Interactive Search with Mixed Attributes

Weicheng Wang*, Raymond Chi-Wing Wong*, Min Xie+
Hong Kong University of Science and Technology, Shenzhen Institute of Computing Sciences, Shenzhen University+*
wwangby@connect.ust.hk, raywong@cse.ust.hk, xiemin@sics.ac.cn

***Abstract*—The problem of extracting the user's favorite tuple from a large dataset has been attracting a lot of attention in the database community. Existing studies attempt to search for the target tuple with the help of user interaction. Specifically, they ask a user several questions, each of which consists of two tuples and asks the user to indicate which one s/he prefers. Based on the user feedback, the user preference is learned implicitly and the target tuple w.r.t. the learned preference is returned. However, they only consider datasets with numerical attributes (e.g., price). In practice, tuples can also be described by categorical attributes (e.g., color), where there is no trivial order in the attribute values. In this paper, we study how to find the user's favorite tuple from datasets with mixed attributes (including both numerical and categorical attributes) by interacting with the user.**

We study our problem progressively. Firstly, we inquiry a special case in which tuples are only described by categorical attributes. We present algorithm *SP-Tree* that asks an asymptotically optimal number of questions. Secondly, we explore the general case in which tuples are described by numerical and categorical attributes. We propose algorithm *GE-Graph* that performs well theoretically and empirically. Experiments are conducted on synthetic and real datasets. The results show that our algorithms outperform existing ones on both the execution time and the number of questions asked. Under typical settings, we reduce dozens of questions asked and speed up by several orders of magnitude.

***Index Terms*—user interaction, categorical attribute**

I. Introduction

A dataset usually consists of millions of tuples described by several attributes. The attributes can be classified into two types, "numerical" attributes and "categorical" attributes [1], [2], based on whether there exist fixed orders on their attribute values. For example, Table I shows a used car dataset. Each car is described by four attributes, namely brand, color, price, and horsepower. The price and horsepower are numerical attributes since there come with fixed orders on their attribute values. Users always desire a cheap car equipped with as much horsepower as possible. In comparison, the brand and color are categorical attributes since they do not refer to fixed orders for all users, i.e., different users have various preferences on the car brands or colors. For instance, one user may prefer Mercedes while another user might be inclined to BMW.

Suppose that Alice wants to buy a used car. She has a trade-off between price and brand and is willing to pay more money for a famous brand than an obscure one. To model her preference, the commonly-used decision-making strategy is to utilize numerical weights in a *linear utility function* [3]–[7]. Precisely, it quantifies Alice's criterion of interest, and characterizes a vector comprising one weight per attribute. Based on the utility function, each tuple is associated with a *utility* (i.e., a function score). It indicates to what extent Alice prefers the tuple, where a higher utility means that the tuple is more favored by her and the tuple with the highest utility is assumed to be her favorite tuple.

Table I: A used car dataset

Car ID	Brand	Color	Price	Horsepower
1	Mercedes	White	2000	300
2	BMW	White	3000	270
3	Mercedes	Black	4500	150
4	BMW	Black	5000	60

Based on the scheme, there are a lot of interactive operators proposed to assist users in finding their favorite tuples. Such operators, regarded as *multi-criteria decision-making tool*, can be applied in various domains, including purchasing a used car, buying a house, and finding a place for a trip. Specifically, they interact with a user and learn the information of the user preference (i.e., the utility function) implicitly from the user feedback. Then, they return the tuple with the highest utility based on the obtained information. However, the existing operators are only designed for the datasets with numerical attributes (e.g., price). In the scenario of purchasing a used car (which is also applied in our user study in Section VI-C), Alice's trade-off involves numerical and categorical attributes. The existing operators have difficulties in taking both types of attributes into account to recommend Alice a satisfactory car.

Similar issues also appear in many other scenarios, e.g., buying a house. Alice may weigh the location (i.e., a categorical attribute) and the size (i.e., a numerical attribute). She may consider a large house in a suburb or a small house in city centre. Only if we handle both types of attributes can we learn the user preference accurately and return an appropriate house.

Motivated by the needs of handling both types of attributes, we propose problem *Interactive Search with Mixed Attributes (ISM)*, which interacts with a user by asking questions to find the user's favorite tuple from the dataset described by mixed attributes (including both numerical attributes and categorical attributes) with as few questions asked as possible. In detail, our interactive framework follows [6]–[8]. Each question is in the form of a *pairwise comparison*. It presents two tuples that are selected based on the user's answers to previous questions, and asks the user to indicate which one s/he prefers. After collecting the user's choice, we learn the user preference implicitly and return the best tuple w.r.t. the learned preference.

The interactive framework has two characteristics worth noting. Firstly, it is based on the assumption proposed by [9] that *users can tell which tuple they prefer the most from a set of tuples*. This assumption is carefully studied by our experiments in Section VI-C. Note that it is possible that tuples

are described by a large number of attributes, which seems to cause trouble for users to answer questions. Nonetheless, [10]–[12] emphasize that users usually pay their attention to a small number attributes that are the most crucial when selecting favorite tuples. Secondly, the questions asked to a user are in the form of pairwise comparisons. In cognitive psychology, the Thurstone's Law of Comparative Judgement indicates that the pairwise comparison is the most effective way to learn the user preference [8], [13]. The user studies in [7], [8] also verified that pairwise comparisons could effectively capture how users assess multi-attributes tuples. This kind of interaction naturally appears in our daily life. For example, a seller shows Alice a black car and a white car, and asks her which one she prefers. A realtor presents two houses near the sea and the park, respectively, and asks Alice: which one do you want to live in?

In the literature of the marketing research [14], [15], it is expected that the questions asked to a user should be as few as possible. Otherwise, users may feel bored and answer questions impatiently, affecting the interaction results. Thus, our problem ISM follows the setting of existing studies [6], [7] to return one tuple instead of multiple tuples (e.g., k tuples), since the latter case requires asking much more questions. Intuitively, the former case only needs to learn the user preference between one tuple and the rest of tuples, while the latter case requires to learn the user preference between k tuples and the rest of tuples. Experiments in [7] verified that the latter case asks 4-10 times more questions than the former case. Its user study also showed that users are willing to answer fewer questions even if fewer tuples are returned. Therefore, we reduce the number of questions asked by just returning one tuple, which is sufficient in many applications, e.g., investing financial products, renting apartments, and purchasing used cars, to strike a balance between the user effort and the result size. For example, Alice plans to rent an apartment for several months. Since she will only live in one apartment for a short time, it suffices to return the best candidate. She might be frustrated due to the long selection process even if finally, several candidates are returned. Note that our proposed algorithms can be easily extended to returning multiple tuples. The extension is shown in Appendix B.

To the best of our knowledge, we are the first to study problem ISM. There are some closely related studies involving user interaction [6], [8], [16] but they are different from ours, by only considering numerical attributes. Our problem ISM involves both numerical and categorical attributes. In this sense, existing studies can be seen as a special case of ours. Note that none of the existing algorithms can be adapted to solve problem ISM satisfactorily. The categorical attributes are discrete and unordered while the numerical attributes are continuous and ordered. The strategies of existing algorithms which were designed for numerical attributes cannot be applied efficiently and effectively to the categorical attributes. In our experiments, the adapted existing algorithms from [6], [8], [16] asked many questions and ran slowly, which is troublesome. For example, on the dataset with two categorical and three numerical attributes, they asked 8 more questions and ran 2 orders of magnitude longer than our proposed algorithms.

Contributions. Our contributions are summarized as follows.

- To the best of our knowledge, we are the first to propose the problem of finding the user's favorite tuple from the dataset with mixed attributes, including both numerical and categorical attributes, by interacting with the user.
- We show a lower bound on the number of questions asked which is needed for finding the user's favorite tuple.
- We propose algorithm *SP-Tree*, for a special case of ISM in which tuples are only described by categorical attributes, which asks an asymptotically optimal number of questions.
- We propose algorithm *GE-Graph* performing well theoretically and empirically for the general case of ISM in which tuples are described by numerical and categorical attributes.
- We conducted experiments to demonstrate the superiority of our algorithms. Under typical settings, the best-known existing algorithm asked 36.1 questions in 239 seconds, which is quite troublesome. Our algorithm asked 25.9 questions in 2.2 seconds. It reduced the number of questions asked by more than 28% and accelerated by two orders of magnitude.

In the following, we discuss the related work in Section II. Section III shows the formal definition of problem ISM. In Section IV, we propose algorithm *SP-Tree* for the special case of ISM. In Section V, we present algorithm *GE-Graph* for the general case of ISM. Extensive experiments are shown in Section VI. Finally, Section VII concludes our paper.

II. Related Work

There are many queries attempting to learn the user preference by interacting with a user. [9] proposed the interactive regret minimizing query. It defined a criterion called the *regret ratio* (which evaluates how regretful a user is when s/he sees the returned tuples instead of the whole dataset), and targeted to return a small size of output while minimizing the regret ratio. However, [9] displayed *fake tuples*, which are artificially constructed (not selected from the dataset), in each question when interacting with a user. This artificial construction might produce unrealistic tuples (e.g., a car with 10USD and 60000 horsepower) and the user may be disappointed if the displayed tuples with which s/he is satisfied do not exist. Besides, it is impractical for users to truly evaluate fakes tuples [6], resulting in the difficulty of applying the algorithm of [9] in real-world applications. Based on these defects, [6] proposed the strongly truthful interactive regret minimization that utilizes *real tuples* (selected from the dataset). Unfortunately, [6], [9] only considered datasets described by numerical attributes. There exist obstacles to applying them to many scenarios that involve both numerical attributes and categorical attributes.

[8] proposed algorithm *Adaptive* that approximates the user preference with the help of user interaction. Nevertheless, instead of recommending tuples, it focused on learning the user preference and thus, it might ask many redundant questions in which our problem ISM is not interested. For example, if Alice prefers car p_1 to both p_2 and p_3, her preference between p_2 and p_3 is less interesting in our problem, but this additional

information might be useful in [8]. [8] proposed to map categorical values to numerical attributes using the standard SVM convention, which is widely used in machine learning area. In this way, the existing algorithms designed for numerical attributes can be adapted to working on the datasets with both categorical and numerical attributes. However, as presented in Section VI, the experimental results show that the adapted algorithms from [6], [8], [16] perform poorly, by asking dozens more questions and taking orders of magnitude longer time than ours. The main reason is that the categorical attributes are in the discrete space but the numerical attributes are in the continuous space. Even if the categorical attributes are transformed to numerical attributes, their nature is unchanged and thus, the optimization strategies for continuous space are not as effective as in discrete space. [17] studied the problem of finding the user's favorite tuple by interacting with a user on the dataset with ordered and unordered attributes. However, the ordered and unordered attribute values are numbers (e.g., the number of seats in a car). Thus, its proposed algorithms rely on the continuous property of attributes and are difficult to be applied to the datasets with categorical attributes.

In the machine learning area, our problem is related to the *learning to rank* problem [16], [18]–[20], which learns the ranking of tuples by pairwise comparison. However, most of the existing algorithms [18]–[20] failed to utilize the inter-relation between tuples. For example, attribute "price" is an inter-relation between cars. Given a \$3000 car p and a \$5000 car q, p is generally considered to be better than q because of the lower price. The existing algorithms neglected this relation and directly asked a question to learn the priority of p and q instead. Thus, they required asking many questions that are unnecessary in our problem ISM. [16] considered the inter-relation between tuples to learn the ranking. However, it was only based on the numerical attributes. Besides, it focused on deriving the order for all pairs of tuples, which needs to ask excessive questions that are not concerned by our problem ISM due to the similar reason stated for [8].

In the information retrieval and search area, there also exist relevant text algorithms, e.g., Rabin Karp [21] and Rossetto [22], since categorical attributes usually show in the form of text. However, the algorithms only pay attention to the text matching. For example, Rabin Karp was designed to find all the sub-strings that are equal to a given pattern and Rossetto studied fuzzy name matching. They neither consider learning the user preference on the text nor involve any user interaction.

Our work is to return the user's favorite tuple from the dataset described by both numerical and categorical attributes by interacting with the user via real tuples. Compared with existing studies, we have the following advantages. (1) We use real tuples during the interaction (unlike [9] that utilizes fake tuples). (2) We consider both numerical attributes and categorical attributes. The existing interactive studies [6], [9], [16] only handle numerical attributes. They can be seen as a special case of our work when the categorical attributes are excluded from consideration. (3) We only involve a few easy questions. Firstly, existing studies like [8], [16], [18]–[20] ask a lot of questions since they either require learning a full ranking of tuples or aim at learning the exact user preference. Secondly, [18]–[20] fail to utilize the inter-relation between tuples and thus, involve a lot of unnecessary interaction.

III. Problem Definition

A. Terminologies

The input to our problem is a set $\mathcal{D}$ of n tuples. Each tuple $p = (p_{cat}[1], p_{cat}[2], ..., p_{cat}[d_{cat}], p_{num}[1], ..., p_{num}[d_{num}])$ is described by d mixed attributes, including d_{cat} categorical attributes and d_{num} numerical attributes ($d = d_{cat} + d_{num}$). For convenience, let $p_{num} = (p_{num}[1], ..., p_{num}[d_{num}])$ and $p_{cat} = (p_{cat}[1], ..., p_{cat}[d_{cat}])$. We denote the number of possible values (i.e., cardinality) in the i-th categorical attribute by s_i for each $i \in [1, d_{cat}]$. Following [6]–[8], we assume that all attributes are independent. The correlated attributes can be handled by considering them together as one attribute.

Example 1. Consider Table II(a). Each tuple p has two categorical attributes and two numerical attributes. Thus, $d_{cat} = d_{num} = 2$. Since each categorical attribute has 2 possible values (i.e., the first one has A_1 and A_2, and the second one has B_1 and B_2), we have $s_1 = s_2 = 2$. □

Following the setting of existing studies [6]–[8], [23], we model the user preference as a *linear utility function*. The linear utility function becomes the most proliferate and effective representation since the inception of utility modeling [3], [4]. The user study conducted by [7], [8] also verified that the linear utility function can effectively capture how real users evaluate tuples. Formally, a linear utility function f is defined to be $f(p) = \sum_{i=1}^{d_{cat}} u_{cat}[i]h(p_{cat}[i]) + \sum_{j=1}^{d_{num}} u_{num}[j]p_{num}[j]$.

- Function $h : p_{cat}[i] \rightarrow \mathbb{R}_+$ maps each categorical value to a real number which indicates to what extent a user favors the categorical value. A larger real number means that the categorical value is more preferred by the user.
- Element $u_{cat}[i]$ (resp. $u_{num}[j]$) measures the importance of the i-th categorical (resp. the j-th numerical) attribute to the user. For the ease of representation, we denote the elements by two vectors in utility function f: *the categorical utility vector* $u_{cat} = (u_{cat}[1], ..., u_{cat}[d_{cat}])$ and *the numerical utility vector* $u_{num} = (u_{num}[1], ..., u_{num}[d_{num}])$. Without loss of generality, following [6], [24], we assume that $\sum_{j=1}^{d_{num}} u_{num}[j] = 1$ and call the domain of u_{num} *the numerical utility space*. Note that u_{cat} and u_{num} are correlated. If we assume that the sum of elements of u_{num} is equal to 1, we cannot make the same assumption for u_{cat}. There are other ways to normalize the utility vectors, e.g., $\sum_{i=1}^{d_{cat}} u_{cat}[i] + \sum_{i=j}^{d_{num}} u_{num}[j] = 1$. Our method considers that p_{num} is fixed while function h varies for different users.
- Function value $f(p)$, called the *utility* of p w.r.t. f, represents how much a user favors tuple p. The tuple that has the highest utility is considered to be the user's favorite tuple.

