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 attracts 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 feedback, the user preference is learned implicitly and the target tuple w.r.t. the learned preference is returned. However, they mainly 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. Even if the categorical attributes can be reduced into numerical ones using conventional strategies (e.g., one-hot encoding), existing methods do not work well. 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 any 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 a user wants a used car. The user has a trade-off between price and brand and is willing to pay more for a famous brand than an obscure one. To model the preference, a commonly-used strategy is to utilize numerical weights in a *linear utility function* [3]–[7]. Precisely, it quantifies the user preference, and characterizes a vector comprising one weight per attribute. Based on the linear utility function, each tuple is associated with a *utility* (i.e., a function score). It indicates to what extent the user prefers the tuple, where a higher utility means that the tuple is more favored by him/her and the tuple with the highest utility is assumed to be the 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 mainly 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), The user's trade-off involves numerical and categorical attributes. The existing operators have difficulties in handling both types of attributes well to recommend the user a satisfactory car.

Motivated by the needs of handling both types of attributes, we propose problem *Interactive Search with Mixed Attributes (ISM)*, which interacts with a user 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. Our interactive framework mainly follows [6]–[8], where 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 an assumption proposed by [9] that *users can tell which tuple they prefer the most from a set of tuples*. This assumption has been studied by many existing works [6], [7]. In Section VI-C, we also studied this assumption by considering the cases in which users may not be able to answer some questions, and may make mistakes during the interaction. Besides, we conducted a user study to evaluate our proposed algorithm in real scenarios. 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], [10]. 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 a user a black car and a white car, and asks which one s/he prefers.

In marketing research [11], [12], 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] verify that the latter case asks 4-10 times more questions than the former case. Its user study also shows 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, a user plans to rent an apartment for several months. Since s/he will only live in one apartment for a short time, it suffices to return the best candidate. The user might be frustrated due to the long selection process even if finally, several candidates are returned. Note that our proposed algorithms can be extended to returning multiple tuples. The extension is shown in our Appendix B.

To the best of our knowledge, we are the first to study problem ISM. There are closely related studies, e.g., [6], [8], [13]. But they are different from ours by mainly considering numerical attributes, while we focus on both numerical and categorical attributes. In this sense, existing studies can be seen as our special case. Note that [6], [8], [13] can be adapted to solve problem ISM by transforming categorical attributes to numerical ones using conventional strategies (e.g., one-hot encoding) [8]. However, none of them work satisfactorily, i.e., as shown empirically, they asked many questions and ran slowly. For example, on the dataset with two categorical and four numerical attributes, they either ask hundreds of questions or execute in thousands of seconds. The main reason behind this is that categorical attributes are discrete and unordered while numerical attributes are continuous and ordered. Even if we transform categorical attributes, the generated attributes cannot get rid of the categorical nature. Their values are still discrete (e.g., 0 or 1) rather than continuous. Thus, the existing strategies designed for numerical attributes cannot be applied efficiently and effectively to categorical attributes.

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 they see 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 users may be disappointed if the displayed tuples with which they are 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] into 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 approximated the user preference by interacting with users. Since it focused on learning the user preference instead of recommending tuples, it might ask many redundant questions in which our problem ISM is not interested. For example, if a user prefers car p_1 to both cars p_2 and p_3, the user preference between p_2 and p_3 is less interesting in our problem, but this additional information may be useful in [8]. [8] proposed to transform categorical attributes to numerical attributes by using the standard SVM convention. 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 shown in Section VI, adapted algorithms from [6], [8], [13] performed poorly. They asked dozens more questions and took orders of magnitude longer time than ours. As remarked in Section I, this is because the categorical nature is unchanged even with transformation. Thus, the optimization strategies for continuous numerical attributes are not effective for discrete categorical attributes. [14] studied to find 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, they rely on the continuous property of attributes and are difficult to be applied to the datasets with categorical attributes.

In the machine learning (ML) area, our problem is related to the *learning to rank* problem [13], [15]–[18], which learns the ranking of tuples by pairwise comparisons. However, most existing algorithms [15]–[18] failed to utilize the inter-relation between tuples. For example, the attribute "price" is an interrelation between cars. Given a \$3000 car p and a \$5000 car q that have the same values in other attributes, 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. [13] considered the inter-relation between tuples to learn the ranking. However, it was only based on the numerical attributes. Besides, [13], [16], [18] focused on deriving the full ranking of tuples, which needs to ask excessive questions that are not concerned by our problem ISM due to the similar reason stated for [8].

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 the 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 studies [6], [9], [13] 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], [13], [15]–[17] ask a lot of questions since they either require learning a full ranking of tuples or aim at learning an exact user preference. Secondly, [15]–[17] fail to utilize the inter-relation between tuples, and thus, involve a lot of unnecessary interaction.

III. Problem Definition

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 $i \in [1, d_{cat}]$.

Table II: Summary of Frequently Used Notations

Notations	Definition
$\mathcal{D}$ and p, q	The dataset and tuples.
d_{cat} / d_{num}	The number of categorical/numerical attributes.
$p_{cat}[i]$ / $p_{num}[j]$	The value of p in the i-th categorical attribute ($i \in [1, d_{cat}]$)/j-th numerical attribute ($j \in [1, d_{num}]$).
$u_{cat}[i]$ / $u_{num}[j]$	The importance of the i-th categorical attribute/j-th numerical attribute to the user.
$g_i(\cdot)$	$g_i(p_{cat}[i]) = u_{cat}[i]h(p_{cat}[i])$ ($i \in [1, d_{cat}]$).
S_p^i	The set $\{p_{cat}[i], p_{cat}[i+1], ..., p_{cat}[d_{cat}]\}$.
h_z^+ / h_z^-	The half-space above/below hyper-plane h_z.
$\mathcal{R}$ / $\mathcal{G}$ / $\mathcal{C}$	The numerical utility range. / The relation graph. / The candidate set.

Example 1. Consider Table III(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 existing setting [6]–[8], [19], 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 studies conducted by [7], [8] also verify 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 $h : p_{cat}[i] \to \mathbb{R}_+$ maps each categorical value to a real number which indicates to what extent a user favors a categorical value. A larger number is more preferred.
- 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], [20], 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 the user's favorite tuple. By definition, rescaling the numerical attributes does not change the ranking of tuples w.r.t. the utility function (i.e., scale invariant [5], [6]). Thus, we assume *w.l.o.g.* that each numerical attribute is normalized to $(0, 1]$ and a larger value is more favored.

