Interactive Search with Mixed Attributes

ABSTRACT

When facing a dataset with millions of tuples, it is not desirable for a user to read the whole dataset to find his/her favorite tuple. Existing studies try to search the target tuple by interacting with the user. Specifically, they ask a user several questions, each of which consists of two tuples and asks the user to indicate which one s/he prefers. Based on the user feedback, the user's preference is learned implicitly and the best tuple w.r.t. the learnt preference is returned. However, they only consider datasets with numerical attributes (e.g., price). In practice, there are many tuples described by categorical attributes (e.g., color), where there is no trivial order on the attribute values. Thus, we study how to find the user's favorite tuple from datasets with mixed attributes (including both numerical and categorical attributes) by interacting with the user.

We study our problem progressively. Firstly, for the dataset, in which the tuples only differ in categorical attributes, we present an algorithm *Tree* that asks an asymptotically optimal number of questions. Secondly, 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 guarantee. 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.

CCS CONCEPTS

• **Information systems → Database query processing**.

KEYWORDS

user interaction; categorical attribute; data analytics

ACM Reference Format:

. 2018. Interactive Search with Mixed Attributes. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Given a dataset with millions of tuples, it is not desirable for a user to read all tuples one by one to find 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. To model her preference, the commonly-used decision making strategy is to utilize numerical weights in a *linear utility function* [11, 17, 24, 29, 31]. Precisely, it quantifies Alice's criterion of interest, and characterizes a vector, comprising one weight per attribute. Based on the utility function, each tuple is associated with a *utility* (i.e., a function score). It indicates to what extent Alice prefers the tuple, where a larger utility means that the tuple is more favored by her.

There are two representative operators for this scenario: *the top-k query* [25] and *the skyline query* [8]. The first operator is the top-k query [25]. It returns the k tuples with the highest utilities, namely the top-k tuples, w.r.t. the preference given by a user. However, users usually have difficulties in specifying their preference explicitly [23, 31]. If the user preference is not known in advance, the top-k query cannot be applied. The second operator is the skyline query [8] which, instead of asking for an explicit user preference, considers all possible user preferences and returns tuples that have the highest utilities (i.e., the 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, [31] 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 user 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 practice, tuples might be also described by categorical attributes (e.g., car brand), where the order of attribute values is uncertain. For example, it is easy to see that a \$8000 car is cheaper than a \$10000 car. 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: *Interactive Search with Mixed Attributes (ISM)*. It interacts with a user for rounds and finds the user's favorite tuple (which follows the setting of [8, 31]) from the dataset described by mixed attributes (including both numerical and categorical attributes) with as little user effort as possible. In each round, following [26, 29, 31], 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 user feedback, the user's preference is implicitly learned and the user's favorite tuple is returned. We stick to the setting in [26, 29, 31] to measure the user effort by *the number of questions asked to a user (i.e., the number of rounds during the interaction)*.

We study problem ISM progressively. We first 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. ISM has two characteristics: (1) it learns the user preference by asking questions, and (2) it considers both numerical and categorical attributes.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Woodstock '18, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Regarding the first characteristic, the questions asked to a user are in the form of pairwise comparison [14, 26, 29]. It follows a basic assumption in [23] that *users are able to tell which tuple they prefer the most from a set of tuples*. 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, 26]. 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, 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, 13, 20] emphasize that users usually pay their attention to a few attributes that are the most crucial when selecting favorite tuples.

In marketing research [19, 27], 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 1 tuple and the rest of tuples, while the latter case requires to learn the user preference between k tuples and the rest of tuples. [29] conducted experiments and verified that the latter case asks 4-10 times more questions than the former case. Its user study also showed that users are willing to answer a smaller number of questions even if they are recommended with fewer tuples. 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.

As for the second characteristic, considering both numerical and categorical attributes is critical in real applications. For instance, in the scenario of purchasing a used car (which is also applied in our user study in Section 6.4), there might be numerous candidates in the market and each one is described by some numerical attributes (e.g., price) and categorical attributes (e.g., brand). Alice might have a trade-off between price and brand. She may be willing to pay more money for a famous brand than an obscure one. This trade-off involves numerical and categorical attributes. To recommend Alice a satisfactory car, it is crucial 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 studies involving user interaction [14, 26, 31] but they are different from ours, by only considering numerical attributes. In contrast, our problem ISM considers both categorical and numerical attributes. In this sense, existing studies can be seen as a special case of problem ISM. Besides, [14] learns the ranking of all tuples and [26] searches for the user preference. Since their goals diverge from us, there could be a large amount of redundant questions asked to a user (to be discussed in Section 2), which may not be relevant to our desired answer (i.e., the user's favorite tuple).

Note that none of the existing algorithms could be adapted to solve problem ISM satisfactorily. The categorical attributes are discrete and unordered while the numerical attributes are continuous and have trivial orders. The strategies of existing algorithms which are designed for numerical attributes cannot be applied efficiently and effectively to the categorical attributes. In our experiments, the adapted existing algorithms ask many questions and spend a long execution time, which is troublesome. For example, when there are 2 categorical attributes (each containing 5 values) and 3 numerical attributes, they ask 30% more questions and takes 2 orders of magnitude longer time than our proposed algorithms.

