2 lie on). □

Instead of maintaining the boundary facets of $C_{p,\mathcal{V}}$ for each p in C, we focus on $C_{O,\mathcal{V}}$, which can be regarded as a conical hull "translated" from p to O, so that the computation can be re-used. Formally, we have the following lemma.

LEMMA 5.8. *$q \in C_{p,\mathcal{V}}$ if and only if $q - p \in C_{O,\mathcal{V}}$.*

The boundary facets of $C_{O,\mathcal{V}}$ depend only on $\mathcal{V}$ (not on p). Thus, we compute its boundary facets and use them for all points in C. When $\mathcal{R}$ is updated after the user answers a question, the boundary facets of $C_{O,\mathcal{V}}$ will be updated accordingly. We can maintain the boundary facets incrementally instead of building them from scratch at each round.

In the worst case, we have to check all facets of $C_{O,\mathcal{V}}$ to conclude $q \notin C_{p,\mathcal{V}}$. We present a necessary condition for conical hull pruning in Appendix B for lack of space so that we can determine that $q \notin C_{p,\mathcal{V}}$ efficiently in $O(1)$ time.

Complexity Analysis. Since $|\mathcal{V}| = t$, there are $O(t^{\lfloor \frac{d-1}{2} \rfloor})$ boundary facets in $C_{O,\mathcal{V}}$ in the worst case. Thus, it takes $O(t^{\lfloor \frac{d-1}{2} \rfloor})$ time to determine if $q \in C_{p,\mathcal{V}}$. However, if the necessary condition is satisfied, we conclude $q \notin C_{p,\mathcal{V}}$ in $O(1)$ time.

Maintain C based on the Pruning Strategies. Based on the update on $\mathcal{R}$, we maintain C with the help of our pruning strategies. Specifically, when $\mathcal{R}$ is updated, we remove each point from C that can be pruned by hyperplane pruning or conical hull pruning and thus, each remaining point in C is the candidate maximum utility tuple w.r.t. some utility vectors in $\mathcal{R}$. A straight-forward way of removing non-maximum utility points is to perform a pairwise checking of points in C, which, however, can be very inefficient.

Note that the maintenance on C is a generalization of skyline computation where $\mathcal{V} = \{-e_i | \ i \in [1, d], e_i[i] = 1 \text{ and } e_i[j] = 0 \text{ if } i \neq j\}$. Some algorithms for skyline computation can be adapted to maintain C. Due to the lack of space, we explain how the branch-and-bound skyline (BBS) algorithm [24] is applied to maintain C in Appendix C.

5.2 Stopping Condition

In this section, we define two stopping conditions based on $\mathcal{R}$ and C. Recall that in IRM/MUD, we want to guarantee an ϵ regret ratio for a user. Thus, we stop the interaction when we determine a point p in C whose regret ratio is at most ϵ.

The First Stopping Condition. If there is only one point p in C, we conclude that p is the maximum utility point with a 0% regret ratio. Thus, we stop the interaction immediately (since $\mathrm{rr}_D(\{p\}, u) = 0\% \leq \epsilon$) and return p to the user.

The Second Stopping Condition. Given $\mathcal{R}$, we define the *diameter* of $\mathcal{R}$, denoted by $||\mathcal{R}||_1$, to be the maximum L1-distance between any two vectors in $\mathcal{R}$. That is, $||\mathcal{R}||_1 = \max_{u,v \in \mathcal{R}} ||u - v||_1$. Recall that the user's utility function u is in $\mathcal{R}$. The following lemma shows that we can determine a p in C so that $\mathrm{rr}_D(\{p\}, u)$ is bounded proportionally by $||\mathcal{R}||_1$.

LEMMA 5.9. *Let v be a vector in $\mathcal{R}$ and $p = \arg\max_{q \in C} v \cdot q$. $\mathrm{rr}_D(\{p\}, u) \leq 2d||\mathcal{R}||_1$ where u is the user's utility vector.*

Initially, $\mathcal{R} = \{u \in \mathbb{R}^d_+ | \ ||u||_1 = 1\}$. When a user (with a utility vector u) answers more questions, the corresponding utility hyperplanes partition $\mathcal{R}$ and $||\mathcal{R}||_1$ is smaller. With Lemma 5.9, we find a point p in C by utilizing a vector v in $\mathcal{R}$ so that $\mathrm{rr}_D(\{p\}, u)$ is bounded proportionally by $||\mathcal{R}||_1$. To guarantee a regret ratio ϵ, we interact with the user until $||\mathcal{R}||_1 \leq \frac{\epsilon}{2d}$ and return $p = \arg\max_{q \in C} v \cdot q$ where v is in $\mathcal{R}$.

5.3 Point Selection

In this section, we present two ways of displaying points in the interactive framework. In particular, we always display true points in C. We first present a heuristic approach which performs well empirically. Then, we present an approach with provable guarantees on the number of questions asked.

The First Approach: Random. At each round, we randomly select s points from C and display them to the user. Let p be the favorite point of the user among them. For each of the remaining $s - 1$ points, namely q, we construct a utility hyperplane $h_{p,q}$, resulting in $s - 1$ utility hyperplanes in total, which update $\mathcal{R}$ and C accordingly (see Section 5.1).

The Second Approach: Simplex. The idea is borrowed from the SIMPLEX method for Linear Programming problems (LP) [8]. We always present points in C which are also vertices in $\mathrm{Conv}(D)$ (note that the maximum utility point must be a vertex in $\mathrm{Conv}(D)$). Specifically, we maintain the vertex $p \in C$ in $\mathrm{Conv}(D)$ with the highest utility displayed so far. Denote the set of all *neighboring vertices* of p in $\mathrm{Conv}(D)$ by N_p (e.g., the neighboring vertices of p_4 is $N_{p_4} = \{p_3, p_5\}$ in Figure 2). We interactively check if there is a vertex in N_p with a higher utility than p by displaying p and at most $s - 1$ neighboring vertices in N_p to the user at each round. Each non-favorite point displayed corresponds to a new utility hyperplane which will then update $\mathcal{R}$ and C. Intuitively, we display p and its neighboring vertices in this approach because they provide directive information to locate the maximum utility point. Formally, we have the following lemma.

LEMMA 5.10. *Given a utility vector u and a vertex $p \in C$ of $\mathrm{Conv}(D)$, either p is the maximum utility point w.r.t. u or, there is vertex in N_p, whose utility is larger than that of p.*

To determine N_p of a vertex p, we can compute the exact convex hull $\mathrm{Conv}(D)$. However, computing the exact $\mathrm{Conv}(D)$ can be time-consuming in high dimensional spaces. In the following, we present an alternative way of obtaining N_p .

Given a vertex p in $\mathrm{Conv}(D)$, we let $V = \{q - p | \ \forall q \in D/\{p\}\}$. A set $V_F \subseteq V$ is defined to be a *frame* of V if V_F is the *minimal* subset of V such that $C_{p,V} = C_{p,V_F}$. Note that for each vector $v \in V_F$, we have $v \notin C_{p,V/\{v\}}$. The following lemma shows that the frame of V is closely related to N_p.