In the above formulation, each term in f related to the categorical attributes appear 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

Table III: 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) Dataset and Utilities

$g_i(\cdot)$
$g_1(A_1) = 0.2$
$g_1(A_2) = 0.3$
$g_2(B_1) = 0.4$
$g_2(B_2) = 0.5$

(b) $g_i(\cdot)$

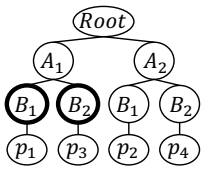


Figure 1: C-Tree

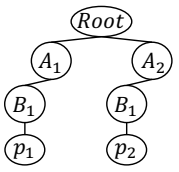


Figure 2: Pruned C-Tree

effect without considering the interplay between $u_{cat}[i]$ and $h(p_{cat}[i])$. Specifically, we use a *single* function g_i to denote both $u_{cat}[i]$ and $h(p_{cat}[i])$ (i.e., we use a *single* unknown variable) such that function $g_i : p_{cat}[i] \to \mathbb{R}_+$ is defined to be: $g_i(p_{cat}[i]) = u_{cat}[i]h(p_{cat}[i])$ and $i \in [1, d_{cat}]$. In this way, we focus on learning $g_i(p_{cat}[i])$ 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 frequently used notations are shown in Table II.

Example 2. Continue Example 1. Let $u_{num} = (0.5, 0.5)$. The utility function f for a tuple p is expressed as $f(p) = g_1(p_{cat}[1]) + g_2(p_{cat}[2]) + 0.5 \times p_{num}[1] + 0.5 \times p_{num}[2]$. 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 III(b). Since $g_1(A_2) = 0.3$ and $g_2(B_1) = 0.4$, the utility of p_2 w.r.t. f is $f(p_2) = 0.3+0.4+0.5\times0.6+0.5\times0.9 = 1.45$. 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. □

Our interactive framework follows [6]–[9] and works on the dataset with mixed attributes. Specifically, we interact with a user for rounds until we find the user's favorite tuple. Each round consists of three components. (1) *Tuple selection*. Based on the user's previous answers, we select two tuples and ask the user to pick the one s/he prefers. Each tuple is a combination of categorical and numerical values. For example, a user 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 stopping condition is satisfied. If so, we terminate the interaction process and return the result. Otherwise, we start a new interactive round. Formally, we are interested in the following problem. For the lack of space, the complete proofs of some theorems/lemmas can be found in Appendix D .

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 [21]. For example, in the job application scenario, the candidates may provide their gender, speciality, and hobbies, which are all categorical attributes.

Intuitively, our algorithm *SP-Tree* maintains tuples in a tree data structure, called *categorical value tree*, or *C-Tree* in short. During the interaction, tuples are 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 that 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).

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 and its stored categorical value is 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 III. 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 III, $S_{p_1}^2 = \{B_1\}$. 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 1. If a user prefers p to q, then $\sum_{c_j \in S_p^i} g_j(c_j) > \sum_{c_j \in S_q^i} g_j(c_j)$ (c_j is the value in the j-th categorical attribute).

For simplicity, we denote $\sum_{c_j \in S_p^i} g_j(c_j) > \sum_{c_j \in S_q^i} g_j(c_j)$ 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 2. 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 work on 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 work on 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.

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 asked 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 $S_p^i \succ S_{q'}^i$.

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 to be processed. Initially, we insert $S_p^i \succ S_q^i$ into Q. Then, we continually pop out relations from Q until Q is empty. For each popped out relation, we use rules (1)-(3) to derive more relations and insert the derived ones (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 Mixed Attributes

We consider the general case of ISM in which tuples are described by categorical and numerical attributes, and propose algorithm *GE-Graph*. Intuitively, we maintain three data structures: (a) A numerical range $\mathcal{R} \subseteq \mathbb{R}^{d_{num}}$, which maintains

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 s.t.* $p_{cat}[l] = q_{cat}[l]$ $\forall l \in [1, i-1]$, *and* $S_p^i = S_{q_j}^i$ $S_q^i = S_{q_k}^i$ **then**
        Show p, q to users. Update the C-Tree.
return The only tuple left in the C-Tree

the learned preference on numerical attributes. (b) A hyper-graph $\mathcal{G}$, which maintains the learned preference on categorical attributes. Here, we use two distinct data structures to handle numerical and categorical attributes, respectively, since the two types of attributes possess different characteristics (continuous vs. discrete). (c) A tuple set $\mathcal{C}$ comprising tuples in $\mathcal{D}$ which are the candidates as the final answer. During the interaction, we select tuples from $\mathcal{C}$ as questions (tuple selection). According to the user feedback, we update $\mathcal{R}$ and $\mathcal{G}$ accordingly and prune from $\mathcal{C}$ the tuples that 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 finally left tuple (stopping condition).

A. Preliminaries

Hyper-plane. In a d_{num}-dimensional geometric space $\mathbb{R}^{d_{num}}$, given a d_{num}-length vector z in $\mathbb{R}^{d_{num}}$, we can build a *hyper-plane* h_z: $\{r \in \mathbb{R}^{d_{num}} \mid r \cdot z = 0\}$, which passes through the origin with its unit normal in the same direction as z [22]. Hyper-plane h_z divides $\mathbb{R}^{d_{num}}$ into two halves, called *half-spaces* [22]. The half-space above h_z (resp. below h_z), denoted by h_z^+ (resp. h_z^-), contains all the points $r \in \mathbb{R}^{d_{num}}$ such that $r \cdot z > 0$ (resp. $r \cdot z < 0$). In geometry, a *polyhedron* $\mathcal{P}$ is an intersection of a set of hyper-planes and half-spaces [22].

Multiset. A *multiset* L is a collection of elements, in which elements are allowed to repeat [23], [24]. 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 is *absent* in a multiset. The latter means that an element is not in the multiset. We store categorical values as elements in multisets. 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 two 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 $h_{p_{num} - q_{num}}$ based on vector $p_{num} - q_{num}$, and update $\mathcal{R}$ following Lemma 3. In the following, if tuples p and q are the same in all categorical attributes, we denote the hyper-plane by h_{p-q} for simplicity.

Lemma 3. 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 $\mathcal{G} = (V, E)$ [25], [26], called the *relational graph*, to store the relations between categorical values based on the learned user preference (e.g., the difference between $g_1(A_1)$ and $g_1(A_2)$, where A_1 and A_2 are two categorical values). V is a set of nodes. 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 to update $\mathcal{G}$.

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 4. If a user prefers p to q (i.e., $f(p) > f(q)$), we obtain an inequality $\sum_{c_i \in L} \mathfrak{m}_L(c_i) g_i(c_i) > u_{num} \cdot (q_{num} - p_{num})$, where c_i denotes a value in the i-th categorical attribute.