To better interact with users, we give two settings to optimize f for categorical and numerical attributes, respectively.

The first setting is concerned with the categorical attributes. In utility function f, each term related to the categorical

Table II: Dataset, utilities and function g

p	$p_{cat}[1]$	$p_{cat}[2]$	$p_{num}[1]$	$p_{num}[2]$	$f(p)$
p_1	A_1	B_1	0.4	1	1.30
p_2	A_2	B_1	0.6	0.9	1.45
p_3	A_1	B_2	0.9	0.5	1.40
p_4	A_2	B_2	1	0.2	1.40

(a) Table

$g(\cdot)$
$g(A_1) = 0.2$
$g(A_2) = 0.3$
$g(B_1) = 0.4$
$g(B_2) = 0.5$

(b) $g(\cdot)$

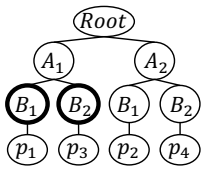


Figure 1: C-Tree

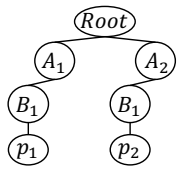


Figure 2: Pruned C-Tree

attributes is shown in the form of $u_{cat}[i]h(p_{cat}[i])$, where $i \in [1, d_{cat}]$. *Both* $u_{cat}[i]$ and $h(p_{cat}[i])$ are unknown at the beginning (i.e., we have *two* unknown variables). Since knowing either one of them does not suffice to determine the user's favorite tuple, we directly study their co-influencing effect without considering the interplay between $u_{cat}[i]$ and $h(p_{cat}[i])$. Specifically, we use a *single* function g to denote both $u_{cat}[i]$ and $h(p_{cat}[i])$ (i.e., we use a *single* unknown variable) such that function $g : p_{cat}[i] \to \mathbb{R}_+$ is defined to be: $g(p_{cat}[i]) = u_{cat}[i]h(p_{cat}[i])$ and $i \in [1, d_{cat}]$. In this way, we focus on learning $g(p_{cat}[i])$ instead of $u_{cat}[i]$ and $h(p_{cat}[i])$. This will help to reduce the number of questions asked to a user since we intend to learn one variable rather than two.

The second setting refers to the numerical attributes. Consider datasets $\mathcal{D} = \{p_1, p_2, ..., p_n\}$ and $\mathcal{D}' = \{p'_1, p'_2, ..., p'_n\}$. Suppose that $\mathcal{D}'$ is a numerical rescaling of $\mathcal{D}$, i.e., there exist positive real numbers λ_1, λ_2, ..., $\lambda_{d_{num}}$ such that $p'_{k\ num}[j] = \lambda_j p_{k\ num}[j]$, $\forall k \in [1, n]$ and $\forall j \in [1, d_{num}]$.

Lemma 1. Given a utility function f for $\mathcal{D}$, there exists a corresponding utility function f' for $\mathcal{D}'$ such that $\forall p_k, p_l \in \mathcal{D}$, if $f(p_k) > f(p_l)$, then $f'(p'_k) > f'(p'_l)$, where $k, l \in [1, n]$.

Due to the lack of space, the complete proofs of some theorems/lemmas in this paper can be found in Appendix D. Lemma 1 indicates that the numerical rescaling on the numerical attributes does not change the ranking of tuples w.r.t. the utility function. Thus, we assume that each numerical attribute is normalized to $(0, 1]$ and a larger value is more favored by users. If a smaller value is more preferable in an attribute (e.g., price), the attribute can be modified by subtracting each value from 1 to satisfy the assumption.

Example 2. Let us continue Example 1. Assume that $u_{num} = (0.5, 0.5)$. Utility function f for a tuple p is expressed as $f(p) = g(p_{cat}[1]) + g(p_{cat}[2]) + 0.5 \times p_{num}[1] + 0.5 \times p_{num}[2]$. Suppose that function g has its values over different categorical values (i.e., A_1, A_2, B_1, and B_2) as shown in Table II(b). Since $g(A_2) = 0.3$ and $g(B_1) = 0.4$, the utility of p_2 w.r.t. f is $f(p_2) = 0.3 + 0.4 + 0.5 \times 0.6 + 0.5 \times 0.9 = 1.45$. The utilities of the other tuples can be computed similarly. Tuple p_2 is the user's favorite tuple, since it has the highest utility w.r.t. f. □

B. Problem ISM

Our interactive framework follows [6]–[9] and works on the dataset with mixed attributes. Specifically, we interact with a user for rounds until we determine the user's favorite tuple. Each round mainly consists of three components. (1) *Tuple selection*. Based on the user's answers in previous rounds, we select two tuples as a question presented to the user and ask him/her to pick the one s/he prefers. Each tuple is a combination of categorical and numerical values. For example, in the purchasing cars scenario, Alice might be presented with two cars: <Mercedes, White, $2000, 300> and <BMW, Black, $5000, 60>, where the last value in the combination represents the horsepower. (2) *Information maintenance*. According to the user's choice, we update the information maintained for learning the user preference. (3) *Stopping condition*. We check whether the maintained information is sufficient. If it satisfies the stopping condition, we terminate the interaction process and return the result. Otherwise, we start a new interactive round. Formally, we are interested in the following problem.

Problem 1. (**Interactive Search with Mixed Attributes (Problem ISM)**) Given a tuple set $\mathcal{D}$ described by mixed attributes, we want to ask a user as few questions as possible to determine the user's favorite tuple in $\mathcal{D}$.

Theorem 1. There exists a dataset of n tuples such that any algorithm needs to ask $\Omega(s_1 + \prod_{i=2}^{d_{cat}} s_i^2 + \log(n - s_1 - \prod_{i=2}^{d_{cat}} s_i^2))$ questions to determine the user's favorite tuple.

IV. ISM Considering Categorical Attributes Only

In this section, we consider a special case of ISM in which tuples are only described by categorical attributes. This case exists in multiple scenarios, e.g., job application, online dating, and customer service [25]. For example, in the job application scenario, the candidates may provide their gender, speciality, and hobbies, which are all categorical attributes. In the online dating scenario, users of online dating platforms are usually required to input categorical data to match them with the right person. The data may include appearance, job and personality.

We present algorithm *SP-Tree* that asks an asymptotically optimal number of questions. Intuitively, we maintain tuples in a tree data structure, called *categorical value tree*, or *C-Tree* in short. During the interaction, tuples are continually selected from the C-Tree as questions to be asked to a user (tuple selection). Based on the user's answer, we update the C-Tree by pruning from it the tuples which cannot be the user's favorite tuple (information maintenance). When there is only one tuple left in the C-Tree, we stop and return the tuple as the answer (stopping condition). In the following, we elaborate in detail how we maintain tuples in the C-Tree in Section IV-A, and how we select tuples as questions from the C-Tree in Section IV-B. The theoretical analysis is shown in Section IV-C.

A. Information Maintenance

Since tuples are described by categorical attributes only, we define a tree data structure *C-Tree* to maintain tuples

based on the categorical attributes. It has $d_{cat}+2$ levels and satisfies the following properties. (1) The root is in the 0-th level. (2) Each node in the i-th level stores a categorical value in the i-th categorical attribute, where $i \in [1, d_{cat}]$. It is possible that different nodes store the same categorical value. Note that we allow attributes to be appeared in the C-Tree in arbitrary orders, e.g., attributes with small cardinalities appear in low-levels of the C-Tree. (3) Each tuple in the dataset is recorded in a leaf (i.e., the node in the $(d_{cat}+1)$-th level). There is only one path from the root to the leaf and the categorical values of the tuple are stored in the path in order.

Example 3. Figure 1 shows a C-Tree. The nodes in the first-level store the categorical values in the first attribute (i.e., A_1 and A_2). The nodes in the second-level store the categorical values in the second attribute (i.e., B_1 and B_2). Each tuple (i.e., p_1, p_2, p_3, and p_4) is stored in a leaf, e.g., p_1 is stored in the leftmost leaf. The path from the root to the leftmost leaf contains the categorical values of p_1 (i.e., A_1 and B_1). □

Construction of C-Tree. The C-Tree is constructed progressively. It starts with a root and the tuples in $\mathcal{D}$ are inserted into it one by one. For each tuple $p \in \mathcal{D}$, we traverse the C-Tree once from the root to the bottom. Suppose that the traversal is at a node N in the $(i-1)$-th level, where $i \in [1, d_{cat}]$. We check the children of N. If there is a child of N which stores the categorical value equal to $p_{cat}[i]$, we move to this child. Otherwise, we create a new child for N to store $p_{cat}[i]$ and move to this new child. The traversal process continues until we reach a node N' in the d_{cat}-th level. Then, we create a child (i.e., a leaf) for node N' to record tuple p.

Example 4. Consider Table II. Assume that the tuples are only described by categorical attributes. The C-Tree begins with a root. Let us insert tuple p_1 into the C-Tree. Since the root does not contain any child, we build (1) 2 new nodes in the first and second levels, respectively, which include A_1 and B_1 in order, and (2) a leaf that contains p_1. The other tuples can be inserted similarly. The C-Tree is shown in Figure 1. □

Update on C-Tree. For the ease of representation, let S_p^i denote the set that contains the values of p from the i-th categorical attribute to the d_{cat}-th categorical attribute, i.e., $S_p^i = \{p_{cat}[i], p_{cat}[i+1], ..., p_{cat}[d_{cat}]\}$. For example, in Table II, $S_{p_1}^2 = \{B_1\}$. Consider two tuples $p, q \in \mathcal{D}$ which are the same in the first $i-1$ categorical attributes, i.e., $\forall j \in [1, i-1]$, $p_{cat}[j] = q_{cat}[j]$. We present them as a question to a user.

Lemma 2. If a user prefers p to q, $\sum_{c \in S_p^i} g(c) > \sum_{c \in S_q^i} g(c)$.

For simplicity, let us denote $\sum_{c \in S_p^i} g(c) > \sum_{c \in S_q^i} g(c)$ by $S_p^i \succ S_q^i$ in the following. It indicates that the user favors more the categorical values in S_p^i than those in S_q^i.

Example 5. In Figure 1, tuples p_1 and p_3 have the same value A_1 in the first categorical attribute. If a user prefers p_1 to p_3, then $S_{p_1}^2 \succ S_{p_3}^2$ (where $S_{p_1}^2 = \{B_1\}$ and $S_{p_3}^2 = \{B_2\}$). □

Based on the learned information (i.e., $S_p^i \succ S_q^i$), we update the C-Tree by pruning tuples which cannot be the favorite one.

Lemma 3. Suppose that $S_p^i \succ S_q^i$. If tuples p and q are the same in the first $i-1$ categorical attributes, q cannot be the user's favorite tuple and thus, it is pruned from the C-Tree.

Intuitively, for the first $i-1$ categorical attributes, tuples p and q have the same values. For the remaining categorical attributes, since $S_p^i \succ S_q^i$, the user favors more the remaining values of p than those of q. Thus, the user must prefer p to q.

Example 6. In Figure 1, suppose that $S_{p_1}^2 \succ S_{p_3}^2$. Since p_1 and p_3 have the same categorical value A_1 in the first attribute, tuple p_3 cannot be the user's favorite tuple and thus, p_3 is pruned from the C-Tree. Consider tuples p_2 and p_4. Since $S_{p_2}^2 = S_{p_1}^2$ and $S_{p_4}^2 = S_{p_3}^2$, we derive $S_{p_2}^2 \succ S_{p_4}^2$ (we will show the derivation rules later). Since p_2 and p_4 have the same categorical value A_2 in the first attribute, tuple p_4 cannot be the user's favorite tuple and thus, p_4 is also pruned from the C-Tree. The updated C-Tree is shown in Figure 2. □

B. Tuple Selection

We learn the user preference on categorical values from the bottom to the top of the C-Tree. Concretely, we process the C-Tree level by level, starting from the d_{cat}-th level. During the process, we move to the level closest to the root (i.e., the i-th level with the smallest value of i, where $i \in [1, d_{cat}]$) such that each node in the level can only reach one leaf. For example, in Figure 2, we move to the first level since it is the level closest to the root and each node in the first level can reach only one leaf (i.e., the node that stores categorical value A_1 (resp. A_2) can only reach one leaf containing p_1 (resp.p_2)).

Suppose that we are at the i-th level of of the C-Tree, where $i \in [1, d_{cat}]$. Let $\mathbf{S}_i = \{S_p^i \mid \text{Tuple } p \text{ is in the C-Tree}\}$. Note that there might be tuples, e.g., p and q, such that $S_p^i = S_q^i$. We only maintain one set (S_p^i or S_q^i) in $\mathbf{S}_i$. Assume that $\mathbf{S}_i = \{S_{q_1}^i, S_{q_2}^i, ..., S_{q_{m_i}}^i\}$. Our algorithm scans from $S_{q_2}^i$ to $S_{q_{m_i}}^i$. For each set $S_{q_j}^i$, where $j \in [2, m_i]$, we consider sets $S_{q_1}^i, S_{q_2}^i, ..., S_{q_{j-1}}^i$ one by one. Suppose that we are at $S_{q_k}^i$, where $k \in [1, j-1]$. We check if there exist two tuples p and q in the C-Tree such that (1) p and q are the same in the first $i-1$ attributes, and (2) $S_p^i = S_{q_j}^i$ and $S_q^i = S_{q_k}^i$. If such p and q exist, we present p and q as a question to the user. Note that $|S_p^i \cap S_q^i|$ ranges from 0 to $i-1$ and thus, the user may require to consider multiple categorical attributes to make a decision.

Example 7. For the second level of the C-Tree in Figure 1, $\mathbf{S}_2 = \{S_{p_1}^2, S_{p_3}^2\}$ ($S_{p_1}^2 = \{B_1\}$ and $S_{p_3}^2 = \{B_2\}$). $S_{p_2}^2$ (resp. $S_{p_4}^2$) is not stored in $\mathbf{S}_2$ since $S_{p_1}^2 = S_{p_2}^2$ (resp. $S_{p_3}^2 = S_{p_4}^2$). Based on our tuple selection strategy, we consider sets $S_{p_1}^2$ and $S_{p_3}^2$, and select p_1 and p_3 as a question presented to the user. □

When obtaining an initial relation $S_p^i \succ S_q^i$, we can derive more relations based on it as follows to update the C-Tree.

Firstly, we ensure the *transitivity* of user preference on the categorical values. For instance, if $S_{p'}^i \succ S_p^i$ and $S_p^i \succ S_q^i$, then we obtain a new relation $S_{p'}^i \succ S_q^i$. Formally, we define the following derivation rules based on $S_p^i \succ S_q^i$.