Algorithm 3 The UH-Random Algorithm

```
Input: $D$, a regret ratio $\epsilon$ and an unknown utility vector $u$
Output: A point $p$ in $D$ with $\mathrm{rr}_D(\{p\}, u) \leq \epsilon$
1: $\mathcal{R} \leftarrow \{u \in \mathbb{R}^d_+ | \sum_{i=1}^d u[i] = 1\}$
2: $C \leftarrow$ the set of all skyline points in $D$
3: while $\|\mathcal{R}\|_1 > \frac{\epsilon}{2d}$ and $|C| > 1$ do
4:     Randomly display $s$ points in $C$
5:     Update $\mathcal{R}$ and $C$ based on utility hyperplanes
6: return $p = \arg\max_{q \in C} v \cdot q$ where $v \in \mathcal{R}$.
```

Lemma 5.11. *Given a vertex p in* $\mathrm{Conv}(D)$, *$q \in N_p$ if and only if $q - p \in V_F$ where $V = \{q - p | \forall q \in D/\{p\}\}$.*

Example 5.12. Consider p_4 and $V = \{p_i - p_4 | \forall p_i \in D/\{p_4\}\}$ in Figure 6. $V_F = \{p_3 - p_4, p_5 - p_4\}$ is a frame of V since it is the minimal subset of V such that $C_{p_4,V} = C_{p_4,V_F}$ (showed in shaded). Thus, $N_{p_4} = \{p_3, p_5\}$ as shown in Figure 2. □

In most cases, computing V_F is cheaper compared with computing the exact $\mathrm{Conv}(D)$ since the number of vectors in V_F is usually smaller than the number of facets in $\mathrm{Conv}(D)$. For example, there are two vectors in the frame in Figure 6, while there are 7 facets in $\mathrm{Conv}(D)$ in Figure 2. Specifically, we can compute the whole V_F by LP [8] in $O(|V||V_F|)$ time. However, we can construct V_F as needed [8] (i.e., do not generate the whole V_F all at once) since we need at most $s - 1$ neighboring vertices of a given vertex p for each question.

5.4 Algorithm Summary and Analysis

We summarize our algorithms by combining the techniques presented in previous sections. Denote the algorithm with "random" point selection by UH-Random and denote the algorithm with "simplex" point selection by UH-Simplex where the prefix "UH" stands for "Utility Hyperplanes".

UH-Random randomly displays points in C to the user and update the data structures based on the user feedback until the stopping conditions are satisfied. Note that $\mathcal{R}$ is strictly smaller after each question. This is because that given two points p and q displayed by UH-Random in a round, neither p nor q is pruned by hyperplane pruning at the beginning of this round and thus, $h_{p,q}$ must divide $\mathcal{R}$ into two smaller partitions. Similarly, $|C|$ is strictly smaller since we can prune at least $s - 1$ non-favorite points after each question. The pseudocode is presented in Algorithm 3.

UH-Simplex works in a similar manner. Intuitively, it starts with a vertex p of $\mathrm{Conv}(D)$. It determines if p has a larger utility than all its neighboring vertices in $\mathrm{Conv}(D)$ by displaying p and the vertices in $N_p \cap C$ to the user. If this is the case, we return p as the maximum utility point (in this case, $C = \{p\}$ according to Lemma 5.10 and the stopping condition is satisfied). Otherwise, we update p to be one of

Algorithm 4 The UH-Simplex Algorithm

```
Input: $D$, a regret ratio $\epsilon$ and an unknown utility vector $u$
Output: A point $p$ in $D$ with $\mathrm{rr}_D(\{p\}, u) \leq \epsilon$
1: $\mathcal{R} \leftarrow \{u \in \mathbb{R}^d_+ | \sum_{i=1}^d u[i] = 1\}$
2: $C \leftarrow$ the set of all skyline points in $D$
3: $p \leftarrow$ a vertex of $\mathrm{Conv}(D)$
4: while $\|\mathcal{R}\|_1 > \frac{\epsilon}{2d}$ and $|C| > 1$ do
5:     Display $p$ and $s - 1$ points in $N_p \cap C$
6:     if the user favors $p$ among these $s$ points then
7:         if $p$ has a higher utility than points in $N_p$ then
8:             return $p$
9:     else
10:        $p \leftarrow$ a vertex in $N_p$ with a higher utility
11:    Update $\mathcal{R}$ and $C$ based on utility hyperplanes
12: return $p = \arg\max_{q \in C} v \cdot q$ where $v \in \mathcal{R}$.
```

its neighboring vertices with a higher utility and repeat the process. The pseudocode is presented in Algorithm 4 and its theoretical performance is provided in Theorem 5.13.

Theorem 5.13. *UH-Simplex is strongly truthful and it determines the favorite point in $O(n/s)$ rounds in the worst case and $O(\deg_{\max} \sqrt[d]{n}/s)$ rounds on average, where $\deg_{\max}$ is the maximum number of neighboring vertices for a vertex in* $\mathrm{Conv}(D)$.

Comparison. Compared with the existing method [22], we have some attractive advantages. Firstly, we maintain more useful information. Specifically, we maintain a region $\mathcal{R}$ which contains the user's true utility vector u and a candidate set C of favorite points but they only maintain an estimate utility vector $\hat{u}$. Note that each vector in $\mathcal{R}$ can be regarded as such an estimate utility vector. Secondly, we support new stopping conditions (e.g., $|C| \leq 1$) while they cannot since they do not maintain any information on C. Thirdly, we always show *true* points in the database while they show *fake* points outside the database. Finally, we can guarantee a 0% regret ratio efficiently, in which case they perform poorly.

6 EXPERIMENT

We conducted experiments on a machine with 1.60GHz CPU and 8GB RAM. All programs were implemented in C/C++.
Datasets. We conducted experiments on *synthetic* and *real* datasets. Specifically, we generated *anti-correlated* datasets by a dataset generator developed for skyline operators [4]. Besides, we adopted 3 real datasets commonly used in the existing studies. They are *Island*, *NBA* and *Household*. Island is 2-dimensional, which contains 63,383 geographic positions. NBA contains 21,961 player/season combinations from 1946 to 2009. Four attributes are used to represent the performance of each player. Household consists of 1,048,576 family tuples in US in 2012. Each family is associated with seven

attributes, showing the economic characteristic. The information about real datasets is summarized in Table 2. For all datasets, each attribute is normalized to (0, 1] and we preprocessed each dataset so that it contains skyline points only.
Algorithms. We evaluated our 2-dimensional algorithms, namely MEDIAN and HULL, and our d-dimensional algorithms, namely UH-SIMPLEX and UH-RANDOM. The competitor algorithms are (1) the only existing interactive algorithm for IRM [22], UTILITYAPPROX, which approximates the user's utility vector by presenting s fake points to the user; (2) a single round algorithm CORESETHS [15] which guarantees an ϵ regret ratio by returning a solution set with the minimum number of points; and (3) a single round algorithm SPHERE [30] which returns a solution with at most k points so that the regret ratio of a user is minimized. Note that although CORESETHS and SPHERE are both the state-of-the-art single round algorithms for the k-regret query, they are applied under different scenarios. Specifically, we minimize the number of points shown while fixing the regret ratio ϵ in CORESETHS, but we minimize the regret ratio while fixing the number of points shown in SPHERE. Depending on the parameters we vary, we compared either CORESETHS or SPHERE accordingly to demonstrate the effectiveness of user interactions in reducing the number of points shown and the regret ratio, respectively. Other preference learning algorithms (not for IRM/MUD) [11, 27] were also compared experimentally by conducting a user study in a real scenario. For lack of space, the user study is shown in Appendix D.1.
Parameter Setting. We evaluated the performance of each algorithm by varying different parameters. Specifically, we studied the effect of (1) different pruning strategies, (2) the number of points displayed per question (i.e., the question size s), (3) the dataset size n, (4) the dimensionality d, (5) the target regret ratio ϵ, and (6) the number of questions we can ask (i.e., the maximum number of points we display). Unless stated explicitly, we set the number of points displayed per question to be 3 (i.e., $s = 3$), set the target regret ratio ϵ to be 1% (i.e., $\epsilon = 1\%$), and we use hyperplane pruning as the default pruning method. For each synthetic dataset, the number of points in the dataset is set to be 100,000 (i.e., n = 100,000) and the dimensionality is set to be 4 (i.e., d = 4).
Performance Measurement. We evaluated the performance of each algorithm using 5 different measurements (experiments were conducted 5 times where we generated 5 user utility vectors independently and reported the average performance): (1) *Execution time.* The execution time of an algorithm is the time needed to guarantee a certain regret ratio ϵ or to display a certain number of points to the user. (2) *Candidate set size.* The candidate set size of an algorithm is the size of C maintained during the interaction. We reported the percentage of remaining points in C after each interaction. For UTILITYAPPROX and the single round algorithms, which do not support pruning, the candidate set size is always 100%. (3) *Regret ratio.* The regret ratio of an interactive algorithm (a single round algorithm) is the regret ratio of the final point suggested (the solution set returned). (4) *The total number of points displayed.* For a single round algorithm, the number of points displayed is the size of the solution set returned. For an interactive algorithm, the number of points displayed is at most the number of questions multiplied by the question size s. (5) *The number of questions asked.* This quantifies user effort. If s is fixed, the number of questions asked is proportional to the number of points displayed.

