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

When facing a dataset with millions of tuples, it is not desirable for a user to read the whole dataset finding his/her favorite tuple. Existing studies try to find the user's preference by interacting with the user and then return tuples based on the learnt preference. 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's preference is learned implicitly and the best tuple w.r.t. the learnt preference is returned. However, they only consider the dataset with numerical attributes (e.g., price). In real life, there are many tuples also described by categorical attributes (e.g., color), where there is no trivial order on the values of those attributes. Thus, we study how to find the user's favorite tuple from the dataset with mixed attributes (including both numerical and categorical attributes) by interacting with the user.

We study our problem progressively. First, for the dataset, in which the tuples only differ in categorical attributes, we present algorithm *Tree* that asks an asymptotically optimal number of questions. Then, for the dataset, in which the tuples can differ in both numerical and categorical attributes, we propose two algorithms *Combination* and *Separation* with provable performance guarantees. Experiments were conducted on synthetic and real datasets, showing that our algorithms outperform existing algorithms both on the execution time and the number of questions asked. Under typical settings, our algorithms ask up to 10 times fewer questions and take several orders of magnitude less time than existing algorithms.

PVLDB Reference Format:
Weicheng Wang*, Raymond Chi-Wing Wong*, Min Xie+. Interactive Search with Mixed Attributes. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

When facing a dataset with millions of tuples, it is not desirable for a user to read all tuples one by one finding the tuple s/he is interested in. Many operators have been proposed to assist users in finding the target tuple. Such operators, regarded as *multi-criteria decision-making tool*, can be applied in various domains, including purchasing a used car, buying a house and searching a partner for dating. For example, in a used car dataset, each car could be described by some attributes, e.g., brand, horse power and price. Alice wants to buy a cheap used car with a famous brand. Based on her preference, each tuple is associated with a *utility* (i.e., score). It indicates to what extent Alice prefers the tuple, where a larger utility means that the tuple is more favored by her.

There are two representative operators for this scenario: *the top-k query* [23] and *the skyline query* [7]. The first operator is the top-k query [23]. It returns the k tuples with the highest utilities, namely top-k tuples, w.r.t. the preference given by a user. However, users usually have difficulties in specifying their preference explicitly [21, 29]. If the user preference is not known in advance, the top-k query cannot be applied. The second operator is the skyline query [7] which, instead of asking for an explicit user preference, considers all possible user preferences and returns tuples that have the highest utilities (i.e., top-1 tuple) w.r.t. at least one preference. Unfortunately, the output size of the skyline query is uncontrollable and it often overwhelms users with excessive results.

Recently, [29] proposed to find the tuple with the (close to) highest utility w.r.t. the user's preference, i.e., the user's (close to) favorite tuple by interacting with the user. Specifically, it asks a user several questions. Based on the feedback, it learns the user's preference implicitly and returns the tuple with the highest utility w.r.t. the learnt preference. It overcomes the deficiencies of the top-k query (that requires the user to provide an exact preference) and the skyline query (whose output size is uncontrollable). However, it only works for the dataset with numerical attributes (e.g., horse power and price), where there are trivial orders on these attribute values. In real life, there are many tuples also described by categorical attributes (e.g., car brand), where the order of values in those attributes are uncertain. For example, it is easy to see that a car with price \$8000 is cheaper than a car with price \$10000. But, we cannot directly tell that a car with brand Porsche is how much better or worse than a car with brand Benz.

Motivated by this limitation, we propose a problem called *Interactive Search with Mixed Attributes (ISM)* which, by interacting with the user, finds the user's favorite tuple (i.e., the tuple with the highest utility w.r.t. the user's preference) from the dataset described by mixed attributes (including both numerical and categorical attributes) with as little user effort as possible. In detail, we interact with a user for rounds. In each round, following [24, 27, 29], we ask the user a question *that consists of two tuples and asks the user to indicate which one s/he prefers (i.e., pairwise comparison)*. Based on the feedback, the user's preference is implicitly learned and the user's favorite tuple is returned. We stick to the setting in [24, 27, 29] to measure the user effort by *the number of questions asked to a user (i.e., the number of rounds in the interaction)*. Thus, we want to answer the question: *how many questions do we need to ask to determine the favorite tuple for a user when tuples in the dataset are described by mixed attributes?*

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

We study problem ISM progressively. First, we consider a *special case* that all tuples in the dataset only differ in categorical attributes. Then, we look into the *general case* that all tuples in the dataset can differ in both numerical and categorical attributes. There are two characteristics of problem ISM. One is that it involves a number of questions asked to a user and the other is that it considers both numerical and categorical attributes.

The questions asked to a user are in the form of pairwise comparison [13, 24, 27]. It follows a basic assumption made by [21] that *users are able to tell which tuple they prefer the most from a set of tuples presented to them.* This assumption is also studied by our experiments in Section 6.4. In cognitive psychology, the Thurstone's Law of Comparative Judgement indicates that pairwise comparison is a more effective way to learn the user preference than the other methods [4, 24]. This kind of interaction naturally appears in our daily life. For example, a car seller might show Alice a black car and a white car and ask her which one she prefers. A realtor may present two houses near the sea and the park, respectively and ask Alice: which one would you desire to live in? There might be a case that the tuples are described by a large number of attributes. [2, 11, 18] emphasize that users usually focus their attention on a few attributes that are most crucial when selecting favorite tuples.

In the literature of the marketing research [17, 25], it is expected that the number of questions asked should be as few as possible. Problem ISM follows the skyline query to find the top-1 tuple instead of the top-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. [27] conducted experiments and verified that the latter case needs to ask 4-10 times more questions than the former case. Its user study also showed that instead of obtaining more recommended tuples, users are willing to answer fewer questions. Thus, we reduce the user effort by just returning one tuple, which is sufficient in many applications, to strike for a balance between the user effort and the result size. Consider a scenario that Alice plans to rent an apartment for several months. Since she could only live in one apartment, it is enough returning the best candidate. Besides, this is a short-term need. It is not necessary for her to spend a lot of effort cherry-picking. She could be frustrated due to the long selection process even if finally, several candidate apartments are returned. Moreover, due to the high-frequency of trading, it is possible that the apartment chosen by her has already been rented. Note that our proposed algorithms can be easily extended to returning the top-k tuples. The extension is shown in Section 5.5.

Considering both numerical and categorical attributes is critical in real applications. Suppose Alice would like to purchase a used car (this scenario is also applied in our user study in Section 6.4.2). There could be numerous candidate used cars in the market and each one is described by some numerical (e.g., price) and categorical (e.g., brand) attributes. Alice might have a trade-off between price and brand, i.e., she may be willing to pay more money for a famous brand than an obscure one. This trade-off involves numerical and categorical attributes. In order to recommend a satisfactory car to Alice, it is important to consider both kinds of attributes.

To the best of our knowledge, we are the first to study problem ISM. There are some closely related problems [13, 21, 24, 29] but they are different from ours. [13] finds the ranking of all tuples (the favorite tuple ranks the first) by interacting with the user. Since the ranking of all tuples has to be determined, there are a lot of questions asked during the interaction. [24] learns the user preference by interacting with the user. Similar to [13], since [24] has to find the user's precise preference, there could be a large amount of redundant questions asked during the interaction, which may not be relevant to our desired answer (i.e., the favorite tuple). [21, 29] propose to find the user's (close to) favorite tuple as the answer by interacting with the user. However, they only focus on the dataset with numerical attributes, which could not be applied in many scenarios. Besides, [21] involves "fake" tuples (i.e., tuples not in the dataset) in the questions during the interaction, which suffers from the *truthfulness problem* as described in Section 2. In contrast, problem ISM utilizes "real" tuples (i.e, tuples in the dataset).

Note that none of the existing algorithms could be adapted to solve problem ISM satisfactorily. For [21], its algorithm that constructs "fake" tuples can only be applied to numerical attributes. [13, 24, 29] ask many questions in a long execution time, which is quite troublesome. In our experiments, when there are 2 categorical attributes (each contains 5 values) and 3 numerical attributes, the adapted algorithms ask a user more than 43 questions in hundreds of seconds, which are too many and slow. In comparison, our algorithms only ask around 30 questions within 1 second.

Contributions. Our contributions are described 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 (problem ISM).
- We show lower bounds on the number of questions needed to determine the favorite tuple in the dataset with mixed attributes.
- We propose algorithm *Tree* for the special case of ISM, which asks an asymptotically optimal number of questions.
- We propose two algorithms *Combination* and *Separation* for the general case of ISM, which have provable guarantees on the number of questions asked and perform well empirically.
- We conducted experiments to demonstrate the superiority of our algorithms. The results show that under typical settings, our algorithms ask up to 10 times fewer questions and take several orders of magnitude less time than existing algorithms.

In the following, we discuss the related work in Section 2. The formal problem definition and the interactive framework are shown in Section 3. In Section 4, we propose algorithm *Tree* for the special case of ISM. In Section 5, we present algorithms *Combination* and *Separation* for the general case of ISM. Experiments are shown in Section 6, and Section 7 concludes our paper.

2 RELATED WORK

Various queries were proposed to assist the multi-criteria decision making, which consider categorical attributes or user interaction.

[26, 28] considered the skyline query on the dataset with categorical attributes. The skyline query returns all the tuples in the dataset that are not *dominated* by other tuples. A tuple p dominates another tuple q if the value of p in each attribute is not better than that of q, and strictly worse in at least one attribute. For example, suppose

there are two cars described by two attributes: p with 5000USD and 300 horse power and q with 3000USD and 350 horse power. Since q is cheaper and has a larger horse power than p, q dominates p. [28] considered that different users may have different preference on categorical attributes. It found the skyline tuples when the user preference on categorical attributes dynamically changes. For example, Alice may prefer color green to color red and Tom might favors more red than green. The skyline tuples w.r.t. the preference of Alice and Tom could be totally different. [26] maintained the skyline tuples when the tuples in the dataset dynamically change. In summary, [28] assumes that the dataset remains unchanged but the user preference varies, while [26] considers the dataset changes but the preference remains unchanged. However, both of them rely on the assumption that the user preference is known in advance.

To overcome the deficiency, [4, 5] involved user interaction. They proposed the interactive skyline query, which learns the user preference on the categorical attributes (e.g., the user preference on the color attribute) by interacting with the user and returns the skyline tuples based on the learnt preference. However, [4, 5] cannot avoid the drawback of the skyline query that the output size is uncontrollable. It is possible that the number of skyline tuples is arbitrarily large [22] and thus, the output overwhelms users.

There are interactive queries [21, 29, 31] whose output size is controllable. [21] proposed the interactive regret minimizing query. It defined a criterion called the regret ratio (which evaluates the returned tuples and represents how regretful a user is when s/he sees the returned tuples instead of the whole dataset) and targeted to return a small size of output whose regret ratio is as small as possible. However, it displayed *fake tuples* to interact with a user. The fake tuples are artificially constructed (not selected from the dataset). This might produce unrealistic tuples (e.g., a car with 10USD and 60000 horse power) and the user may be disappointed if the displayed tuples with which s/he is satisfied do not exist. Besides, it is impractical for users to truly evaluate fakes tuples [29], resulting in the difficulty of applying the algorithm of [21] in real-world applications. Based on these defects, [29] proposed algorithms *UH-Simplex* and *UH-Random* that utilize *real tuples* (selected from the dataset) for the interactive regret minimization query. However, they require users to answer many questions and wait a long time for algorithm processing. Recently, [31] improved the algorithms of [29]. Instead of asking for the favorite one among the displayed tuples, it asks a user to give an order of the displayed tuples. Unfortunately, although this improvement reduces the number of questions asked, it does not reduce the user effort essentially, since giving an order among tuples is equivalent to picking the favorite tuple several times. Moreover, [21, 29, 31] only considered the dataset described by numerical attributes. In contrast, our problem *ISM* focuses on both numerical and categorical attributes.

There are alternative approaches [14, 24] that focus on learning the user preference by interacting with the user. [24] proposed algorithm *Adaptive* that approximates the user preference with the help of user interaction. Nevertheless, instead of recommending tuples, it focuses on learning the user preference and thus, it might ask the user much more questions [27]. For example, if Alice prefers car p_1 to both p_2 and p_3, her preference between p_2 and p_3 is less interesting in our problem, but this additional information might be useful in [24]. [14] learned the user preference on tuples with both numerical and categorical attributes. However, it only paid attention to extract the user preference from the given information and neglected how to select tuples as questions asked to a user. Besides, the learnt information is limited on the order of categorical values instead of their differences. For example, consider two categorical values *red* and *green* in the color attribute. [14] can only learn that *red* is better than *green*, while we may obtain that a user is willing to pay more 100USD to buy a *red* product than a *green* one.