Contributions. Our contributions are summarized as follows.

- To the best of our knowledge, we are the first to propose the problem of finding the user's favorite tuple from the dataset with mixed attributes, including both numerical and categorical attributes, by interacting with the user (problem ISM).
- We show a lower bound on the number of questions needed to determine the favorite tuple in the dataset with mixed attributes.
- We propose an 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 guarantee 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. Section 3 shows the problem definition. 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. 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.

[28, 30] studied the skyline query on the dataset with categorical attributes. The skyline query returns all the tuples, called skyline tuples, that are not *dominated* by others. A tuple p dominates another tuple q if the value of q in each attribute is not better than that of p, and strictly worse in at least one attribute. For example, consider two cars described by two attributes: p with 3000USD and 350 horse power and q with 5000USD and 300 horse power. Since p is cheaper and has a larger horse power than q, p dominates q. [28] maintained the skyline tuples when the tuples in the dataset dynamically change. [30] proposed that different users may have different preference on categorical attributes, e.g., Alice may prefer color green to color red and Tom might favor more red than green. The skyline tuples w.r.t. different user preferences could be totally different. [30] found the skyline tuples when the user preference on categorical attributes dynamically changes. However, both [30] and [28] require that the user preference is known in advance.

To overcome the deficiency, [4, 5] utilized the 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 [24] and thus, the output overwhelms users.

There are interactive queries [23, 31, 33] whose output size is controllable. [23] 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* in each question 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 [31], resulting in the difficulty of applying the algorithm of [23] in real-world applications. Based on these defects, [31] proposed algorithms *UH-Simplex* and *UH-Random* that utilize *real tuples* (selected from the dataset) for the interactive regret minimization query. To reduce the number of questions asked, [33] improved the algorithms of [31]. 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, [23, 31, 33] only considered the dataset described by numerical attributes. In contrast, our problem *ISM* focuses on both numerical and categorical attributes.

There are alternative approaches [15, 26] that aim to learn the user preference by interacting with the user. [26] 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 many redundant questions which our problem ISM is not interested in. 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 [26]. Moreover, [26] proposed to map categorical values to numerical attributes using the standard SVM convention, which is widely used in machine learning area. In this way, the existing algorithms designed for numerical attributes can be adapted to working on the datasets with both categorical and numerical attributes. However, as presented in Section 6, our experimental results show that the adapted algorithms [14, 26, 31] perform poorly, by asking several times more questions and take orders of magnitude longer time than us. The main reason is that the categorical attributes are in discrete space while the numerical attributes are in continuous space. Even if the categorical attributes are transformed to numerical attributes, their nature is unchanged and thus, the strategies designated for continuous space are not as effective as in discrete space.

[15] 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 in questions to be 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. [15] 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 *learning to rank* problem [12, 14, 18, 21], which learns the ranking of tuples by pairwise comparison. However, most of the existing methods [12, 18, 21] failed to utilize the inter-relation between tuples. For example, attribute "price" is an inter-relation between cars. Given a 3000USD car p and a 5000USD car q, p is generally considered to be better than q because of the lower price. The existing algorithms neglected this relation and directly asked a question to learn the priority of p and q instead. Thus, they require to ask much more questions than necessary in our problem ISM. [14] considered the inter-relation between tuples on the learning to rank problem. However, it was based on the numerical attributes and lost sight of the categorical attributes. Besides, it focused on deriving the order for all pairs of tuples, which requires asking much more questions than necessary in our problem ISM due to the similar reason stated for [26].

Our work is on target to return the user's favorite tuple from the dataset described by both numerical and categorical attributes by interacting with the user via real tuples. Compared with existing methods, we have the following advantages. (1) We do not require pre-knowledge of the user preference (but [28, 30] 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 [23] that utilizes fake tuples). (4) We consider both numerical attributes and categorical attributes. The existing interactive works [14, 23, 31, 33] only handle numerical attributes. They can be seen as a special case of our work when the categorical attributes are excluded from consideration. (5) We only involve a few easy questions. Firstly, existing studies like [12, 14, 18, 21, 26] 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, [12, 18, 21] 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 [33], since it requires users to give orders for the displayed tuples. In contrast, we only ask a user to pick his/her favorite tuple between two tuples in each question. These questions are much easier for the user to handle.

3 PROBLEM DEFINITION

3.1 Terminologies

The input to our problem is a set $\mathcal{D}$ of n tuples (i.e., $|\mathcal{D}| = n$). 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 convenience, let $p_{num} = (p_{num}[1], p_{num}[2], ..., p_{num}[d_{num}])$. The number of values in the i-th categorical attribute is denoted by s_i for each $i \in [1, d_{cat}]$.

Example 3.1. Consider 4 tuples, namely p_1, p_2, p_3 and p_4, in Table 1. Each tuple p has 2 categorical attributes (i.e., $p_{cat}[1]$ and $p_{cat}[2]$) and 2 numerical attributes (i.e., $p_{num}[1]$ and $p_{num}[2]$).