6.1 Results on Synthetic Datasets

We first compared our 2-dimensional algorithms, MEDIAN and HULL, against the existing ones on a 2-dimensional dataset in Figure 7 by varying the number of points we can display. For completeness, we also compared the d-dimensional algorithms, UH-SIMPLEX and UH-RANDOM (however, their performance will be analyzed later). Since s is fixed to 2 in MEDIAN, we set s to 2 in all other algorithms for fair comparison. Besides, we also reported the performance of HULL and UTILITYAPPROX when s is set to 4 to demonstrate the effect of doubling the question size. Note that when fixing the maximum number of points displayed, the interactive algorithms with different values of s can ask different numbers of questions (see Figure 7(d)). For example, if the maximum number of points displayed is set to 8, HULL($s = 2$) can ask 4 questions while HULL($s = 4$) can only ask 2 questions. Figure 7(a) depicts the execution time. All algorithms are fast and they take only a few milliseconds to execute. In Figure 7(b), all our algorithms perform similarly in terms of the candidate set size. In particular, MEDIAN and HULL quickly reduce the candidate set size by a factor of 5 with only 1-2 questions while all existing approaches fail to provide any reduction on the candidate set size. As a by-product, MEDIAN and HULL can achieve a significantly smaller empirical regret ratio compared with UTILITYAPPROX within a few rounds of interactions (see Figure 7(c)). In particular, MEDIAN and HULL suggest the maximum utility point (0% regret ratio) after presenting 4 points to the user (but since there could be more than one point in the candidate set, we need more questions to conclude that it is indeed the maximum utility point) while UTILITYAPPROX can guarantee a 5% regret ratio at best in this case. In this 2-dimensional dataset, s does not have a significant impact on the performance, but we will show later in d-dimensional datasets that a moderate s is helpful in reducing the overall user effort.

We studied the performance of different pruning strategies in Figure 8 where we evaluated the execution time by augmenting UH-SIMPLEX($s = 3$, $\epsilon = 1\%$) with hyperplane pruning and conical hull pruning. Since these two pruning strategies differ in implementations and they produce the

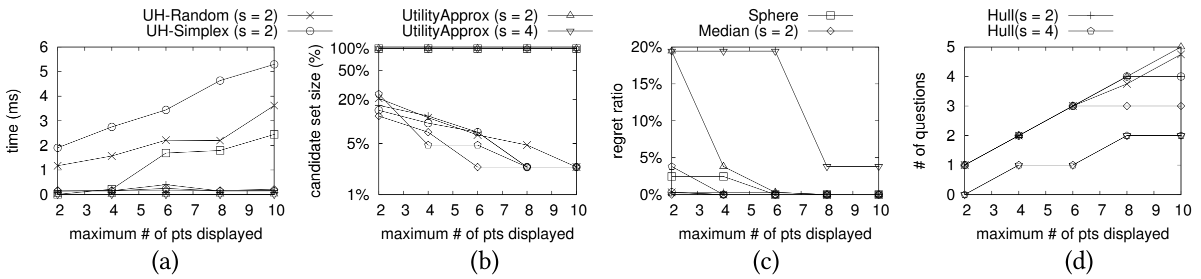


Figure 7: Results on 2-dimensional Synthetic Datasets ($d = 2, n = 100$k)

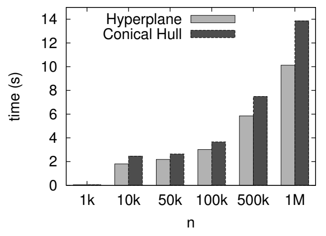


Figure 8: Pruning Strategies

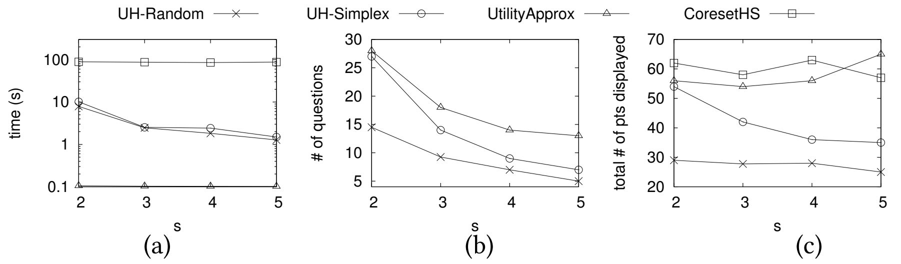
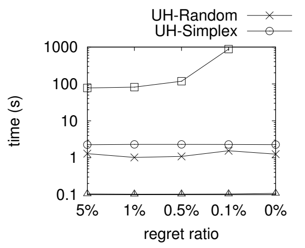


Figure 9: Vary s ($d = 4, \epsilon = 1\%, n = 100$k)

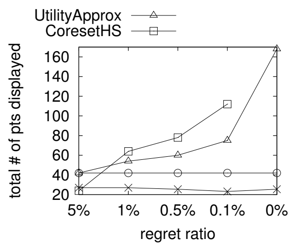
same result, we only reported the execution time in Figure 8. Both pruning strategies are efficient and they can guarantee the desired regret ratio in around 10 seconds even when the dataset contains more than 1 million tuples. However, hyperplane pruning consumes slightly less time than conical hull pruning in most cases empirically. Thus, we stick to hyperplane pruning in the rest of the experiments.

In Figure 9, we studied the effect of s on different interactive algorithms on a 4-dimensional dataset. We also compared the single round algorithm CoresetHS to demonstrate the usefulness of user interactions in reducing the number of points displayed while achieving the same regret ratio. To guarantee a 1% regret ratio, CoresetHS requires longer execution time and a larger output size compared with the interactive algorithms (Figure 9(a)(c)). Specifically, it takes 90 seconds to return 60 tuples to the user in order to achieve a 1% regret ratio while our UH-Random algorithm needs as few as 10 seconds and 30 tuples to guarantee the same regret ratio. Note that UtilityApprox is faster than UH-Simplex and UH-Random (Figure 9(a)). This is because that it constructs fake points and does not heavily rely on the input dataset. However, as we argue in Section 3, this weak truthfulness is not desirable. Although UH-Simplex and UH-Random spend slightly more time than UtilityApprox, their execution times are small and reasonable given that we can achieve a much stronger result. In particular, when $s = 2$ in UH-Random, we need less than 0.5 second on average to process each answer provided by the user, which is acceptable in real scenarios. Despite the slightly worse performance in terms of the execution time, UH-Simplex and UH-Random ask fewer questions and display fewer points to the user under all values of s compared with UtilityApprox (Figure 9(b)(c)), which is more crucial in interactive algorithms due to the human effort in asking the user questions. We can also observe in the figure that, although UH-Random does not provide provable guarantee on the number of questions as UH-Simplex does, its empirical performance is good. When the question size s increases, UH-Simplex and UH-Random need fewer questions and fewer points to solve IRM/MUD while the total number of points displayed by UtilityApprox even increases slightly (which is also observed in [22]). To perform fair comparison, we set the default value of s to be 3 which gives reasonable performance for all interactive algorithms in the rest of the experiments.