In machine learning area, our problem is related to the problem of *learning to rank* [10, 13, 16, 19], which learns the ranking of tuples by pairwise comparison. However, most of the existing methods [10, 16, 19] failed to utilize the inter-relation between tuples. For example, tuples with attribute "horse power" have inter-relation since a car with 350 horse power is generally considered to be better than a car with 300 horse power. Thus, they require to ask much more questions to a user [29]. [13] considered the inter-relation between tuples on the learning to rank problem. However, it was proposed based on the numerical attributes and did not consider the inter-relation on the categorical attributes. Besides, it focused on deriving the order for all pairs of tuples, which requires asking much more questions due to the similar reason stated for [24].

Our work focuses on returning the user's favorite tuple from the dataset described by both numerical and categorical attributes by interacting with the user via real tuples. Compared with existing methods, we have the following advantages. (1) We do not require pre-knowledge of the user preference (while [26, 28] need to know the user preference in advance). (2) We return only one tuple that is the user's favorite (but the output size of the skyline query is uncontrollable). (3) We use real tuples during the interaction (unlike [21] that utilizes fake tuples). (4) We consider both numerical attributes and categorical attributes (but [13, 21, 29, 31] only handle numerical attributes). (5) We only involve a few easy questions. Firstly, existing studies like [10, 13, 16, 19, 24] ask a lot of questions since they either require to learn a full ranking of tuples or aim to learn the exact user preference. Secondly, [10, 16, 19] fail to utilize the inter-relation between tuples and thus, involve some unnecessary interaction. Thirdly, there might exist difficulties for users to answer questions in [31], since it requires users to give orders for the displayed tuples. In contrast, we only need a user to pick his/her favorite tuple between two candidates in each question. The questions are much easier for the user to handle.

3 PROBLEM DEFINITION

3.1 Terminologies

The input to our problem is a tuple set D of size n (i.e., $|D| = n$). Each tuple p is described by d mixed attributes, including d_{cat} categorical attributes and d_{num} numerical attributes ($d = d_{cat} + d_{num}$). We denote the i-th categorical attribute value of p and the j-th numerical attribute value of p by $p_{cat}[i]$ ($i \in [1, d_{cat}]$) and $p_{num}[j]$ ($j \in [1, d_{num}]$), respectively. For the sake of convenience, let $p_{num} = (p_{num}[1], p_{num}[2], ..., p_{num}[d_{num}])$. Following [1, 24, 29], we model the user preference by a *utility function* $f(p) = \sum_{i=1}^{d_{cat}} u_{cat}[i]h(p_{cat}[i]) + \sum_{j=1}^{d_{num}} u_{num}[j]p_{num}[j]$ where:

- u_{cat} and u_{num} are a d_{cat}-length and a d_{num}-length non-negative vectors, namely the *categorical utility vector* and the *numerical* *utility vector*, respectively. $u_{cat}[i]$ (resp. $u_{num}[j]$) measures the importance of the i-th categorical attribute (resp. the j-th numerical attribute) to the user. Without loss of generality, following [29, 30], we assume that $\sum_{j=1}^{d_{num}} u_{num}[j] = 1$ and call the domain of u_{num} the *numerical utility space*.

- Function $h : p_{cat}[i] \rightarrow \mathbb{R}_+$, called *transferring function*, maps each categorical value to a real number which indicates to what extent a user favors the categorical value. A larger real number means that the categorical value is more preferred by the user. For the ease of representation, we use function $g : p_{cat}[i] \rightarrow \mathbb{R}_+$ to denote $g(p_{cat}[i]) = u_{cat}[i]h(p_{cat}[i])$, where $i \in [1, d_{cat}]$.

The function value $f(p)$, called the *utility* of p w.r.t. f, represents how much a user favors the tuple. The tuple that has the highest utility is the user's favorite tuple.

Lemma 3.1 shows that rescaling the numerical attributes (scaling by a positive real number to maintain monotonicity) does not change the ranking of tuples w.r.t. the user preference. Formally, consider two datasets $D = \{p_1, p_2, ..., p_n\}$ and $D' = \{p'_1, p'_2, ..., p'_n\}$. Assume that D' is a numerical rescaling of D, i.e., 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 $\forall j \in [1, d_{num}]$. For any utility function f for D, there exists a corresponding utility function f' for D' such that $\forall p_k, p_l \in D$, if $f(p_k) > f(p_l)$, $f'(p'_k) > f'(p'_l)$, where $k, l \in [1, n]$.

Lemma 3.1. *Rescaling the numerical attributes of tuples in the dataset does not change the ranking of tuples.*

Based on Lemma 3.1, we assume that each numerical attribute is normalized to $(0, 1]$ and a larger value is more favored by users. If a smaller value is preferable (e.g., price), the attribute could be modified by subtracting each value from 1 to satisfy the assumption.

Example 3.2. Let $f(p) = g(p_{cat}[1]) + g(p_{cat}[2]) + 0.5p_{num}[1] + 0.5p_{num}[2]$ (i.e., $u_{num} = (0.5, 0.5)$) and function g is shown in Table 1. Consider the tuples shown in Table 1. The utility of p_2 w.r.t. f is $f(p_2) = 0.3 + 0.4 + 0.5 \times 0.6 + 0.5 \times 0.9 = 1.45$. The utilities of the other tuples can be computed similarly. Tuple p_2 is the user's favorite tuple, since it has the highest utility.

3.2 Interactive Framework

Our interactive framework follows [27, 29]. We interact with a user for rounds until we can determine which tuple has the highest utility w.r.t. the user preference. In each round,

- (**Tuple selection**) We select two tuples presented to the user and ask the user which one s/he prefers.
- Then, the user picks his/her favorite tuple.
- (**Information maintenance**) Based on the feedback, the maintained information for learning the user preference is updated.
- (**Stopping condition**) Finally, we check the stopping condition. If it is satisfied, we terminate the interaction and return the result. Otherwise, we start a new interactive round.

Formally, we are interested in the following problem.

Problem 1. *(*Interactive Search with Mixed Attributes (ISM)*) Given a tuple set D with mixed attributes, we are to ask a user as few questions as possible to determine the user's favorite tuple in D.*

3.3 Lower Bound

We present lower bounds on the number of questions asked progressively. Theorem 3.3 considers the special case of ISM, where all tuples in the dataset only differ in categorical attributes. Theorem 3.4 considers the general case of ISM, where tuples in the dataset can differ in both numerical and categorical attributes. Note that s_i is the cardinality of the i-th categorical attribute, where $i \in [1, d_{cat}]$. Due to the lack of space, the proofs of some theorems/lemmas in this paper can be found in the technical report [3].

Theorem 3.3. *If the tuples only differ in categorical attributes, there exists a dataset such that any algorithm needs to ask $\Omega(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions to determine the user's favorite tuple.*

Proof Sketch. We show a dataset with $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ tuples and prove that the user feedback of any question can only help to identify one tuple that cannot be the user's favorite tuple. □

Theorem 3.4. *If the tuples can differ in both numerical and categorical attributes, 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.*

Proof Sketch. We consider the categorical and numerical attributes separately. We show that to find user's favorite tuple, any algorithm needs to ask $\Omega(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions to learn the user preference on categorical attributes and $\Omega(\log(n - s_1 - \prod_{i=2}^{d_{cat}} s_i^2))$ questions to learn the user preference on numerical attributes. □

4 SPECIAL ISM: ALGORITHM TREE

In this section, we consider the special case of ISM that all tuples only differ in categorical attributes, i.e., $\forall p, q \in D$ and $\forall j \in [1, d_{num}]$, $p_{num}[j] = q_{num}[j]$. We present the *Tree* algorithm that asks an asymptotically optimal number of questions.

Intuitively, we maintain tuples in a tree data structure. During the interaction, tuples are continually selected from the tree as questions asked to a user. Based on the feedback, the tuples which cannot be the favorite tuple are pruned from the tree. Finally, we stop when there is only one tuple left in the tree and return the tuple as the answer. In the following, we show *Tree* by elaborating the three components of the interactive framework in Section 4.1, Section 4.2 and Section 4.3, and summarizing them in Section 4.4.

4.1 Information Maintenance

Since tuples only differ in categorical attributes, the numerical attributes cannot contribute to determining the favorite tuple in the dataset. Thus, we focus on the categorical attributes only.

4.1.1 Tuple Maintenance. We maintain a tree data structure $\mathcal{T}$ to record tuples, called *categorical value tree*, or *C-Tree* in short, which 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. (3) Each tuple in the dataset is recorded by a leaf (i.e., the node in the $(d_{cat} + 1)$-th level). Note that for each leaf, there is only one path

$g(\cdot)$
$g(A) = 0.2$
$g(B) = 0.3$
$g(C) = 0.4$
$g(D) = 0.5$

p	$p_{cat}[1]$	$p_{cat}[2]$	$p_{num}[1]$	$p_{num}[2]$	$f(p)$
p_1	A	C	0.4	1	1.30
p_2	B	C	0.6	0.9	1.45
p_3	A	D	0.9	0.5	1.40
p_4	B	D	1	0.2	1.40

Table 1: Function g, dataset and utilities

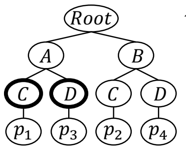


Figure 1: C-Tree

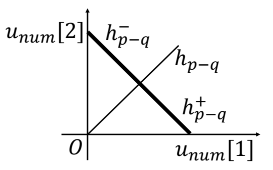
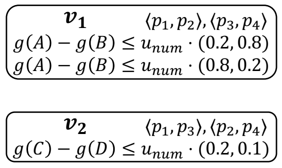


Figure 2: Space $\mathbb{R}^{d_{num}}$



v_1 $\langle p_1, p_2\rangle, \langle p_3, p_4\rangle$
$g(A) - g(B) \leq u_{num} \cdot (0.2, 0.8)$
$g(A) - g(B) \leq u_{num} \cdot (0.8, 0.2)$
v_2 $\langle p_1, p_3\rangle, \langle p_2, p_4\rangle$
$g(C) - g(D) \leq u_{num} \cdot (0.2, 0.1)$

Figure 3: Bounds in nodes

from the root to it. The categorical values of the tuple recorded by the leaf are stored by the nodes in the path in order.

We build $\mathcal{T}$ as follows. $\mathcal{T}$ is initialized with a root. Then, we insert tuples in D into it one by one. Specifically, for each tuple p, $\mathcal{T}$ is traversed once from top to bottom. Suppose we are at a node N in the $(i-1)$-th level, where $i \in [1, d_{cat}]$. We check whether the categorical value stored by one of the children of N equals to $p_{cat}[i]$. If yes, we move to the child. Otherwise, we create a new child for N to store $p_{cat}[i]$ and move to this new child. This process continues until we reach a node N_d in the d_{cat}-th level. Finally, we create a child (i.e., a leaf) for N_d to record p.

Example 4.1. Consider Table 1. We assume that all tuples only differ in categorical attributes. Initially, the C-Tree has a root in the 0-th level. Consider tuple p_1. Since the root does not contain any child, we build (1) 2 new nodes in the first and second levels, respectively, that include categorical values A and C in order and (2) a leaf that contains p_1. The C-Tree is shown in Figure 1.

4.1.2 User Preference Maintenance. Our strategy is to learn the user preference on the categorical values from bottom to top of $\mathcal{T}$. Concretely, we start considering the nodes in the d_{cat}-th level and gradually take the nodes in the i-th level into account if each node N in the i-th level can only reach one leaf, where $i \in [1, d_{cat}]$. We call the path from N to the only leaf the *unique path* of N.

Suppose that the nodes in the i-th level of $\mathcal{T}$ are being processed. Consider two nodes N_1 and N_2 that have the same parent. Use P_1 and P_2 to represent the unique path of N_1 and N_2, respectively. Denote by S_1 (resp. S_2) the set containing all categorical values in P_1 (resp. in P_2), and let p (resp. q) to be the tuple recorded by the leaf in P_1 (resp. in P_2). Since N_1 and N_2 have the same parent, $\forall j \in [1, i-1], p_{cat}[j] = q_{cat}[j]$. If a user prefers p to q, we can learn that $\sum_{c \in S_1} g(c) > \sum_{c \in S_2} g(c)$, denoted by $S_1 \succ S_2$ for simplicity.

For each node N in the i-th level, we build a set S that contains all the categorical values in the unique path P of N if there does not exist a set including all the categorical values in P. Assume that there are m_i sets $\mathcal{S}_i = \{S_1, S_2, ..., S_{m_i}\}$. Given $S_1 \succ S_2$, we can derive the user preference on other sets in $\mathcal{S}_i$ in two steps. Firstly, we build a queue Q and insert into Q all the pairs of sets $S_l, S_k \in \mathcal{S}_i$ such that (1) $S_l \setminus S_k = S_1 \setminus S_2$ and (2) $S_k \setminus S_l = S_2 \setminus S_1$. Since $S_1 \succ S_2$, we have $S_l \succ S_k$. Secondly, we continually pop out a pair of sets from Q until Q is empty. Suppose that $S_l \succ S_k$ is popped out. We derive new preference of other sets in $\mathcal{S}_i$ and insert them into Q as follows. (1) If $\exists S \in \mathcal{S}_i$ such that $S \succ S_l$, then $S \succ S_k$. (2) If $\exists S \in \mathcal{S}_i$ such that $S_k \succ S$, then $S_l \succ S$. (3) If $\exists S_j, S_t \in \mathcal{S}_i$ such that $S_j \succ S_l$ and $S_k \succ S_t$, then $S_j \succ S_t$.