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

(a) Table

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

(b) $g(\cdot)$

Table 1: Dataset, utilities and function g

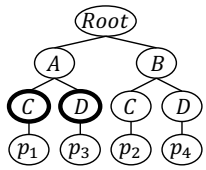


Figure 1: C-Tree

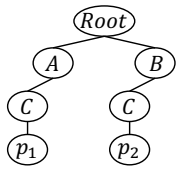


Figure 2: Pruned C-Tree

Thus, $d_{cat} = d_{num} = 2$. In this example, $s_1 = s_2 = 2$ since $p_{cat}[1]$ has 2 values, say A and B, and $p_{cat}[2]$ has 2 values, say C and D.

We model the user preference as a *linear utility function*, which is widely considered in existing work [1, 26, 29, 31]. The class of linear utility function becomes the most proliferate and effective representation since the inception of utility modeling [11, 17]. The user study conducted by [26] also verified that the linear utility function can effectively capture how real users assess the suitability of multi-attribute tuples. Formally, a linear utility function f is defined to be $f(p) = \sum_{i=1}^{d_{cat}} u_{cat}[i]h(p_{cat}[i]) + \sum_{j=1}^{d_{num}} u_{num}[j]p_{num}[j]$, where

- 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.
- We can see $u = (u_{cat}[1], ...u_{cat}[d_{cat}], u_{num}[1], ..., u_{num}[d_{num}])$ as a d-length non-negative vector. Each element represents the user's preference to an attribute. For the ease of representation, we denote it by two utility vectors in the function: u_{cat} (d_{cat}-length) and u_{num} (d_{num}-length), 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 [31, 32], we assume that $\sum_{j=1}^{d_{num}} u_{num}[j] = 1$ and call the domain of u_{num} the *numerical utility space* [1]. There are other ways to normalize the utility vectors, e.g., $\sum_{i=1}^{d} u[i] = 1$. Our normalization considers that p_{num} is fixed while function $h(\cdot)$ varies for different users.

Note that in the above form, each term related to the categorical attribute is represented in the form of $u_{cat}[i]h(p_{cat}[i])$ where $i \in [1, d_{cat}]$. Since *both* $u_{cat}[i]$ and $h(p_{cat}[i])$ are unknown at the beginning (i.e., we have *two* unknown variables), in this paper, for the ease of representation, we use a *single* function $g(\cdot)$ to denotes both $u_{cat}[i]$ and $h(p_{cat}[i])$ (i.e., we use a *single* unknown variable). Specifically, function $g : p_{cat}[i] \rightarrow \mathbb{R}_+$ is defined to be: $g(p_{cat}[i]) = u_{cat}[i]h(p_{cat}[i])$, 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 considered to be the user's favorite tuple.

Note that rescaling the numerical attributes does not change the ranking of tuples w.r.t. the utility function, i.e., the utility function maintains monotonicity when p_{num} is scaled by positive real numbers. Formally, consider two datasets $\mathcal{D} = \{p_1, p_2, ..., p_n\}$ and $\mathcal{D}' = \{p'_1, p'_2, ..., p'_n\}$. Assume that $\mathcal{D}'$ is a numerical rescaling of $\mathcal{D}$, 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 $\forall j \in [1, d_{num}]$. Given a utility function f for $\mathcal{D}$, there exists a corresponding utility function f' for $\mathcal{D}'$ such that $\forall p_k, p_l \in \mathcal{D}$, if $f(p_k) > f(p_l)$, then $f'(p'_k) > f'(p'_l)$, where $k, l \in [1, n]$.

Lemma 3.2. *Rescaling the numerical attributes of tuples in the dataset does not change the ranking of tuples.*

For the lack of space, the complete proofs of the theorems/lemmas in this paper can be found in the technical report [3]. Based on Lemma 3.2, 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 in an attribute (e.g., price), the attribute could be modified by subtracting each value from 1 to satisfy the assumption.

Example 3.3. Consider back our example. Let $u_{num} = (0.5, 0.5)$. The utility function f for a point p is expressed as follows after we adopt function $g(\cdot)$. $f(p) = g(p_{cat}[1]) + g(p_{cat}[2]) + 0.5 \cdot p_{num}[1] + 0.5 \cdot p_{num}[2]$. Suppose that function g has its values over different categorical values (i.e., A, B, C and D) as shown in Table 1.

Since $g(B) = 0.3$ and $g(C) = 0.4$, the utility of p_2 w.r.t. f is $f(p_2) = 0.3 + 0.4 + 0.5 \times 0.6 + 0.5 \times 0.9 = 1.45$. The utilities of the other tuples can be computed similarly. Tuple p_2 is the user's favorite tuple, since it has the highest utility.

3.2 Problem ISM

Our interactive framework follows [29, 31]. We interact with a user for rounds until we can determine the user's favorite tuple. Each round consists of three major components. (1) **Tuple selection**. Based on the user's answers in previous rounds, we select two tuples presented to the user and ask the user to pick the one s/he prefers. (2) **Information maintenance**. According to the user feedback, the information maintained for learning the user preference is updated. (3) **Stopping condition**. We check the stopping condition. If it is satisfied, we terminate the interaction process and return the result. Otherwise, we start a new interactive round. Formally, we are interested in the following problem.