For simplicity, with a slight abuse of notations, we denote $\sum_{c_i \in L} \mathfrak{m}_L(c_i) g_i(c_i)$ in Lemma 4 by $g(L)$. We 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 III that have categorical values A_1, B_1 and A_2, B_1, respectively. We build

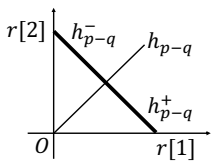


Figure 3: Space $\mathbb{R}^{d_{num}}$

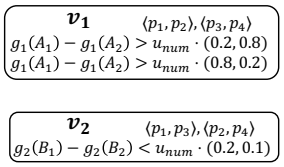


Figure 4: Bounds in Nodes

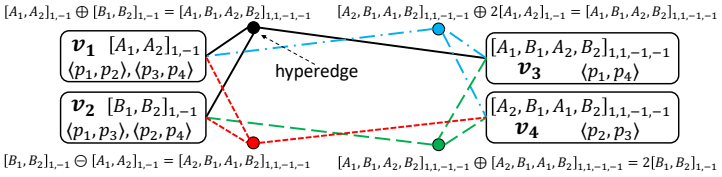


Figure 5: Relational Graph

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., $g_1(A_1) - g_1(A_2) > u_{num} \cdot (p_{2\ num} - p_{1\ num})$. 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 the 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 $u_{num} \cdot \triangle x$. If $\mathcal{R}$ is smaller, it is more likely 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 use linear programming to check if a bound is untight. For the lack of space, see Appendix A.

Hyper-edge. Each hyper-edge in E connects three nodes in V, indicating 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 multisets can 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 two multisets cannot derive the third one. Without loss of generality, suppose that L_1 and L_2 can deduce L_3. Let $\odot \in \{\oplus, \ominus\}$. 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 the 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. Other hyper-edges are similarly computed and 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 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 point r in the updated $\mathcal{R}$, $r \cdot (\triangle x_2 - \triangle x_1) > 0$. 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 tuple pair $\langle p, q\rangle$ and multiset L. If a user prefers p to q, we can obtain a lower bound $u_{num} \cdot (q_{num} - p_{num})$ for $g(L)$, denoted by $u_{num} \cdot \triangle x$ for simplicity. We add $u_{num} \cdot \triangle x$ into v and update v in two steps. (1) We utilize all 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 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 $g_2(B_1) - g_2(B_2)$, 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 in v_2 based on the updated $\mathcal{R}$. □

After the update of node v, we can utilize bound $u_{num} \cdot \triangle x$ in v to generate new bounds for the neighbor nodes of v connected by hyper-edges. Assume that nodes $v_1, v_2 \in V$ with multisets L_1 and L_2 are two neighbor nodes of v. 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 following other rules, we can generate new bounds similarly. Please see Appendix A.

After obtaining the newly generated bounds for the neighbor nodes v_c of v, we can update v_c in the same way as we update v. The newly generated bounds may further *trigger* the update of the neighbor nodes of v_c. The update will be continually triggered and multiple nodes in $\mathcal{G}$ will be updated.

Specifically, we process the nodes in *batches* to update $\mathcal{G}$. The nodes to be updated in the b-th batch ($b \geq 1$) are stored in queue Q_b. Initially, node v (containing $\langle p, q\rangle$ that are presented to a user) is inserted into queue Q_1, and node v is updated in the first batch. In the b-th batch, we pop out nodes in Q_b until Q_b is empty. For each popped-out node v_p, we update it and generate new bounds for each of its neighbor node v'_p. We insert v'_p into Q_{b+1} and node v'_p is to be updated in the $(b+1)$-th batch if the newly generated bounds are tight. 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. Other acceleration strategies (e.g., indexing) are discussed in Appendix A.

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 process, we determine and prune from $\mathcal{C}$ the tuples that cannot be the user's favorite tuple based on $\mathcal{R}$ and $\mathcal{G}$.

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

Lemma 5. 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 as shown 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 6. 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 m_i vertices in $\mathcal{R}_i$, to identify if $\mathcal{R}_i \subseteq h^-_i$, we need $O(m_i)$ time to check if all vertices of $\mathcal{R}_i$ are in h^-_i. The total time is $O(\sum_{i=1}^{l} m_i)$. 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 node v as shown 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 $\mathcal{C}$. □

Algorithm 2: The *GE-Graph* Algorithm

```
Input: A tuple set D, two update parameters α and ε
Output: The user's favorite tuple in D
R ← {r ∈ R_+^{d_num} | Σ_{i=1}^{d_num} r[i] = 1}.
Initialize G and C ← D.
while |C| > 1 do
    v_r ← the selected reference node.
    if Gain(p2,q2,v_r) / Gain(p1,q1,v_r) ≤ ε then
        Display ⟨p1, q1⟩ and obtain the user's answer A.
        Update R with h^+_{p1−q1}/h^+_{q1−p1} according to A.
    else
        Display ⟨p2, q2⟩ and obtain the user's answer A.
        v ← the node in G containing ⟨p2, q2⟩.
        b ← 1 and Q_b ← {v}.
        while b ≤ α do
            while |Q_b| ≠ ∅ do
                Pop a node v_p from Q_b.
                Update bounds in v_p according to A.
                Insert all neighbor nodes of v_p in Q_{b+1}.
            b ← b + 1.
        Update R based on the updated nodes in G.
    for each tuple p ∈ C do
        Check if p can be pruned from C based on C_p/C'_p.
return The only tuple left in C.
```

C. Tuple Selection

In each round, we select two tuples from $\mathcal{C}$ as a question. The restriction of selecting tuples from $\mathcal{C}$ ensures that $|\mathcal{C}|$ becomes strictly smaller after each question. This is because we can know the user preference on the two selected tuples, and thus, prune at least one tuple from $\mathcal{C}$. Our tuple selection strategy has 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 values, it is more difficult to analyze the user preference, since it is hard to learn the preference difference is caused by which pairs of categorical values.

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

Consider a reference node $v_r \in V$. We define the *gain* of $\langle p, q\rangle$ w.r.t. v_r, denoted by $Gain(p, q, v_r)$, to be the number of tuples stored in v_r that can be pruned from $\mathcal{C}$ if we present $\langle p, q\rangle$ as a question to a user. If $\frac{Gain(p_2,q_2,v_r)}{Gain(p_1,q_1,v_r)}$ is large than a given parameter ϵ, we use $\langle p_2, q_2\rangle$ as a question. Otherwise, we turn to $\langle p_1, q_1\rangle$. In practice, we can use any node (e.g., the one storing $\langle p_2, q_2\rangle$) as reference node v_r. 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.

D. Analysis

The pseudocode of *GE-Graph* is presented in Algorithm 2 and Theorem 3 gives a worst-case guarantee for *GE-Graph*.

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

In an extreme case where each tuple is described by a *unique* categorical value, any algorithms need to ask $\Omega(n)$ questions, since each categorical value has to be asked at least once. Otherwise, its utility cannot be determined. However in practice, *GE-Graph* performs much better as shown in Section VI. It can prune many tuples from $\mathcal{C}$ in each round and thus, easily achieve the stopping condition (i.e., only one tuple left in $\mathcal{C}$).