Example 4.2. In Figure 1, there are two sets $S_1 = \{C\}$ and $S_2 = \{D\}$ in the second level of the C-tree. If a user prefers p_1 to p_3, we can learn that $g(C) > g(D)$, i.e., $S_1 \succ S_2$. Since there are only two sets, no user preference on other sets is derived.

4.1.3 Tuple Pruning. Based on the learnt user preference on sets in $\mathcal{S}_i$, the unique paths P of some nodes in the i-th level can be removed from $\mathcal{T}$. In this way, the number of children of some nodes in the $(i-1)$-th level are reduced and the tuples recorded by the leaves in P are pruned. Specifically, consider two nodes N_1 and N_2 in the i-th level that have the same parent. Denote by P_1 and P_2 the unique paths of N_1 and N_2, respectively.

Lemma 4.3. *P_2 can be removed from $\mathcal{T}$ if $S_l \succ S_k$, where sets S_l and S_k contain all the categorical values in P_1 and P_2, respectively.*

Example 4.4. The second level of the C-Tree in Figure 1 has two sets $S_1 = \{C\}$ and $S_2 = \{D\}$. If $S_1 \succ S_2$, the unique paths of the nodes in the second level which contain categorical value D are removed. All nodes in the first level have only one child, and tuples p_3 and p_4 are pruned.

4.2 Tuple Selection

Recall that we learn the user preference on the categorical values from bottom to top of $\mathcal{T}$. Suppose that we at the i-th level of $\mathcal{T}$ and there are m_i sets $\mathcal{S}_i = \{S_1, S_2, ..., S_{m_i}\}$, where $i \in [1, d_{cat}]$. Start from S_2. For each set S_j ($j \in [2, m_i]$), we consider sets S_k ($k \in [1, j-1]$) one by one. For each pair S_j and S_k, we check whether there exist two nodes N_1 and N_2 in the i-th level of $\mathcal{T}$ such that (1) N_1 and N_2 have the same parent and (2) S_j and S_k store all the categorical values in the unique path of N_1 and N_2, respectively. If there exist such two nodes, we present the user with the two tuples recorded by the leaves in the unique paths of N_1 and N_2, respectively, and ask the user to pick the tuple s/he prefers.

Example 4.5. The second level of the C-Tree in Figure 1 has two sets $S_1 = \{C\}$ and $S_2 = \{D\}$. There exists two nodes (represented by bold circles) such that they have the same parent, and S_1 and S_2 contain the categorical values in the unique paths of these two nodes, respectively. We present tuples p_1 and p_3 that are recorded by the leaves in the unique paths of these two nodes.

4.3 Stopping Condition

During the interaction, tuples are continually pruned from $\mathcal{T}$. If $\mathcal{T}$ contains only one tuple p, we conclude that p is the user's favorite tuple. Thus, we stop the interaction and return p to the user.

4.4 Summary and Analysis

The pseudocode of algorithm *Tree* is shown in Algorithm 1. Its theoretical analysis is presented in Theorem 4.6.

Theorem 4.6. *Algorithm* Tree *solves the special case of ISM by asking the user $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions.*

Proof Sketch. We prove that *Tree* needs to ask $O(\prod_{k=i}^{d_{cat}} s_k^2)$ questions in the i-th level of the C-Tree, where $i \in [2, d_{cat}]$, and $O(s_1)$ questions in the first level of the C-Tree. □

Algorithm 1: The *Tree* Algorithm

```
Input: A tuple set $D$
Output: The user's favorite tuple in $D$
1 Build the C-Tree $\mathcal{T}$ for all tuples in $D$
2 for $i \leftarrow d_{cat}$ to 1 do
3     Build set $\mathcal{S}_i = \{S_1, S_2, ..., S_{m_i}\}$
4     for $j \leftarrow 2$ to $m_i$ do
5         for $k \leftarrow 1$ to $j-1$ do
6             if there exist two nodes satisfying the two
              conditions associated with $S_j$ and $S_k$ then
7                 Ask the user a question
8                 Learn the user preference and update $\mathcal{T}$
9 return the only tuple left in the C-tree $\mathcal{T}$
```

Corollary 4.7. *Algorithm* Tree *is asymptotically optimal in terms of the number of questions asked for the special case of ISM.*

5 GENERAL ISM: ALGORITHMS COMBINATION AND SEPARATION

In this section, we describe our algorithms for the general case of ISM by showing how we address the three components of the interactive framework. Then, we show the theoretical analysis and their extension of returning top-k tuples.

5.1 Information Maintenance

We define three data structures for learning the user preference: (1) a *candidate numerical utility range* $\mathcal{R}$ which contains the user's numerical utility vector; (2) a *relational graph* $\mathcal{G}$ which maintains the learnt user preference on categorical values; and (3) a *candidate set* $C \subseteq D$ which includes the user's favorite tuple.

5.1.1 Maintenance on $\mathcal{R}$. Given a d_{num}-length vector p in a d_{num}-dimensional geometric space $\mathbb{R}^{d_{num}}$, we can build a *hyperplane* [9], denoted by h_p, which passes through the origin with its unit normal in the same direction as p. Hyperplane h_p divides $\mathbb{R}^{d_{num}}$ into two halves, called *halfspace* [9]. The halfspace above h_p (resp. below h_p), denoted by h_p^+ (resp. h_p^-), contains all the points $u_{num} \in \mathbb{R}^{d_{num}}$ such that $u_{num} \cdot p > 0$ (resp. $u_{num} \cdot p < 0$). In geometry, a *polyhedron* $\mathcal{P}$ is the intersection of a set of halfspaces and a point in $\mathcal{P}$ is said to be a *vertex* of $\mathcal{P}$ if it is a corner point of $\mathcal{P}$ [9].

The *candidate numerical utility range* $\mathcal{R}$ is a polyhedron in the numerical utility space, which contains the user's numerical utility vector. Initially, $\mathcal{R}$ is the entire numerical utility space, i.e., $\mathcal{R} = \{u_{num} \in \mathbb{R}_+^{d_{num}} \mid \sum_{i=1}^{d_{num}} u_{num}[i] = 1\}$. As shown in Figure 2, when $d_{num} = 2$, $\mathcal{R}$ is initialized to be a line segment. During the interaction, based on the user feedback, we can build hyperplanes and update $\mathcal{R}$ in two ways. First, consider two tuples p and q presented to a user such that $\forall i \in [1, d_{cat}]$, $p_{cat}[i] = q_{cat}[i]$. We can build a hyperplane $h_{p_{num}-q_{num}}$, denoted by h_{p-q} for simplicity, based on vector $p_{num} - q_{num}$ and update $\mathcal{R}$ following Lemma 5.1.

Lemma 5.1. *If a user prefers p to q, $\mathcal{R}$ is updated to be $\mathcal{R} \cap h_{p-q}^+$.*

Example 5.2. Figure 2 shows hyperplane h_{p-q} based on vector $p_{num} - q_{num}$, where $p_{num} = (\frac{1}{2}, 0)$ and $q_{num} = (0, \frac{1}{2})$. If a user prefers p to q, $\mathcal{R}$ is updated to be $\mathcal{R} \cap h_{p-q}^+$ (right line segment).

The second way to update $\mathcal{R}$ is based on the relational graph. We postpone the discussion to the next section.

5.1.2 Maintenance on $\mathcal{G}$. The *relational graph* $\mathcal{G} = (V, E)$ is a hypergraph storing the learnt user preference on categorical values. V is the set of nodes and E is the set of hyperedges each of which connects 3 nodes in V. In the following, we first define $\mathcal{G}$ in terms of nodes and hyperedges, and then show the way to update $\mathcal{G}$.

Node. Given a pair of tuples $p, q \in D$, we build a node v that contains a *categorical pair* (L_1, L_2). L_1 (resp. L_2) is a *multiset* that includes all the categorical values in p but not in q (resp. in q but not in p), and allows for multiple instances of each categorical value [12]. The number of instances of each categorical value c is called the *multiplicity* of c and denoted by $\mathfrak{m}_L(c)$. Node v also stores pair $\langle p, q \rangle$. If there are multiple pairs of tuples that share the same categorical pair, they are merged/stored in one node.

Example 5.3. Consider tuples p_1 and p_4 in Table 1 that have categorical values A, C and B, D, respectively. We build a node v_3 that stores categorical pair $(\{A, C\}, \{B, D\})$ and pair $\langle p_1, p_4 \rangle$. All nodes in the relational graph of Table 1 are shown in Figure 4.

Each node contains several *upper bounds* and *lower bounds* which describe the user preference on the categorical values in the categorical pair. Specifically, consider a node v that contains categorical pair (L_1, L_2) and tuple pair $\langle p, q \rangle$. If a user prefers p to q, i.e., $f(p) > f(q)$, we can obtain an inequality $\sum_{c \in L_1} g(c) - \sum_{c \in L_2} g(c) > u_{num} \cdot (q_{num} - p_{num})$. With a slight abuse of notations, we denote the inequality by $g(L_1) - g(L_2) > u_{num} \cdot (q_{num} - p_{num})$ for simplicity and call $u_{num} \cdot (q_{num} - p_{num})$ the *lower bound* of $g(L_1) - g(L_2)$. Similarly, if a user prefers q to p, we can derive an upper bound $u_{num} \cdot (q_{num} - p_{num})$ for $g(L_1) - g(L_2)$. During the interaction, there might be many upper and lower bounds in each node, which affects the execution time and the storage space. To reduce the number of bounds, we remove the *untight bounds* in each node.

Definition 5.4 (Untight Bound). Bound $u_{num} \cdot x$ is an untight lower bound for $g(L_1) - g(L_2)$ if $\forall u_{num} \in \mathcal{R}$, there exists a lower bound $u_{num} \cdot x'$ for $g(L_1) - g(L_2)$ such that $u_{num} \cdot (x' - x) > 0$. Similarly for upper bounds [3].

Intuitively, if $u_{num} \cdot x$ is untight, $\forall u_{num} \in \mathcal{R}$, there exists a lower bound $u_{num} \cdot x'$ which bounds $g(L_1) - g(L_2)$ more tightly than $u_{num} \cdot x$. We utilize the linear programming algorithm (LP) to check whether a lower bound $u_{num} \cdot x$ is untight (similarly for upper bounds [3]). Specifically, assume that there are r other lower bounds $u_{num} \cdot x'$. For each of them, we build a constraint $u_{num} \cdot x' > u_{num} \cdot x$. Then, we check whether $\forall u_{num} \in \mathcal{R}$, there exists a constraint that can be satisfied. Due to the lack of space, the detailed LP formulation is shown in the technical report [3]. Since there are r constraints and d_{num} variables, an LP solver (e.g., Simplex [6]) needs $O(r^2 d_{num})$ time in practice, which is time-consuming if r is large. To accelerate, we present a sufficient condition for Definition 5.4.

Lemma 5.5. *Bound $u_{num} \cdot x$ is an untight lower bound for $g(L_1) - g(L_2)$ if there exists a lower bound $u_{num} \cdot x'$ for $g(L_1) - g(L_2)$ such that $\forall u_{num} \in \mathcal{R}$, $u_{num} \cdot (x' - x) > 0$. Similarly for upper bounds [3].*

Lemma 5.5 compares $u_{num} \cdot x$ with the other lower bounds one by one. It checks whether there exists a lower bound $u_{num} \cdot x'$ that bounds $g(L_1) - g(L_2)$ tighter than $u_{num} \cdot x$ for any $u_{num} \in \mathcal{R}$. Suppose there are v vertices u_v in $\mathcal{R}$. Each comparison only takes $O(\text{v})$ time to check whether all vertices satisfy $u_v \cdot (x' - x) > 0$.

Hyperedge. Consider two categorical pairs (L_1, L_2) and (L_3, L_4). We first introduce a few operations on multisets and categorical pairs, and then discuss the hyperedges based on the operations.