1) If $\exists S_{p'}^i \in \mathbf{S_i}$ such that $S_{p'}^i \succ S_p^i$, then $S_{p'}^i \succ S_q^i$.
2) If $\exists S_{q'}^i \in \mathbf{S_i}$ such that $S_q^i \succ S_{q'}^i$, then $S_p^i \succ S_{q'}^i$.

Algorithm 1: The *SP-Tree* Algorithm

Input: A tuple set $\mathcal{D}$
Output: The user's favorite tuple in $\mathcal{D}$
Build the C-Tree for all tuples in $\mathcal{D}$
for $i \leftarrow d_{cat}$ *to* 1 **do**
  Build set $\mathbf{S}_i = \{S_{q_1}, S_{q_2}, ..., S_{q_{m_i}}\}$
  for $j \leftarrow 2$ *to* m_i **do**
    for $k \leftarrow 1$ *to* $j-1$ **do**
      if $\exists p, q$ *in the C-Tree such that they are the same in the first* $i-1$ *attributes, and* $S_p^i = S_{q_j}^i$ *and* $S_q^i = S_{q_k}^i$ **then**
        Present p and q to the user
        Update the C-Tree accordingly
return The only tuple left in the C-Tree

Example 8. Suppose that there are three sets $S_{q_1}^i = \{B_1\}$, $S_{q_2}^i = \{B_2\}$, and $S_{q_3}^i = \{B_3\}$. Assume that $S_{q_1}^i \succ S_{q_2}^i$ (i.e., $\{B_1\} \succ \{B_2\}$). If we learn that $S_{q_2}^i \succ S_{q_3}^i$ (i.e., $\{B_2\} \succ \{B_3\}$), we can derive $S_{q_1}^i \succ S_{q_3}^i$ (i.e., $\{B_1\} \succ \{B_3\}$). □

Secondly, since S_p^i and S_q^i may have the same values, we only consider their distinct values to derive new relations.

3) If $\exists S_{p'}^i, S_{q'}^i \in \mathbf{S_i}$ such that $S_{p'}^i \setminus S_{q'}^i = S_p^i \setminus S_q^i$ and $S_{q'}^i \setminus S_{p'}^i = S_q^i \setminus S_p^i$, then $S_{p'}^i \succ S_{q'}^i$.

Example 9. Suppose that there are four sets $S_{q_1}^i = \{A_1, B_1\}$, $S_{q_2}^i = \{A_1, B_2\}$, $S_{q_3}^i = \{A_2, B_1\}$, and $S_{q_4}^i = \{A_2, B_2\}$. If we learn that $S_{q_1}^i \succ S_{q_2}^i$ (i.e., $\{A_1, B_1\} \succ \{A_1, B_2\}$), we can derive $S_{q_3}^i \succ S_{q_4}^i$ (i.e., $\{A_2, B_1\} \succ \{A_2, B_2\}$). □

Based on the derivation rules, we derive more relations based on $S_p^i \succ S_q^i$ as follows. We build a queue Q storing the relations based on which we will derive more relations. Initially, we insert relation $S_p^i \succ S_q^i$ into Q. Then, we continually pop out relations from Q until Q is empty. For each popped out relation, we use the above three rules to derive more relations and insert the derived relations (if any) into Q.

C. Analysis

The pseudocode of our algorithm *SP-Tree* is shown in Algorithm 1. The theoretical analysis is presented in Theorem 2.

Theorem 2. Algorithm *SP-Tree* solves the special case of ISM by asking a user $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions.

Corollary 1. *SP-Tree* is asymptotically optimal in terms of the number of questions asked for the special case of ISM.

V. ISM Considering Both Categorical and Numerical Attributes

We consider the general case of ISM in which tuples are described by categorical and numerical attributes, and propose algorithm *GE-Graph*. At a high-level, we maintain three data structures. The first one is a numerical range $\mathcal{R} \subseteq \mathbb{R}^{d_{num}}$, which maintains the learned information of the user preference on numerical attributes. The second one is a hypergraph $\mathcal{G}$, which maintains the learned information of the user preference on categorical attributes. Here, we use two distinct data structures to handle numerical and categorical attributes, respectively, since they possess different characteristics. The numerical attributes are ordered and continuous but the categorical attributes are unordered and discrete. The third one is a tuple set $\mathcal{C}$ comprising tuples in $\mathcal{D}$ which are candidates as the final answer. During the interaction, we select tuples from $\mathcal{C}$ as questions asked to a user (tuple selection). According to the user feedback, we update $\mathcal{R}$ and $\mathcal{G}$ accordingly, and prune the tuples from $\mathcal{C}$ which cannot be the user's favorite tuple based on the updated $\mathcal{R}$ and $\mathcal{G}$ (information maintenance). When there is only one tuple left in $\mathcal{C}$, we stop the interaction and return the tuple as the answer (stopping condition). In the following, we first discuss the preliminaries in Section V-A. Then, we present the information maintenance and tuple selection strategies of algorithm *GE-Graph* in Section V-B and Section V-C, respectively. The theoretical analysis appears in Section V-D.

A. Preliminaries

Hyper-plane. In a d_{num}-dimensional geometric space $\mathbb{R}^{d_{num}}$, given a d_{num}-length vector p in $\mathbb{R}^{d_{num}}$, we can build a *hyper-plane* h_p: $\{r \in \mathbb{R}^{d_{num}} \mid r \cdot p = 0\}$, which passes through the origin with its unit normal in the same direction as p [26]. Hyper-plane h_p divides $\mathbb{R}^{d_{num}}$ into two halves, called *half-spaces* [26]. The half-space above h_p (resp. below h_p), denoted by h_p^+ (resp. h_p^-), contains all the points $r \in \mathbb{R}^{d_{num}}$ such that $r \cdot p > 0$ (resp. $r \cdot p < 0$). In geometry, a *polyhedron* $\mathcal{P}$ is an intersection of a set of hyper-planes and half-spaces [26].

Multiset. A *multiset* L is a collection of elements, in which elements are allowed to repeat [27], [28]. The number of times an element A repeats in a multiset L is called the *multiplicity* of A and is denoted by $\mathfrak{m}_L(A)$. Note that the multiplicity can be negative ($\mathfrak{m}_L(A) < 0$) or zero ($\mathfrak{m}_L(A) = 0$). The former can be interpreted as the number of times an element *absent* in a multiset. The latter means that an element is not in the multiset. We use multisets to store categorical values as elements. A multiset is represented as $L = [A_1, A_2, ...]_{\mathfrak{m}_L(A_1), \mathfrak{m}_L(A_2), ...}$. For example, $[A_1, A_2]_{-1,2}$ denotes a multiset with 1 absences of A_1 and 2 repeats of A_2. Let L_1 and L_2 be two multisets. There are several operations on multisets L_1 and L_2:

- $L = L_1 \oplus L_2$ is the multiset containing the elements in L_1 or L_2, where $\forall A \in L$, $\mathfrak{m}_L(A) = \mathfrak{m}_{L_1}(A) + \mathfrak{m}_{L_2}(A)$. For example, if $L_1 = [A_2, A_3]_{-2,-1}$ and $L_2 = [A_2, A_3]_{2,2}$, then $L = [A_3]_1$. For convenience, we denote $L_1 \oplus L_1$ by $2L_1$.
- The inverse of a multiset L, denoted by $\overline{L}$, contains exactly the elements of L, where $\forall A \in \overline{L}$, $\mathfrak{m}_{\overline{L}}(A) = -\mathfrak{m}_L(A)$. For example, if $L = [A_1, A_2]_{1,-2}$, then $\overline{L} = [A_1, A_2]_{-1,2}$. For simplicity, we denote $L_1 \oplus \overline{L_2}$ by $L_1 \ominus L_2$.

B. Information Maintenance

1) Numerical Utility Range $\mathcal{R}$: Recall that $\sum_{i=1}^{d_{num}} u_{num} = 1$. The user's numerical utility vector u_{num} can be seen as a point in space $\mathbb{R}^{d_{num}}$. We maintain a polyhedron $\mathcal{R}$ in $\mathbb{R}^{d_{num}}$, called the *numerical utility range*, which contains u_{num}. Initially, $\mathcal{R}$ is the entire numerical utility space, i.e., $\mathcal{R} = \{r \in$

$\mathbb{R}_+^{d_{num}} | \sum_{i=1}^{d_{num}} r[i] = 1\}$. During the interaction, we interact with a user and build hyper-planes to update $\mathcal{R}$ in two ways.

Firstly, consider tuples $p, q \in \mathcal{D}$ presented as a question to a user, where $\forall i \in [1, d_{cat}]$, $p_{cat}[i] = q_{cat}[i]$. We build a hyper-plane $h_{p_{num}-q_{num}}$ and update $\mathcal{R}$ following Lemma 4. In the following, if p and q are the same in all categorical attributes, we denote the hyper-plane by h_{p-q} for simplicity.

Lemma 4. If a user prefers p to q, $\mathcal{R}$ is updated to be $\mathcal{R} \cap h_{p-q}^+$.

Example 10. In Figure 3, $d_{num} = 2$. $\mathcal{R}$ is initialized to be a line segment. Suppose that $p_{num} = (\frac{3}{4}, \frac{1}{4})$ and $q_{num} = (\frac{1}{4}, \frac{3}{4})$. If a user prefers p to q, we build a hyper-plane h_{p-q} based on $p_{num} - q_{num}$ and $\mathcal{R} \leftarrow \mathcal{R} \cap h_{p-q}^+$ (right line segment). □

Secondly, $\mathcal{R}$ can also be updated based on the *relational graph* $\mathcal{G}$. We postpone its description to the next section.

2) Relational Graph $\mathcal{G}$: We maintain a hyper-graph $\mathcal{G} = (V, E)$ [29], [30], called the *relational graph*, to store the relations between categorical values based on the learned user preference (e.g., the difference between $g(A_1)$ and $g(A_2)$, where A_1 and A_2 are two categorical values). V is a set of nodes and E is a set of hyper-edges each of which connects 3 nodes in V. In the following, we first define $\mathcal{G}$ in terms of nodes and hyper-edges, and then show the way of updating $\mathcal{G}$.

Node. Given two tuples $p, q \in \mathcal{D}$ that have different values in at least one categorical attribute, we build a node v containing the following information. (1) A multiset $L = L_1 \ominus L_2$ (or $L_1 \oplus \overline{L_2}$). L_1 (resp. L_2) is a multiset that stores all the categorical values of p (resp. q) as elements with multiplicity 1. (2) Pair $\langle p, q \rangle$. If multiple pairs correspond to the same multiset, they are stored in the same node. (3) Several *upper bounds* and *lower bounds*, which describe the user preference on the categorical values in the multiset. Consider two tuples $p, q \in \mathcal{D}$.

Lemma 5. If a user prefers p to q (i.e., $f(p) > f(q)$), we obtain an inequality $\sum_{A \in L} \mathfrak{m}_L(A) g(A) > u_{num} \cdot (q_{num} - p_{num})$.

With a slight abuse of notations, we denote the inequality in Lemma 5 by $g(L) > u_{num} \cdot (q_{num} - p_{num})$ for simplicity and call $u_{num} \cdot (q_{num} - p_{num})$ the lower bound of $g(L)$. Similarly, if a user prefers q to p, we can derive an upper bound $u_{num} \cdot (q_{num} - p_{num})$ for $g(L)$, i.e., $g(L) < u_{num} \cdot (q_{num} - p_{num})$.

Example 11. Consider tuples p_1 and p_2 in Table II that have categorical values A_1, B_1 and A_2, B_1, respectively. We build a node v_1 in Figure 5 that stores multiset $L = [A_1, A_2]_{1,-1}$ and pair $\langle p_1, p_2 \rangle$. Since pair $\langle p_3, p_4 \rangle$ corresponds to the same multiset L, it is also stored in node v_1. If a user prefers p_1 to p_2, we obtain a lower bound $u_{num} \cdot (p_{2\ num} - p_{1\ num})$ for $g(L)$, i.e., $g(A_1) - g(A_2) > u_{num} \cdot (p_{2\ num} - p_{1\ num})$. Similarly, we can build the other nodes shown in Figure 5. □

There may be many upper or lower bounds in each node, which affects the execution time and the storage space. To save cost, we do not store the *untight bounds*. In the following, when p and q are clear, we denote a bound (e.g., $u_{num} \cdot (p_{num} - q_{num})$) by $u_{num} \cdot \triangle x$ or $u_{num} \cdot \triangle y$ for simplicity.

Definition 1. $u_{num} \cdot \triangle x$ is an *untight lower bound* for $g(L)$ if $\forall r \in \mathcal{R}$, there is a lower bound $u_{num} \cdot \triangle x'$ for $g(L)$ such that $r \cdot (\triangle x' - \triangle x) > 0$. Similarly for upper bounds in Appendix A.

Intuitively, since $u_{num} \in \mathcal{R}$, if $u_{num} \cdot \triangle x$ is untight, there must be a lower bound $u_{num} \cdot \triangle x'$ such that $u_{num} \cdot (\triangle x' - \triangle x) > 0$, i.e., $u_{num} \cdot \triangle x'$ bounds $g(L)$ more tightly than $u_{num} \cdot \triangle x$. It is easy to see that if $\mathcal{R}$ is smaller, there will be higher possibility that the condition (i.e., $\forall r \in \mathcal{R}$, there is a lower bound $u_{num} \cdot \triangle x'$ for $g(L)$ such that $r \cdot (\triangle x' - \triangle x) > 0$) is satisfied. Thus, if $\mathcal{R}$ becomes smaller, a lower bound might turn to be an untight bound. Similar to upper bounds.

We utilize the linear programming algorithm (LP) to check if a lower bound $u_{num} \cdot \triangle x$ is untight (similarly for upper bounds in Appendix A). Specifically, we define a variable w to be the objective value. Assume that there are l other lower bounds $u_{num} \cdot \triangle x_i$ for $g(L)$, where $i \in [1, l]$. For each $u_{num} \cdot \triangle x_i$, we build a constraint $r \cdot (\triangle x_i - \triangle x) < w$, where $r \in \mathcal{R}$. Formally, we construct an LP as follows.

$$\begin{aligned} \text{minimize} \quad & w \\ \text{subject to} \quad & r \cdot (\triangle x_i - \triangle x) < w, \text{ where } i \in [1, l] \\ & r \in \mathcal{R} \end{aligned}$$

If the objective value $w > 0$, then $\forall r \in \mathcal{R}$, $\exists i \in [1, l]$ such that $r \cdot (\triangle x_i - \triangle x) > 0$. Thus, $u_{num} \cdot \triangle x$ is an untight lower bound. Since there are l constraints and d_{num} variables, an LP solver (e.g., Simplex [31]) needs $O(r^2 d_{num})$ time in practice. It is costly if l is large. Therefore, we propose a sufficient condition for identifying untight bounds. At a high-level, we try to find only *one* lower bound $u_{num} \cdot \triangle x'$ such that $r \cdot (\triangle x' - \triangle x) > 0$ for all $r \in \mathcal{R}$. In this way, we may only need to consider a few lower bounds $u_{num} \cdot \triangle x_i$ instead of all of them and thus, accelerate the process. Due to the lack of space, the techniques can be found in Appendix A.