VI. Experiment

Our experiments were conducted 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 synthetic datasets, we considered both *anti-correlated* and *correlated* distributions for numerical attributes [27]. Their categorical attributes were generated according to a Zipfian distribution [2], [28], where the Zipfian parameter is set to 1. For real datasets, we used *Car* [6], *Diamond* [29], *Earthquake* [30], and *Pollution* [31]. Dataset *Car* includes 69,052 used cars with 3 categorical (fuel type, vehicle type, and gearbox) and 4 numerical attributes (price, date of manufacture, horsepower, and used kilometers). Dataset *Diamond* has 119,308 records with 2 categorical (shape and color) and 2 numerical attributes (price and carat). Dataset *Earthquake* contains 62,665 records with 2 categorical (magType and type) and 4 numerical attributes (depth, mag, rms, and gap). Dataset *Pollution* has 608,700 records with 2 categorical attributes (state and year) and 3 numerical attributes (CO, SO_2, and NO_2).

For all datasets, each numerical attribute was normalized to $(0, 1]$. Following prior settings [6], [7], which preprocessed datasets to include only skyline tuples, we also preprocessed datasets in the same manner to fairly compare our algorithms with existing ones in a similar environment as described in the original studies. Specifically, tuple $p \in \mathcal{D}$ was retained if $\nexists q \in \mathcal{D}$ such that (1) p and q were the same in all categorical attributes and (2) q was not worse than p in all numerical attributes and was strictly better than p in at least one numerical attribute. After preprocessing, there were 3639, 513, 741, and 5150 skyline tuples in datasets *Car*, *Diamond*, *Earthquake*, and *Pollution*, respectively. The synthetic datasets typically had $10^2 - 10^4$ skyline tuples. The exact statistics of synthetic datasets can be found in Appendix C.

Algorithms. We compared *SP-Tree* and *GE-Graph* against *ActiveRanking* [13], *QuickSort* [16], *AR* [18], *Adaptive* [8], *UH-Random* [6], and *UH-Simplex* [6]. The first four competitors were ML methods adopting popular ML techniques to learn the rank or the user preference via user interaction. The last two were proposed for the preference queries, that utilized user interaction to identify desired tuples for users.

For the existing algorithms designed for numerical attributes only, we applied the commonly used *one-hot encoding* technique [8], [32], to transform the values of all categorical attributes into the numerical form. Specifically, for each categorical value from all categorical attributes, we created 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.

Since the existing algorithms were not proposed to find the user's favorite tuple directly, we adapted them as follows. (1) *ActiveRanking* [13] and *QuickSort* [16] learned the ranking of tuples. We returned the top-ranked tuple when obtaining the ranking. (2) *AR* [18] divided tuples into l sets, where $l \leq n$ and tuples in the i-th set were more preferred by the user than those in the j-th set for all $i, j \in [1, l]$ and $i < j$. We set $l = 2$ and set the sizes of two sets to be 1 and $n-1$, respectively. (3) *UH-Simplex* and *UH-Random* [6] returned a tuple whose so called *regret ratio* satisfied a given threshold. We set the threshold to 0 so that the returned tuple must be the user's favorite one. (4) *Adaptive* [8] learned the user preference. According to [8], the learned user preference was 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 top-ranked tuple w.r.t. the learned user preference. Besides, we created a version of *Adaptive*, called *Adapt-Prune*, which incorporated the tuple pruning strategy from [6] for comparison.

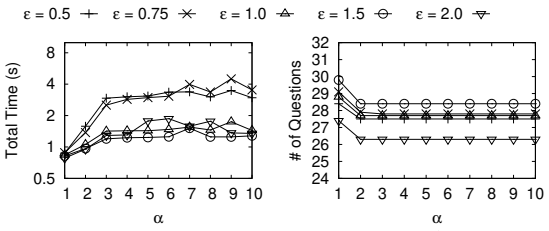
Parameter Setting. We evaluated the algorithms by varying: (1) The dataset size n. (2) The number of numerical (resp. categorical) attributes d_{num} (resp. d_{cat}). (3) The cardinality of each categorical attribute c_{val} (i.e., the number of possible values in each categorical attribute). Following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

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*.

Unless stated explicitly, each algorithm was conducted ten times with different utility functions. We stopped an algorithm if its total execution time was more than 10^4 seconds. Since different utility functions may cause different convergence results, we randomly generated the utility functions and expected to see the average convergence performance. In the following experiments, we reported the average results.

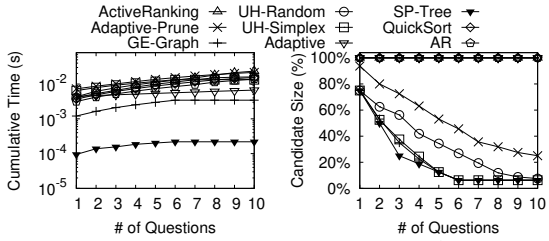
A. Results on Synthetic Datasets

Performance Study. Recall that we introduce two parameters α and ϵ for our algorithm *GE-Graph*, where α controls the update of $\mathcal{G}$ and ϵ is used for tuple selection. We explored their value settings, by varying α from 1 to 10 and ϵ from 0.5 to 2 on synthetic datasets; other parameters were set by default. Figure 6 shows the total time and the number of questions



(a) (b)

Figure 6: Update Strategy



(a) (b)

Figure 7: Synthetic (Categorical-only & Anti)

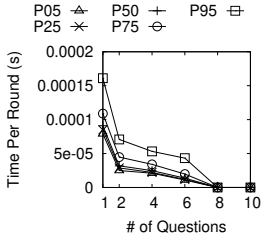
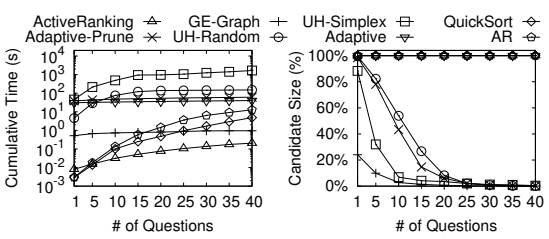


Figure 8: SP-Tree (Time per round statistic)



(a) (b)

Figure 9: Synthetic (Mixed & Anti)

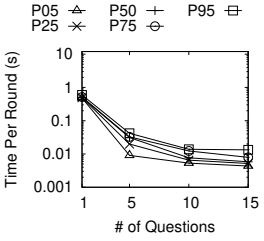
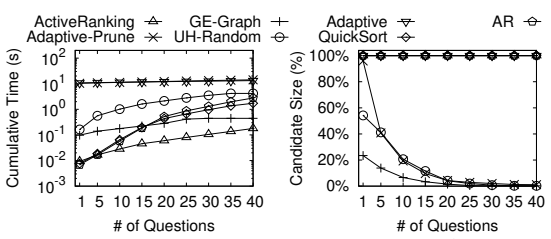


Figure 10: GE-Graph (Time per round statistic)



(a) (b)

Figure 11: Synthetic (Mixed & Correlated)

asked. When α increased, the total 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 many updated nodes in $\mathcal{G}$ helped little to prune tuples. Thus, we set $\alpha = 2$ in the rest experiments. Similarly, we set $\epsilon = 2$ since algorithm *GE-Graph* asked the fewest questions when $\epsilon = 2$.