- Define $L = L_1 + L_2$ to be the multiset containing the categorical values in L_1 or L_2. If $c \in L_1$ and $c \in L_2$, then $c \in L$ and $\mathfrak{m}_L(c) = \mathfrak{m}_{L_1}(c) + \mathfrak{m}_{L_2}(c)$. E.g., if $L_1 = \{A\}$ and $L_2 = \{A\}$, $L = \{A, A\}$.
- Define $L = L_1 - L_2$ to be the multiset containing the categorical values c such that $\mathfrak{m}_{L_1}(c) > \mathfrak{m}_{L_2}(c)$. $\forall c \in L$, $\mathfrak{m}_L(c) = \mathfrak{m}_{L_1}(c) - \mathfrak{m}_{L_2}(c)$. E.g., if $L_1 = \{A, A\}$ and $L_2 = \{A\}$, then $L = \{A\}$.
- Define $2L_1 = L_1 + L_1$. E.g., if $L_1 = \{A\}$, then $2L_1 = \{A, A\}$.
- Define $(L_1, L_2) + (L_3, L_4) = (L_1 + L_3 - L_2 - L_4, L_2 + L_4 - L_1 - L_3)$. E.g., $(\{A\}, \{B\}) + (\{B, D\}, \{A, C\}) = (\{D\}, \{C\})$.
- Define $(L_1, L_2) - (L_3, L_4) = (L_1, L_2) + (L_4, L_3)$.

Consider any three nodes $v_1, v_2, v_3 \in V$ containing (L_1, L_2), (L_3, L_4) and (L_5, L_6), respectively. If any two of the categorical pairs could deduce the third one based on the above introduced operations, there exists a hyperedge connecting v_1, v_2 and v_3. Without loss of generality, suppose (L_1, L_2) and (L_3, L_4) could deduce (L_5, L_6). There are the following deducing options, where $\odot \in \{+, -\}$.

$$(L_1, L_2) \odot (L_3, L_4) = (L_5, L_6) \text{ or } (2L_5, 2L_6) \text{ or } (L_6, L_5) \text{ or } (2L_6, 2L_5)$$

$$(2L_1, 2L_2) \odot (L_3, L_4) = (L_5, L_6) \text{ or } (L_6, L_5)$$

$$(L_1, L_2) \odot (2L_3, 2L_4) = (L_5, L_6) \text{ or } (L_6, L_5)$$

Example 5.6. Consider Figure 4. There exists a hyperedge connecting nodes v_1 (with $(\{A\}, \{B\})$), v_2 (with $(\{C\}, \{D\})$) and v_3 (with $(\{A, C\}, \{B, D\})$) since $(\{A\}, \{B\}) + (\{C\}, \{D\}) = (\{A, C\}, \{B, D\})$. The hyperedges are shown in small black dots.

If there exist three nodes that are connected by a hyperedge, we can use the bounds in any two nodes to generate new bounds for the third one. The generation is based on how the categorical pairs in the two nodes deduce the categorical pair in the third node. Concretely, consider nodes $v_1, v_2, v_3 \in V$ with categorical pairs (L_1, L_2), (L_3, L_4) and (L_5, L_6), respectively. Suppose that $(L_1, L_2) + (L_3, L_4) = (L_5, L_6)$ and nodes v_1 and v_2 have bounds as follows.

$$v_1 : x_1 \cdot u_{num} < g(L_1) - g(L_2) < y_1 \cdot u_{num}$$

$$v_2 : x_2 \cdot u_{num} < g(L_3) - g(L_4) < y_2 \cdot u_{num}$$

Then, we could generate new bounds for v_3.

$$v_3 : (x_1 + x_2) \cdot u_{num} < g(L_5) - g(L_6) < (y_1 + y_2) \cdot u_{num}$$

Note that the generated bounds might be untight. We only insert into nodes the tight generated bounds.

If (L_1, L_2) and (L_3, L_4) deduce (L_5, L_6) in other ways, we can also generate bounds for v_3 similarly. The detailed generation for other kinds of deduction is shown in the technical report [3].

Update. In each round of the interaction, we present two tuples $p, q \in D$ and ask a user to pick the one s/he prefers. According to the user feedback, $\mathcal{G}$ is updated. The update can be divided into two cases based on p and q: (1) p and q are the same in categorical attributes (i.e., $\forall i \in [1, d_{cat}]$, $p_{cat}[i] = q_{cat}[i]$) and (2) p and q differ in categorical attributes.

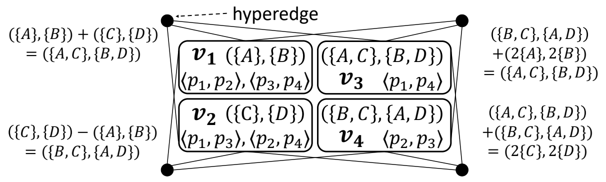

hyperedge
$(\{A\}, \{B\}) + (\{C\}, \{D\}) = (\{A, C\}, \{B, D\})$
v_1 $(\{A\}, \{B\})$ $\langle p_1, p_2 \rangle, \langle p_3, p_4 \rangle$
$(\{A, C\}, \{B, D\})$ v_3 $\langle p_1, p_4 \rangle$
$(\{B, C\}, \{A, D\}) + (2\{A\}, 2\{B\}) = (\{A, C\}, \{B, D\})$
$(\{C\}, \{D\}) - (\{A\}, \{B\}) = (\{B, C\}, \{A, D\})$
v_2 $(\{C\}, \{D\})$ $\langle p_1, p_3 \rangle, \langle p_2, p_4 \rangle$
$(\{B, C\}, \{A, D\})$ v_4 $\langle p_2, p_3 \rangle$
$(\{A, C\}, \{B, D\}) + (\{B, C\}, \{A, D\}) = (2\{C\}, 2\{D\})$

Figure 4: Relational graph

Consider the first case. As mentioned in Section 5.1.1, $\mathcal{R}$ will be updated to be $\mathcal{R} \cap h^+_{p-q}$ or $\mathcal{R} \cap h^+_{q-p}$. Based on the updated $\mathcal{R}$, we check and remove untight bounds in each node.

Example 5.7. Figure 3 shows two upper bounds in v_1. Suppose $\mathcal{R}$ is updated to be a line segment from $(0.5, 0.5)$ to $(1, 0)$. Bound $u_{num} \cdot (0.2, 0.8)$ is removed, since $u_{num} \cdot (0.8, 0.2)$ bounds $g(A) - g(B)$ more tightly than it for any u_{num} in the updated $\mathcal{R}$.

Consider the second case. Without loss of generality, suppose a user prefers p to q, i.e., $f(p) > f(q)$. We can obtain a lower bound $u_{num} \cdot (q_{num} - p_{num})$, denoted by $u_{num} \cdot x$ for simplicity, for $\sum_{i=1}^{d_{cat}} g(p_{cat}[i]) - g(q_{cat}[i])$. In the following, we first show how to update a single node in $\mathcal{G}$ as well as $\mathcal{R}$ based on the obtained lower bound $u_{num} \cdot x$, and then discuss the way to update multiple nodes in $\mathcal{G}$. Similarly for the updating based on upper bounds.

Assume that node v contains categorical pair (L_1, L_2) and tuple pair $\langle p, q \rangle$. We first add $u_{num} \cdot x$ into v and then update v by two steps. Firstly, all the upper bounds in v are used to update $\mathcal{R}$. Suppose there is an upper bound $g(L_1) - g(L_2) < u_{num} \cdot y$ in v. Since $u_{num} \cdot x < g(L_1) - g(L_2) < u_{num} \cdot y$, we have $u_{num} \cdot (y - x) > 0$. In this way, we could build a hyperplane h_{y-x} based on vector $y - x$ and update $\mathcal{R}$ to be $\mathcal{R} \cap h^+_{y-x}$. Secondly, all the bounds in v that are untight based on the updated $\mathcal{R}$ are removed.

Example 5.8. Figure 3 shows an upper bound in v_2. If we obtain a lower bound $u_{num} \cdot (0.1, 0.6)$ for $g(C) - g(D)$, we can build a hyperplane h based on vector $(0.1, -0.5)$ and update $\mathcal{R}$ to be $\mathcal{R} \cap h^+$.

When node v is updated, its updated bounds can be used to generate new bounds for the nodes $v_c \in V$ that are connected with v via the hyperedges. We say v triggers the update of nodes v_c. Then, v_c might trigger the update of other nodes. The update will be continually triggered and multiple nodes in $\mathcal{G}$ will be updated. Our method to update multiple nodes in $\mathcal{G}$ works as follows. We build a queue Q to store the nodes that have been updated. Initially, node v is inserted into Q. Then, we recursively pop out a node in Q until Q is empty. For each popped out node v_p, we generate new bounds for the nodes connected with v_p. If the generated bounds are tight, those nodes are updated and inserted into Q.

Since there could be many nodes updated, the execution time might be long. Thus, we utilize a strategy to control the updating process to strike for a trade off between the number of nodes updated and the time cost. We update based on the number of *levels*. In the first level, we update the node that contains the pair of tuples presented to a user as a question. In the i-th level, we update the nodes that are connected with the nodes updated in the $(i-1)$-th level ($i = 2, 3, 4, \ldots$). The update terminates when all the nodes in the α levels are updated, where $\alpha \geq 1$ is a given parameter. The setting of parameter α will be discussed in Section 6.1.

5.1.3 Maintenance on C. The *candidate set* $C \subseteq D$ contains the user's favorite tuple. During the interaction, based on the user feedback, we determine and prune from C the tuples that cannot be the user's favorite tuple. In the following, we first present several pruning strategies and then show the detailed procedure of maintaining C based on the proposed pruning strategies.

Same Categorical Values. Given a tuple $p \in C$, consider the set C_p of tuples that have the same categorical values as p. We can safely prune p from C if no matter which numerical utility vector the user uses in $\mathcal{R}$, there exists a tuple $q \in C_p$ such that s/he prefers q to p. Formally, we summarize as follows.

LEMMA 5.9. *Given $\mathcal{R}$, tuple p can be pruned if $\mathcal{R} \subseteq \cup_{q \in C_p} h^+_{q-p}$, where h_{q-p} is a hyperplane based on vector $q_{num} - p_{num}$.*

To check if $\mathcal{R} \subseteq \cup_{q \in C_p} h^+_{q,p}$, we utilize LP similar to Definition 5.4. In practice, the time complexity of a LP solver (e.g., Simplex [6]) is $O(|C_p|^2 d_{num})$. The detailed description can be found in [3].

Different Categorical Values. Given a tuple $p \in C$, consider the set C'_p of tuples that differ in the categorical attributes with p. For each $q \in C'_p$, we first find the node $v \in V$ that contains pair $\langle p, q \rangle$. Then, we check whether p can be pruned because of q based on the upper bounds in v. Specifically, suppose there are r upper bounds $u_{num} \cdot y_i$ in v, where $i \in [1, r]$. We divide $\mathcal{R}$ into r disjoint smaller polyhedrons $\mathcal{R}_i$ such that (1) each of them corresponds to exactly *one* upper bound $u_{num} \cdot y_i$ and (2) $\forall u_{num} \in \mathcal{R}_i$, $u_{num} \cdot y_i$ is the tightest upper bound, i.e., $\forall j \in [1, r]$ and $j \neq i$, $(y_j - y_i) \cdot u_{num} > 0$. For each $\mathcal{R}_i$, we build a hyperplane h_i based on vector $p_{num} - q_{num} + y_i$. Then, we can check whether p can be pruned from C based on $\mathcal{R}_i$ and h_i using the following lemma.

LEMMA 5.10. *Given polyhedrons $\mathcal{R}_i$ where $i \in [1, r]$, tuple p can be pruned from C if $\forall i \in [1, r]$, $\mathcal{R}_i \subseteq h^-_i$.*

If there are v_i vertices in $\mathcal{R}_i$, to identify whether $\mathcal{R}_i \subseteq h^-_i$, we need $O(\mathrm{v}_i)$ time to check whether all the vertices of $\mathcal{R}_i$ are in h^-_i. The total time complexity is $O(\sum_{i=1}^r \mathrm{v}_i)$. Similarly, we can find node v that contains pair $\langle q, p \rangle$ and check whether p can be pruned because of q based on the lower bounds in v [3].

Example 5.11. In Figure 3, node 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 hyperplane h_1 based on vector $(-0.2, 0.8)$ and let $\mathcal{R}_1 = \mathcal{R}$. Since $\mathcal{R}_1 \subseteq h^-_1$, p_2 can be pruned.

Maintain C based on the Pruning Strategies. Based on $\mathcal{R}$ and $\mathcal{G}$, C is maintained with the help of our pruning strategies. Specifically, we check each tuple $p \in C$ to see whether it cannot be the favorite tuple. We first utilize Lemma 5.9 to compare p with tuples in C_p. If Lemma 5.9 is satisfied, p is directly pruned. Otherwise, we compare p with tuples in C'_p according to Lemma 5.10.

5.2 Tuple Selection

In this section, we present two approaches of selecting tuples as questions in the interactive framework, which preform well empirically and have provable guarantees on the number of questions asked. In each round, two tuples are selected from set C and displayed to a user. Based on the categorical values of the selected tuples, the tuple selection can be classified into two types.