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

Theorem 3.4. *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. Consider a dataset. (1) There are $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ tuples each of which has a different value in a categorical attribute. We show that the user feedback of any question can only help to identify one tuple that cannot be the user's favorite tuple. (2) The rest $O(n - s_1 - \prod_{i=2}^{d_{cat}} s_i^2)$ tuples differ in numerical attributes. Any algorithm needs to find the target tuple in the form of a binary tree. Thus, there would be $\Omega(\log(n - s_1 - \prod_{i=2}^{d_{cat}} s_i^2))$ questions asked. □

[1] Note that u_{cat} and u_{num} are correlated. If the sum of weights over u_{num} is assumed to 1, we cannot simply assume the same for u_{cat}.

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., $\forall p, q \in \mathcal{D}$ and $\forall j \in [1, d_{num}], p_{num}[j] = q_{num}[j]$. We present an algorithm *Tree* 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 to be asked to a user. Based on the user feedback, the tuples which cannot be the favorite tuple are pruned from the tree. When there is only one tuple left in the tree, we stop 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 determine 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 from the root to it. The categorical values of the tuple that is recorded by the leaf are stored by the nodes in the path in order.

We build $\mathcal{T}$ as follows. Initially, $\mathcal{T}$ starts with a root. Then, tuples in $\mathcal{D}$ are inserted into $\mathcal{T}$ one by one. For each tuple $p \in \mathcal{D}$, $\mathcal{T}$ is traversed once from top to bottom. Suppose that traversing is 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 $p_{cat}[i]$. 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, $\mathcal{T}$ has a root in the 0-th level. Let us insert tuple p_1 into $\mathcal{T}$. 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. For example, in Figure 2, the path, which goes from the node N_A containing A to the leaf storing p_1, is the unique path of N_A.

Assume that the nodes in the i-th level of $\mathcal{T}$ are being considered, where $i \in [1, d_{cat}]$. Denote the two nodes with the same parent by N_1 and N_2, and let P_1 and P_2 be 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) 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 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 such a set does not exist. 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 2, there are two nodes in the first level of C-Tree which have the same parent (i.e., root) and contain categorical values A and B, respectively. Each node has a unique path. The path containing A (resp. B) ends at the leaf with p_1 (resp. p_2). We build two sets $S_1 = \{A, C\}$ and $S_2 = \{B, C\}$. If a user prefers p_1 to p_2, we can learn that $S_1 \succ S_2$. 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 the C-Tree. In this way, the number of children of some nodes in the $(i-1)$-th level is 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 $\mathcal{T}$ has two sets $S_1 = \{C\}$ and $S_2 = \{D\}$ in Figure 1. If $S_1 \succ S_2$, the unique paths of the nodes in the second level which contain D are removed and thus, tuple p_3 and tuple p_4 are pruned. Figure 2 shows $\mathcal{T}$ after the pruning.

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 are 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.

Algorithm 1: The *Tree* Algorithm

Input: A tuple set $\mathcal{D}$
Output: The user's favorite tuple in $\mathcal{D}$
Build the C-Tree $\mathcal{T}$ for all tuples in $\mathcal{D}$
for $i \leftarrow d_{cat}$ *to* 1 **do**
  Build set $\mathcal{S}_i = \{S_1, S_2, ..., S_{m_i}\}$
  for $j \leftarrow 2$ *to* m_i **do**
    for $k \leftarrow 1$ *to* $j-1$ **do**
      if S_j *and* S_k *correspond to two qualified nodes* **then**
        Ask the user a question and update $\mathcal{T}$
return The only tuple left in the C-tree $\mathcal{T}$

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 a user* $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ *questions.*

Proof Sketch. Algorithm *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. □

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 consider the general case of ISM. We first discuss the preliminaries, and then present our algorithms by addressing the three components of the interactive framework. Finally, we summarize our algorithms and analyze their theoretical performance.

5.1 Preliminaries

A *multiset* L is a collection of elements, in which elements are allowed to repeat [7]. The number of times an element A repeats in a multiset is called the *multiplicity* of A and denoted by $\mathfrak{m}_L(A)$. We call $\mathfrak{m}_L(\cdot)$ the multiplicity function of L. The multiplicity can be classified to positive ($\mathfrak{m}_L(A) > 0$), negative ($\mathfrak{m}_L(A) < 0$) or zero multiplicity. The zero multiplicity means that the element is not in the multiset. In literature, a multiset is represented as $L = [A_1, A_2, ...]_{\mathfrak{m}_L(A_1), \mathfrak{m}_L(A_2), ...}$, e.g., $[A, B, C]_{-1,2,-2}$ denotes a multiset with -1 copies of A, 2 copies of B, and -2 copies of C.

A set is a multiset in which each element has multiplicity 1, i.e., a multiset $[A, B, C]_{1,1,1}$ is equal to a set $\{A, B, C\}$. The operations on sets can be extended to multisets with the help of the multiplicity. In the following, assume that L_1 and L_2 are two multisets with their multiplicity functions $\mathfrak{m}_{L_1}(\cdot)$ and $\mathfrak{m}_{L_2}(\cdot)$, respectively.