Progress Analysis. We showed how the algorithms progressed during the interaction process. We reported the middle results, i.e., the candidate size and execution time (including total time, cumulative time, and time per round). Note that the candidate size was an indicator of termination (i.e., it implied the number of questions asked). If the candidate size was reduced at a faster pace, our algorithms would stop more quickly. We evaluated our algorithms on several datasets.

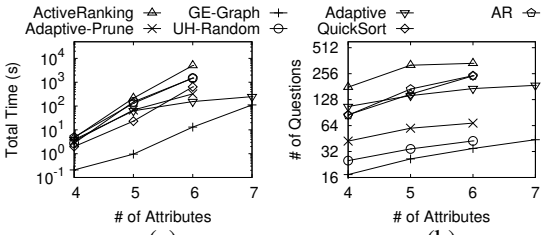
The first dataset was a categorical-only dataset in Figure 7. Our algorithms were the fastest and reduced the candidate size the most effectively. In particular, *SP-Tree* designed for the special case of ISM only took 10^{-4} seconds to ask 6 questions. It was the only one that reduced the candidates by 50% after 2 questions. To show how *SP-Tree* interacted with users, we ran it 100 times with different utility functions and plotted the distribution of time per round (P05, P25, P50, P75, and P95) in Figure 8. The time per round of *SP-Tree* decreased and it only spent 10^{-5}−10^{-4} seconds to interact with users in each round.

The second one was a mixed-attribute dataset in Figure 9. Since *SP-Tree* only worked for the special case of ISM, it was excluded. *UH-Simplex* failed to reduce the candidate size effectively and ran the slowest. This was because its tuple selection strategy was developed based on the *convex hull* technique, which was costly for multiple attributes. It took more than 1,000 seconds to ask 25 questions, while the second slowest algorithm only took 142 seconds. Since *UH-Simplex* was too slow, 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 each question and *Adaptive* maintained a costly data structure. *Adaptive-Prune* inherited both time-intensive issues. Besides, *Adaptive* did not reduce the candidate size since it only learned the user preference. *Adaptive-Prune* and *UH-Random* reduced the candidate size ineffectively. Their pruning strategy treated all attributes equally and neglected the different characteristics of categorical and numerical attributes as remarked in Section I. *GE-Graph* performed the best in reducing the candidate size. It pruned more than 90% tuples from $\mathcal{C}$ after 5 questions. *ActiveRanking*, *QuickSort*, and *AR* ran faster than *GE-Graph* in the first few questions, since they did not have a tuple pruning step, and thus, did not need to process many tuples in the beginning. However, without exploiting the relations between tuples, they asked more questions. As the process continued, *GE-Graph* needed less time for each round since fewer and fewer tuples were left to process (see Figure 10). Note that Figure 9 only showed the cumulative time up to 40 questions, where *GE-Graph* already terminated (since its lines became flat) but the others did not. The *total* time of *GE-Graph* was the smallest (see Figure 12). The time distribution of *GE-Graph* was shown in Figure 10. When the number of rounds increased, *GE-Graph* took less time (0.003-0.6 seconds on average) for each round.

We also tested algorithms on other datasets with different distributions, number of attributes, and cardinality. For example, in Figure 11, we showed the performances of algorithms on a mixed-attribute correlated dataset. The results verified that our proposed techniques did not rely on attribute independence and can be applied to various types of datasets. For the lack of space, the results on some other datasets were shown in Appendix C.

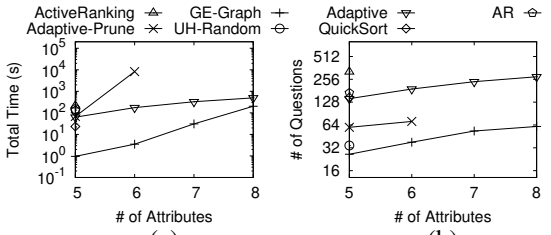
Scalability. We next studied the scalability of algorithms by varying different parameters, e.g., d_{num}, d_{cat}, c_{val}, and n.

Varying d_{num}. In Figure 12, we varied the number of numerical attributes d_{num} from 2 to 5. The results showed that our algorithm *GE-Graph* preformed well when we scaled the



(a) (b)

Figure 12: Vary d_{num} (Mixed & Anti)



(a) (b)

Figure 13: Vary d_{cat} (Mixed & Anti)

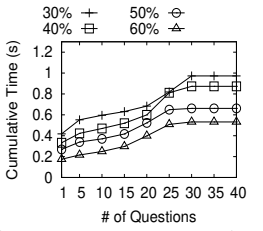
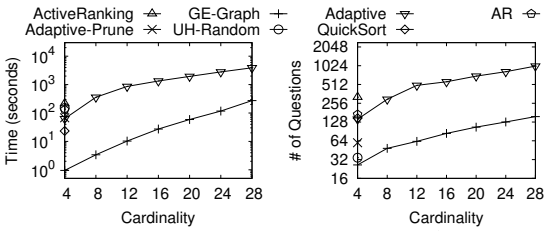
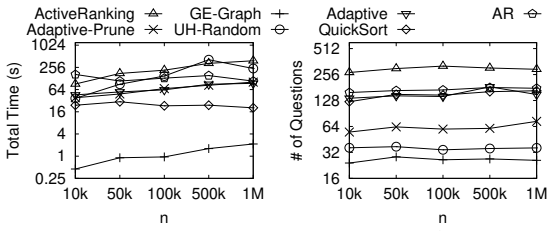


Figure 14: Vary Sparsity (Time)



(a) (b)

Figure 15: Vary c_{val} (Mixed & Anti)



(a) (b)

Figure 16: Vary n (Mixed & Anti)

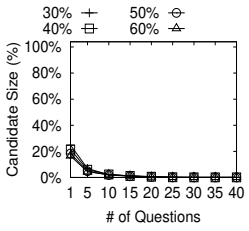


Figure 17: Vary Sparsity (Candidate size)

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 13, we varied the number of categorical attributes d_{cat} from 2 to 5. Most existing algorithms could not complete when $d_{cat} > 3$ due to the excessive time ($\geq 10^4$ seconds). They handled a large amount of attributes since the one-hot encoding mapped each categorical value to an attribute. In contrast, *GE-Graph* utilized the relational graph to handle categorical values. It scaled well w.r.t. d_{cat}, e.g., when $d_{cat} = 3$, it asked $22\% - 82\%$ fewer questions and ran $1 - 4$ orders of magnitude faster than the existing ones.

Varying c_{val}. In Figure 15, we varied the cardinality c_{val} of categorical attributes from 4 to 28. *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 other competitors (e.g., *AR*) when $c_{val} = 4$, since they took too much time due to the excessive attribute issue as stated previously.

Varying n. In Figure 16, 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.

Vary Sparsity. We evaluated *GE-Graph* on sparse data by varying the *sparse ratio*, defined to be the portion of zero or null values, from 30% to 60% in Figure 14 and 17. When the data was sparser, tuples were less different (since they had zero or null values in more attributes). Thus, the information maintenance step (which dominates the time cost) in *GE-Graph* became easier, e.g., relational graph $\mathcal{G}$ only built nodes for tuples with different categorical values, resulting in a lower time cost. Meanwhile, *GE-Graph* reduced the candidate size effectively for different sparse ratios, leading to a small number of questions asked (about 20-24 questions were asked).