Algorithm 2: The *Combination* Algorithm

```
Input: A tuple set $D$
Output: The user's favorite tuple in $D$
1 $\mathcal{R} \leftarrow \{u_{num} \in \mathbb{R}^{d_{num}}_+ | \sum_{i=1}^{d_{num}} u_{num}[i] = 1\}$, $C \leftarrow D$
2 Initialize relational graph $\mathcal{G}$, $i \leftarrow 1$
3 while $|C| > 1$ do
4     if $i = 1, 3, 5, ...$ then
5         Display two tuples in type 1
6     else if $i = 2, 4, 6, ...$ then
7         Display two tuples in type 2
8     Update $\mathcal{R}$, $\mathcal{G}$ and $C$ based on the user feedback, $i \leftarrow i + 1$
9 return the only tuple left in $C$
```

- **Type 1.** We randomly select two tuples $p, q \in C$ that are the same in categorical attributes, i.e., $\forall i \in [1, d_{cat}], p_{cat}[i] = q_{cat}[i]$.
- **Type 2.** We randomly select two tuples $p, q \in C$ such that (1) they differ in categorical attributes and (2) the number of different categorical values between p and q is the least among all the pairs of tuples in C. The reason of (2) is that if there are many different categorical values, it is hard to learn the difference in preference (e.g., $g(L_1) - g(L_2) > u_{num} \cdot (q_{num} - p_{num})$) is caused by which pairs of categorical values and thus, makes it more difficult to analyze the user preference on the categorical values.

The First Approach: Combination. We alternate the two types of tuple selection. In the i-th round, where $i = 1, 3, 5, ...$, we use the tuple selection of type 1. In the j-th round, where $j = 2, 4, 6, ...$, we use the tuple selection of type 2. Based on the user feedback, $\mathcal{R}$, $\mathcal{G}$ and C are updated accordingly (see Section 5.1).
The Second Approach: Separation. We separate the two types of tuple selection. We first use the tuple selection of type 1 until each pair of tuples in C differ in at least one categorical attribute. For type 1, each pair of selected tuples only differ in the numeric attributes. Thus, we only need to build hyperplanes and update $\mathcal{R}$ and C (see Section 5.1). Secondly, we use the tuple selection of type 2. Since the selected tuples have different categorical values, we update $\mathcal{R}$, $\mathcal{G}$ and C accordingly (see Section 5.1).

5.3 Stopping Condition

During the interaction, we maintain a candidate set C that contains the user's favorite tuple. If there is only one tuple p in C, i.e., $|C| = 1$, we claim that p is the user's favorite tuple and thus, we stop the interaction and return p to the user.

5.4 Summary and Analysis

We summarize our algorithms by combining the techniques presented in previous sections. Denote by *Separation* the algorithm with the "separation" tuple selection approach and by *Combination* the algorithm with the "combination" tuple selection approach.

Combination and *Separation* display tuples selected from C to the user and update the data structures (e.g., $\mathcal{R}$, $\mathcal{G}$ and C) based on the user feedback until the stopping condition is satisfied. If the selected tuples p and q are the same in categorical attributes, hyperplane h_{p-q} will be built and divide $\mathcal{R}$ into two smaller polyhedrons. Based

Algorithm 3: The *Separation* Algorithm

```
Input: A tuple set $D$
Output: The user's favorite tuple in $D$
1 $\mathcal{R} \leftarrow \{u_{num} \in \mathbb{R}_+^{d_{num}} \mid \sum_{i=1}^{d_{num}} u_{num}[i] = 1\}$, $C \leftarrow D$
2 Initialize relational graph $\mathcal{G}$
3 while $\exists p, q \in C, \forall i \in [1, d_{cat}], p_{cat}[i] = q_{cat}[i]$ do
4     Display two tuples in type 1
5     Update $\mathcal{R}$ and $C$ based on the user feedback
6 while $|C| > 1$ do
7     Display two tuples in type 2
8     Update $\mathcal{R}$, $\mathcal{G}$ and $C$ based on the user feedback
9 return the only tuple left in $C$
```

on the user feedback, one of the polyhedron will be pruned and $\mathcal{R}$ will be strictly smaller. If the selected tuples differ in categorical attributes, there will be bounds added to $\mathcal{G}$ and thus, the bounds on some categorical values will be more tight. In both cases, $|C|$ is strictly smaller after each question, since we can know the user preference on a pair of tuples and thus, prune at least one tuple.

The pseudocode of algorithms *Combination* and *Separation* are presented in Algorithm 2 and Algorithm 3, respectively. Their theoretical analysis are summarized in Theorem 5.12.

Theorem 5.12. *Algorithm* Combination *and algorithm* Separation *solve ISM by asking the user $O(n)$ questions.*

5.5 Extension to Top-k

We show the way to extend our algorithms to return the top-k tuples. Our extension strategy is to iterate the algorithms k times and output the tuples returned by each iteration. Recall that our algorithms maintain (1) C, which contains the user's favorite tuple and (2) $\mathcal{R}$ and $\mathcal{G}$, which store the learnt information of user preference. We maintain a tuple set $\mathcal{B}$ as the output. In the end of each iteration (i.e., when $|C| = 1$), we put the tuple in C into $\mathcal{B}$. Then, we start a new iteration by keeping $\mathcal{R}$ and $\mathcal{G}$ of the last iteration and initializing C to be $D \setminus \mathcal{B}$. We continue this process until $|\mathcal{B}| = k$.

6 EXPERIMENTS

We conducted experiments on a machine with 3.10GHz CPU and 16GB RAM. All programs were implemented in C/C++.

Datasets. Following existing studies, we conducted experiments on synthetic and real datasets used in [28, 29]. The synthetic datasets contain both numerical and categorical attributes. The numerical attributes are anti-correlated and generated by a generator developed for skyline operators [7]. The categorical attributes are generated according to a Zipfian distribution [15, 28]. By default, the Zipfian parameter is set to 1. The real datasets are *Nursery* and *Car* that are commonly used in [28, 29]. Dataset *Nursery* contains 12,960 tuples with 8 attributes. Following [28], we transformed 6 attributes (parents, has_nurs, housing, finance, social and health) to numerical attributes since their values are ordered. For example, attribute "parents" has 3 values: usual, pretentious, great_pretentious. We transformed them to numbers 0.33, 0.66 and 1, respectively. The remaining 2 attributes (the form of family and the number of children) were used as categorical attributes, since there is no trivial order on their values. One example is attribute "the number of children". Although its values are numbers, it is not clear whether a family with one child is "better" than a family with two children. Dataset *Car* includes 69,052 used cars. We maintained its 4 numerical attributes (price, date of manufacture, horse power and used kilometers) and 3 categorical attributes (fuel type, vehicle type, gearbox). For all datasets, each numerical attribute was normalized to $(0, 1]$. Besides, to be consistent with the experimental setting in [8, 29], we preprocessed all the datasets to contain skyline tuples only (which are the user's possible favorite tuples) since we are interested in the user's favorite tuple. Specifically, tuple p in the dataset is maintained if $\nexists q \in D$ such that (1) p and q are the same in categorical attributes and (2) p is not better than q in each numerical attribute, and strictly worse in at least one numerical attribute.

Algorithms. We evaluated our algorithms *Tree*, *Combination* and *Separation* with existing algorithms: *ActiveRanking* [13], *Adaptive* [24], *UH-Random* [29] and *UH-Simplex* [29]. Note that the existing algorithms are designed for numerical attributes. Following the commonly used approach for handling categorical attributes in preference queries [24] and machine learning area [20], we transformed each tuple to contain numerical attributes only for existing algorithms processing. Specifically, we removed all the categorical attributes. Then, for each categorical *value*, we built a new numerical *attribute*. If a tuple is described by this categorical value originally, we set its value of the newly built attribute to 1. Otherwise, we set it to 0. In this way, the original numerical attributes remain and the values of the newly built attributes are either 0 or 1. Besides, since none of the existing algorithms aims to find the favorite tuple directly, we adapted them as follows. (1) *ActiveRanking* learns the full ranking of all tuples by interacting with the user. We return the tuple that ranks the first when the ranking is obtained. (2) *Adaptive* learns the user preference by interacting with the user. According to the experimental results in [24], the learnt user preference is very close to the theoretical optimal if the error threshold σ of the learnt user preference is set to $1 - 10^{-5}$. Thus, we set $\sigma = 1 - 10^{-5}$ and return the tuple that ranks the first w.r.t. the learnt user preference. (3) *UH-Simplex* and *UH-Random* find a tuple such that a criterion called the *regret ratio*, evaluating how "regretful" a user is when s/he sees the resulting tuple instead of the whole dataset, is minimized by interacting with the user. They stop the interaction when they can find a tuple whose regret ratio satisfies a given threshold. We set the threshold to 0, since this guarantees that the returned tuple is the user's favorite tuple.

Parameter Setting. We evaluated the performance of each algorithm by varying different parameters: (1) the number of questions we can ask; (2) the dataset size n; (3) the number of numerical attributes d_{num}; (5) the number of categorical attributes d_{cat}; (6) the cardinality of each categorical attribute c_{val} (i.e., the number of values in each categorical attribute). Following [28, 29], unless stated explicitly, we set $n = 100,000$, $d_{cat} = 3$, $d_{num} = 2$ and $c_{val} = 4$. (As shown in Section 6.2, the existing algorithms have difficulties to run when the default parameters are set to be large.)

Performance Measurement. We evaluated the performance of each algorithm by 3 measurements: (1) Execution time. The processing time. (2) Candidate size. The number of tuples in C during the interaction. We reported the percentage of remaining

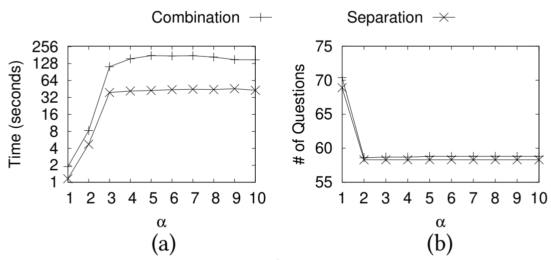


Figure 5: Update strategy

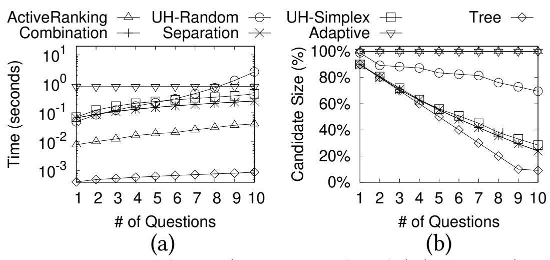


Figure 6: Synthetic (categorical only) ($c_{val} = 10$)

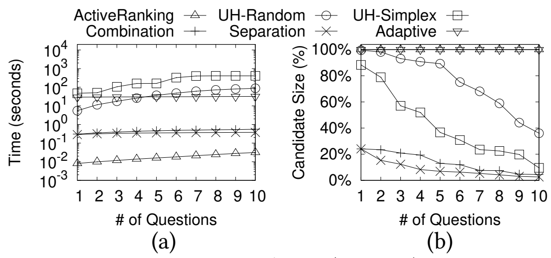


Figure 7: Synthetic ($c_{val} = 4$)

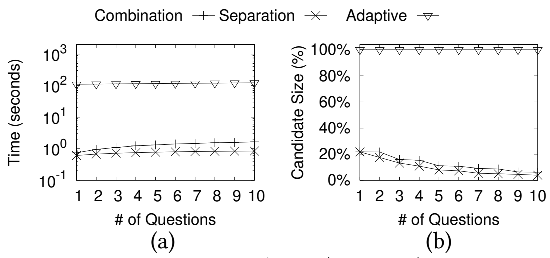


Figure 8: Synthetic ($c_{val} = 10$)

tuples in C in each interactive round. (3) The number of questions asked. The number of the interactive rounds. Each algorithm was conducted 10 times and we reported its average performance.

6.1 Performance Study of Our Algorithms

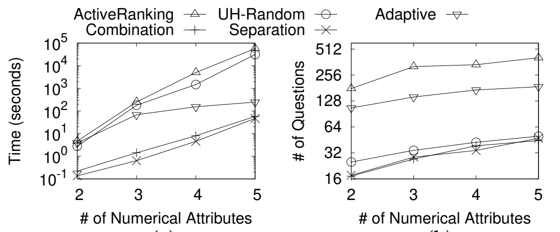
We evaluated our strategy used to control the updating process of $\mathcal{G}$ (shown in Section 5.1.2) via the execution time and the number of questions asked. Figure 5 shows its performance by varying parameter α on the dataset, where $c_{val} = 10$ and the other parameters are set by default. When α increases, the execution time increases since there are more nodes updated in $\mathcal{G}$. The number of questions asked only decreases when α increases from 1 to 2. This means that the more nodes updated help little to prune tuples. When $\alpha = 2$, the number of questions asked is the least and the execution time is small. Thus, we set $\alpha = 2$ in the following experiments.