We proceed with the performance evaluation of UH-Simplex and UH-Random by varying the regret ratio ϵ from 5% to 0% in Figure 10 on a 4-dimensional dataset. Note that s is fixed to 3 for all algorithms in Figure 10. Thus, we did not report the number of questions asked by each algorithm since they can be easily inferred from the total number of points displayed. According to the results presented in Figure 10, our interactive algorithms are less sensitive to the target regret ratio ϵ and achieve orders of improvement in both execution time and output size. This is because our pruning strategies are very effective and we can quickly reduce the candidate set size after only a few questions (to be shown shortly) so that we can guarantee the required regret ratio efficiently. In comparison, the execution time of CoresetHS increases rapidly when ϵ decreases. In particular, it takes more than 1000 seconds and returns more than 1000 points to achieve a 0% regret ratio, i.e, to obtain the favorite point of a user. Thus, we exclude its performance when $\epsilon = 0\%$ in the figure for better visualization. Note that the number of points

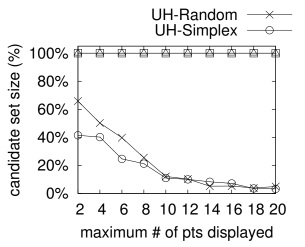


(a)

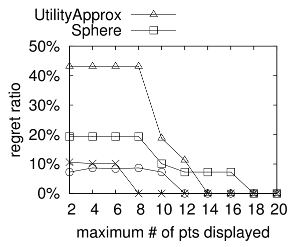


(b)

Figure 10: Vary Regret Ratio ($d = 4, s = 3, n = 100$k)



(a)



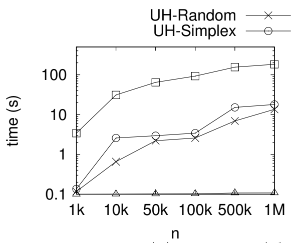
(b)

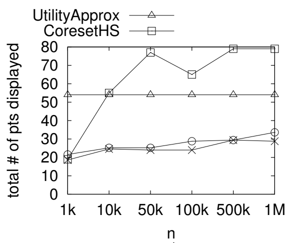
Figure 11: Vary Points Displayed ($d = 5, s = 2, n = 100$k)

displayed by UtilityApprox also increases rapidly when ϵ is close to 0%. It conforms with our claim in Section 3 that the number of questions asked by UtilityApprox can be extremely large (i.e., $O(\log_s(1/\epsilon))$) for MUD where $\epsilon = 0\%$.

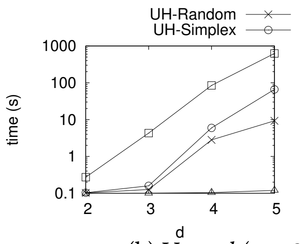
In Figure 11, we vary the maximum number of tuples we can display to the user (which is proportional to the number of questions we can ask the user) on a 5-dimensional dataset. The purpose of this experiment is to verify the effectiveness of our pruning strategies and to show how we can guarantee a small regret ratio, so we set s to be 2 so that the effect of each user interaction can be clearly observed. When the user answers more questions, we learn more about the user's utility vector and we can prune more points in the candidate set, as shown in Figure 11(a). Our pruning strategies are very effective. In particular, we can prune 40% of points in C by asking only 1 question (i.e., displaying 2 points) and prune 90% of points by asking 5 questions (i.e., displaying 10 points). Besides, UH-Simplex and UH-Random are very effective in reducing the user regret (see Figure 11(b)). Specifically, their regret ratios drop to 0% more quickly than UtilityApprox and Sphere. In particular, UH-Random suggests the maximum utility point (0% regret ratio) to the user after 4 questions (i.e., 8 points) and UH-Simplex suggests the maximum utility point after 6 questions (i.e., 12 points). In comparison, UtilityApprox and Sphere locate the maximum utility point after 14 (i.e., 7 questions) and 18 points.

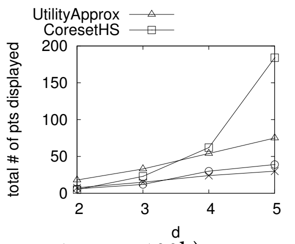
We also evaluated the scalability of UH-Simplex and UH-Random in Figure 12. In Figure 12(a), we studied the scalability of each algorithm on the dataset size n. UH-Simplex and UH-Random scale well in terms of the execution time while showing the smallest amount of points to the user. In particular, to guarantee a 1% regret ratio on a dataset with





(a) Vary n ($d = 4, s = 3, \epsilon = 1\%$)





(b) Vary d ($s = 3, \epsilon = 1\%, n = 100$k)

Figure 12: Scalability Test

1,000,000 points, the number of points we display is half of that of UtilityApprox and one third of that of CoresetHS. Besides, all interactive algorithms are significantly faster than the single round algorithm CoresetHS. In Figure 12(b), we studied the scalability of each algorithm on the dimensionality d. Compared with UtilityApprox, UH-Simplex and UH-Random consistently show fewer points in all dimentionalites. Compared with CoresetHS, UH-Simplex and UH-Random are two orders faster, verifying the usefulness of user interactions in guaranteeing the user regret.

Finally, we observe that the performance of an interactive algorithm might vary from one user to another whose utility vector and maximum utility point is different. Thus, we generated 100 utility vectors randomly and studied the performance of UH-Random over different users (the performance of UH-Simplex is similar). Due to the lack of space, the results are shown in Appendix D.2 (see Figure 18).

6.2 Results on Real Datasets

In this section, we studied the performance of the proposed algorithms on real datasets. Note that all our algorithms perform very efficiently (e.g., they take only a few seconds to execute) on real datasets. This is because that the number of skyline points in a real dataset is usually smaller than that in the synthetic dataset. Thus, we omit the results on execution time on real datasets. Similarly, we did not vary the regret ratio on real datasets since all algorithms can achieve a small regret ratio by displaying a small number of points.

The results on the Island dataset are shown in Figure 13 where we vary the maximum number of points displayed. All our algorithms effectively reduce the candidate set size. However, different from the results presented on the synthetic datasets, the candidate set size of UH-Random is slightly

Dataset	d	$\|D\|$
Island	2	63,383
NBA	4	21,961
Household	7	1,048,578

Table 2: Real Datasets

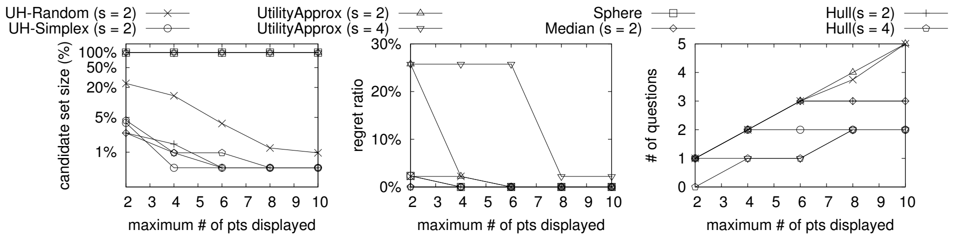
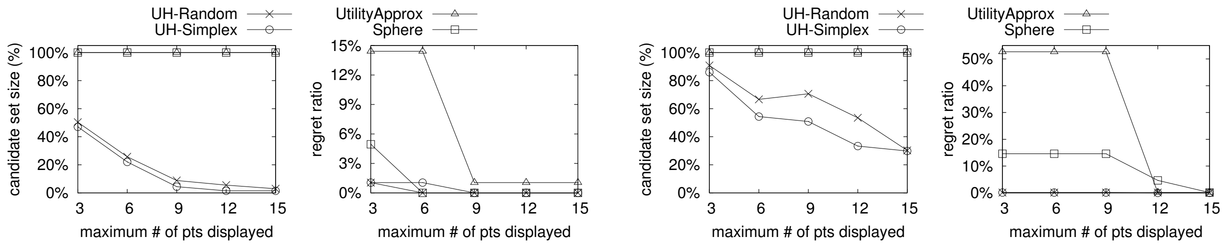


Figure 13: Results on the Island Dataset



(a) NBA (b) Household

Figure 14: Vary Maximum Number of Points Displayed on NBA and Household ($s = 3$)

larger than the other algorithms, but it is still much smaller than that of UtilityApprox and Sphere. When considering the regret ratio, Median and Hull achieve a much smaller empirical regret ratio compared with UtilityApprox.