B. Results on Real Datasets

In Figure 18, we explored how algorithms progressed during the interaction process on real datasets by reporting the candidate size. For dataset *Car*, we processed it into two versions. The first version was exactly *Car*. The second version, namely *Car*(Cat), only maintained the categorical attributes in dataset *Car*. The results of some existing algorithms was incomplete since they spent more than 10^4 seconds.

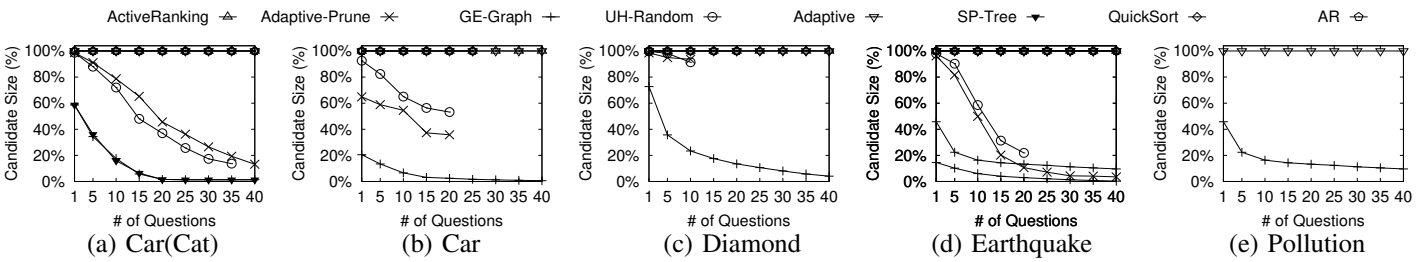
Our algorithms consistently reduced the candidate size the most effectively on all datasets. For example, after asking 15 questions on *Pollution*, *GE-Graph* had fewer than 15% tuples in $\mathcal{C}$. The advantage of *GE-Graph* was more evident on some datasets, e.g., *Car* and *Earthquake* ($\geq 80\%$ tuples were pruned after the first question) since there were many tuple pairs, say $\langle p, q\rangle$ and $\langle p', q'\rangle$ in these datasets, such that p differed from q in the same way as how p' differed from q'. When a user preferred p to q, we could prune q as well as q' from $\mathcal{C}$ (since the user would also prefer p' to q'). This led to a large portion of tuples to be pruned after each question. On average, *GE-Graph* asked 18.2, 54.8, 54.8, 38.7, and 166.6 questions on *Car(Cat)*, *Car*, *Diamond*, *Earthquake*, and *Pollution*, respectively, while the best competitor *Adaptive* asked 107.1, 287.4, 272.8, 206.1, and 985.3 questions.

The cumulative time, as shown in Appendix C, was consistent to that in Section VI-A. As remarked there, *ActiveRanking*, *QuickSort*, and *AR* were faster in the first few questions since they did not need to process many tuples at the beginning. However, they required asking more questions and running in longer total times than ours. The details were shown in Appendix C.

C. Interaction Study

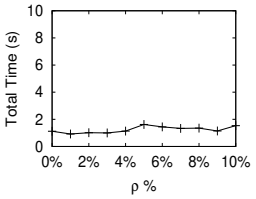
We next justified our interactive assumption (Section I) and presented a user study for *GE-Graph* (*SP-Tree* is similar).

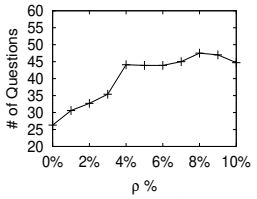
Confidence Study. We studied the case on a synthetic dataset ($c_{val} = 10$) that a user was unable to answer some questions.



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

Figure 18: Real Datasets





(a) (b)

Figure 19: Confidence Study

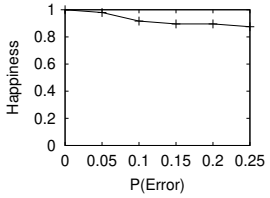


Figure 20: Error Study

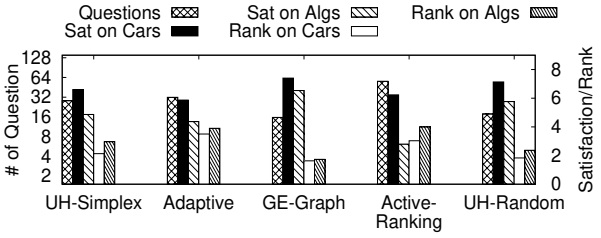


Figure 21: User Study

Given a tuple pair $\langle p, q\rangle$, if *the ratio of their utility difference* $\frac{|f(p)-f(q)|}{\max\{f(p),f(q)\}}$ was smaller than a threshold ρ, we assumed that there was a 50% chance that a user could not answer this question; in this case, another question would be asked. As shown in Figure 19, the total time was almost unchanged and the number of questions only slightly increased. This indicated that unanswered questions affected little to our algorithm.

Error Study. We considered the case where users might make mistakes. We assumed that there was a possibility, denoted by $P(Error)$, that a user gave wrong answers that were different from their real preferences. By varying $P(Error)$, we reported the *happiness* of users, defined to be $\frac{f(p)}{f(p^*)}$ [33], where p (resp. p^*) is the tuple returned by *GE-Graph* (resp. the user's true favorite tuple). The closer the happiness to 1, the better the result. As shown in Figure 20, although the happiness decreased slightly, it was over 0.87 even if the possibility of giving wrong answers was up to 0.25. This showed that our algorithm was affected little by the user mistakes.

User Study. We conducted a user study on dataset *Car* to demonstrate the robustness of our algorithm during real interaction. Following [6], [7], we randomly selected 1000 candidate cars from the dataset described by 5 attributes (price, year of manufacture, horsepower, used kilometer, and fuel type). We recruited 30 participants and reported their average result.

We compared *GE-Graph* against *UH-Random*, *UH-Simplex*, *Adaptive* (the ML method that asked the fewest questions), and *ActiveRanking* the fastest ML method). Each algorithm aimed at finding the user's favorite car. Since the user preference was unknown for real users, we re-adapted *Adaptive* following [6], [7] (instead of the way described previously). We compared the user's answer to some randomly selected questions with the prediction w.r.t. an estimated preference u_e. If 75% answers were correctly predicted, *Adaptive* returned the top-ranked car w.r.t. u_e. Other algorithms followed our original adaptation.

Each algorithm was evaluated by five metrics: (1) an objective metric, which was *The number of questions*, and two subjective metrics (2) *Satisfaction on cars* and (3) *Satisfaction on algorithms*, evaluating how satisfied a user was with the returned cars and the algorithms, respectively. Both subjective metrics asked users to give scores from 1 to 10. A higher score indicated that the user felt more satisfied with the returned car (resp. the algorithm). Since a user may give the same score to different cars/algorithms, we asked each user to give (4) *a rank on the returned cars* and (5) *a rank on the algorithms*.