- $L = L_1 \oplus L_2$ is the multiset containing elements in L_1 or L_2, where $\forall A \in L$, $\mathfrak{m}_L(A) = \mathfrak{m}_{L_1}(A) + \mathfrak{m}_{L_2}(A)$. For example, if $L_1 = [A, B, C]_{1,-2,-1}$ and $L_2 = [B, C]_{2,2}$, then $L = [A, C]_{1,1}$. If L_1 is the same as L_2, we denote $2L_1 = L_1 \oplus L_1$ for convenience.
- The inverse of a multiset L, denoted by $\overline{L}$, contains exactly the elements of L with multiplicities that are the additive inverses of their multiplicities in L. For example, if $L = [A, B, C]_{1,-2,-1}$, then $\overline{L} = [A, B, C]_{-1,2,1}$. For simplicity, we denote $L_1 \oplus \overline{L_2}$ by $L_1 \ominus L_2$.

5.2 Information Maintenance

We define three data structures for learning the user preference. (1) A *candidate numerical utility range* $\mathcal{R}$ contains the learnt user preference on numerical attributes. (2) A *relational graph* $\mathcal{G}$ stores the learnt user preference on categorical attributes. Here, we maintain the collected information of user preference on the numerical and categorical attributes using $\mathcal{R}$ and $\mathcal{G}$, respectively, since they enjoy different characteristics. The values of numerical attributes are continuous while the values of categorical attributes are discrete. (3) A *candidate set* $\mathcal{C} \subseteq \mathcal{D}$ comprises the user's favorite tuple.

5.2.1 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* which passes through the origin with its unit normal in the same direction as p [10]. We denote the hyperplane by h_p and say h_p is constructed based on vector p. Hyperplane h_p divides $\mathbb{R}^{d_{num}}$ into two halves, called *halfspace* [10]. 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 $\mathcal{P}$ [10].

The *candidate numerical utility range* $\mathcal{R}$ is a polyhedron in the numerical utility space and contains the user's numerical utility vector. Initially, $\mathcal{R}$ is the entire numerical utility space, i.e., $\mathcal{R} = \{u_{num} \in \mathbb{R}_+^{d_{num}} | \sum_{i=1}^{d_{num}} u_{num}[i] = 1\}$. As shown in Figure 3, 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. Firstly, 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 $h_{p_{num} - q_{num}}$ based on vector $p_{num} - q_{num}$ and update $\mathcal{R}$ following Lemma 5.1. Since p and q only differ in numerical attributes, we denote the hyperplane by h_{p-q} for simplicity.

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 3 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 $\mathcal{G}$. We postpone the discussion to the next section.

5.2.2 Maintenance on $\mathcal{G}$. The *relational graph* $\mathcal{G} = (V, E)$ is a hypergraph storing the learnt user preference on categorical attributes. 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 $p, q \in \mathcal{D}$, we build a node v that contains a multiset $L = L_1 \oplus L_2$. L_1 (resp. L_2) is a multiset that stores

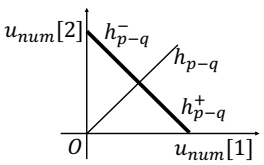


Figure 3: Space $\mathbb{R}^{d_{num}}$

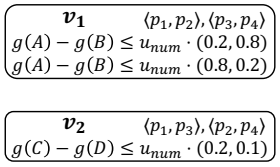


Figure 4: Bounds in nodes

all the categorical values of p (resp. q) as elements with multiplicity 1 (resp. -1). Node v also stores pair $\langle p, q\rangle$. If there are multiple pairs of tuples sharing the same multiset, they are stored in one node. Besides, node v contains several *upper bounds* and *lower bounds* which describe the user preference on the categorical values in the multiset. Suppose a user prefers p to q, i.e., $f(p) > f(q)$. We can obtain an inequality $\sum_{A\in L} \mathfrak{m}_L(A)g(A) > u_{num} \cdot (q_{num} - p_{num})$. With a slight abuse of notations, we denote the inequality by $g(L) > u_{num} \cdot (q_{num} - p_{num})$ for simplicity and call $u_{num} \cdot (q_{num} - p_{num})$ the lower bound of $g(L)$. Similarly, if a user prefers q to p, we can derive an upper bound $u_{num} \cdot (q_{num} - p_{num})$ for $g(L)$.

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 multiset $L = [A, C, B, D]_{1,1,-1,-1}$ and pair $\langle p_1, p_4\rangle$. All nodes in the relational graph of Table 1 are shown in Figure 5. If a user prefers p_1 to p_4, we could obtain a lower bound $u_{num} \cdot (p_{4\ num} - p_{1\ num})$ for $g(L)$, i.e., $g(A) + g(C) - g(B) - g(D)$.

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

Definition 5.4 (Untight Bound). Bound $u_{num} \cdot x$ is an untight lower bound for $g(L)$ if $\forall u_{num} \in \mathcal{R}$, there is a lower bound $u_{num} \cdot x'$ for $g(L)$ 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 $g(L)$ 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 identifying untight bounds.

Lemma 5.5. *Bound $u_{num} \cdot x$ is an untight lower bound for $g(L)$ if there exists a lower bound $u_{num} \cdot x'$ for $g(L)$ 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 $g(L)$ 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 $O(\mathrm{v})$ time to check whether all vertices satisfy $u_v \cdot (x' - x) > 0$.