Consider the performance of our d-dimensional algorithms, UH-Random and UH-Simplex, on NBA and Household where we vary the number of points displayed in Figure 14. We depicted the candidate set size in Figure 14. When the user is presented with more points (i.e., is asked more questions), the candidate set size is reduced rapidly. In particular, after 5 questions (i.e., 15 points since $s = 3$), we prune 97% and 70% of points in the candidate set on NBA and Household, respectively. When considering the regret ratio in Figure 14, UH-Simplex and UH-Random suggest the point with a 0% regret ratio after 2 questions (i.e., 6 points), while the regret ratio of UtilityApprox is much larger (e.g., greater than 50% on Household). Similar to the synthetic datasets, although we suggest the maximum utility point after a few questions empirically, we might need to ask more questions to conclude that it is indeed the desired point guaranteeing the regret ratio ϵ since it could happen that the upper bound on the regret ratio is still greater than ϵ. That is, there could be another utility vector in $\mathcal{R}$ whose maximum utility is much larger than the utility of the point suggested.

6.3 Summary

The experiments showed the superiority of our 2-dimensional algorithms, Median and Hull, and our d-dimensional algorithms, UH-Simplex and UH-Random, over the best-known previous approaches: (1) We are both efficient and effective. In particular, UH-Simplex and UH-Random achieve orders of improvement in execution time compared with CoresetHS (e.g., when $s = 2$ on the 4-dimensional dataset) and ask the smallest number of questions (i.e., present the least points) compared with UtilityApprox (e.g., one fourth of questions on a 4-dimensional dataset compared with UtilityApprox when $\epsilon = 0\%$). (2) The scalability of UH-Simplex and UH-Random is demonstrated. Specifically, they are scalable to both n and d. It only takes them around 15 seconds to execute (12 times faster than CoresetHS) when $n = 1,000,000$. (3) Our pruning strategies are useful. For example, we prune 97% of points in the candidate set by asking only 5 questions (i.e., 15 points since $s = 3$) on NBA. (4) We guarantee a small empirical regret ratio with a few questions. For example, on Household, we suggest a point with 0% regret ratio after 2 questions (i.e., 6 points) while the regret ratios of UtilityApprox and Sphere are 50% and 15%, respectively.

7 CONCLUSION

We present an interactive framework for IRM/MUD in this paper, under which we model the user preference by utility hyperplanes and present two effective pruning strategies for constructing the candidate set of maximum utility points. We propose two asymptotically optimal 2-dimensional algorithms and two d-dimensional algorithms with provable guarantees and superior empirical performance. In particular, we always present true points in the database and thus, our solutions are *strongly truthful*. Extensive experiments showed that our algorithms are very useful in achieving small regret ratios with a few rounds of interactions. As for future research, we consider adapting the ranking algorithms studied in machine learning into our problem.

ACKNOWLEDGMENTS

The research of Min Xie and Raymond Chi-Wing Wong is supported by HKRGC GRF 16214017 and ITS/227/17FP.

REFERENCES

[1] Hyundai offers 85 million to settle horsepower suit: Automaker overstated the horsepower of hyundai and kia cars exported to the u.s., https://www.consumeraffairs.com/news04/hyundai_settlement.html.

[2] A. Bhargava, R. Ganti, and R. Nowak. Bandit approaches to preference learning problems with multiple populations. *stat*, 1050:14, 2016.

[3] K. Borgwardt. The average number of pivot steps required by the simplex-method is polynomial. *Mathematical Methods of Operations Research*, 26(1):157–177, 1982.

[4] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings. 17th International Conference on Data Engineering*, 2001.

[5] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2006.

[6] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. On high dimensional skylines. In *Advances in Database Technology-EDBT 2006*, pages 478–495. Springer, 2006.

[7] Y. Chang, L. Bergman, V. Castelli, C. Li, M. Lo, and J. Smith. The onion technique: Indexing for linear optimization queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2000.

[8] J. Dulá, R. Helgason, and N. Venugopal. An algorithm for identifying the frame of a pointed finite conical hull. In *INFORMS Journal on Computing*, volume 10, pages 323–330. INFORMS, 1998.

[9] M. Goncalves and M. Yidal. Top-k skyline: a unified approach. In *On the Move to Meaningful Internet System 2005*, 2005.

[10] A. Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[11] K. Jamieson and R. Nowak. Active ranking using pairwise comparisons. In *Advances in Neural Information Processing Systems*, 2011.

[12] K. Jamieson and R. Nowak. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *Annual Conference on Information Sciences and Systems (CISS)*, 2014.

[13] B. Jiang, J. Pei, X. Lin, D. Cheung, and J. Han. Mining preferences from superior and inferior examples. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 390–398. ACM, 2008.

[14] K. Ju, K. Jamieson, R. Nowak, and X. Zhu. Top arm identification in multi-armed bandits with batch arm pulls. In *AISTATS*, 2016.

[15] N. Kumar and S. Sintos. Faster approximation algorithm for the k-regret minimizing set and related problems. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 62–74. SIAM, 2018.

[16] J. Lee, G. won You, and S. won Hwang. Personalized top-k skyline queries in high-dimensional space. In *Information Systems*, 2009.

[17] X. Lian and L. Chen. Top-k dominating queries in uncertain databases. In *Proceedings of International Conference on Extending Database Technology: Advances in Database Technology*, 2009.

[18] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *Proceedings of International Conference on Data Engineering*, 2007.

[19] T. Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[20] B. Mason, L. Jain, and R. Nowak. Learning low-dimensional metrics. In *Advances in Neural Information Processing Systems*, 2017.

[21] D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. In *Proceedings of the VLDB Endowment*, 2009.

[22] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino. Interactive regret minimization. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.

[23] D. Nanongkai, A.D. Sarma, A. Lall, R.J. Lipton, and J. Xu. Regret-minimizing representative databases. In *Proceedings of the VLDB Endowment*, 2010.

[24] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. In *ACM Transactions on Database Systems (TODS)*, volume 30, pages 41–82. ACM, 2005.

[25] A. N. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos. Domination mining and querying. In *DaWaK*, 2007.

[26] P. Peng and R.C.W Wong. Geometry approach for k regret query. In *Proceedings of International Conference on Data Engineering*, 2014.

[27] L. Qian, J. Gao, and H.V. Jagadish. Learning user preferences by adaptive pairwise comparison. In *Proceedings of the VLDB Endowment*, 2015.

[28] R. Rockafellar. *Convex analysis*. Princeton university press, 2015.

[29] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proceedings of International Conference on Data Engineering*, pages 896–905. IEEE, 2007.

[30] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proceedings of the 2018 ACM International Conference on Management of Data*. ACM, 2018.

A SUMMARY OF NOTATIONS

We summarize the frequently used notations in Table 3.