Figure 21 showed that our algorithm performed the best for all metrics. For example, it asked about $11\% - 71\%$ fewer questions than the competitors. Its satisfactions (resp. ranks) on both returned cars and algorithms were the best, which were 7.4 and 6.5 (resp. 1.6 and 1.7), respectively. This verified that our algorithm could work well in real-world scenarios.

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 real users may make mistakes 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 show that our algorithms are efficient and effective. As for future work, we consider providing guarantees in the case where users may make mistakes in answering questions.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Weicheng Wang and Raymond Chi-Wing Wong is supported by PRP/026/21FX. The research of Min Xie is supported in part by Guangdong Basic and Applied Basic Research Foundation 2022A1515010120 and China NSFC 62202313.

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] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] R. Heckel, N. B. Shah, K. Ramchandran, and M. J. Wainwright, "Active ranking from pairwise comparisons and when parametric assumptions do not help," *The Annals of Statistics*, vol. 47, no. 6, pp. 3099–3126, 2019.

[19] 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.

[20] 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.

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

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

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

[24] L. d. F. Costa, "An introduction to multisets," *arXiv preprint arXiv:2110.12902*, 2021.

[25] 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.

[26] 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.

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

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

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

[30] Dataset earthquake, 2022. [Online]. Available: https://www.kaggle.com/datasets/thedevastator/uncovering-geophysical-insights-analyzing-usgs-e

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

[32] 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.

[33] M. Xie, R. C.-W. Wong, P. Peng, and V. J. Tsotras, "Being happy with the least: Achieving α-happiness with minimum number of tuples," in *Proceedings of the International Conference on Data Engineering*, 2020, pp. 1009–1020.

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

Table IV: Summary of Used Notations

Notations	Definition
$\mathcal{D}$ and p, q	The dataset and the tuples.
n	The number of tuples in the dataset.
d_{cat} / d_{num}	The number of categorical/numerical attributes.
$p_{cat}[i]$ / $p_{num}[j]$	The value of p in the i-th categorical attribute ($i \in [1, d_{cat}]$)/j-th numerical attribute ($j \in [1, d_{num}]$).
s_i	The number of possible values (i.e., the cardinality) in the i-th categorical attribute, where $i \in [1, d_{cat}]$).
A, B	The categorical values.
$f(\cdot)$	The linear utility function.
$u_{cat}[i]$ / $u_{num}[j]$	The importance of the i-th categorical attribute/j-th numerical attribute to the user.
$h(\cdot)$	A function that maps each categorical value to a real number, indicating the user preference on the value.
$g_i(\cdot)$	$g_i(p_{cat}[i]) = u_{cat}[i]h(p_{cat}[i])$ ($i \in [1, d_{cat}]$).
S_p^i	The set $\{p_{cat}[i], p_{cat}[i+1], ..., p_{cat}[d_{cat}]\}$.
h_z	A hyper-plane which passes through the origin with its unit normal in the same direction as vector z.
h_z^+ / h_z^-	The half-space above/below hyper-plane h_z.
L	A multiset.
$m_L(A)$	The multiplicity of categorical value A in L.
$\mathcal{R}$	The numerical utility range.
$\mathcal{G}$	The relation graph.
v	A node in the relation graph $\mathcal{G}$.
$\triangle x$, $\triangle y$	The d_{num}-length vectors.
$\mathcal{C}$	The candidate set.

APPENDIX A
TECHNIQUES SUPPLEMENT

A. Notation Table

We present a notation table, as shown in Table IV, to facilitate the understanding of the techniques.

B. Scale Invariant

In our problem setting, we normalize each numerical attribute from $(0, 1]$ since rescaling the numerical attributes does not change the ranking of tuples w.r.t. the utility function. 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 7. 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 $k, l \in [1, n]$.

Proof. 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 $p, q \in \mathcal{D}$ such that $f(p) > f(q)$. We have the conclusion as follows.

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

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. □

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

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

Lower Bounds. We utilize the linear programming algorithm (LP) to check if a lower bound $u_{num} \cdot \triangle x$ is untight. 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 [34]) needs $O(r^2 d_{num})$ time in practice. It is costly if l is large. Therefore, we present the sufficient

condition of 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 [22]. 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. In this part, we first show the definition of the untight upper bound. Then, we discuss the way to identify whether an 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 an LP as follows.

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

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 the untight upper bounds, 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 $\sum_{i=1}^{m} t_i = 1$ [22]. 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.

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

D. Relational Graph $\mathcal{G}$

Update with Derivation Rules. 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 could use the bounds in any two nodes to derive new bounds for the third one. Without loss of generality, let us focus on how nodes v_1 and v_2 derive new bounds for v_3. Suppose v_1 and v_2 have bounds as follows.

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

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

Based on the way of L_1 and L_2 deducing L_3, we can generate new bounds for 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 $L_1 \oplus 2L2 = \overline{L_3}$.

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

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

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

Acceleration Strategies. We implement two strategies to accelerate the use of the relational graph. (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.

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

In this section, we complement our strategies of pruning tuples from $\mathcal{C}$. First, we show the detailed LP formulation for Lemma 5. Then, we present a lemma that is used to prune tuples from $\mathcal{C}$ with the help of lower bounds.

Linear Programming for Lemma 5. We utilize the linear programming algorithm (LP) based on Lemma 5 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 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) $\forall r \in \mathcal{R}_i$, $\forall j \in [1, l]$ 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 m_i vertices in $\mathcal{R}_i$, we need $O(m_i)$ 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, $\mathcal{R}_i \not\subseteq h^+_i$. Since there are l polyhedrons $\mathcal{R}_i$, the total time complexity is $O(\sum_{i=1}^{l} m_i)$.

d_{cat}	d_{num}	n	c_{val}	The number of skyline tuples
2	2	100000	4	230
2	3	100000	4	1574
2	3	10000	4	672
2	3	50000	4	1260
2	3	500000	4	2547
2	3	1000000	4	3076
2	4	100000	4	6047
2	5	100000	4	16518
3	3	100000	4	3668
4	3	100000	4	7882
5	3	100000	4	16583
2	3	100000	8	3887
2	0	100000	10	100
2	3	100000	10	4923
2	3	100000	12	5934
2	3	100000	16	7899
2	3	100000	20	9740
2	3	100000	24	11547
2	3	100000	28	13270

Table V: The Number of Skyline Tuples

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 tuple 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 maintain 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

Skyline Tuples. As mentioned in Section VI, we processed each dataset to only contain skyline tuples following the existing setting [6], [7]. In Table V, we showed the number of skyline tuples for typical synthetic datasets after preprocessing.