Hyper-edge. Each hyper-edge in E connects three nodes in V, which indicates the relation of the categorical values stored in the nodes. Consider any three nodes $v_1, v_2, v_3 \in V$ with their multisets L_1, L_2, and L_3, respectively. If any two of the multisets could deduce the third one based on one of the following derivation rules, there exists a hyper-edge connecting v_1, v_2 and v_3. Note that not every three nodes are connected by a hyper-edge. It is possible that any two of the multisets cannot derive the third one. Without loss of generality, suppose that L_1 and L_2 can deduce L_3. Let $\odot \in \{\oplus, \ominus\}$ ($L_1 \oplus \overline{L_2} = L_1 \ominus L_2$). We have the following derivation rules.

1) $L_1 \odot L_2 = L_3$ or $2L_3$ or $\overline{L_3}$ or $2\overline{L_3}$
2) $2L_1 \odot L_2 = L_3$ or $\overline{L_3}$
3) $L_1 \odot 2L_2 = L_3$ or $\overline{L_3}$

Consider derivation rule $L_1 \oplus L_2 = L_3$ as an example. In Figure 5, there is a hyper-edge connecting nodes v_1, v_2 and v_3, since their multisets fulfill the derivation rule, i.e., $[A_1, A_2]_{1,-1} \oplus [B_1, B_2]_{1,-1} = [A_1, B_1, A_2, B_2]_{1,1,-1,-1}$. We use a small black circle in Figure 5 to represent the hyper-edge connecting v_1, v_2 and v_3. Similarly, the other hyper-edges based on other derivation rules are also shown in small circles in Figure 5.

Update. Suppose that tuples $p, q \in \mathcal{D}$ are selected as a question to be asked to a user. Following the user feedback, relational graph $\mathcal{G}$ can be updated. We consider the update in

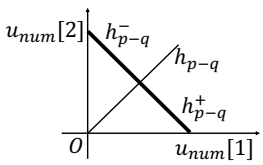


Figure 3: Space $\mathbb{R}^{d_{num}}$

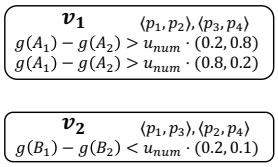


Figure 4: Bounds in nodes

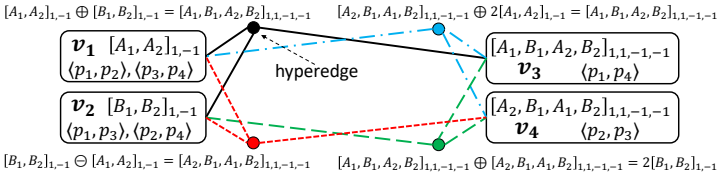


Figure 5: Relational graph

two cases: (1) p and q are the same in all categorical attributes and (2) p and q differ in at least one categorical attribute.

Consider the first case. Based on the user feedback, we can update $\mathcal{R}$ to be smaller (i.e., $\mathcal{R}\cap h^{+}_{p-q}$ or $\mathcal{R}\cap h^{+}_{q-p}$, see Section V-B1,), which may make some upper or lower bounds untight. We remove those untight bounds from each node.

Example 12. Figure 4 shows two lower bounds in v_1, where $\triangle x_1 = (0.2, 0.8)$ and $\triangle x_2 = (0.8, 0.2)$. Suppose that $\mathcal{R}$ is updated to be a line segment from $(0.5, 0.5)$ to $(1, 0)$. For any r in the updated $\mathcal{R}$, $r\cdot(\triangle x_2 - \triangle x_1) > 0$ and thus, bound $u_{num}\cdot(0.2, 0.8)$ is untight and is removed from v_1. □

Consider the second case. Assume that node v contains the tuple pair $\langle p, q\rangle$ and a multiset L. If a user prefers p to q, we can obtain an inequality $g(L) > u_{num}\cdot(q_{num} - p_{num})$, i.e., a lower bound $u_{num}\cdot(q_{num}-p_{num})$ for $g(L)$. For simplicity, we denote $u_{num}\cdot(q_{num}-p_{num})$ by $u_{num}\cdot\triangle x$ in the following. We add lower bound $u_{num}\cdot\triangle x$ into v and update v in two steps. (1) We utilize all the upper bounds in v with $u_{num}\cdot\triangle x$ to update $\mathcal{R}$. Suppose that there is an upper bound $u_{num}\cdot\triangle y$ for $g(L)$ in v (i.e., $g(L) < u_{num}\cdot\triangle y$). Since $u_{num}\cdot\triangle x < g(L) < u_{num}\cdot\triangle y$, we have $u_{num}\cdot(\triangle y - \triangle x) > 0$. In this way, we can build a hyper-plane $h_{\triangle y-\triangle x}$ based on vector $\triangle y - \triangle x$ and update $\mathcal{R}$ to be $\mathcal{R}\cap h^{+}_{\triangle y-\triangle x}$. (2) We remove all the bounds in v that become untight based on the updated $\mathcal{R}$.

Example 13. Figure 4 shows an upper bound in v_2, where $\triangle y = (0.2, 0.1)$. Suppose that we obtain a lower bound $u_{num}\cdot(0.1, 0.6)$ for $g(B_1) - g(B_2)$, i.e., $\triangle x = (0.1, 0.6)$. Since $\triangle y - \triangle x = (0.1, -0.5)$, we can build a hyper-plane h based on vector $(0.1, -0.5)$ and update $\mathcal{R}$ to be $\mathcal{R}\cap h^{+}$. Then, we remove the untight bounds based on the updated $\mathcal{R}$. □

After node v is updated, we can utilize bound $u_{num}\cdot\triangle x$ in v to generate new bounds for the nodes $v_c \in V$ that are connected with v via the hyper-edges. Assume that nodes $v_1, v_2 \in V$ with multisets L_1 and L_2 are connected with v by a hyper-edge. Without loss of generality, suppose that $L_1\oplus L_2 = L$ and nodes v_1 and v_2 have bounds $g(L_1) < u_{num}\cdot\triangle y_1$ and $g(L_2) < u_{num}\cdot\triangle y_2$, respectively. Since $g(L) > u_{num}\cdot\triangle x$, we could generate the following new bounds for v_1 and v_2.

- v_1: $g(L_1) = g(L) - g(L_2) > u_{num}\cdot(\triangle x - \triangle y_2)$
- v_2: $g(L_2) = g(L) - g(L_1) > u_{num}\cdot(\triangle x - \triangle y_1)$

Note that we will not use the generated bounds if they are untight. If L_1, L_2 and L are derived using other rules, we can also generate new bounds similarly. For the lack of space, the other generation can be found in Appendix A.

After obtaining the new generated bounds for node v_c, we can update v_c in the same way as we update v. Then, we can use the new generated bounds to *trigger* the update of other nodes that are connected with v_c. The update will be continually triggered and multiple nodes in $\mathcal{G}$ will be updated.

Specifically, we update $\mathcal{G}$ as follows. We build a queue Q to store the nodes that have been updated. Initially, node v (containing $\langle p, q\rangle$ that are presented as a question to a user) is updated and inserted into Q. We say that node v is updated in the first *batch*. Then, we recursively pop out nodes in Q. For each popped out node v_p, we generate new bounds for the nodes v'_p that are connected with v_p. If the generated bounds are tight, we update v'_p with these bounds and insert v'_p into Q. We say that node v'_p is updated in the i-th batch if node v_p is updated in the $(i-1)$-th batch. The update process continues until all the nodes in the α-th batch are updated, where $\alpha \geq 1$ is a given parameter. If α is large, the update of $\mathcal{G}$ may be costly. If it is small, we may fail to collect sufficient information from the user feedback. The setting of parameter α will be discussed in Section VI-A.

To accelerate the updating process of $\mathcal{G}$, we implemented two optimization strategies in practice. (1) We build an index (similar to the C-Tree shown in Section IV-A) for the multisets in nodes. In this way, given a tuple pair $\langle p, q\rangle$, we can quickly locate the node that stores the pair. (2) The relational graph is constructed progressively. Only if a node is updated will the hyper-edges related to this node be built. Our experiments verified that the strategies help our algorithm proceed efficiently.

3) Candidate Set $\mathcal{C}$: The *candidate set* $\mathcal{C} \subseteq \mathcal{D}$ contains the user's favorite tuple. Initially, $\mathcal{C}$ is set to be $\mathcal{D}$. During the interaction, based on $\mathcal{R}$ and $\mathcal{G}$, we determine and prune from $\mathcal{C}$ the tuples that cannot be the user's favorite tuple.

Pruning Strategy 1. For any $p \in \mathcal{C}$, consider the set $\mathcal{C}_p$ of tuples in $\mathcal{C}\setminus\{p\}$ that have the same values with p in all categorical attributes.

Lemma 6. Given $\mathcal{R}$, p can be pruned if $\mathcal{R} \subseteq \cup_{q\in\mathcal{C}_p} h^{+}_{q-p}$, where h_{q-p} is a hyper-plane based on vector $q_{num} - p_{num}$.

Intuitively, since $u_{num} \in \mathcal{R}$, if $\mathcal{R} \subseteq \cup_{q\in\mathcal{C}_p} h^{+}_{q-p}$, we can find a point $q \in \mathcal{C}_p$ such that $f(q) > f(p)$. To check if $\mathcal{R} \subseteq \cup_{q\in\mathcal{C}_p} h^{+}_{q-p}$, we utilize the LP similar to Definition 1. The time complexity of a LP solver (e.g., Simplex [31]) is $O(|\mathcal{C}_p|^2 d_{num})$. The details can be found in Appendix A.

Pruning Strategy 2. For any $p \in \mathcal{C}$, consider the set $\mathcal{C}'_p$ of tuples in $\mathcal{C}$ that have different values in at least one categorical attributes with p. For each $q \in \mathcal{C}'_p$, we find the node $v \in V$ that contains pair $\langle p, q\rangle$. Suppose that there are l upper bounds $u_{num}\cdot\triangle y_i$ in v, where $i \in [1, l]$. We divide $\mathcal{R}$ into l disjoint smaller polyhedrons, namely $\mathcal{R}_1$, $\mathcal{R}_2$, ..., $\mathcal{R}_l$, such that (1) each $\mathcal{R}_i$ corresponds to exactly one upper bound $u_{num}\cdot\triangle y_i$ and (2) $\forall r \in \mathcal{R}_i$, $\forall j \in [1, l]$ and $j \neq i$, we have $r\cdot(\triangle y_j -$

$\triangle y_i) > 0$. For each $\mathcal{R}_i$, we build a hyper-plane h_i that passes through the origin with its norm as $p_{num} - q_{num} + \triangle y_i$.

Lemma 7. Given polyhedrons $\mathcal{R}_i$, where $i \in [1, l]$, tuple p can be pruned from $\mathcal{C}$ if $\forall i \in [1, l]$, $\mathcal{R}_i \subseteq h_i^-$.

If there are w_i vertices in $\mathcal{R}_i$, to identify if $\mathcal{R}_i \subseteq h_i^-$, we need $O(\mathrm{w}_i)$ time to check whether all the vertices of $\mathcal{R}_i$ are in h_i^-. The total time complexity is $O(\sum_{i=1}^{l} \mathrm{w}_i)$. Similarly, we can find node v containing pair $\langle q, p\rangle$ and check if p can be pruned because of q based on the lower bounds in v. The detail can be found in Appendix A.

Example 14. In Figure 4, v_2 contains pair $\langle p_2, p_4\rangle$ and an upper bound. Assume that $\mathcal{R}$ is a line segment from $(0.8, 0.2)$ to $(1, 0)$. We build a hyper-plane h_1 based on vector $(-0.2, 0.8)$ and set $\mathcal{R}_1 = \mathcal{R}$. Since $\mathcal{R}_1 \subseteq h_1^-$, p_2 can be pruned from $\mathcal{C}$. □

C. Tuple Selection

We now present our strategy of selecting tuples as questions to be asked to a user. In each interactive round, we select two tuples from $\mathcal{C}$ and display them to the user. The restriction of selecting tuples from $\mathcal{C}$ makes sure that $|\mathcal{C}|$ is strictly smaller after each question, since we can know the user preference on the two selected tuples and thus, prune at least one tuple from $\mathcal{C}$. The tuple selection can be classified into two types.

- **Same-Categorical Type.** We randomly select two tuples $p, q \in \mathcal{C}$ such that $\forall i \in [1, d_{cat}], p_{cat}[i] = q_{cat}[i]$.
- **Different-Categorical Type.** We randomly select two tuples $p, q \in \mathcal{C}$ such that (1) they differ in at least one categorical attributes and (2) they have the fewest different categorical values among all pairs in $\mathcal{C}$. The intuition of (2) is that if there are many different categorical values, it is hard to learn the difference in preference is caused by which pairs of categorical values and thus, it is more difficult to analyze the user preference on the categorical values.

Our idea is to alternate flexibly the two selection types to ask a user questions. Intuitively, in each interactive round, we find $p_1, q_1 \in \mathcal{C}$ following the same-categorical type, and $p_2, q_2 \in \mathcal{C}$ following the different-categorical type. Then, we evaluate the two pairs $\langle p_1, q_1\rangle$ and $\langle p_2, q_2\rangle$, and choose the one as question which can help to prune more tuples from $\mathcal{C}$.

Specifically, consider a node $v \in V$ with several tuple pairs. Given a tuple pair $\langle p, q\rangle$, we define the *gain* of $\langle p, q\rangle$ w.r.t. v, denoted by $Gain(p, q, v)$, to be the number of tuples that (1) are stored in the tuple pairs of v and (2) can be pruned from $\mathcal{C}$ if we present p and q as a question to be asked to a user. Since there are two possible user answers (the user prefers p to q or prefers q to p), the number of pruned tuples is uncertain. Thus, we use the average number of pruned tuples of both answers. Our method finds node v_2 that stores the pair $\langle p_2, q_2\rangle$. Since p_1 and q_1 are the same in all categorical attributes, we cannot find a node in V storing pair $\langle p_1, q_1\rangle$. Then, if $\frac{Gain(p_2,q_2,v_2)}{Gain(p_1,q_1,v_2)}$ is large than a given parameter ϵ, we use p_2 and q_2 as a question. Otherwise, we turn to p_1 and q_1. Note that the value of ϵ should be set properly. If $\epsilon = \infty$, same-categorical type takes priority. If $\epsilon \leq 0$, different-categorical type dominates the tuple selection. We will explore the setting of ϵ in Section VI-A.