6.2 Results on Synthetic Datasets

We studied our algorithm *Tree* on the synthetic dataset, in which the tuples only differ in categorical attributes. We set $c_{val} = 10$ and the other parameters by default. For completeness, our algorithms *Combination* and *Separation*, and existing ones were also involved. Figure 6 shows the execution time and the candidate size by varying the number of questions we can ask. Except *UH-Random* that costs several seconds, the other algorithms can finish within 1 second. Algorithm *Tree* only takes 10^{-3} seconds, which performs the best among all algorithms. Besides, our algorithms reduce the candidate size the most effectively. In particular, *Tree* is the only algorithm that reduces the candidate size more than 50% after asking 5 questions.

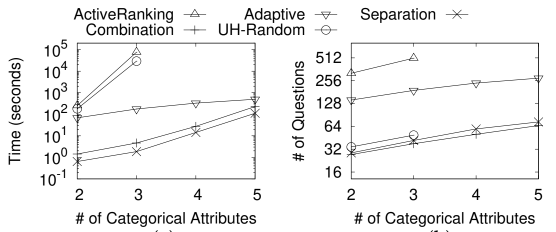
We compared *Combination* and *Separation* against existing algorithms on the synthetic datasets, in which the tuples differ in both numerical and categorical attributes, by varying the number of questions we can ask. Figure 7 demonstrates the performance on the datasets where all parameters are set by default. *UH-Simplex* does not reduce the candidate size effectively and runs the slowest among all algorithms. It takes up to 410 seconds to ask 10 questions. In comparison, the second slowest existing algorithm takes within 90 seconds. Since *UH-Simplex* is time-consuming, we do not put it into comparison in the rest of the experiments. *UH-Random* and *Adaptive* are also slow. They take tens of seconds, since *UH-Random* spends much time pruning tuples after asking each question and *Adaptive* maintains a costly data structure. Besides, *UH-Random* reduces the candidate size ineffectively. After 10 questions asked, there are still more than 36% of tuples left. *Adaptive* only focuses on learning the user preference and fails to provide any reduction on the candidate size. Our algorithms reduce the candidate size effectively and efficiently. They prune more than 90% of tuples by asking 7 questions within 0.5 seconds. *ActiveRanking* is faster than our algorithms, since it does not prune tuples, which neglects the connection between tuples and results in asking more questions. Although our algorithms spend slightly more time, their execution time are small and reasonable given that they can effectively reduce the candidate size and obtain the user's favorite tuple by asking a few questions. Figure 8 shows the result on the dataset where $c_{val} = 10$ and the other parameters are set by default. Except *Adaptive*, none of the existing algorithm can finish the computation. They either cost more than 10^5 seconds or run out of the memory. Thus, we set $c_{val} = 4$ by default in the rest of the experiments.

We studied the scalability of *Combination* and *Separation* on the number of attributes with existing algorithms. Each algorithm is measured by the execution time and the number of questions asked by varying d_{num} and d_{cat}, respectively. In Figure 9, we fixed $d_{cat} = 2$ and increased d_{num} from 2 to 5. The results show that our algorithms ask the least number of question within the shortest time. When $d_{num} = 4$, they ask about 38 questions in 8 seconds. In comparison, the fastest existing algorithm *Adaptive* takes 154 seconds and *UH-Random* that asks the least number of questions among existing algorithms asks around 43 questions. Although the number of questions asked by *UH-Random* is close to our algorithms, it takes about 1482 seconds, which is 2 orders of magnitude longer than our algorithms. In Figure 10, we fixed $d_{num} = 3$ and varied d_{cat} from 2 to 5. *UH-Random* and *ActiveRanking* cannot run when $d_{cat} > 3$ due to the excessive execution time (more than 10^6



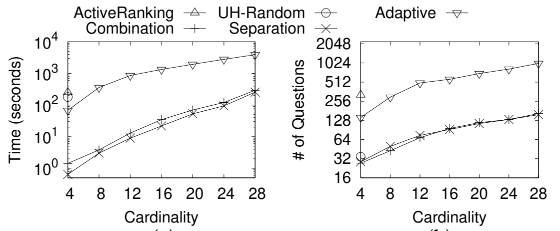
(a) (b)

Figure 9: Vary d_{num}



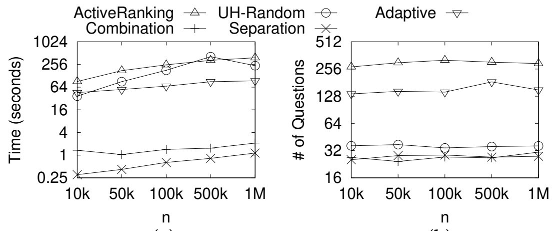
(a) (b)

Figure 10: Vary d_{cat}



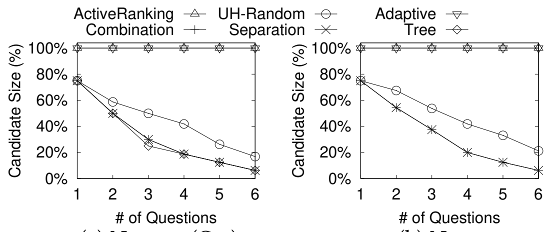
(a) (b)

Figure 11: Vary c_{val}



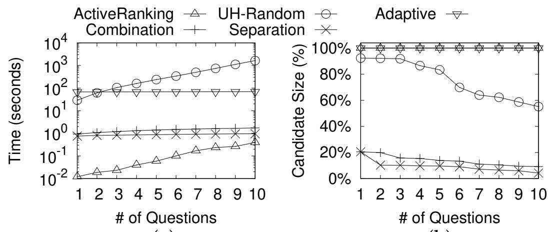
(a) (b)

Figure 12: Vary n



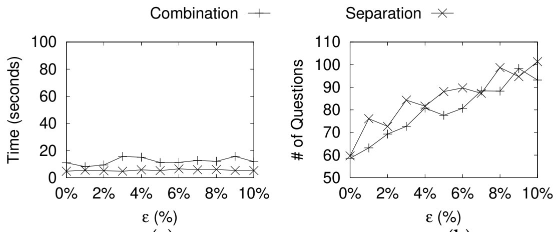
(a) Nursery(Cat) (b) Nursery

Figure 13: Nursery



(a) (b)

Figure 14: Car



(a) (b)

Figure 15: Confidence Study

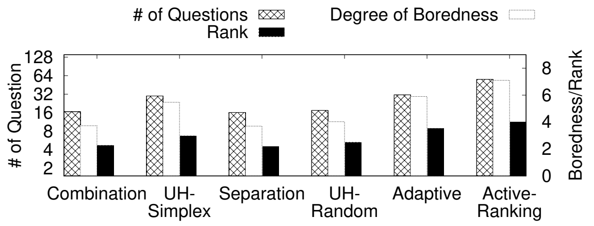


Figure 16: User Study

seconds). Our algorithms ask the least number of questions within the shortest time. When $d_{cat} = 3$, they ask around 41 questions within 5 seconds. In contrast, *UH-Random* that asks the least number of questions among existing algorithms needs 49 questions, and the fastest existing algorithm *Adaptive* takes 175 seconds.

We evaluated the scalability of our algorithms against existing algorithms on the cardinality of categorical attributes c_{val}. In Figure 11, c_{val} is varied from 4 to 28. The results show that *Combination* and *Separation* perform the best on both the execution time and the number of questions asked. When the cardinality increases, the execution time and the number of questions required by them increase slowly. For example, when $c_{val} = 16$, our algorithms ask 95 questions within 35 seconds, while *Adaptive* asks 566 questions in 1340 seconds. Note that we only show the performance of *UH-Random* and *ActiveRanking* when $c_{val} = 4$, since they take too much time (more than 10^6 seconds) when the cardinality increases.

In Figure 12, we evaluated the scalability of our algorithms on the data size n. *Combination* and *Separation* scale the best in terms of the execution time and the number of questions asked. Their execution times are less than 2 seconds even if $n = 1,000,000$, while the existing algorithms spend at least 94 seconds. *ActiveRanking* even takes more than 385 seconds. Besides, our algorithms also ask at least 6 fewer questions than the existing algorithms for arbitrary n. When $n = 500,000$, they ask around 27 questions, while *ActiveRanking* asks 305 questions and *Adaptive* asks 184 questions.

6.3 Results on Real Datasets

We studied the performance of *Combination* and *Separation* against existing algorithms, namely *UH-Random*, *Adaptive* and *ActiveRanking*, on 2 real datasets: *Nursery* and *Car* by varying the number of questions we can ask. Besides, we constructed another real dataset, namely *Nursery(Cat)*, by only maintaining the categorical attributes in the *Nursery* dataset, and put algorithm *Tree* into comparison.

The results on datasets *Nursery(Cat)* and *Nursery* are shown in Figure 13. All algorithms only take within 0.05 seconds to execute. This is because that the number of skyline tuples in a real dataset is usually smaller than that in a synthetic dataset. Thus, we omit the results on the execution time. Figure 13 (a) and (b) show the candidate size of each algorithm on dataset *Nursery(Cat)* and dataset *Nursery*, respectively. Our algorithms *Tree*, *Combination* and *Separation* reduces the candidate size the most effectively. On both datasets, after 6 questions, our algorithms only contain 6.25% of tuples in C. In contrast, the existing algorithm which reduces the candidate size the most effectively contains 17% of tuples in C.

The results on dataset *Car* are shown in Figure 14. All algorithms only cost a few seconds. Note that *ActiveRanking* takes less time to process after each question than our algorithms, since it does not need to prune tuples after each question. Although our algorithms take slightly more time, the execution time is small and reasonable since our algorithms are able to reduce the candidate size the most effectively. After 9 questions, our algorithms can prune about 90% of tuples in C, while the best existing algorithm can only prune around 40% of tuples in C. Besides, since *ActiveRanking* asks more questions than our algorithm, its total execution time is much longer.

6.4 Interaction Study

6.4.1 Confidence Study. We studied the situation that a user might be unable to answer some questions during the interaction. Our study is conducted on a synthetic dataset, where $c_{val} = 10$ and the other parameters are set by default. Given two tuples p and q presented as a question, we define the utility difference to be $(f(p) - f(q))/f(p)$, where $f(p) > f(q)$. If the utility difference is smaller than a given threshold ϵ, we assume that there is a 50% chance that a user cannot answer the question. Our algorithms *Combination* and *Separation* were adapted so that if the question cannot be answered, they will select and ask another question based on their tuple selection strategy. Figure 15 shows the execution time and the number of questions asked with the increasing threshold ϵ. The results show that the execution time remains almost unchanged and the number of questions asked slightly increases. This verifies that unanswered questions affect little to our algorithms.

6.4.2 User study. We conducted a user study on dataset *Car* to demonstrate the robustness of our algorithms, since users might make mistakes or provide inconsistent feedback during the interaction. As shown in Figure 14, existing algorithms ask many questions in a long time. The user study cannot be conducted on *Car* directly. Thus, following [27, 29], we randomly selected 1000 candidate used cars from the dataset described by 5 attributes, namely price, year of manufacture, horse power, used kilometers and fuel type. 30 participants were recruited and their average result was reported.

We compared our algorithms *Combination* and *Separation*, against 4 existing algorithms: *UH-Random*, *UH-Simplex*, *Adaptive* and *ActiveRanking*. Each algorithm aims at finding the user's favorite used car. Since the user preference is unknown, we re-adapted *Adaptive* (instead of the way described previously). *Adaptive* maintains an estimated user preference u_e during the interaction. We compared the user's answer of some randomly selected questions with the prediction w.r.t. u_e. If 75% questions [29] can be correctly predicted, we stop and return the car with the highest utility w.r.t. u_e. The other existing algorithms follow the adaptation described at the beginning of Section 6. Each algorithm was evaluated by 3 measurements. (1) *The number of questions asked.* (2) *The degree of boredness* which is a score from 1 to 10 given by each participant. It indicates how bored the participant feels when s/he sees the returned used car after answering several questions (1 denotes the least bored and 10 means the most bored). (3) *Rank* which is the ranking of algorithms given by each participant. Since participants sometimes gave the same score for different algorithms, we asked each participant to give a ranking of algorithms to distinguish different algorithms.

Figure 16 shows the results. The number of questions required by *Combination* and *Separation* are 16.2 and 16.1, respectively, while existing algorithms ask more than 17.5 questions. In particular, the number of questions asked by *Adaptive* and *ActiveRanking* are up to 31.3 and 55.9, respectively. Besides, *Combination* and *Separation* are the least boring and rank the best. Their degrees of boredness are 3.73 and 3.70, and their rankings are 2.3 and 2.2, respectively. In comparison, for the most boring algorithm *ActiveRanking*, its degree of boredness and ranking are 7.1 and 4, respectively.