Notation	Meaning
D	The set of d-dimensional points with $\|D\| = n$
f and u	$f(p) = u \cdot p$ where $\sum_{i=1}^{d} u[i] = 1$
$\mathrm{rr}_D(S, u)$	The regret ratio of S over D w.r.t. u
s	The number of points displayed per question
ϵ	The target regret ratio we want to guarantee
$\mathrm{Conv}(D)$	The convex hull of D
H	A hyperplane $H = \{p \in \mathbb{R}^d \| \sum_{i=1}^{d} p[i] = 1\}$
$\mathcal{R}$	The candidate utility range on H which contains the user's true utility vector u
C	The candidate set of maximum utility points (i.e., if $p = \arg\max_{q \in D} u \cdot q$ where $u \in \mathcal{R}$, $p \in C$)
$h_{p,q}$	The utility hyperplane passing through the origin with its normal in the same direction as $p - q$
$h^+_{p,q}$	The half space above $h_{p,q}$
$\\|\mathcal{R}\\|_1$	The L1-diameter of $\mathcal{R}$ (i.e., $\\|\mathcal{R}\\|_1 = \max_{p,q \in \mathcal{R}} \\|p - q\\|_1$)
$\mathcal{V}$	The extreme vector set $\mathcal{V} = \{-n_i \| \forall h_i \in \mathcal{H}\}$ where $\mathcal{H}$ is the set of hyperplanes that form $\mathcal{R}$
$C_{p,\mathcal{V}}$	The conical hull of a point $p \in D$ w.r.t. $\mathcal{V}$ ($C_{p,\mathcal{V}} = \{q \in \mathbb{R}^d \| (q - p) = \sum_{v_i \in \mathcal{V}} w_i v_i$ where $w_i \geq 0\}$)
$H_{\mathcal{V}}$	A hyperplane with the normal $n_{\mathcal{V}}$ where $n_{\mathcal{V}} \cdot v_i > 0, \forall v_i \in \mathcal{V}$, and the offset $c_{\mathcal{V}} = \min_{v_i \in \mathcal{V}} n_{\mathcal{V}} \cdot v_i$
N_p	The set of all neighboring vertices of a vertex p in $\mathrm{Conv}(D)$
V_F	The frame of a vector set V (V_F is the minimal subset of V such that $C_{p,V} = C_{p,V_F}$)

Table 3: Summary of Frequently Used Notations

B A NECESSARY CONDITION

We formally define the necessary condition in conical hull pruning so that we can determine that $q \notin C_{p,\mathcal{V}}$ efficiently. We define a hyperplane, denoted by $H_{\mathcal{V}}$, for $\mathcal{V}$. The normal of $H_{\mathcal{V}}$, denoted by $n_{\mathcal{V}}$, is a unit vector such that $n_{\mathcal{V}} \cdot v_i > 0$ for each v_i in $\mathcal{V}$ (we describe how to compute $n_{\mathcal{V}}$ later). The offset of $H_{\mathcal{V}}$, denoted by $c_{\mathcal{V}}$, is $\min_{v_i \in \mathcal{V}} n_{\mathcal{V}} \cdot v_i$. Note that the ray shooting from O in the direction of v_i must intersect $H_{\mathcal{V}}$ at v_i' where $v_i' = c_i v_i$ and $c_i > 0$ (since $n_{\mathcal{V}} \cdot v_i > 0$). Similarly, we assume that the ray shooting from O in the direction of $q - p$ intersects $H_{\mathcal{V}}$ at q' where $q' = c_q(q - p)$ and $c_q \geq 0$ (if the ray does not intersect $H_{\mathcal{V}}$, c_q is positive infinity).

To illustrate, assume that $p = O$ and consider $C_{O,\mathcal{V}}$ where $\mathcal{V} = \{-n_1, -n_2\}$ in Figure 15. $H_{\mathcal{V}}$ is shown in a dashed line. Consider q_1 and q_2 (drawn in dot points). The ray shooting from O to q_1 and the ray shooting from O to q_2 intersect $H_{\mathcal{V}}$ at q_1' and q_2' (drawn in cross points), respectively.

Lemma B.1. *If $q \in C_{p,\mathcal{V}}$, $\|q'\|_2 \leq 1$.*

According to Lemma B.1, if $\|q'\|_2 > 1$, we directly conclude that $q \notin C_{p,\mathcal{V}}$ without checking the facets of $C_{O,\mathcal{V}}$. For example, in Figure 15, we draw a circle centered at O with radius 1. The point q_1 is in $C_{O,\mathcal{V}}$ and thus, $\|q_1'\|_2 \leq 1$ (inside the circle) according to Lemma B.1. However, since $\|q_2\|_2 > 1$ (outside the circle), we know that $q_2 \notin C_{O,\mathcal{V}}$.

Next, we formally define $n_{\mathcal{V}}$. According to our problem definition, $C_{O,\mathcal{V}}$ must be *pointed* (or *acute*). That is, there must be a hyperplane that supports it only at O (i.e., all points in $C_{O,\mathcal{V}}$ are on or below the hyperplane). Define the average vector of $\mathcal{V}$ to be $\bar{v} = \frac{1}{|\mathcal{V}|}\sum_{v_i \in \mathcal{V}} v_i$. Consider the following primal/dual LP with an arbitrary point b in $\mathbb{R}^d$ [8]:

$$\textbf{(P)} \quad \min \theta \quad \text{subjected to} \quad -\theta\bar{v} + \sum_{v_i \in \mathcal{V}} w_i v_i = b$$
$$\theta \geq 0 \text{ and } w_i \geq 0, \forall v_i \in \mathcal{V}$$

$$\textbf{(D)} \quad \max \pi \cdot b \quad \text{subjected to} \quad -\pi \cdot \bar{v} \leq 1$$
$$\pi \cdot v_i \leq 0, \forall v_i \in \mathcal{V}$$

Note that for an arbitrary b in $\mathbb{R}^d$, (P) is always feasible [8]. Consider a non-zero solution π of (D). We can define $n_{\mathcal{V}}$ to be $-\pi$ since $\pi \cdot v_i \leq 0$ for each $v_i \in \mathcal{V}$ (if ignoring the equality case). Denote the optimal solutions of (P) and (D) by θ^* and π^* and thus, $\theta^* = \pi^* \cdot b$. The following lemma from [8] shows how to obtain a non-zero solution π^* of (D).

Lemma B.2 ([8]). *b is exterior to $C_{O,\mathcal{V}}$ if and only if $\theta^* > 0$.*

According to Lemma B.2, we compute a non-zero solution π^* of (D) by substituting a b exterior to $C_{O,\mathcal{V}}$ to (P) (since $\theta^* = \pi^* \cdot b > 0$). We can define such a point b to be $-v_i$ for a v_i in $\mathcal{V}$ since $C_{O,\mathcal{V}}$ is pointed and $-v_i$ is exterior to $C_{O,\mathcal{V}}$.

Note that we want a vector $n_{\mathcal{V}}$ whose dot product with v_i is *strictly* greater than 0 for each $v_i \in \mathcal{V}$. Otherwise, $c_{\mathcal{V}} = \min_{v_i \in \mathcal{V}} n_{\mathcal{V}} \cdot v_i = 0$ and the hyperplane $\mathcal{H}_{\mathcal{V}}$ passes through the origin O. If this is the case, for any two points p and q, the ray $q - p$ intersects $\mathcal{H}_{\mathcal{V}}$ at O ($= q'$) and Lemma B.1 is useless since $\|q'\|_2$ is always 0.

Let $\mathcal{V}'$ be subset of $\mathcal{V}$ such that for each v_i in $\mathcal{V}$, there is a vector v_j in $\mathcal{V}'$ and $\pi_j^* \cdot v_i < 0$ where π_j^* is the optimal solution of (D) by setting b to be $-v_j$. Then, we define $n_{\mathcal{V}}$ to be the unit vector in the same direction as $-\frac{1}{|\mathcal{V}'|}\sum_{v_j \in \mathcal{V}'} \pi_j^*$. It can be verified that $n_{\mathcal{V}} \cdot v_i > 0$ for each v_i in $\mathcal{V}$ as desired.

C CANDIDATE SET MAINTENANCE

We explain how the branch-and-bound skyline (BBS) algorithm [24] is applied to maintain C. Consider conical hull pruning as an example (similar if we use hyperplane pruning). We index C using an R-tree [10] (or some other data partitioning methods) where each *intermediate entry* is a *minimum bounding rectangle (MBR)* and each *leaf entry* is a data point in C. Besides, we maintain a minimum heap of the entries in the R-tree according to their distance to a virtual point Q with $Q[i] = \max_{p \in D} p[i]$ (the "favorite" point of all u in $\mathcal{R}$). Initially, we insert the root of the R-tree into the minimum heap and the new candidate set C' is empty. The algorithm works in iterations. In each iteration, we pop the entry e from the heap with the minimum distance to Q:

(1) **Case 1 (e is a leaf entry):** If $e \notin C_{p,\mathcal{V}}$ $\forall p \in C'$, we insert e to C' and remove points in C' which are in $C_{e,\mathcal{V}}$.
(2) **Case 2 (e is an intermediate entry):** If e (an MBR) is totally contained in the conical hull of some points in C', the whole e is pruned. Otherwise, e is "expanded" and its children are inserted into the heap.