Progress Analysis. We showed how our algorithms *SP-Tree* and *GE-Graph* progressed during the interaction process on two synthetic datasets, where $c_{val} = 10$ and the other parameters were set by default. We compared our algorithms against existing ones by varying the number of questions we can ask. The cumulative time and the candidate size after each question was asked were reported.

The first dataset was a categorical-only dataset. Figure **??** showed the results. Except for algorithm *UH-Random*, the other algorithms only took within 1 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 justified the superiority of *SP-Tree*.

The second dataset was a mixed attribute dataset in Figure 23. Except for *Adaptive*, none of the existing algorithm could finish the computation. They either cost more than 10^4 seconds or ran out of memory. In comparison, our algorithm *GE-Graph* reduced the candidate size effectively within the shortest time.

B. Results on Real Datasets

We compared our algorithms *GE-Graph* and *SP-Tree* with existing algorithms on four real datasets: *Car* [6], *Diamond* [29], *Earthquake* [30], and *Pollution* [31].

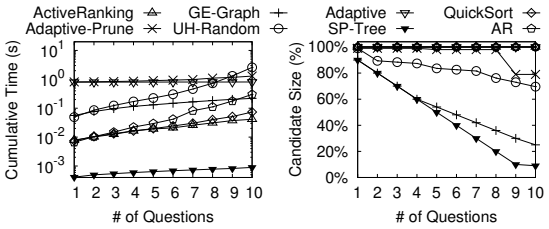
The cumulative times on the real datasets were shown in Figure 24 (the candidate sizes have been reported in Figure 18). According to the results, our algorithms were slower than some existing algorithms in the first few questions, since our algorithms had a tuple pruning step. They needed to process many tuples in the beginning. However, as the process continued, they needed less and less time for each round since fewer and fewer tuples were left to process, e.g., the cumulative time of *GE-Graph* was faster than algorithm *ActiveRanking* after asking 15 questions on the *Diamond* dataset.

For better visualization, we summarized the number of questions eventually asked for all algorithms on all real datasets in Figure 25. Our algorithms asked the fewest questions. On datasets *Car*, *Diamond*, *Earthquake*, and *Pollution*, we did not show the results of some existing algorithms since their execution times were more than 10^4 seconds.

APPENDIX D
PROOFS

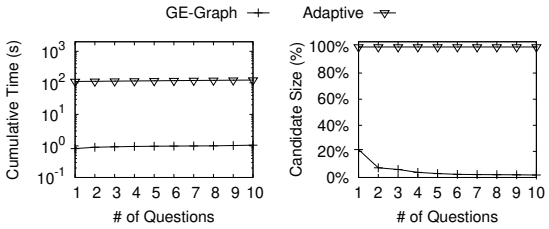
Proof of Theorem 1. Consider a dataset with n tuples as follows. Use $\mathcal{H}_i$ to denote the set which contains all the categorical values in the i-th categorical attribute ($|\mathcal{H}_i| = s_i$), where $i \in [1, d_{cat}]$. (1) For the second categorical attribute, we divide the categorical 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 could 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 combination that consist of the second categorical attribute to the d_{cat}-th categorical attribute $< A_i, X_j >$, where $A_i \in \mathfrak{S}_1$ and $X_j \in \mathcal{X}$, it corresponds to $\frac{s_2}{2}(\prod_{i=3}^{d_{cat}} s_i - 1)$ combinations $< B_k, X_l >$, where $B_k \in \mathfrak{S}_2$ and $X_l \in \mathcal{X} \setminus \{X_j\}$. 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 in $< A_i, X_j >$ and q contains the values in $< B_k, X_l >$. Each tuple in the dataset contains a different categorical value in the first attribute. Thus, the dataset contains



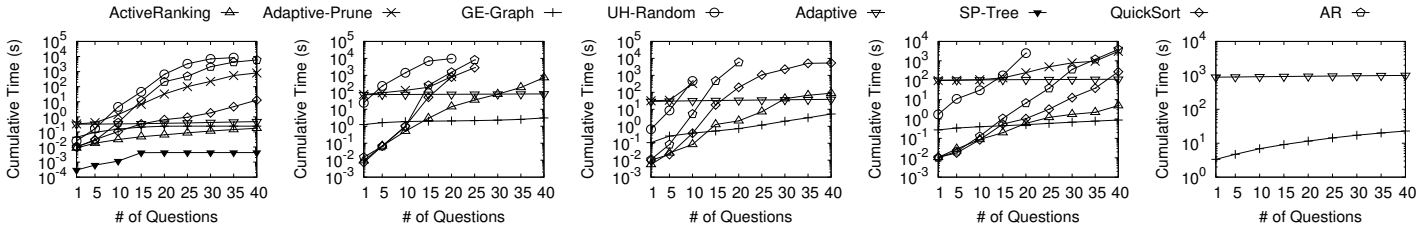
(a) (b)

Figure 22: Synthetic (categorical only & Anti) ($c_{val} = 10$)



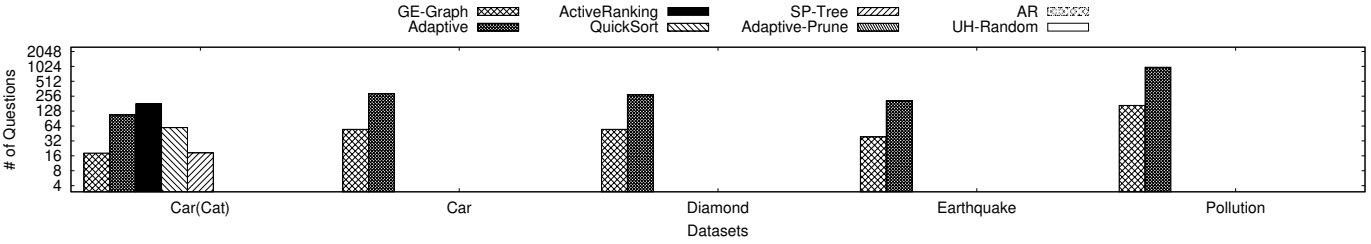
(a) (b)

Figure 23: Synthetic (Mixed & Anti) ($c_{val} = 10$)



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

Figure 24: Real Datasets (Time)



(a) (b)

Figure 25: Real Datasets (Summary)

$\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ 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 1. Since a user prefers p to q, we have $f(p) > f(q)$, i.e., we have an inequality as follows.

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

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

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

□

Proof of Lemma 2. Since p and q are the same in the first $i-1$ categorical attributes, we have the following equality.

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

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

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

Thus, by summarizing equality (1) and inequality (2), we have $\sum_{j=1}^{d_{cat}} g_j(p_{cat}[j]) > \sum_{j=1}^{d_{cat}} g_j(q_{cat}[j])$. 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 3. If a user prefers p to q, then $f(p) > f(q)$. Since tuples 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 4. 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.

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

□

Proof of Lemma 5. For any $q \in \mathcal{C}_p$, it has the same values as p in all categorical attributes. The difference between $f(q)$ and $f(p)$ only depends on the numerical attributes. Consider hyper-plane h_{q-p} 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 6. 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. □