6.5 Summary

The experiments showed the superiority of our algorithms over the best-known existing ones. (1) We are effective and efficient. Compared with existing algorithms, our algorithms achieve orders of improvement on execution time and ask fewer questions (e.g., under typical settings, our algorithms ask up to 10 times fewer questions than existing algorithms). (2) Our algorithms scale well on the number of attributes, the cardinality of categorical attributes and the dataset size (e.g., *Separation* asks 59 questions within 15 seconds on the dataset with 3 numerical attributes and 4 categorical attributes, while *Adaptive* asks 238 questions in 329 seconds). (3) The pruning strategies are useful (e.g., our pruning strategies can reduce the candidate size to 20% by only asking 3 questions on the dataset with 3 numerical attributes and 2 categorical attributes, while existing algorithms can only reduce to around 60%). In summary, *Tree* asks the least number of questions in the shortest time for the special ISM, where all tuples only differ in categorical attributes. *Combination* and *Separation* run the fastest and ask the least number of questions for the general ISM, where the tuples can differ in both numerical and categorical attributes.

7 CONCLUSION

In this paper, we present interactive algorithms for searching the user's favorite tuple in the dataset described by both numerical and categorical attributes, pursing as little user effort as possible. For the special case of ISM, we propose algorithm *Tree* which asks an asymptotically optimal number of questions. For the general case of ISM, two algorithms *Combination* and *Separation* are presented, which perform well on the execution time and the number of questions asked. Extensive experiments showed that our algorithms are efficient and effective for problem ISM. As for future work, we are looking forward to enhancing the robustness of the algorithms, i.e., considering the case that users may answer questions mistakenly.

REFERENCES

[1] Abolfazl Asudeh, Azade Nazi, Nan Zhang, Gautam Das, and H. V. Jagadish. 2019. RRR: Rank-Regret Representative. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 263–280.

[2] Product Attributes. 2021. https://repository.up.ac.za/bitstream/handle/2263/27411/04chapter4.pdf?sequence=5&isAllowed=y

[3] Anonymous Author(s). 2021. *Interactive Search with Mixed Attributes.* Technical Report.

[4] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. 2007. Eliciting Matters – Controlling Skyline Sizes by Incremental Integration of User Preferences. In *Advances in Databases: Concepts, Systems and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 551–562.

[5] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. 2007. User Interaction Support for Incremental Refinement of Preference-Based Queries. In *Proceedings of the First International Conference on Research Challenges in Information Science.* 209–220.

[6] Dimitris Bertsimas and John Tsitsiklis. 1997. *Introduction to Linear Optimization* (1st ed.). Athena Scientific.

[7] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *Proceedings of the International Conference on Data Engineering.* 421–430.

[8] Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan. 2017. k-Regret Minimizing Set: Efficient Algorithms and Hardness. In *20th International Conference on Database Theory.* Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 11:1–11:19.

[9] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. 2008. *Computational geometry: Algorithms and applications.* Springer Berlin Heidelberg.

[10] Brian Eriksson. 2013. Learning to Top-k Search Using Pairwise Comparisons. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, Vol. 31. PMLR, Scottsdale, Arizona, USA, 265–273.

[11] Joseph F Hair. 2009. Multivariate data analysis. (2009).

[12] James L. Hein. 2002. *Discrete Mathematics* (2nd ed.). Jones and Bartlett Publishers, Inc., USA.

[13] Kevin G. Jamieson and Robert D. Nowak. 2011. Active Ranking Using Pairwise Comparisons. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2240–2248.

[14] Bin Jiang, Jian Pei, Xuemin Lin, David W. Cheung, and Jiawei Han. 2008. Mining Preferences from Superior and Inferior Examples. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 390–398.

[15] Norman Lloyd Johnson, Samuel Kotz, and Adrienne W. Kemp. 1992. *Univariate Discrete Distributions.* Wiley-Interscience.

[16] Tie-Yan Liu. 2010. Learning to Rank for Information Retrieval. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM, New York, NY, USA, 904.

[17] Alchemer LLC. 2021. https://www.alchemer.com/resources/blog/how-many-survey-questions/

[18] Scott B MacKenzie. 1986. The role of attention in mediating the effect of advertising on attribute importance. *Journal of Consumer Research* 13, 2 (1986), 174–195.

[19] Lucas Maystre and Matthias Grossglauser. 2017. Just Sort It! A Simple and Effective Approach to Active Preference Learning. In *Proceedings of the 34th International Conference on Machine Learning*. 2344–2353.

[20] Boriana L Milenova, Joseph S Yarmus, and Marcos M Campos. 2005. 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*. 1152–1163.

[21] Danupon Nanongkai, Ashwin Lall, Atish Das Sarma, and Kazuhisa Makino. 2012. Interactive Regret Minimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 109–120.

[22] Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J. Lipton, and Jun Xu. 2010. Regret-Minimizing Representative Databases. In *Proceedings of the VLDB Endowment*, Vol. 3. VLDB Endowment, 1114–1124.

[23] Peng Peng and Raymong Chi-Wing Wong. 2015. K-Hit Query: Top-k Query with Probabilistic Utility Function. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 577–592.

[24] Li Qian, Jinyang Gao, and H. V. Jagadish. 2015. Learning User Preferences by Adaptive Pairwise Comparison. In *Proceedings of the VLDB Endowment*, Vol. 8. VLDB Endowment, 1322–1333.

[25] QuestionPro. 2021. https://www.questionpro.com/blog/optimal-number-of-survey-questions/

[26] Nikos Sarkas, Gautam Das, Nick Koudas, and Anthony K. H. Tung. 2008. Categorical Skylines for Streaming Data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada). Association for Computing Machinery, New York, NY, USA, 239–250.

[27] Weicheng Wang, Raymond Chi-Wing Wong, and Min Xie. 2021. Interactive Search for One of the Top-k. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 13.

[28] Raymond Chi-Wing Wong, Jian Pei, Ada Wai-Chee Fu, and Ke Wang. 2009. Online Skyline Analysis with Dynamic Preferences on Nominal Attributes. *IEEE Transactions on Knowledge and Data Engineering* 21, 1 (2009), 35–49.

[29] Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. 2019. Strongly Truthful Interactive Regret Minimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 281–298.

[30] Min Xie, Raymond Chi-Wing Wong, Jian Li, Cheng Long, and Ashwin Lall. 2018. Efficient K-Regret Query Algorithm with Restriction-Free Bound for Any Dimensionality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 959–974.

[31] Jiping Zheng and Chen Chen. 2020. Sorting-Based Interactive Regret Minimization. In *Web and Big Data-4th International Joint Conference, APWeb-WAIM*. Springer, 473–490.

Appendix A MAINTENANCE ON $\mathcal{G}$

A.1 Node

In this section, we are supplementing our strategies of maintaining the lower bounds and upper bounds in each node of $\mathcal{G}$. First, we show the implementation of Definition 5.4 and Lemma 5.5 that apply to lower bounds. Then, we present a definition and a lemma for upper bounds with discussion in detail.

A.1.1 Lower Bounds. We are to show the linear programming for Definition 5.4. We define a variable w to be the objective function. Assume there are r other lower bounds $u_{num} \cdot x_i$ for $g(L_1) - g(L_2)$, where $i \in [1, r]$. For each of them, we build a constraint $u_{num} \cdot (x - x_i) \geq w$. Formally, we construct a LP as follows.

$$\begin{aligned} &\text{maximize } w \\ &\text{subject to } u_{num} \cdot (x - x_1) \geq w \\ &\qquad u_{num} \cdot (x - x_2) \geq w \\ &\qquad \ldots\ldots \\ &\qquad u_{num} \cdot (x - x_r) \geq w \\ &\qquad u_{num} \in \mathcal{R} \end{aligned}$$

If the objective function $w < 0$, then $\forall u_{num} \in \mathcal{R}$, $\exists i \in [1, r]$ such that $u_{num} \cdot (x - x_i) < 0$. Thus, $\forall u_{num} \in \mathcal{R}$, there exists a lower bound $u_{num} \cdot x_i$ such that $u_{num} \cdot (x_i - x) > 0$.

In the following, we present the implementation for Lemma 5.5. Recall that $\mathcal{R}$ is a polyhedron, which is an intersection of a set of halfspaces. One property of the polyhedron is that it is convex [9]. Specifically, suppose $\mathcal{R}$ has v vertices u_{vi}, where $i \in [1, \text{v}]$. Each u_{num} in $\mathcal{R}$ can be represented by the vertices of $\mathcal{R}$, i.e., $u_{num} = \sum_{i=1}^{\text{v}} t_i u_{vi}$, where t_i is a positive real number such that $\sum_{i=1}^{\text{v}} t_i = 1$.

Lemma 5.5 compares lower bound $u_{num} \cdot x$ with the other lower bounds $u_{num} \cdot x'$ one by one. For each comparison, e.g., $u_{num} \cdot x$ and $u_{num} \cdot x'$, we check whether all vertices $u_{vi} \in \mathcal{R}$, where $i \in [1, \text{v}]$, satisfy $u_{vi} \cdot (x' - x) > 0$, . If yes, $\forall u_{num} \in \mathcal{R}$, we have

$$\begin{aligned} u_{num} \cdot (x' - x) &= (\sum_{i=1}^{\text{v}} t_i u_{vi}) \cdot (x' - x) \\ &= \sum_{i=1}^{\text{v}} (t_i u_{vi} \cdot (x' - x)) \\ &> 0 \end{aligned}$$

A.1.2 Upper Bounds. In the following, we first show the formal definition of an untight upper bound and then present Lemma A.2 which is a sufficient condition for the definition and is used to accelerate.

Definition A.1. Bound $u_{num} \cdot y$ is an untight upper bound for $g(L_1) - g(L_2)$ if $\forall u_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot y'$ for $g(L_1) - g(L_2)$ such that $u_{num} \cdot (y' - y) < 0$.

Intuitively, if $u_{num} \cdot y$ is an untight upper bound, $\forall v_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot y'$ which bounds $g(L_1) - g(L_2)$ more tightly than $u_{num} \cdot y$. We utilize a LP to check whether $u_{num} \cdot y$ is untight. Specifically, we define a variable w to be the objective function. Assume there are r other upper bounds $u_{num} \cdot y_i$ for $g(L_1) - g(L_2)$, where $i \in [1, r]$. For each of them, we build a constraint $u_{num} \cdot (y - y_i) \leq w$. Formally, we construct a LP as follows.

$$\begin{aligned} &\text{minimize } w \\ &\text{subject to } u_{num} \cdot (y - y_1) \leq w \\ &\qquad u_{num} \cdot (y - y_2) \leq w \\ &\qquad \ldots\ldots \\ &\qquad u_{num} \cdot (y - y_r) \leq w \\ &\qquad u_{num} \in \mathcal{R} \end{aligned}$$

If the objective function $w > 0$, then $\forall u_{num} \in \mathcal{R}$, $\exists i \in [1, r]$ such that $u_{num} \cdot (y - y_i) > 0$. Thus, $\forall u_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot y_i$, such that $u_{num} \cdot (y_i - y) < 0$.

Lemma A.2. *Bound $u_{num} \cdot y$ is an untight upper bound for $g(L_1) - g(L_2)$ if there exists an upper bound $u_{num} \cdot y'$ for $g(L_1) - g(L_2)$ such that $\forall u_{num} \in \mathcal{R}$, $u_{num} \cdot (y' - y) < 0$.*

Proof. Since there exists an upper bound $u_{num} \cdot y'$ for $g(L_1) - g(L_2)$ such that $\forall u_{num} \in \mathcal{R}$, $u_{num} \cdot (y' - y) < 0$, i.e., $\forall u_{num} \in \mathcal{R}$, $g(L_1) - g(L_2) < u_{num} \cdot y' < u_{num} \cdot y$. This implies that $\forall u_{num} \in \mathcal{R}$, upper bound $u_{num} \cdot y'$ always bounds $g(L_1) - g(L_2)$ more tightly than $u_{num} \cdot y$. □

Suppose $\mathcal{R}$ has v vertices u_{vi}, where $i \in [1, \text{v}]$. Since $\mathcal{R}$ is a polyhedron, each u_{num} in $\mathcal{R}$ can be represented as $u_{num} = \sum_{i=1}^{\text{v}} t_i u_{vi}$, where t_i is a positive real number such that $\sum_{i=1}^{\text{v}} t_i = 1$ [9].