This process continues until the heap is empty and C' is the updated candidate set. Different from BBS for skyline computations, when a new e is inserted to C', we need to cross check whether some existing points in C' is in $C_{e,\mathcal{V}}$. We refer the readers not familiar with the process above to [24].

D ADDITIONAL EXPERIMENTS

D.1 User Study on Used Car Purchase

Since real users might make mistakes or provide inconsistent feedbacks, we conducted a user study on a used car database[1] to further verify the usefulness of our methods in real scenarios such as car purchase. Same as [27], we randomly selected 1000 cars from the database. Each car is described by 4 attributes, namely price, year of purchase, power and used kilometers. We recruited 30 participants, ranging in age from 18 to 30, and asked them to complete the survey. For the ease of evaluation, we only focus on MUD ($\epsilon = 0\%$) in this user study; that is, we aim at finding the favorite car for each participant among those 1000 candidate cars.

[1]https://www.kaggle.com/orgesleka/used-cars-database

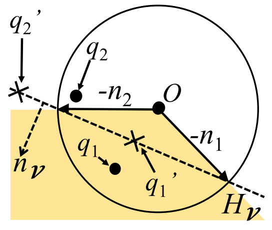


Figure 15: Fast Prune

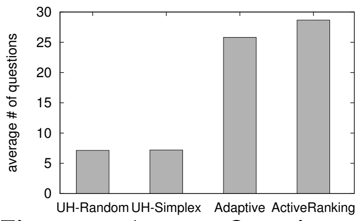


Figure 16: Average Questions

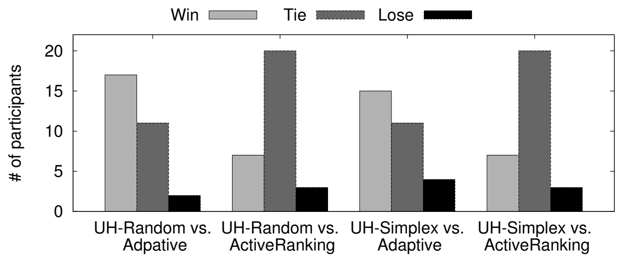


Figure 17: Comparisons between Recommended Cars

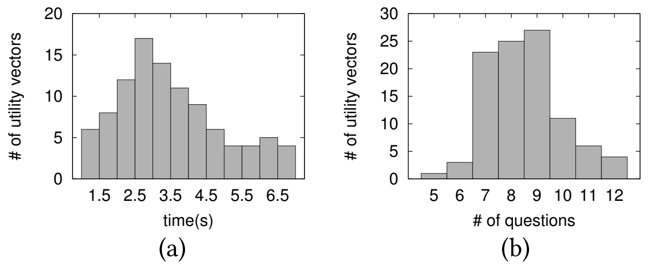


Figure 18: Distribution ($d = 4, s = 3, \epsilon = 1\%, n = 100$k)

We compared our methods, UH-Random and UH-Simplex against two existing preference learning algorithms, namely Adaptive [27] and ActiveRanking [11]. We excluded the existing IRM algorithm [22] in this experiment due to its poor performance in solving MUD, which has been demonstrated in Section 6. Since the existing algorithms [11, 27] are restricted to pairwise comparisons (each comparison corresponds to a question asked to the user), we fixed s to be 2 in our methods for fair comparison. Note that Adaptive and ActiveRanking are not designed for MUD (see Section 2). Thus, we modify them to solve MUD more effectively:

- Adaptive [27] estimates the user preference by selecting comparisons adaptively. We utilize the randomly selected comparisons to test their accuracy and stop the interaction until they correctly identify 75% comparisons (the stable accuracy in [27]). The maximum utility car w.r.t. the estimated preference is returned.
- ActiveRanking [11] identifies the ranking of cars based on their distances to the unknown favorite car. We adapt our hyperplane pruning strategy to reduce their comparisons if a car is definitely not the favorite one. Finally, the top rank car is returned as the favorite car.

We measured the performance of each algorithm by the average number of questions asked and the quality of the car returned (and times are negligible). However, it is difficult to ask the participants to provide their true favorite cars (i.e., the ground truth) by examining all 1000 candidates. Thus, we cross-compared the quality of the cars returned by our algorithms and competitor algorithms. Consider UH-Random and Adaptive as an example. Let p be the car returned by UH-Random and q be the car returned by Adaptive.

- If $p = q$, both UH-Random and Adaptive recommend the same car. We count this comparison as a *"tie"*.
- If $p \neq q$, we ask the participant for his/her preference between p and q. If s/he prefers p to q, our algorithm identifies a better car and this comparison is counted as a *"win"*. Otherwise, it is counted as a *"lose"*.

Finally, we reported the total count of win/tie/lose for each pair of algorithms over the 30 participants.

In Figure 16, we reported the average number of questions. To identify the favorite car among 1000 candidates, UH-Random and UH-Simplex ask the users 7 questions only on average. In comparison, Adaptive and ActiveRanking are not primally designed for MUD and thus, they may ask the users some less interesting/less informative questions, resulting in more questions to be asked (4 times more than us). Despite the significant reduction on the number of question asked, the cars recommended by our methods are of better quality. Specifically, even though real users can provide inconsistent feedbacks during interactions, in all four pairs of comparisons in Figure 17, UH-Random and UH-Simplex consistently outperform Adaptive and ActiveRanking by returning a better car 38% of time and returning the same car 52% of time (i.e., we return a not-worse car 90% of time).

D.2 Distributions over Different Users

In this experiment, we generated 100 utility vectors randomly and studied the performance of UH-Random over different users (UH-Simplex is similar) on a 4-dimensional dataset (other parameters are fixed to the default setting in Section 6). In Figure 18(a), we depicted the distribution of users on execution time. For all users, UH-Random finishes the computation in seconds. In particular, it only takes UH-Random 2∼4 seconds for most users. In Figure 18(b), we depicted the distribution of users on the number of questions. UH-Random asks 5∼12 questions to guarantee the regret. In particular, over 80% of users obtain a solution with at most 9 questions.

E REMAINING PROOFS

Proof of Theorem 3.3. It is easy to see that there is a dataset D such that for each p in D, p is the maximum utility point for some utility functions. Any algorithm that identifies all such points must be in the form of a tree. Consider a s-ary tree of depth r generated by r questions with at least one leaf node for each p (otherwise that point is not the maximum utility point for any utility function). Since it is a s-ary tree with at least n leaves, the height of the tree is $\Omega(\log_s n)$. In other words, any algorithm needs to ask $\Omega(\log_s n)$ questions to identify the maximum utility point in the worst case. □

Proof of Lemma 4.1. This lemma follows directly from the convexity and the linear utility functions. □

Proof of Theorem 4.4. The correctness of MEDIAN follows from Corollary 4.2. Initially, $|C| = m \leq n$ where n is the number of points in D (i.e., $|D| = n$). Since C is reduced by half in every round, it takes $O(\log_2 n)$ rounds to determine the maximum utility point. It takes $O(n \log n)$ time to determine and sort all vertices of $\mathrm{Conv}(D \cup \{b_1, b_2, O\})$. Based on the sorted vertices, each iteration of MEDIAN takes $O(1)$ time, resulting in $O(\log_2 n)$ time. Thus, the total time complexity is $O(n \log n)$. Besides, MEDIAN is strongly truthful since we always present vertices which are points in the dataset. □