Hyperedge. Consider any three nodes $v_1, v_2, v_3 \in V$ containing multiset L_1, L_2 and L_3, respectively. If any two of the multisets could deduce the third one based on one of the following derivation, there

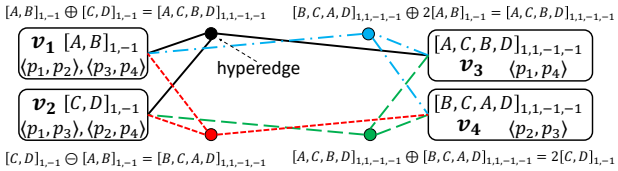


Figure 5: Relational graph

exists a hyperedge connecting v_1, v_2 and v_3, denoted by $e(v_1, v_2, v_3)$. Without loss of generality, suppose L_1 and L_2 could deduce L_3. Let $\odot \in \{\oplus, \ominus\}$ and $L_1 \oplus \overline{L_2} = L_1 \ominus L_2$. We have the following derivation.

$$L_1 \odot L_2 = L_3 \text{ or } 2L_3 \text{ or } \overline{L_3} \text{ or } 2\overline{L_3} \quad (1)$$

$$2L_1 \odot L_2 = L_3 \text{ or } \overline{L_3} \quad (2)$$

$$L_1 \odot 2L_2 = L_3 \text{ or } \overline{L_3} \quad (3)$$

Example 5.6. Consider v_1 (with $[A, B]_{1,-1}$), v_2 (with $[C, D]_{1,-1}$)) and v_3 (with $([A, C, B, D]_{1,1,-1,-1})$ in Figure 5. There exists a hyperedge $e(v_1, v_2, v_3)$ since $[A, B]_{1,-1} + [C, D]_{1,-1} = [A, C, B, D]_{1,1,-1,-1}$. The hyperedges are shown in small circles.

Note that not every three nodes are connected by a hyperedge. It is possible that any two of the multisets cannot derive the third one. If nodes $v_1, v_2, v_3 \in V$ with multisets L_1, L_2 and L_3, respectively, are connected by a hyperedge, we can use the bounds in any two nodes to generate new bounds for the third one, following how the multisets in the two nodes deduce the multiset in the third node. Suppose that $L_1 \oplus L_2 = L_3$ and v_1 and v_2 have bounds as follows.

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

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

Then, we could generate new bounds for v_3.

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

The generated bounds might be untight. We only insert into nodes the tight generated bounds. If L_1 and L_2 deduce L_3 in other ways, we can also generate bounds for v_3 similarly. The generation of the other deduction can be found in the technical report [3].

Update. In each interactive round, we present two tuples $p, q \in \mathcal{D}$ and ask a user to pick the one s/he prefers. Based on the user feedback, $\mathcal{G}$ is updated. The update can be divided into two cases: (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.

Consider the first case. As mentioned in Section 5.2.1, $\mathcal{R}$ will be updated to be $\mathcal{R} \cap h^+_{p-q}$ or $\mathcal{R} \cap h^+_{q-p}$. Based on the updated $\mathcal{R}$, some bounds become untight and they are removed from each node.

Example 5.7. Figure 4 shows two upper bounds in v_1. Suppose that $\mathcal{R}$ 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 multiset L and tuple pair $\langle p, q\rangle$. We first add $u_{num} \cdot x$ into v and then update v through the following

two steps. Firstly, all the upper bounds in v are used to update $\mathcal{R}$. Suppose there is an upper bound $g(L) < u_{num} \cdot y$ in v. Since $u_{num} \cdot x < g(L) < u_{num} \cdot y$, 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 are checked and the untight bounds based on the updated $\mathcal{R}$ are removed.

Example 5.8. Figure 4 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.

The updates on many nodes may lead to a long execution time. 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. Our update is 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 ($i = 2, 3, 4, ...$). The update terminates when all the nodes in the α levels are updated, where $\alpha \geq 1$ is a given parameter. If α is large, multiple nodes updated could cause a long time. If α is small, we may fail to collect sufficient information from the user feedback. The setting of parameter α will be discussed experimentally in Section 6.1.

5.2.3 Maintenance on C. The *candidate set* $C \subseteq \mathcal{D}$ contains the user's favorite tuple. Initially, C is set to $\mathcal{D}$. During the interaction, based on $\mathcal{R}$ and $\mathcal{G}$, we determine and prune from C the tuples that cannot be the user's favorite tuple. Specifically, for each tuple, we compare it with the other tuples by two pruning strategies.

Pruning Strategy 1. 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.

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].

Pruning Strategy 2. 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 which passes through the origin with its normal $p_{num} - q_{num} + y_i$. Then, we can check whether p can be pruned from C based on $\mathcal{R}_i$ and h_i using the lemma below.

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 4, 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 from C.

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, p is the favorite tuple. We stop the interaction and return p to the user.

5.4 Tuple Selection

We present two approaches of selecting tuples during the interaction, which preform well empirically and have provable guarantee on the number of questions asked. In each round, two tuples selected from C are displayed to a user. The restriction of tuple selection from C ensures that $|C|$ 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. Based on the categorical values of the selected tuples, the tuple selection can be classified into two types.