Lemma A.2 compares upper bound $u_{num} \cdot y$ with the other upper bounds $u_{num} \cdot y'$ one by one. For each comparison, e.g., $u_{num} \cdot y$ and $u_{num} \cdot y'$, we check whether all vertices $u_{vi} \in \mathcal{R}$, where $i \in [1, \text{v}]$, satisfy $u_{vi} \cdot (y' - y) < 0$, . If yes, $\forall u_{num} \in \mathcal{R}$, we have

$$\begin{aligned} u_{num} \cdot (y' - y) &= (\sum_{i=1}^{\text{v}} t_i u_{vi}) \cdot (y' - y) \\ &= \sum_{i=1}^{\text{v}} (t_i u_{vi} \cdot (y' - y)) \\ &< 0 \end{aligned}$$

A.2 Update

Consider nodes $v_1, v_2, v_3 \in V$ with categorical pairs (L_1, L_2), (L_3, L_4) and (L_5, L_6), respectively. Suppose v_1 and v_2 have bounds as follows.

$$v_1 : x_1 \cdot u_{num} < g(L_1) - g(L_2) < y_1 \cdot u_{num}$$
$$v_2 : x_2 \cdot u_{num} < g(L_3) - g(L_4) < y_2 \cdot u_{num}$$

Then, based on the way of (L_1, L_2) and (L_3, L_4) deducing (L_5, L_6), we could generate new bounds for v_3 as follows.

(1) Suppose $(L_1, L_2) + (L_3, L_4) = (L_5, L_6)$.

$$v_3 : (x_1 + x_2) \cdot u_{num} < g(L_5) - g(L_6) < (y_1 + y_2) \cdot u_{num}$$

(2) Suppose $(L_1, L_2) - (L_3, L_4) = (L_5, L_6)$.

$$v_3 : (x_1 - y_2) \cdot u_{num} < g(L_5) - g(L_6) < (y_1 - x_2) \cdot u_{num}$$

(3) Suppose $(L_1, L_2) + (L_3, L_4) = (L_6, L_5)$

$$v_3 : -(y_1 + y_2) \cdot u_{num} < g(L_5) - g(L_6) < -(x_1 + x_2) \cdot u_{num}$$

(4) Suppose $(L_1, L_2) - (L_3, L_4) = (L_6, L_5)$

$$v_3 : (x_2 - y_1) \cdot u_{num} < g(L_5) - g(L_6) < (y_2 - x_1) \cdot u_{num}$$

(5) Suppose $(L_1, L_2) + (L_3, L_4) = (2L_5, 2L_6)$

$$v_3 : \frac{1}{2}(x_1 + x_2) \cdot u_{num} < g(L_5) - g(L_6) < \frac{1}{2}(y_1 + y_2) \cdot u_{num}$$

(6) Suppose $(L_1, L_2) - (L_3, L_4) = (2L_5, 2L_6)$

$$v_3 : \frac{1}{2}(x_1 - y_2) \cdot u_{num} < g(L_5) - g(L_6) < \frac{1}{2}(y_1 - x_2) \cdot u_{num}$$

(7) Suppose $(L_1, L_2) + (L_3, L_4) = (2L_6, 2L_5)$

$$v_3 : -\frac{1}{2}(y_1 + y_2) \cdot u_{num} < g(L_5) - g(L_6) < -\frac{1}{2}(x_1 + x_2) \cdot u_{num}$$

(8) Suppose $(L_1, L_2) - (L_3, L_4) = (2L_6, 2L_5)$

$$v_3 : \frac{1}{2}(x_2 - y_1) \cdot u_{num} < g(L_5) - g(L_6) < \frac{1}{2}(y_2 - x_1) \cdot u_{num}$$

(9) Suppose $(2L_1, 2L_2) + (L_3, L_4) = (L_5, L_6)$

$$v_3 : (2x_1 + x_2) \cdot u_{num} < g(L_5) - g(L_6) < (2y_1 + y_2) \cdot u_{num}$$

(10) Suppose $(2L_1, 2L_2) - (L_3, L_4) = (L_5, L_6)$

$$v_3 : (2x_1 - y_2) \cdot u_{num} < g(L_5) - g(L_6) < (2y_1 - x_2) \cdot u_{num}$$

(11) Suppose $(2L_1, 2L_2) + (L_3, L_4) = (L_6, L_5)$

$$v_3 : -(2y_1 + y_2) \cdot u_{num} < g(L_5) - g(L_6) < -(2x_1 + x_2) \cdot u_{num}$$

(12) Suppose $(2L_1, 2L_2) - (L_3, L_4) = (L_6, L_5)$

$$v_3 : (x_2 - 2y_1) \cdot u_{num} < g(L_5) - g(L_6) < (y_2 - 2x_1) \cdot u_{num}$$

(13) Suppose $(L_1, L_2) + (2L_3, 2L_4) = (L_5, L_6)$

$$v_3 : (x_1 + 2x_2) \cdot u_{num} < g(L_5) - g(L_6) < (y_1 + 2y_2) \cdot u_{num}$$

(14) Suppose $(L_1, L_2) - (2L_3, 2L_4) = (L_5, L_6)$

$$v_3 : (x_1 - 2y_2) \cdot u_{num} < g(L_5) - g(L_6) < (y_1 - 2x_2) \cdot u_{num}$$

(15) Suppose $(L_1, L_2) + (2L_3, 2L_4) = (L_6, L_5)$

$$v_3 : -(y_1 + 2y_2) \cdot u_{num} < g(L_5) - g(L_6) < -(x_1 + 2x_2) \cdot u_{num}$$

(16) Suppose $(L_1, L_2) - (2L_3, 2L_4) = (L_6, L_5)$

$$v_3 : (2x_2 - y_1) \cdot u_{num} < g(L_5) - g(L_6) < (2y_2 - x_1) \cdot u_{num}$$

Appendix B MAINTENANCE ON C

In this section, we are supplementing our strategies of pruning tuples in C. First, we show the detailed LP formulation for Lemma 5.9. Then, we present a lemma that is used to prune tuples with the help of lower bounds.

B.1 Linear programming for Lemma 5.9

We define a variable w to be the objective function. For each tuple $q \in C_p$, we build a constraint $u_{num} \cdot (p_{num} - q_{num}) \geq w$. Formally, we construct a LP as follows.

$$\begin{aligned} &\text{maximize } w \\ &\text{subject to } \forall q \in C_p,\ u_{num} \cdot (p_{num} - q_{num}) \geq w \\ &\qquad\qquad u_{num} \in \mathcal{R} \end{aligned}$$

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

B.2 Prune tuples with lower bounds

Given a tuple $p \in C$, consider the set C'_p of tuples that differ in the categorical attributes with p. For each $q \in C'_p$, we first find the node v that contains categorical pair (L_1, L_2) and 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 r lower bounds $u_{num} \cdot x_i$ in v, where $i \in [1, r]$. We divide $\mathcal{R}$ into r disjoint smaller polyhedrons $\mathcal{R}_i$ such that (1) each of them corresponds to exactly *one* lower bound $u_{num} \cdot x_i$ and (2) $\forall u_{num} \in \mathcal{R}_i, u_{num} \cdot x_i$ is the tighest lower bound, i.e., $\forall j \in [1, r]$ and $j \neq i$, $(x_i - x_j) \cdot u_{num} > 0$. For each $\mathcal{R}_i$, (1) we build a hyperplane h_i based on vector $q_{num} - p_{num} + x_i$; Then, we can check whether p can be pruned from C based on $\mathcal{R}_i$ and h_i using the following lemma.

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

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

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

This shows that $\forall u_{num} \in \mathcal{R}$, the utility of q is larger than that of p. Thus, p can be safely pruned from C. □

If there are v_i vertices in $\mathcal{R}_i$, we need $O(\mathrm{v}_i)$ time to check whether all vertices in $\mathcal{R}_i$ are in h^+_i. The total time complexity is $O(\sum_{i=1}^{r} \mathrm{v}_i)$.

Appendix C PROOFS

Proof of Lemma 3.1. Consider two tuple sets $D = \{p_1, p_2, ..., p_n\}$ and $D' = \{p'_1, p'_2, ..., p'_n\}$, where D' is a numerical rescaling of 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 $\lambda_j > 0$, $u'_{num}[j] \geq 0$. Besides, 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 $p, q \in D$ such that $f(p) > f(q)$. We have the conclusion as follows.

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

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

Proof of Lemma 3.3. The special case of ISM assumes that the tuples are the same in numerical attributes. Since the numerical attributes cannot contribute to determining the user's favorite tuple, we focus on the categorical attributes only in the following.

Consider a dataset 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 first categorical attribute, we represent its categorical values by $\mathcal{H}_1 = \{F_1, F_2, F_3, ...\}$. (2) 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., $|\mathfrak{S}_1| = |\mathfrak{S}_2| = \frac{s_2}{2}$). (3) 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 $\frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ pairs of tuples and each tuple has a different categorical value in the first attribute. There are $\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ tuples in the dataset.

During the interaction, each question consists of two tuples p and q and asks the user to tell which one s/he prefers. Since the categorical values in the first attribute of tuples are different, any algorithm could only learn the user preference on a pair of categorical values $F_i, F_j \in \mathcal{H}_i$ by asking a question and thus, prune at most one tuple from the dataset. Since there are $\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ tuples, we need to ask $\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1) - 1$ questions, i.e., $\Omega(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions to find the user's favorite tuple. □

Proof of Lemma 3.4. 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 $\frac{s_1}{2} + \frac{s_2^2}{8}(\prod_{i=3}^{d_{cat}} s_i)(\prod_{i=3}^{d_{cat}} s_i - 1)$ 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 in 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 user. If 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 tuple which have the same categorical values, as proved in [29], 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 4.3. Since $S_l > S_k$, i.e., $\sum_{c\in S_l} g(c) > \sum_{c\in S_k} g(c)$, and S_l and S_k contain all the categorical values in P_1 and P_2, respectively, the user must prefer the categorical values in P_1 to the categorical values in P_2. Thus, unique path P_2 can be removed. □

Proof of Theorem 4.6. Recall that we process the C-Tree from the bottom to the top. In the i-th level of the C-Tree, where $i \in [2, d_{cat}]$, there could be $O(\prod_{k=i}^{d_{cat}} s_k)$ sets, since there might be $O(\prod_{k=i}^{d_{cat}} s_k)$ categorical value combinations which consist of 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 Lemma 5.1. If a user prefers p to q, then $f(p) > f(q)$. Since p and q are the same in 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 h^+_{p-q}. □

Proof of Lemma 5.5. Since there exists a lower bound $u_{num} \cdot x'$ for $g(L_1) - g(L_2)$ such that $\forall u_{num} \in \mathcal{R}$, $u_{num} \cdot (x' - x) > 0$, i.e., $\forall u_{num} \in \mathcal{R}$, $g(L_1) - g(L_2) > u_{num} \cdot x' > u_{num} \cdot x$. This implies that $\forall u_{num} \in \mathcal{R}$, lower bound $u_{num} \cdot x'$ always bounds $g(L_1) - g(L_2)$ more tightly than $u_{num} \cdot x$. □

Proof of Lemma 5.9. For any $q \in C_p$, it is the same as p in all categorical attributes. The difference between $f(q)$ and $f(p)$ only depends on the numerical attributes. Consider hyperplane h_{q-p} that is based on vector $q_{num} - p_{num}$. $\forall u_{num} \in h^+_{q-p}$, $u_{num} \cdot q_{num} > u_{num} \cdot p_{num}$. Since $\mathcal{R} \subseteq \cup_{q\in C_p} h^+_{q-p}$, $\forall u_{num} \in \mathcal{R}$, $\exists q \in C_p$ such that $u_{num} \in h^+_{q-p}$ and thus, $u_{num} \cdot q_{num} > u_{num} \cdot p_{num}$. This means $\forall u_{num} \in \mathcal{R}$, there exists a tuple $q \in C_p$ such that the utility of q is larger than that of p w.r.t. u_{num}, i.e., $f(q) > f(p)$. Thus, p can be safely pruned from C. □

Proof of Lemma 5.10. Consider each polyhedron $\mathcal{R}_i$, where $i \in [1, r]$. $\mathcal{R}_i \subseteq h^-_i$ means that for any numerical utility vector $u_{num} \in \mathcal{R}_i$, $u_{num} \cdot (y_i + p_{num} - q_{num}) < 0$. Since (1) $\forall i \in [1, r]$, $\mathcal{R}_i \subseteq h^-_i$, and (2) $\cup_{i\in[1,r]}\mathcal{R}_i \subseteq \mathcal{R}$, we have $\forall u_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot y_i$, where $i \in [1, r]$, such that $u_{num} \cdot (y_i + p_{num} - q_{num}) < 0$. Then, we have

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

This shows that $\forall u_{num} \in \mathcal{R}$, the utility of p is smaller than that of q. Thus, p can be safely pruned from C. □

Proof of Theorem 5.12. In each round, we can learn the user preference between a pair of tuples and thus, prune at least one tuples from C. Since $|D| = n$, there must be only one tuple left in C after $O(n)$ rounds. □