Proof of Theorem 4.5. The correctness follows from Lemma 4.1. Assume that p_{i_j} is the favorite point of the user. Since $f(p_{i_j}) \geq f(p_{i_{j+1}})$, the points ordered after $p_{i_{j+1}}$, the "rightmost" point in C_j, cannot be the maximum utility point according to Lemma 4.1. That is, the maximum utility point does not lie in C_l for each $l \geq j+1$. Similarly, $f(p_{i_j}) \geq f(p_{i_{j-1}})$ implies that the maximum utility point does not lie in C_l for each $l < j-1$. Then, we focus on $C_j \cup C_{j-1}$ in the next iteration. The remaining proofs are similar to Theorem 4.4. □

Proof of Lemma 5.1. Since the user prefers p to q, $u \cdot p > u \cdot q$. i.e., $u \cdot (p - q) > 0$, which implies $u \in h^+_{p,q}$ according to the definition of $h_{p,q}$. □

Proof of Lemma 5.3. $h^+_{q,p}$ contains the set of all utility vectors such that $u \cdot q > u \cdot p$. Since $h^+_{q,p} \cap \mathcal{R} = \emptyset$, $u \cdot p \geq u \cdot q$ for each u in $\mathcal{R}$. In other words, the utility of p is at least that of q in all cases and thus, q can be safely pruned from C. □

Proof of Lemma 5.6. $q \in C_{p,\mathcal{V}}$ implies $(q-p) = \sum_{v_i \in \mathcal{V}} w_i v_i$. Then, $u \cdot (q-p) = \sum_{v_i \in \mathcal{V}} w_i u \cdot v_i = \sum_{n_i \in \mathcal{H}} w_i u \cdot (-n_i) \leq 0$ since $u \cdot n_i \geq 0$ for each h_i in $\mathcal{H}$. That is, $u \cdot p \geq u \cdot q$. □

Proof of Lemma 5.8. $q \in C_{p,\mathcal{V}}$ implies that $q-p = \sum_{v_i \in \mathcal{V}} w_i v_i$, which can be written as $(q-p) - O = \sum_{v_i \in \mathcal{V}} w_i v_i$. That is, $q - p \in C_{O,\mathcal{V}}$. □

Proof of Lemma 5.9. Let $p^* = \arg\max_{q \in D} u \cdot q$. Firstly, by following a similar analysis as in [22], $u \cdot p^* - u \cdot p \leq 2\|\mathcal{R}\|_1$. Secondly, there must exist a i^* such that $u[i^*] \geq \frac{1}{d}$. Otherwise, $\sum_{i=1}^d u[i] < 1$, which contradicts the definition of u. Let p_{i^*} be the point in D with $p_{i^*}[i^*] = 1$. Then, $u \cdot p^* = \max_{q \in D} u \cdot q \geq u \cdot p_{i^*} \geq u[i^*] p_{i^*}[i^*] \geq 1/d$. Thus, $\mathrm{rr}_D(\{p\}, u) = \frac{u \cdot p^* - u \cdot p}{u \cdot p^*} \leq \frac{2\|\mathcal{R}\|_1}{1/d} \leq 2d\|\mathcal{R}\|_1$. □

Proof of Lemma 5.10. Let p^* be the vertex in $\mathrm{Conv}(D)$ such that its utility is larger than all vertices in N_{p^*}. That is, $u \cdot p^* \geq u \cdot p$ for each p in N_{p^*}, which gives an extreme vector set $\mathcal{V} = \{p - p^* \mid \forall p \in N_{p^*}\}$. Consider the conical hull $C_{p^*,\mathcal{V}}$. It could be easily verified that for each point p in D, p is in $C_{p^*,\mathcal{V}}$ according to the definitions of vertex and convex hull. According to Lemma 5.6, $u \cdot p^* \geq u \cdot p$ for each p in D and thus, p^* is the maximum utility point w.r.t. u. □

Proof of Lemma 5.11. It follows from the convexity of $\mathrm{Conv}(D)$. If q is not a neighbouring vertex of p, $q-p$ is strictly in $C_{p,V}$. Thus, it cannot be in V_F, and vice versa. □

Proof of Theorem 5.13. UH-SIMPLEX displays points inside the database (i.e., strongly truthful). Besides, it prunes at least $s-1$ unqualified points (more points can be pruned with the pruning strategies) from C after each question. Thus, the number of questions is $O(n/s)$ in the worst case. However, UH-SIMPLEX performs much better in practice.

Recall that we maintain a vertex p with the highest utility during the interaction and we update p if there is a neighboring vertex in N_p with a higher utility (which is the *pivot step* in the SIMPLEX method for LP [8]). In the worst case, we update p $O(n)$ times since there are at most n vertices in $\mathrm{Conv}(D)$. However, according to [3], the average number of updates on p (pivot steps) is much smaller, i.e., $O(\sqrt[d]{n})$. To determine one update on p, we ask $O(\deg_{\max}/s)$ questions since there are $O(\deg_{\max})$ neighboring vertices in N_p and we can display at most $s-1$ of them per question. Thus, the number of questions is $O(\deg_{\max} \sqrt[d]{n}/s)$ on average. □

Proof of Lemma B.1. Recall that $|\mathcal{V}| = t$.

Firstly, we prove that if $q \in C_{p,\mathcal{V}}$, we have $q' = \sum_{i=1}^t w'_i v'_i$ with $w'_i \geq 0$ and $\sum_{i=1}^t w'_i = 1$. Note that $q \in C_{p,\mathcal{V}}$ implies $q - p = \sum_{i=1}^t w_i v_i$ where $w_i \geq 0$. Since $v'_i = c_i v_i$,

$$q' = c_q(q-p) = c_q \sum_{i=1}^t w_i v_i = c_q \sum_{i=1}^t \frac{1}{c_i} w_i v'_i = \sum_{i=1}^t w'_i v'_i$$

where $w'_i = \frac{c_q w_i}{c_i} \geq 0$. Since v'_i and q' lie on $H_\mathcal{V}$, $v'_i \cdot n_\mathcal{V} = c_\mathcal{V}$ and $q' \cdot n_\mathcal{V} = c_\mathcal{V}$. Then, $c_\mathcal{V} = q' \cdot n_\mathcal{V} = (\sum_{i=1}^t w'_i v'_i) \cdot n_\mathcal{V} = \sum_{i=1}^t w'_i v'_i \cdot n_\mathcal{V} = \sum_{i=1}^t w'_i c_\mathcal{V}$. That is, $\sum_{i=1}^t w'_i = 1$.

Secondly, we prove $\|v'_i\|_2 \leq 1$ for each $i \in [1, t]$. Consider v_i and its corresponding v'_i on $H_\mathcal{V}$. Denote the angle between v and $n_\mathcal{V}$ by $\theta_{<v, n_\mathcal{V}>}$. By the definition of v_i and v'_i,

$$0 < \cos(\theta_{<v'_i, n_\mathcal{V}>}) = \cos(\theta_{<v_i, n_\mathcal{V}>}) = \frac{v_i \cdot n_\mathcal{V}}{\|v_i\|_2 \|n_\mathcal{V}\|_2} = v_i \cdot n_\mathcal{V}.$$

Since v'_i lies on H, $\|v'_i\|_2 =$

$$\frac{v'_i \cdot n_\mathcal{V}}{\|n_\mathcal{V}\|_2 \cos(\theta_{<v'_i, n_\mathcal{V}>})} = \frac{c_\mathcal{V}}{\cos(\theta_{<v'_i, n_\mathcal{V}>})} = \frac{\min_{j \in [1,t]} n_\mathcal{V} \cdot v_j}{n_\mathcal{V} \cdot v_i} \leq 1.$$

By combining the results above, $\|q'\|_2 = \|\sum_{i=1}^t w'_i v'\|_2 \leq \sum_{i=1}^t \|w'_i v'\|_2 \leq \sum_{i=1}^t w'_i = 1$ and the lemma follows. □