- **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 pairs in C. The reason of (2) is that if there are many different categorical values, it is hard to learn the difference in preference is caused by which pairs of categorical values and thus, makes it more difficult to analyze the user preference on the categorical values.

The First Approach: Combination. Our first approach treats numerical and categorical attributes equally. 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 5.2).

The Second Approach: Separation. Our second approach separates the two types of tuple selection. We first use the tuple selection of type 1 until all tuples in C differ in at least one categorical attribute. Since each selected pair of tuples only differ in the numerical attributes, we only need to build hyperplanes and update $\mathcal{R}$ and C (see Section 5.2). 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 5.2). The second approach gives priority to the numerical attributes so that $\mathcal{R}$ could

Algorithm 2: The *Combination* Algorithm

Input: A tuple set $\mathcal{D}$
Output: The user's favorite tuple in $\mathcal{D}$

$\mathcal{R} \leftarrow \{u_{num} \in \mathbb{R}_+^{d_{num}} | \sum_{i=1}^{d_{num}} u_{num}[i] = 1\}$, $C \leftarrow \mathcal{D}$
Initialize relational graph $\mathcal{G}$, $i \leftarrow 1$
while $|C| > 1$ **do**
  if $i = 1, 3, 5, \ldots$ **then**
    Display two tuples in type 1
  else if $i = 2, 4, 6, \ldots$ **then**
    Display two tuples in type 2
  Update $\mathcal{R}$, $\mathcal{G}$ and C based on the user feedback, $i \leftarrow i + 1$
return The only tuple left in C

Algorithm 3: The *Separation* Algorithm

Input: A tuple set $\mathcal{D}$
Output: The user's favorite tuple in $\mathcal{D}$

$\mathcal{R} \leftarrow \{u_{num} \in \mathbb{R}_+^{d_{num}} | \sum_{i=1}^{d_{num}} u_{num}[i] = 1\}$, $C \leftarrow \mathcal{D}$
Initialize relational graph $\mathcal{G}$
while $\exists p, q \in C, \forall i \in [1, d_{cat}], p_{cat}[i] = q_{cat}[i]$ **do**
  Display two tuples in type 1
  Update $\mathcal{R}$ and C based on the user feedback
while $|C| > 1$ **do**
  Display two tuples in type 2
  Update $\mathcal{R}$, $\mathcal{G}$ and C based on the user feedback
return The only tuple left in C

be quickly narrowed to a small range in the early stage. In this way, the user preference on numerical attributes can be learned first, and moreover, a smaller $\mathcal{R}$ would help to reduce the computation on $\mathcal{G}$, since whether a bound is untight is closely related to $\mathcal{R}$.

5.5 Summary and Analysis

We call as *Separation* and *Combination* the algorithms with the "separation" and "combination" tuple selection approach, respectively. Both algorithms display tuples selected from C 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, h_{p-q} will be built and $\mathcal{R}$ becomes one of the smaller polyhedrons, i.e., $\mathcal{R} \cap h^+_{p-q}$ or $\mathcal{R} \cap h^-_{p-q}$. If p and q differ in categorical attributes, there will be bounds added to $\mathcal{G}$ and thus, the bounds on some categorical values will be tighter. C will also be updated accordingly. The pseudocode of *Combination* and *Separation* are presented in Algorithms 2 and 3, respectively.

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

6 EXPERIMENT

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

Datasets. We conducted experiments on synthetic and real datasets which are commonly used in existing studies [30, 31]. The synthetic datasets contain both numerical and categorical attributes. The numerical attributes are anti-correlated developed for the skyline query [8]. The categorical attributes are generated according to a Zipfian distribution [16, 30], where the Zipfian parameter is set to 1. The real datasets are *Nursery* and *Car* [30, 31]. *Nursery* contains 12,960 tuples with 8 attributes. Following [30], 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 mapped 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 uncertain whether a family with one child is "better" than a family with two children. *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 and gearbox). For all datasets, each numerical attribute was normalized to $(0, 1]$. To be consistent with the experimental setting in [9, 31], we preprocessed all the datasets to contain skyline tuples. Specifically, tuple $p \in \mathcal{D}$ is retained if $\nexists q \in \mathcal{D}$ 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* [14], *Adaptive* [26], *UH-Random* [31] and *UH-Simplex* [31]. Note that the existing algorithms are designed for numerical attributes. We transformed each categorical value to a numerical attribute using the standard SVM convention, which is commonly used in preference queries [26] and machine learning area [22] for handling categorical attributes. Specifically, 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. 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) *UH-Simplex* and *UH-Random* find a tuple such that a criterion, called *regret ratio* (as introduced in Section 2), 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. (3) *Adaptive* learns the user preference by interacting with the user. According to the experimental results in [26], 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. Note that *Adaptive* does not adopt any tuple pruning strategy. For comparison, we also create another version, named *Adapt-Prune*, which includes the tuple pruning strategy proposed by [31].

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 c_{val} of each categorical attribute (i.e., the number of values in each categorical attribute). Following [30, 31], 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 conduct when the default parameters are set to be large.)