Algorithm 2: The *GE-Graph* Algorithm

```
Input: A tuple set D
Output: The user's favorite tuple in D
R ← {r ∈ R_+^{d_num} | Σ_{i=1}^{d_num} r[i] = 1}, C ← D
Initialize relational graph G
while |C| > 1 do
    if Gain(p2,q2,v2) / Gain(p1,q1,v2) > ε then
        Display p1 and q1 to a user
    else
        Display p2 and q2 to a user
    Update R, G and C based on the user feedback
return The only tuple left in C
```

D. Analysis

The pseudocode of *GE-Graph* is presented in Algorithms 2. Theorem 3 gives a theoretical guarantee on the number of questions asked. In practice, its performance is much better than the theoretical analysis, since we can prune many points from $\mathcal{C}$ in each round. We will show this in Section VI-A.

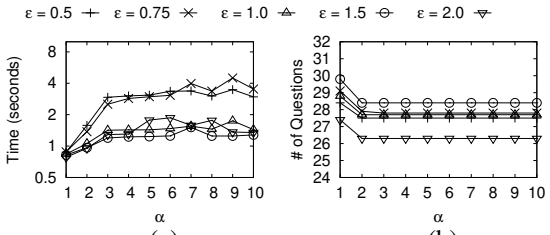
Theorem 3. Algorithm *GE-Graph* solves the general case of ISM by asking a user $O(n)$ questions.

VI. Experiment

We conducted experiments with C/C++ implementation on a machine with 3.10GHz CPU and 16GB RAM.

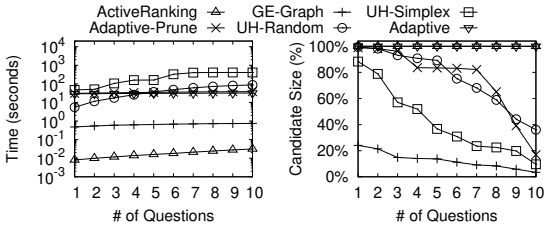
Datasets. We conducted experiments on synthetic and real datasets that were commonly used in existing studies [2], [6]. For the synthetic datasets, the numerical attributes are anti-correlated [32] and the categorical attributes are generated according to a Zipfian distribution [2], [33], where the Zipfian parameter is set to 1. The real datasets are *Nursery* [2], *Car* [6], *Diamond* [34], and *Pollution* [35]. *Nursery* contains $12,960$ tuples with 6 numerical attributes (parents, has_nurs, housing, finance, social, and health) and 2 categorical attributes (the form of family and the number of children). *Car* includes $69,052$ used cars with 4 numerical attributes (price, date of manufacture, horsepower, and used kilometers) and 3 categorical attributes (fuel type, vehicle type, and gearbox). Different from [6] which only selects 1000 cars from the datasets for experiments, we use the dataset directly so that the experiments can be conducted with a large amount of cars. *Pollution* has 608,700 records with 2 categorical attributes (state and year) and 3 numerical attributes (CO, SO_2 and NO_2). *Diamond* has 119,308 records with 2 categorical attributes (shape and color) and 2 numerical attributes (price and carat).

For all datasets, each numerical attribute was normalized to $(0, 1]$. To be consistent with the experimental setting in [6], [36], we preprocessed all the datasets to contain *skyline tuples*. Specifically, tuple $p \in \mathcal{D}$ was retained if there did not exist $q \in \mathcal{D}$ such that (1) p and q were the same in all categorical attributes and (2) p was not better than q in each numerical attribute, and strictly worse in at least one numerical attribute.



(a) (b)

Figure 6: Update strategy



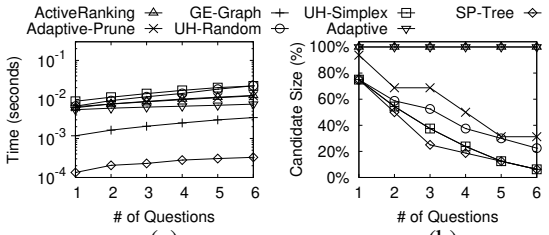
(a) (b)

Figure 8: Synthetic

Algorithms. We evaluated our algorithms *SP-Tree* and *GE-Graph* with existing ones *ActiveRanking* [16], *Adaptive* [8], *UH-Random* [6], and *UH-Simplex* [6]. Since the existing algorithms were designed for numerical attributes, we transformed the values of all categorical attributes to numerical attributes by using the standard SVM convention, which is commonly used in preference queries [8] and machine learning area [37] for handling categorical attributes. Specifically, for each categorical value from all categorical attributes, we built a new numerical attribute. If a tuple was described by this categorical value originally, we set its value of the newly built numerical attribute to 1. Otherwise, we set it to 0.

Note that the existing algorithms were not proposed to find the user's favorite tuple directly. We adapted them as follows. (1) *ActiveRanking* [16] learned the ranking of tuples. We returned the first ranked tuple when obtaining the ranking. (2) *UH-Simplex* and *UH-Random* [6] defined a criterion *regret ratio* to evaluate tuples and returned a tuple whose regret ratio satisfied a given threshold. We set the threshold to 0 so that the returned tuple must be the user's favorite tuple. (3) *Adaptive* [8] learned the user preference. According to the experimental results in [8], the learned user preference was very close to the theoretical optimal if the error threshold σ in the algorithm was set to 10^{-5}. Thus, we set $\sigma = 10^{-5}$ and returned the tuple that ranks the first w.r.t. the learned user preference. Note that *Adaptive* did not adopt any tuple pruning strategies. For comparison, we created another version, namely *Adapt-Prune*, which included the tuple pruning strategy proposed by [6].

Parameter Setting. We evaluated the algorithms by varying different parameters. (1) The dataset size n. (2) The number of numerical attributes d_{num}. (3) The number of categorical attributes d_{cat}. (4) The cardinality of each categorical attribute c_{val} (i.e., the number of possible values in each categorical attribute). Following [2], [6], unless stated explicitly, we set n = 100,000, d_{cat} = 3, d_{num} = 2, and c_{val} = 4 by default.



(a) (b)

Figure 7: Synthetic (categorical only)

Performance Measurement. We evaluated the performances of algorithms by 3 measurements: (1) *Execution time*, (2) *Candidate size*. We reported the percentage of remaining tuples in $\mathcal{C}$ after each question, (3) *The number of questions asked*.

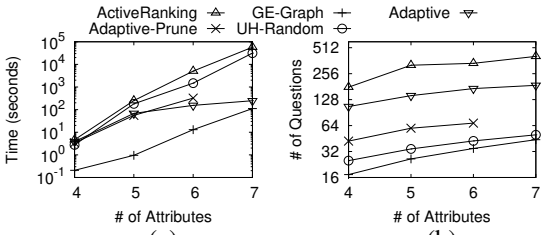
Each algorithm was conducted ten times with different generated utility functions. For u_{cat} and u_{num}, we randomly generated two vectors. For $h(\cdot)$, we randomly assigned a value to each categorical value. It is worth mentioning that although we focus on determining function $g(\cdot)$ in our algorithms, in the experiments, we setup the utility function by generating u_{cat} and $h(\cdot)$, respectively. The generated vectors and the assigned values simulated the user preference which could be regarded as the ground truth in our setup (a common setting in the literature [6], [7]), but they were not disclosed to the algorithms. Different utility functions may cause different convergence of algorithms. Thus, we randomly generated utility functions and expected to see the average convergence performance. We reported the average performance of each algorithm.

A. Results on Synthetic Datasets

Performance Study. Recall that we introduce two parameters α and ϵ for our algorithm *GE-Graph* in Section V-B2 and Section V-C, respectively. α is used to control the update of $\mathcal{G}$ and ϵ is for tuple selection. We explored the value setting of these two parameters, by varying α from 1 to 10 and ϵ from 0.5 to 2 on the synthetic dataset in which the other parameters were set by default. Figure 6 shows the execution time and the number of questions asked. When α increased, the execution time increased since there were more nodes to be updated in $\mathcal{G}$. However, the number of questions asked only decreased when α increased from 1 to 2. This meant that the more updated nodes in $\mathcal{G}$ helped little to prune tuples. Thus, we set $\alpha = 2$ in the rest experiments. For parameter ϵ, when $\epsilon = 2$, *GE-Graph* asked the fewest questions. Thus, we set $\epsilon = 2$ in the following.

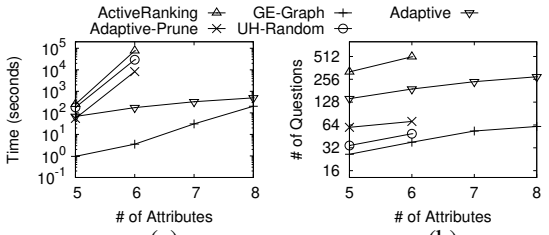
Progress Analysis. We showed how algorithms progress during the interaction process. We reported the middle results, i.e., execution time and the candidate size after each question asked. The second measurement is an indicator showing the number of questions asked by algorithms. If the candidate size can be reduced effectively, we can quickly achieve the stopping condition. Our algorithms *SP-Tree* and *GE-Graph* were compared against existing ones on several synthetic datasets.

The first dataset contained tuples that were described by categorical attributes only, where the other parameters were set by default. Figure 7 shows the results. Our algorithms *SP-Tree* and *GE-Graph* were the fastest and reduced the candidate



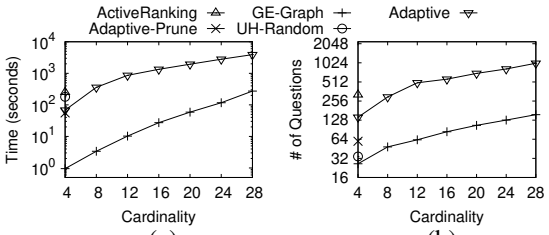
(a) (b)

Figure 9: Vary d_{num}



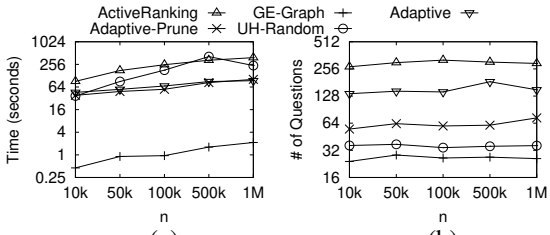
(a) (b)

Figure 10: Vary d_{cat}



(a) (b)

Figure 11: Vary c_{val}



(a) (b)

Figure 12: Vary n

size the most effectively. In particular, our algorithm *SP-Tree* only took 10^{-4} seconds to ask 6 questions, and it was the only algorithm that reduced the candidate size by 50% after asking 2 questions. This justified the superiority of *SP-Tree*.

The second dataset contained tuples that were described by numerical and categorical attributes. Since *SP-Tree* only works for the special case of ISM, we did not include it for comparison. Figure 8 shows the results. *UH-Simplex* did not reduce the candidate size effectively and ran the slowest among all algorithms. Its tuple selection strategy was developed based on *convex hull*, which was costly for multiple attributes. It took up to 410 seconds to ask 10 questions, while the second slowest existing algorithm spent 90 seconds. Since *UH-Simplex* was time-consuming, we excluded it in the rest experiments. *UH-Random*, *Adaptive* and *Adaptive-Prune* were also slow. They took dozens of seconds. *UH-Random* spent much time pruning tuples after asking each question and *Adaptive* maintained a costly data structure. *Adaptive-Prune* inherited both time-intensive issues. Besides, *Adaptive* failed to provide any reduction on the candidate size since it only focused on learning the user preference. *Adaptive-Prune* and *UH-Random* reduced the candidate size ineffectively. Their pruning strategy considered all the attributes equally and thus, neglected the different properties of categorical and numerical attributes as discussed in Section I. Our algorithm *GE-Graph* reduced the candidate size effectively and efficiently. They pruned more than 90% of tuples in $\mathcal{C}$ by asking 7 questions within 0.7 seconds. *ActiveRanking* was faster than *GE-Graph*, since it did not prune tuples, which neglected the connection between tuples and resulted in asking more questions. Although *GE-Graph* took slightly more time, its execution times was small and reasonable given that it could effectively reduce the candidate size and obtain the user's favorite tuple by asking a few questions.

We also conducted experiments on two datasets with $c_{val} = 10$ (the other properties of these two datasets were the same as that of previous two datasets). The results show that the existing algorithms performed badly. Some of them could not finish the computation. Due to the lack of space, the detailed results and analysis can be found in Appendix C.

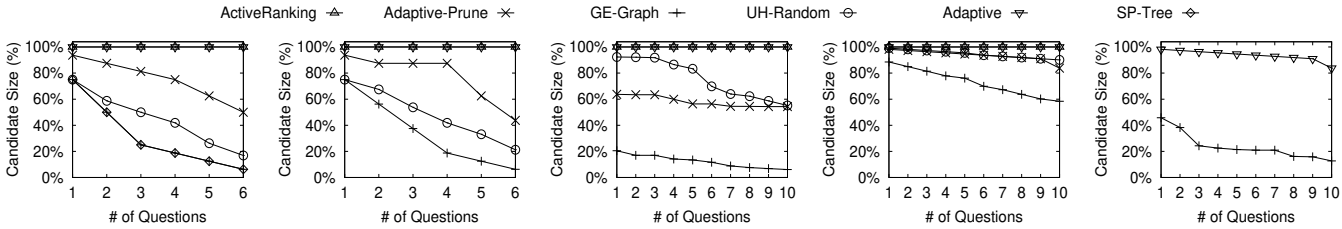
Scalability. We studied the scalability of algorithms by varying parameters (i.e., d_{num}, d_{cat}, c_{val}, and n) and reporting the number of questions asked and the execution time.

Varying d_{num}. In Figure 9, we varied the number of numerical attributes d_{num} from 2 to 5. The results show that our algorithm *GE-Graph* preformed well when we scale the numerical attributes. It asked $12\% - 92\%$ fewer questions and ran 1-3 orders of magnitude faster than the existing algorithms.

Varying d_{cat}. In Figure 10, we varied the number of categorical attributes d_{cat} from 2 to 5. *UH-Random*, *Adaptive-Prune* and *ActiveRanking* could not complete when $d_{cat} > 3$ due to the excessive execution time (more than 10^6 seconds). They handled a large amount of attributes since the standard SVM mapped each categorical value to an attribute. In contrast, our algorithm utilized the relational graph to handle the categorical values. It scaled well w.r.t. the number of categorical attributes. When $d_{cat} = 3$, it asked $22\% - 82\%$ fewer questions and ran 1-4 orders of magnitude faster than the existing algorithms.

Varying c_{val}. In Figure 11, we varied the cardinality of categorical attributes c_{val} from 4 to 28. Our algorithm *GE-Graph* asked the fewest questions within the shortest time. When $c_{val} = 16$, *Adaptive* asked 6 times more questions and took 85 times longer time than *GE-Graph*. Note that we only showed the performances of *UH-Random*, *Adaptive-Prune* and *ActiveRanking* when $c_{val} = 4$, since they took too much time (more than 10^6 seconds) when the cardinality increased. Their inefficiency was caused by the excessive number of attributes as stated previously.

Varying n. In Figure 12, we varied the data size n from $10k$ to $1M$. Our algorithm *GE-Graph* scaled the best. Its execution times was less than 2.2 seconds even if $n = 1M$, while the fastest existing algorithm took 94 seconds. *GE-Graph* also asked at least 8 fewer questions than the existing ones for all n.



(a) Nursery(Cat) (b) Nursery (c) Car (d) Diamond (e) Pollution

Figure 13: Real Datasets

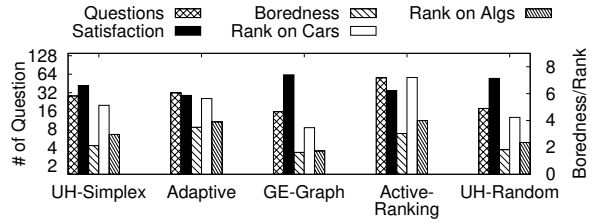


Figure 14: User Study

B. Results on Real Datasets

We explored how algorithms progress during the interaction process on real datasets by reporting the candidate size. Due to the lack of space, the execution times are shown in Appendix C. For dataset *Nursery*, we processed it into two versions. The first one was exactly *Nursery*. The second one, namely *Nursery(Cat)*, only maintained the categorical attributes in *Nursery*. Figure 13 shows the results. Our algorithm *GE-Graph* reduced the candidate size the most effectively on all datasets. For example, on dataset *Car*, after asking 4 questions, *GE-Graph* only contained fewer than 15% tuples in $\mathcal{C}$, while existing algorithms contained at least 59% tuples in $\mathcal{C}$.

C. Interaction Study

We conducted a user study on dataset *Car* to demonstrate the robustness of our algorithm, since users might make mistakes or provide inconsistent feedback during real interaction. Following [6], [7], we randomly selected 1000 candidate used cars from the dataset described by 5 attributes (price, year of manufacture, horse power, used kilometer, and fuel type). We recruited 30 participants and reported their average result.

We compared our algorithm *GE-Graph* against existing ones, *UH-Random*, *UH-Simplex*, *Adaptive*, and *ActiveRanking*. Each algorithm aimed at finding the user's favorite used car. Since the user preference is unknown, we re-adapted *Adaptive* following [6], [7] (instead of the way described previously). *Adaptive* maintained an estimated user preference u_e during the interaction. We compared the user's answer of some randomly selected questions with the prediction w.r.t. u_e. If 75% questions were correctly predicted, we stopped and returned the first ranked car w.r.t. u_e. The other existing algorithms followed the adaptation described at the beginning of Section VI.

Each algorithm was evaluated by 5 measurements. (1) *The number of questions asked*. (2) *The satisfaction* which is a score from 1 to 10 given by each participant. It evaluates how satisfied a participant is with the returned car. (3) *The boredness* which is a score from 1 to 10 given by each participant. A higher score indicates that the participant feels more bored when s/he sees the returned used car after answering several questions. (4) *The rank on cars*. We asked each participant to give an explicit ranking of the cars returned by different algorithms. (5) *The rank on algorithms*. We asked each participant to give an explicit ranking of all algorithms.

Figure 14 shows the results. Our algorithm *GE-Graph* performed the best in terms of each measurement. It asked about $11\% - 71\%$ fewer questions than the existing algorithms. Its satisfaction was 7.4 and its boredness was 3.4. In comparison, for the most boring algorithm *ActiveRanking*, its satisfaction was 6.2 and its boredness was 7.4. The results show that our algorithm can work well in real-world scenario.

We also conducted a confidence study to explore the case that a user may be unable to answer some questions. For the lack of space, the details are shown in Appendix C.

D. Summary

The experiments showed the superiority of our algorithms over the best-known existing ones. (1) Our algorithms were effective and efficient. On the dataset with size 1M, the best-known existing algorithm *UH-Random* asked 36.1 questions in 239 seconds, which is unacceptable in practice. In comparison, *GE-Graph* asked 25.9 questions in 2.2 seconds. It reduced the number of questions asked by more than 28% and accelerated by two orders of magnitude. (2) *GE-Graph* scaled well in terms of d_{cat}, d_{num}, c_{val}, and n (e.g., it asked 78% fewer questions than *Adaptive* on the dataset with $d_{num} = 5$). (3) *GE-Graph* was robust (e.g., in our user study, even if users may provide inconsistent feedbacks during the interaction, it achieved the best degree of satisfaction than existing ones). In summary, *SP-Tree* asked the fewest questions in the shortest time for the special case of ISM, and *GE-Graph* ran the fastest and asked the fewest questions for the general case of ISM.

VII. Conclusion

In this paper, we present interactive algorithms for searching the user's favorite tuple in the dataset with numerical and categorical attributes by asking as few questions as possible. For the special case of ISM, we propose algorithm *SP-Tree* that asks an asymptotically optimal number of questions. For the general case of ISM, we propose algorithm *GE-Graph*, which performs well theoretically and empirically. Extensive experiments showed that our algorithms were efficient and effective. As for future work, we consider the case that users may make mistakes when answering questions.

REFERENCES

[1] N. Sarkas, G. Das, N. Koudas, and A. K. H. Tung, "Categorical skylines for streaming data," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2008, p. 239–250.

[2] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang, "Online skyline analysis with dynamic preferences on nominal attributes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 1, pp. 35–49, 2009.

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

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

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

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

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

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

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

[10] J. F. Hair, "Multivariate data analysis," 2009.

[11] S. B. MacKenzie, "The role of attention in mediating the effect of advertising on attribute importance," *Journal of Consumer Research*, vol. 13, no. 2, pp. 174–195, 1986.

[12] Product attributes, 2022. [Online]. Available: https://repository.up.ac.za/bitstream/handle/2263/27411/04chapter4.pdf?sequence=5&isAllowed=y

[13] W.-T. Balke, U. Güntzer, and C. Lofi, "Eliciting matters – controlling skyline sizes by incremental integration of user preferences," in *Advances in Databases: Concepts, Systems and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 551–562.

[14] Alchemer llc, 2022. [Online]. Available: https://www.alchemer.com/resources/blog/how-many-survey-questions/

[15] Questionpro, 2022. [Online]. Available: https://www.questionpro.com/blog/optimal-number-of-survey-questions/

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

[17] W. Wang and R. C.-W. Wong, "Interactive mining on ordered and unordered attributes," in *Proceedings of the 27th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 2022, p. 201–210.

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

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

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

[21] Rabinkarp, 2022. [Online]. Available: https://pit-claudel.fr/clement/blog/linear-time-probabilistic-pattern-matching-and-the-rabin-karp-algorithm

[22] Rossetto, 2022. [Online]. Available: https://www.rosette.com/name-matching-algorithms

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

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

[25] Formplus, 2022. [Online]. Available: https://www.formpl.us/blog/categorical-data

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

[27] W. D. Blizard, "Negative membership." *Notre Dame Journal of formal logic*, vol. 31, no. 3, pp. 346–368, 1990.

[28] L. da F. Costa, "An introduction to multisets," 10 2021.

[29] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in Neural Information Processing Systems*, vol. 19, 2006, pp. 1601–1608.

[30] P. Valdivia, P. Buono, C. Plaisant, N. Dufournaud, and J.-D. Fekete, "Analyzing dynamic hypergraphs with parallel aggregated ordered hypergraph visualization," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019.

[31] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.

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

[33] N. L. Johnson, S. Kotz, and A. W. Kemp, *Univariate Discrete Distributions*. Wiley-Interscience, 1992.

[34] Dataset diamond, 2022. [Online]. Available: https://www.kaggle.com/datasets/miguelcorraljr/brilliant-diamonds

[35] Dataset pollution, 2022. [Online]. Available: https://www.kaggle.com/datasets/alpacanonymous/us-pollution-20002021

[36] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R. C.-W. Wong, and W. Zhan, "k-Regret Minimizing Set: Efficient Algorithms and Hardness," in *20th International Conference on Database Theory*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 11:1–11:19.

[37] B. L. Milenova, J. S. Yarmus, and M. M. Campos, "Svm in oracle database 10g: removing the barriers to widespread adoption of support vector machines," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 1152–1163.

APPENDIX A
TECHNIQUES SUPPLEMENT

A. Nodes in Relational Graph $\mathcal{G}$

In this section, we complement the details of maintaining the lower bounds and upper bounds in the nodes of $\mathcal{G}$.

Lower Bounds. We present the sufficient condition for identifying the untight lower bound. Consider a lower bound $u_{num} \cdot \triangle x$ in node $v \in V$. Assume that there are l other lower bounds $u_{num} \cdot \triangle x'$ in v.

Lemma 8. Bound $u_{num} \cdot \triangle x$ is an untight lower bound for $g(L)$ if there exists a lower bound $u_{num} \cdot \triangle x'$ for $g(L)$ such that $\forall r \in \mathcal{R}$, $r \cdot (\triangle x' - \triangle x) > 0$.

Proof. Since $\forall u_{num} \in \mathcal{R}$ and there exists a lower bound $u_{num} \cdot \triangle x'$ for $g(L)$ such that $\forall r \in \mathcal{R}$, $r \cdot (\triangle x' - \triangle x) > 0$, we have $u_{num} \cdot (\triangle x' - \triangle x) > 0$. This implies that $g(L) > u_{num} \cdot \triangle x' > u_{num} \cdot \triangle x$, i.e., lower bound $u_{num} \cdot \triangle x'$ bounds $g(L)$ more tightly than $u_{num} \cdot \triangle x$. □

Lemma 8 tries to search for a lower bound $u_{num} \cdot \triangle x'$ such that $\forall r \in \mathcal{R}$, $r \cdot (\triangle x' - \triangle x) > 0$. In practice, we implement the lemma as follows. Recall that $\mathcal{R}$ is a polyhedron, which is an intersection of a set of hyper-planes and half-spaces. One property of the polyhedron is that it is convex [26]. Specifically, suppose that $\mathcal{R}$ has m vertices r_i (i.e., the corner points of $\mathcal{R}$), where $i \in [1, m]$. Each point $r \in \mathcal{R}$ can be represented by the vertices of $\mathcal{R}$, i.e., $r = \sum_{i=1}^{m} t_i r_i$, where t_i is a positive real number such that $\sum_{i=1}^{m} t_i = 1$. We compare lower bound $u_{num} \cdot \triangle x$ with the other lower bounds $u_{num} \cdot \triangle x'$ one by one. For each comparison, e.g., $u_{num} \cdot \triangle x$ and $u_{num} \cdot \triangle x'$, we check whether all vertices $r_i \in \mathcal{R}$, where $i \in [1, m]$, satisfy $r_i \cdot (\triangle x' - \triangle x) > 0$. If yes, $\forall r \in \mathcal{R}$, we have the following derivation.

$$\begin{aligned} r \cdot (\triangle x' - \triangle x) &= (\sum_{i=1}^{m} t_i r_i) \cdot (\triangle x' - \triangle x) \\ &= \sum_{i=1}^{m} (t_i r_i \cdot (\triangle x' - \triangle x)) \\ &> 0 \end{aligned}$$

Upper Bounds. We first show the definition of the untight upper bound, and then discussed the way to identify whether a upper bound is untight.

Definition 2 (Untight Upper Bound). Bound $u_{num} \cdot \triangle y$ is an untight upper bound for $g(L)$ if $\forall r \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot \triangle y'$ for $g(L)$ such that $r \cdot (\triangle y' - \triangle y) < 0$.

Intuitively, if $u_{num} \cdot \triangle y$ is untight, since $u_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot \triangle y'$ such that $u_{num} \cdot (\triangle y' - \triangle y) < 0$, i.e., $u_{num} \cdot \triangle y'$ bounds $g(L)$ more tightly than $u_{num} \cdot \triangle y$. We utilize the LP algorithm to check whether an upper bound $u_{num} \cdot \triangle y$ is untight. Specifically, we define a variable w to be the objective value. Assume that there are l other upper bounds $u_{num} \cdot \triangle y_i$ for $g(L)$, where $i \in [1, l]$. For each $u_{num} \cdot \triangle y_i$, we build a constraint $r \cdot (\triangle y_i - \triangle y) > w$. Formally, we construct a LP as follows.

$$\begin{aligned} \text{maximize} \quad & w \\ \text{subject to} \quad & r \cdot (\triangle y_i - \triangle y) > w, \text{ where } i \in [1, l] \\ & r \in \mathcal{R} \end{aligned}$$

If the objective function $w < 0$, then $\forall r \in \mathcal{R}$, $\exists i \in [1, l]$ we have $r \cdot (\triangle y_i - \triangle y) < 0$. Thus, $u_{num} \cdot \triangle y$ is an untight upper bound. To accelerate the process of identifying untight bound, we propose a sufficient condition for identifying the upper bounds.

Lemma 9. Bound $u_{num} \cdot \triangle y$ is an untight upper bound for $g(L)$ if there exists an upper bound $u_{num} \cdot \triangle y'$ for $g(L)$ such that $\forall r \in \mathcal{R}$, $r \cdot (\triangle y' - \triangle y) < 0$.

Proof. Since $\forall u_{num} \in \mathcal{R}$ and there exists an upper bound $u_{num} \cdot \triangle y'$ for $g(L)$ such that $\forall r \in \mathcal{R}$, $r \cdot (\triangle y' - \triangle y) < 0$, we have $u_{num} \cdot (\triangle y' - \triangle y) < 0$. This implies that $g(L) < u_{num} \cdot \triangle y' < u_{num} \cdot \triangle y$, i.e., upper bound $u_{num} \cdot \triangle y'$ bounds $g(L)$ more tightly than $u_{num} \cdot \triangle y$. □

Lemma 9 tries to search for an upper bound $u_{num} \cdot \triangle y'$ such that $\forall r \in \mathcal{R}$, $r \cdot (\triangle y' - \triangle y) < 0$. In practice, we implement the lemma similarly to Lemma 8. Suppose that $\mathcal{R}$ has m vertices r_i, where $i \in [1, m]$. Since $\mathcal{R}$ is a polyhedron, each point $r \in \mathcal{R}$ can be represented as $r = \sum_{i=1}^{m} t_i r_i$, where t_i is a positive real number such that $\sum_{i=1}^{m} t_i = 1$ [26]. We compare upper bound $u_{num} \cdot \triangle y$ with the other upper bounds $u_{num} \cdot \triangle y'$ one by one. For each comparison, e.g., $u_{num} \cdot \triangle y$ and $u_{num} \cdot \triangle y'$, we check whether all vertices $r_i \in \mathcal{R}$, where $i \in [1, m]$, satisfy $r \cdot (\triangle y' - \triangle y) < 0$. If yes, $\forall r \in \mathcal{R}$, we have the following derivation.

$$\begin{aligned} r \cdot (\triangle y' - \triangle y) &= (\sum_{i=1}^{m} t_i r_i) \cdot (\triangle y' - \triangle y) \\ &= \sum_{i=1}^{m} (t_i r_i \cdot (\triangle y' - \triangle y)) \\ &< 0 \end{aligned}$$

B. Hyper-edges in Relational Graph $\mathcal{G}$

Consider nodes $v_1, v_2, v_3 \in V$ with multisets L_1, L_2 and L_3, respectively. If there exists a hyper-edge connecting the three nodes, we can use the bounds in any two nodes to derive new bounds for the third one. Without loss of generality, let us focus on how node v_1 and v_2 derive new bounds for v_3. Suppose that v_1 and v_2 have bounds as follows.

$$v_1 : \triangle x_1 \cdot u_{num} < g(L_1) < \triangle y_1 \cdot u_{num}$$

$$v_2 : \triangle x_2 \cdot u_{num} < g(L_2) < \triangle y_2 \cdot u_{num}$$

Then, based on the way of L_1 and L_2 deducing L_3, we can generate new bounds for node v_3 as follows. Note that $L_1 \ominus L_2 = L_1 \oplus \overline{L_2}$.

(1) Suppose that $L_1 \oplus L_2 = L_3$.

$$v_3 : (\triangle x_1 + \triangle x_2) \cdot u_{num} < g(L_3) < (\triangle y_1 + \triangle y_2) \cdot u_{num}$$

(2) Suppose that $L_1 \ominus L_2 = L_3$.

$$v_3 : (\triangle x_1 - \triangle y_2) \cdot u_{num} < g(L_3) < (\triangle y_1 - \triangle x_2) \cdot u_{num}$$

(3) Suppose that $L_1 \oplus L_2 = \overline{L_3}$.

$$v_3 : -(\triangle y_1 + \triangle y_2) \cdot u_{num} < g(L_3) < -(\triangle x_1 + \triangle x_2) \cdot u_{num}$$

(4) Suppose that $L_1 \ominus L_2 = \overline{L_3}$.

$$v_3 : (\triangle x_2 - \triangle y_1) \cdot u_{num} < g(L_3) < (\triangle y_2 - \triangle x_1) \cdot u_{num}$$

(5) Suppose that $L_1 \oplus L_2 = 2L_3$.

$$v_3 : \frac{1}{2}(\triangle x_1 + \triangle x_2) \cdot u_{num} < g(L_3) < \frac{1}{2}(\triangle y_1 + \triangle y_2) \cdot u_{num}$$

(6) Suppose that $L_1 \ominus L_2 = 2L_3$.

$$v_3 : \frac{1}{2}(\triangle x_1 - \triangle y_2) \cdot u_{num} < g(L_3) < \frac{1}{2}(\triangle y_1 - \triangle x_2) \cdot u_{num}$$

(7) Suppose that $L_1 \oplus L_2 = 2\overline{L_3}$.

$$v_3 : -\frac{1}{2}(\triangle y_1 + \triangle y_2) \cdot u_{num} < g(L_3) < -\frac{1}{2}(\triangle x_1 + \triangle x_2) \cdot u_{num}$$

(8) Suppose that $L_1 \ominus L_2 = 2\overline{L_3}$.

$$v_3 : \frac{1}{2}(\triangle x_2 - \triangle y_1) \cdot u_{num} < g(L_3) < \frac{1}{2}(\triangle y_2 - \triangle x_1) \cdot u_{num}$$

(9) Suppose that $2L_1 \oplus L_2 = L_3$.

$$v_3 : (2 \triangle x_1 + \triangle x_2) \cdot u_{num} < g(L_3) < (2 \triangle y_1 + \triangle y_2) \cdot u_{num}$$

(10) Suppose that $2L_1 \ominus L_2 = L_3$.

$$v_3 : (2 \triangle x_1 - \triangle y_2) \cdot u_{num} < g(L_3) < (2 \triangle y_1 - \triangle x_2) \cdot u_{num}$$

(11) Suppose that $2L_1 \oplus L_2 = \overline{L_3}$.

$$v_3 : -(2\triangle y_1 + \triangle y_2) \cdot u_{num} < g(L_3) < -(2\triangle x_1 + \triangle x_2) \cdot u_{num}$$

(12) Suppose that $2L_1 \ominus L_2 = \overline{L_3}$.

$$v_3 : (\triangle x_2 - 2 \triangle y_1) \cdot u_{num} < g(L_3) < (\triangle y_2 - 2 \triangle x_1) \cdot u_{num}$$

(13) Suppose that $L_1 \oplus 2L_2 = L_3$

$$v_3 : (\triangle x_1 + 2 \triangle x_2) \cdot u_{num} < g(L_3) < (\triangle y_1 + 2 \triangle y_2) \cdot u_{num}$$

(14) Suppose that $L_1 \ominus 2L_2 = L_3$.

$$v_3 : (\triangle x_1 - 2 \triangle y_2) \cdot u_{num} < g(L_3) < (\triangle y_1 - 2 \triangle x_2) \cdot u_{num}$$

(15) Suppose that $L_1 \oplus 2L_2 = \overline{L_3}$.

$$v_3 : -(\triangle y_1 + 2\triangle y_2) \cdot u_{num} < g(L_3) < -(\triangle x_1 + 2\triangle x_2) \cdot u_{num}$$

(16) Suppose that $L_1 \ominus 2L_2 = \overline{L_3}$

$$v_3 : (2 \triangle x_2 - \triangle y_1) \cdot u_{num} < g(L_3) < (2 \triangle y_2 - \triangle x_1) \cdot u_{num}$$

C. Update on Candidate set $\mathcal{C}$

In this section, we complement our strategies of pruning tuples in $\mathcal{C}$. First, we show the detailed LP formulation for Lemma 6. Then, we present a lemma that is used to prune tuples with the help of lower bounds.

Linear Programming for Lemma 6. We utilize the linear programming algorithm (LP) based on Lemma 6 to check if tuple p can be pruned. We define a variable w to be the objective value. For each tuple $q \in \mathcal{C}_p$ (note that q has the same values with p in all categorical attributes), we build a constraint $r \cdot (p_{num} - q_{num}) > w$, where $r \in \mathcal{R}$. Formally, we construct a LP as follows.

$$\begin{aligned} \text{maximize} \quad & w \\ \text{subject to} \quad & r \cdot (p_{num} - q_{num}) > w, \text{ where } q \in \mathcal{C}_p \\ & r \in \mathcal{R} \end{aligned}$$

If $w < 0$, then $\forall r \in \mathcal{R}$, $\exists q \in \mathcal{C}_p$ such that $r \cdot (p_{num} - q_{num}) < 0$ i.e., $r \cdot q_{num} > r \cdot p_{num}$. This means $\forall r \in \mathcal{R}$, $\exists q \in \mathcal{C}_p$ such that $r \in h^+_{q-p}$. Thus, $\mathcal{R} \subseteq \cup_{q \in \mathcal{C}_p} h^+_{q-p}$.

Prune tuples with lower bounds. Given a tuple $p \in \mathcal{C}$, consider the set $\mathcal{C}'_p$ of tuples in $\mathcal{C}$ that differ in at least one categorical attribute with p. For each $q \in \mathcal{C}'_p$, we first find the node v that contains tuple pair $\langle q, p \rangle$. Then, we check whether p can be pruned because of q based on the lower bounds in v. Specifically, suppose there are l lower bounds $u_{num} \cdot \triangle x_i$ in v, where $i \in [1, l]$. We divide $\mathcal{R}$ into l disjoint smaller polyhedrons $\mathcal{R}_i$ such that (1) each of them corresponds to exactly one lower bound $u_{num} \cdot \triangle x_i$ and (2) $\forall r \in \mathcal{R}_i$, $\forall j \in [1, r]$ and $j \neq i$, $r \cdot (\triangle x_i - \triangle x_j) > 0$. For each $\mathcal{R}_i$, we build a hyperplane h_i which passes through the origin with its unit norm in the same direction as vector $q_{num} - p_{num} + \triangle x_i$;

Lemma 10. Given polyhedrons $\mathcal{R}_i$, where $i \in [1, l]$, tuple p can be pruned from $\mathcal{C}$ if $\forall i \in [1, l]$, $\mathcal{R}_i \subseteq h^+_i$.

Proof. Consider polyhedron $\mathcal{R}_i$, where $i \in [1, l]$. $\mathcal{R}_i \subseteq h^+_i$ means that $\forall r \in \mathcal{R}_i$, $r \cdot (q_{num} - p_{num} + \triangle x_i) > 0$. Since (1) $\cup_{i \in [1,l]} \mathcal{R}_i = \mathcal{R}$ and (2) $\forall i \in [1, l]$, $\mathcal{R}_i \subseteq h^+_i$, we have $\forall r \in \mathcal{R}$, there exists a lower bound $u_{num} \cdot \triangle x_i$, where $i \in [1, l]$, such that $r \cdot (q_{num} - p_{num} + \triangle x_i) > 0$. Because $u_{num} \in \mathcal{R}$, we have

$$\begin{aligned} f(q) - f(p) &= g(L_1) - g(L_2) + u_{num} \cdot (q_{num} - p_{num}) \\ &> \triangle x_i \cdot u_{num} + u_{num} \cdot (q_{num} - p_{num}) \\ &= u_{num} \cdot (\triangle x_i + q_{num} - p_{num}) \\ &> 0 \end{aligned}$$

This shows that the utility of q is larger than that of p w.r.t. the user preference. Thus, p can be safely pruned from $\mathcal{C}$. □

If there are w_i vertices in $\mathcal{R}_i$, we need $O(\mathrm{w}_i)$ time to check whether all vertices in $\mathcal{R}_i$ are in h^+_i. If there is a vertex of $\mathcal{R}_i$ not in h^+_i, $\mathcal{R}_i \not\subseteq h^+_i$. The total time complexity is $O(\sum_{i=1}^{l} \mathrm{w}_i)$.

APPENDIX B
EXTENSION TO TOP-K

In this section, we show the way to extend our algorithm *GE-Graph* to returning the user's favorite k tuples, where $k \in [1, n]$. Our extension strategy is to iterate the algorithm k times and output the tuples returned by each iteration. Recall that *GE-Graph* maintains (1) a tuple set $\mathcal{C}$, which contains the user's favorite tuple, and (2) two data structures $\mathcal{R}$ and $\mathcal{G}$, which store the learned information of user preference w.r.t. the numerical attributes and categorical attributes, respectively. We add a data structure, a tuple set $\mathcal{B}$ as the output. In the end of each iteration (i.e., when $|\mathcal{C}| = 1$), we put the finally left tuple in $\mathcal{C}$ into $\mathcal{B}$. Then, we start a new iteration by keeping $\mathcal{R}$ and $\mathcal{G}$ of the last iteration and initializing $\mathcal{C}$ to be $\mathcal{D} \setminus \mathcal{B}$. We continue this process until $|\mathcal{B}| = k$.

APPENDIX C
EXPERIMENTS

A. Results on Synthetic Datasets

Progress Study. We conducted experiments to show how algorithms progress during the interaction process on two synthetic datasets. We compared our algorithms *SP-Tree* and *GE-Graph* against existing ones by varying the number of questions we could ask. The execution time and the candidate size were reported after each question asked.

The first dataset contained tuples which were only described by categorical attributes, where $c_{val} = 10$ and the other parameters were set by default. Figure 15 shows the results. Except for algorithm *UH-Random*, the other algorithms only took within 1.3 second to ask 10 questions. *UH-Random* took more than 2.5 seconds, since its tuple selection strategy was not effective and its pruning strategy was time-consuming. Our algorithm *SP-Tree* only took 10^{-4} seconds to ask 10 questions, which performed the best among all algorithms. Besides, our algorithms *SP-Tree* and *GE-Graph* reduced the candidate size the most effectively. In particular, *SP-Tree* was the only algorithm that reduced the candidate size by 50% after asking 5 questions. This justifies the superiority of *SP-Tree*.

The second dataset contained tuples which were described by both numerical and categorical attributes, where $c_{val} = 10$ and the other parameters were set by default. Figure 16 shows the results. Except for algorithm *Adaptive*, none of the existing algorithms could finish the computation. They either cost more than 10^5 seconds or ran out of memory. In comparison, our algorithm *GE-Graph* reduced the candidate size effectively within a short time.

B. Results on Real Datasets

Figure 17 shows the interaction time of algorithms when we increased the number of questions we could ask on all real datasets. All the algorithms ran fast on datasets *Nursery(Cat)*, *Nursery*, and *Diamond*. On dataset *Car*, *ActiveRanking* took less time to process each question than our algorithm *GE-Graph*, since it did not need to prune tuples after each question. Although *GE-Graph* took slightly more time, its execution time was small and reasonable given that it reduced the candidate size the most effectively. Since *ActiveRanking* asked more questions than our algorithm *GE-Graph*, its total execution time was longer. On dataset *Pollution*, except for *Adaptive*, none of the existing algorithms could run due to the large time cost. In contrast, our algorithm *GE-Graph* spent only a few seconds.

C. Interaction Study

Confidence Study. We studied the situation that a user might be unable to answer some questions. Our study was conducted on a synthetic dataset, where $c_{val} = 10$ and the other parameters were set by default. Given two tuples p and q presented as a question to a user, we define the *utility difference* to be $(f(p) - f(q))/f(p)$, where $f(p) > f(q)$. If the utility difference is smaller than a given threshold ρ, we assume that there is a 50% chance that a user cannot answer the question. Our algorithm *GE-Graph* was adapted so that if the question could not be answered, it would ask another question based on its tuple selection strategy. Figure 18 shows the performance of our algorithm *GE-Graph* by varying ρ. The execution time remained almost unchanged and the number of questions asked slightly increased. This shows that the unanswered questions affect little to our algorithm.

APPENDIX D
PROOFS

Proof of Lemma 1. Consider datasets $\mathcal{D} = \{p_1, p_2, ..., p_n\}$ and $\mathcal{D}' = \{p'_1, p'_2, ..., p'_n\}$, where $\mathcal{D}'$ is a numerical rescaling of $\mathcal{D}$, i.e., (1) for the numerical attributes, there exist positive real numbers $\lambda_1, \lambda_2, ..., \lambda_{d_{num}}$ such that $p'_{k\ num}[j] = \lambda_j p_{k\ num}[j]$, $\forall k \in [1, n]$ and $\forall j \in [1, d_{num}]$; and (2) for the categorical attributes, $p'_{k\ cat}[i] = p_{k\ cat}[i]$, $\forall k \in [1, n]$ and $\forall i \in [1, d_{cat}]$.

Denote $U = \sum_{j=1}^{d_{num}} \frac{u_{num}[j]}{\lambda_j}$. For any utility function f, there exists a corresponding utility function f' such that (1) $\forall j \in [1, d_{num}]$, $u'_{num}[j] = \frac{u_{num}[j]}{U\lambda_j}$; and (2) $\forall i \in [1, d_{cat}]$, $u'_{cat}[i] = \frac{u_{cat}[i]}{U}$. Since $u_{num}[j] \geq 0$ and $\lambda_j > 0$, we have $u'_{num}[j] \geq 0$. The following shows that $\sum_{j=1}^{d_{num}} u'_{num}[j] = 1$.

$$\begin{aligned}\sum_{j=1}^{d_{num}} u'_{num}[j] &= \sum_{j=1}^{d_{num}} \frac{u_{num}[j]}{U\lambda_j} \\ &= \frac{1}{U} \sum_{j=1}^{d_{num}} \frac{u_{num}[j]}{\lambda_j} \\ &= 1\end{aligned}$$

Consider a pair of tuples $p, q \in \mathcal{D}$. If $f(p) > f(q)$, we have the conclusion as follows.

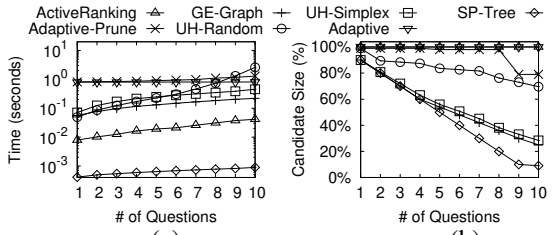


Figure 15: Synthetic (categorical only) ($c_{val} = 10$)

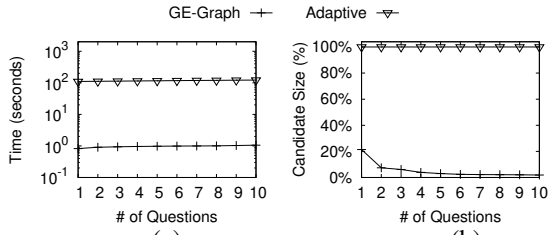


Figure 16: Synthetic ($c_{val} = 10$)

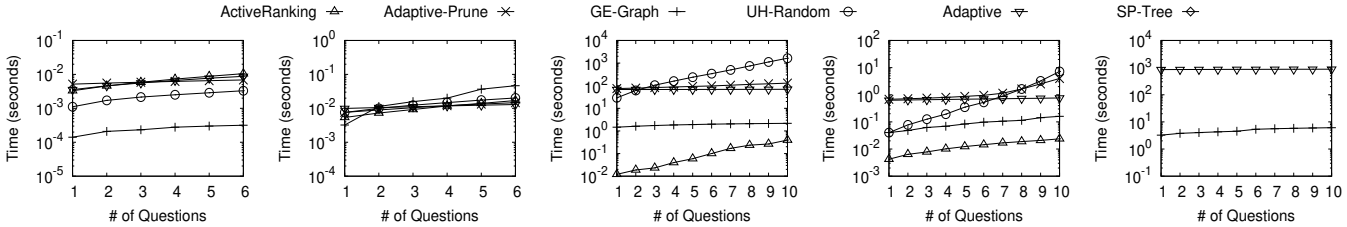


Figure 17: Real Dataset (Execution Time)

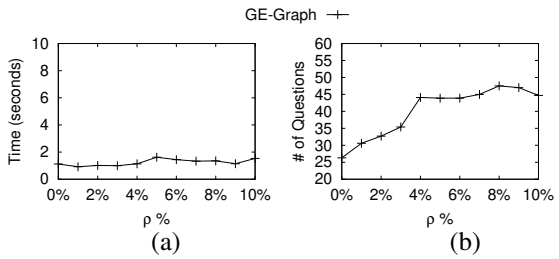


Figure 18: Confidence Study

$$
\begin{aligned}
& f(p) > f(q) \\
\Longrightarrow & \sum_{i=1}^{d_{cat}} u_{cat}[i]h(p_{cat}[i]) + \sum_{j=1}^{d_{num}} u_{num}[j]p_{num}[j] > \\
& \sum_{i=1}^{d_{cat}} u_{cat}[i]h(q_{cat}[i]) + \sum_{j=1}^{d_{num}} u_{num}[j]q_{num}[j] \\
\Longrightarrow & \sum_{i=1}^{d_{cat}} U\frac{u_{cat}[i]}{U}h(p_{cat}[i]) + \sum_{j=1}^{d_{num}} U\frac{u_{num}[j]}{U\lambda_j}\lambda_j p_{num}[j] > \\
& \sum_{i=1}^{d_{cat}} U\frac{u_{cat}[i]}{U}h(q_{cat}[i]) + \sum_{j=1}^{d_{num}} U\frac{u_{num}[j]}{U\lambda_j}\lambda_j q_{num}[j] \\
\Longrightarrow & U\sum_{i=1}^{d_{cat}} u'_{cat}[i]h(p'_{cat}[i]) + U\sum_{j=1}^{d_{num}} u'_{num}[j]p'_{num}[j] > \\
& U\sum_{i=1}^{d_{cat}} u'_{cat}[i]h(q'_{cat}[i]) + U\sum_{j=1}^{d_{num}} u'_{num}[j]q'_{num}[j] \\
\Longrightarrow & Uf'(p') > Uf'(q') \\
\Longrightarrow & f'(p') > f'(q')
\end{aligned}
$$

Therefore, for any utility function f for $\mathcal{D}$, there exists a corresponding utility function f' for $\mathcal{D}'$ such that $\forall p_k, p_l \in \mathcal{D}$, if $f(p_k) > f(p_l)$, then $f'(p'_k) > f'(p'_l)$, where $k, l \in [1, n]$. This proves that rescaling the numerical attributes will not change the ranking of tuples. □

Proof of Theorem 1. Consider a dataset with n tuples as follows. Use $\mathcal{H}_i$ to denote the set which contains all the possible values in the i-th categorical attribute ($|\mathcal{H}_i| = s_i$), where $i \in [1, d_{cat}]$. (1) For the second categorical attribute, we divide the categorical values in $\mathcal{H}_2$ into two sets $\mathfrak{S}_1 = \{A_1, A_2, A_3, ...\}$ and $\mathfrak{S}_2 = \{B_1.B_2, B_3, ...\}$ in equal size (i.e., $|\mathfrak{S}_1| = |\mathfrak{S}_2| = \frac{s_2}{2}$). (2) For the third categorical attribute to the d_{cat}-th categorical attribute, there can be $\prod_{i=3}^{d_{cat}} s_i$ categorical value combinations (i.e., the Cartesian product of $\mathcal{H}_3 \times \mathcal{H}_4 \times ... \times \mathcal{H}_{d_{cat}}$). For simplicity, we represent the set which contains all the combinations by $\mathcal{X} = \{X_1, X_2, X_3, ...\}$.

For each categorical value combination $< A_i, X_j >$, where $A_i \in \mathfrak{S}_1$ and $X_j \in \mathcal{X}$, it corresponds to $\frac{s_2}{2}(\prod_{i=3}^{d_{cat}} s_i - 1)$ categorical values combinations $< B_k, X_l >$, where $B_k \in \mathfrak{S}_2$ and $X_l \in \mathcal{X} \backslash \{X_j\}$. For each pair $(< A_i, X_j >, < B_k, X_l >)$, there exists a pair of tuples p and q such that p contains the values of $< A_i, X_j >$ and q contains the values of $< B_k, X_l >$. Each tuple in the dataset contains a different categorical value in the first attribute. Thus, the dataset contains $\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ different categorical value combinations which consist of the first attribute to the d_{cat}-th attribute in the dataset. Denote the set which contains all the combinations by $\mathcal{Y} = \{Y_1, Y_2, Y_3, ...\}$.

Except Y_1, for each combination $Y_i \in \mathcal{Y}$, there is only one tuple which contains the categorical values in Y_i, where $i = 2, 3, 4, ...$ The rest of the tuples contain the categorical values of Y_1. Besides, all the tuples are different in numerical attributes. Therefore, the dataset contains $\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ tuples with different categorical values and the rest $n - (\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1))$ tuples are the same in categorical attributes.

Consider the questions asked to a user. If the two tuples presented to the user have different categorical values, any algorithm can only prune one tuple after each question and thus, it needs to ask $\Omega(s_1+\prod_{i=2}^{d_{cat}} s_i^2)$ questions. To determine the rest of the tuples which have the same categorical values, as proved in [6], any algorithm that identifies all the tuples which differ in numerical attributes must be in the form of a binary tree. If the binary tree has $n-(\frac{s_1}{2}+\frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i-1))$ leaves, the height of the tree is $\Omega(\log(n-s_1-\prod_{i=2}^{d_{cat}} s_i^2))$. Thus, any algorithm needs to ask $\Omega(\log(n-s_1-\prod_{i=2}^{d_{cat}} s_i^2))$ questions. In total, it needs to ask $\Omega(s_1+\prod_{i=2}^{d_{cat}} s_i^2+\log(n-s_1-\prod_{i=2}^{d_{cat}} s_i^2))$ questions to determine the user's favorite tuple. □

Proof of Lemma 2. Since a user prefers p to q, we have $f(p) > f(q)$, i.e., we have an inequality as follows.

$$\sum_{j=1}^{d_{cat}} g(p_{cat}[j]) > \sum_{j=1}^{d_{cat}} g(q_{cat}[j])$$

Since p and q have the same categorical values in the first $i-1$ attributes, we have the following inequality.

$$\sum_{j=i}^{d_{cat}} g(p_{cat}[j]) > \sum_{j=i}^{d_{cat}} g(q_{cat}[j]) \Longrightarrow \sum_{c\in S_p^i} g(c) > \sum_{c\in S_q^i} g(c)$$

□

Proof of Lemma 3. Since p and q are the same in the first $i-1$ categorical attributes, we have the following equality.

$$\sum_{j=1}^{i-1} g(p_{cat}[j]) = \sum_{j=1}^{i-1} g(q_{cat}[j]) \quad (1)$$

Because $S_p^i \succ S_q^i$, we can obtain another inequality.

$$\sum_{j=i}^{d_{cat}} g(p_{cat}[j]) > \sum_{j=i}^{d_{cat}} g(q_{cat}[j]) \quad (2)$$

Thus, by summarizing equality (1) and inequality (2), we have $\sum_{j=1}^{d_{cat}} g(p_{cat}[j]) > \sum_{j=1}^{d_{cat}} g(q_{cat}[j])$. Note that p and q are only described by categorical attributes. Thus $f(p) > f(q)$, which indicates that p is more preferred by the user than q. □

Proof of Theorem 2. Recall that we process the C-Tree from the bottom to the top. In the i-th level of the C-Tree, where $i \in [2, d_{cat}]$, the size of $\mathbf{S}_i$ is $O(\prod_{k=i}^{d_{cat}} s_k)$, since there might be $O(\prod_{k=i}^{d_{cat}} s_k)$ categorical value combinations which consist of the values from the i-th attribute to the d_{cat}-th attribute. For each pair of sets, there might exist two nodes in the i-th level of the C-Tree and we may need to ask a question. Thus, we need to ask $O(\prod_{k=i}^{d_{cat}} s_k^2)$ questions in the i-th level of the C-Tree. When we process the first level of the C-Tree, each node contains only one categorical value and has one unique path. There will be $O(s_1)$ tuples left in the C-Tree. For each question asked, we could prune at least one tuple. Thus, we need to ask $O(s_1)$ questions in the first level of the C-Tree. In summary, we interact with a user for $O(s_1+\sum_{i=2}^{d_{cat}}\prod_{k=i}^{d_{cat}} s_k^2)$ rounds, i.e., $O(s_1+\prod_{i=2}^{d_{cat}} s_i^2)$ rounds. □

Proof of Corollary 1. As shown in Theorem 1, when $n = s_1 + \prod_{i=2}^{d_{cat}} s_i^2$, the lower bound of the number of questions asked is $\Omega(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$. Since algorithm *SP-Tree* determines the user's favorite point by asking $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions, it is asymptotically optimal in terms of the number of questions asked. □

Proof of Lemma 4. If a user prefers p to q, then $f(p) > f(q)$. Since p and q are the same in all categorical attributes, we have $u_{num} \cdot (p_{num} - q_{num}) > 0$. This indicates that the user's numerical utility vector u_{num} is in h^+_{p-q} according to the definition of hyper-plane h_{p-q}. □

Proof of Lemma 5. Based on the definition of the node, multiset $L = L_1 \ominus L_2$ (or $L_1 \oplus \overline{L_2}$). L_1 (resp. L_2) is a multiset that stores all the categorical values of p (resp. q) as elements with multiplicity 1. If a user prefers p to q, we have the following derivation.

$$\begin{aligned}
& f(p) > f(q) \\
\Longrightarrow & \sum_{i=1}^{d_{cat}} g(p_{cat}[i]) + u_{num} \cdot p_{num} > \\
& \sum_{i=1}^{d_{cat}} g(q_{cat}[i]) + u_{num} \cdot q_{num} \\
\Longrightarrow & \sum_{i=1}^{d_{cat}} g(p_{cat}[i]) - \sum_{i=1}^{d_{cat}} g(q_{cat}[i]) > u_{num} \cdot (q_{num} - p_{num}) \\
\Longrightarrow & \sum_{A\in L_1} \mathfrak{m}_{L_1}(A) g(A) - \sum_{A\in L_2} \mathfrak{m}_{L_2}(A) g(A) > \\
& u_{num} \cdot (q_{num} - p_{num}) \\
\Longrightarrow & \sum_{A\in L} \mathfrak{m}_{L}(A) g(A) > u_{num} \cdot (q_{num} - p_{num})
\end{aligned}$$

□

Proof of Lemma 6. For any $q \in \mathcal{C}_p$, it is the same as p in all categorical attributes. The difference between $f(q)$ and $f(p)$ only depends on the numerical attributes. Consider hyperplane $h_{q_{num}-p_{num}}$ that is based on vector $q_{num} - p_{num}$. $\forall r \in h^+_{q-p}$, $r \cdot q_{num} > r \cdot p_{num}$. Since $\mathcal{R} \subseteq \cup_{q\in\mathcal{C}_p} h^+_{q-p}$, $\forall r \in \mathcal{R}$, $\exists q \in \mathcal{C}_p$ such that $r \cdot q_{num} > r \cdot p_{num}$. Because $u_{num} \in \mathcal{R}$, there exists a tuple $q \in \mathcal{C}_p$ such that $u_{num} \cdot q_{num} > u_{num} \cdot p_{num}$, i.e., $f(q) > f(p)$. Thus, p can be safely pruned from $\mathcal{C}$. □

Proof of Lemma 7. Consider each polyhedron $\mathcal{R}_i$, where $i \in [1, l]$. $\mathcal{R}_i \subseteq h_i^-$ means that for any point $r \in \mathcal{R}_i$, $r \cdot (\triangle y_i + p_{num} - q_{num}) < 0$. Since (1) $\forall i \in [1, l]$, $\mathcal{R}_i \subseteq h_i^-$, and (2) $\cup_{i\in[1,l]}\mathcal{R}_i \subseteq \mathcal{R}$, we have $\forall r \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot \triangle y_i$, where $i \in [1, l]$, such that $r \cdot (\triangle y_i + p_{num} - q_{num}) < 0$. Because $u_{num} \in \mathcal{R}$, we have

$$\begin{aligned}
f(p) - f(q) &= g(L_1) - g(L_2) + u_{num} \cdot (p_{num} - q_{num}) \\
&< \triangle y_i \cdot u_{num} + u_{num} \cdot (p_{num} - q_{num}) \\
&= u_{num} \cdot (\triangle y_i + p_{num} - q_{num}) \\
&< 0
\end{aligned}$$

This shows that the utility of p is smaller than that of q w.r.t. the user preference. Thus, p can be safely pruned from $\mathcal{C}$. □

Proof of Theorem 3. In each interactive round, we can learn the user preference between a pair of tuples and thus, prune at least one tuple from $\mathcal{C}$. Since $|\mathcal{D}| = n$, there must be only one tuple left in $\mathcal{C}$ after $O(n)$ rounds. □