Performance Measurement. We evaluated the performance of each algorithm by 3 measurements: (1) *Execution time*; (2) *Candidate size*: We reported the percentage of remaining tuples in C in each interactive round; (3) *The number of questions asked*: Each algorithm was conducted 10 times with different generated utility functions: (1) for u_{cat} and u_{num}, we randomly generate two vectors; (2) for $h(\cdot)$, we randomly assign a value to each categorical value. It is worth mentioning that although we focus on determining function $g(\cdot)$, in the experiments we setup the utility function based on u_{num} and $h(\cdot)$ described above, since they uniquely determine $g(\cdot)$. We reported the average performance of algorithms.

6.1 Performance Study of Our Algorithms

Recall that, in Section 5.2.2, we introduce a parameter α to control the number of nodes to be updated in $\mathcal{G}$. In this section, we explore how α affects the updating process of $\mathcal{G}$. Figure 6 shows the performance of our algorithms by varying parameter α on the synthetic 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 to be updated in $\mathcal{G}$. However, the number of questions asked only decreases when α increases from 1 to 2. This means that more nodes updated in $\mathcal{G}$ 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 rest experiments.

6.2 Results on Synthetic Datasets

Progress analysis. We demonstrate how the algorithms progress during the interaction process. Varying the number of questions we can ask, we reported the execution time and the candidate size. Our algorithms *Combination* and *Separation* were compared against existing algorithms on several synthetic datasets.

The first dataset contains tuples which only differ in categorical attributes. We set $c_{val} = 10$ and the other parameters by default. For completeness, our algorithm *Tree* was also involved. Figure 7 shows the results. Except *UH-Random*, the other algorithms only take within 1 second to ask 10 questions. *UH-Random* takes more than 2.5 seconds, since its pruning strategy is time-consuming. Our algorithm *Tree* only takes 10^{-3} seconds to ask 10 questions, which performs the best among all algorithms. Besides, our 3 algorithms reduce the candidate size the most effectively. In particular, *Tree* is the only algorithm that reduces the candidate size by 50% after asking 5 questions. This justifies the superiority of *Tree*.

The second dataset contains tuples which differ in both numerical and categorical attributes. All the parameters are set by default, shown in Figure 8. *UH-Simplex* does not reduce the candidate size effectively and runs the slowest among all algorithms. Its tuple selection strategy is developed based on *convex hull*, which is costly for multiple attributes. It takes up to 410 seconds to ask 10 questions. In comparison, the second slowest existing algorithm takes 90 seconds. Since *UH-Simplex* is time-consuming, we exclude it for comparison in the rest experiments. *UH-Random*, *Adaptive* and *Adaptive-Prune* are also slow. They take tens of seconds. *UH-Random* takes much time for pruning tuples after asking each question and *Adaptive* maintains a costly data structure. *Adaptive-Prune* inherits both time-intensive issues. Besides, *Adaptive* fails to provide any reduction on the candidate size since it only focuses on learning the user preference. *Adaptive-Prune* and *UH-Random* reduce the candidate size ineffectively. They utilize the same existing prune strategy, which considers all the attributes equally and thus, neglects the different properties of categorical and numerical attributes as discussed in Section 1. Our algorithms reduce the candidate size effectively and efficiently. They prune more than 90% of tuples in C 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 take slightly more time, their execution times 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.

The third dataset is the same as the second one except $c_{val} = 10$. As shown in Figure 9, none of the existing algorithm, except *Adaptive*, finish the computation. They either cost more than 10^5 seconds or run out of memory. Thus, we set $c_{val} = 4$ by default.

Scalability. We studied the scalability of algorithms by varying d_{num}, d_{cat}, c_{val} and n, and reporting the number of questions asked and the execution time. Unless stated explicitly, the parameters were set by default (i.e., $d_{num} = 2$, $d_{cat} = 3$, $c_{val} = 4$ and $n = 100k$).
Varying d_{num}. In Figure 10, we varied the number of numerical attributes d_{num} from 2 to 5. The results show that our algorithm asks $6\% - 88\%$ fewer questions and runs 1-3 orders of magnitude shorter than the existing algorithms. This verifies that our algorithms preform well when we scale the numerical attributes.
Varying d_{cat}. In Figure 11, we varied the number of categorical attributes 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 seconds). Our algorithms scale the best. When $d_{cat} = 3$, they ask $15\% - 92\%$ fewer questions and run 2-4 orders of magnitude faster than the existing algorithms. This is because the existing algorithms handle a large amount of attributes (since the standard SVM maps each categorical value to an attribute), while our algorithms utilize the relational graph to handle the categorical values.
Varying c_{val}. In Figure 12, we varied the cardinality of categorical attributes c_{val} from 4 to 28. Our algorithms ask the fewest questions within the shortest time and they are not very sensitive to the increase of c_{val} in both execution time and the number of question asked. For example, when $c_{val} = 16$, *Adaptive* asks 6 times more questions and takes 38 times longer time than our algorithms. Note that we only show the performance of *UH-Random*, *Adaptive-Prune* and *ActiveRanking* when $c_{val} = 4$, since they take too much time (more than 10^6 seconds) when the cardinality increases. Their inefficiency is caused by the excessive attribute issue as stated previously.
Varying n. In Figure 13, we varied the data size n from $10k$ to $1M$. *Combination* and *Separation* scale the best. Their execution times are less than 2 seconds even if $n = 1M$, while the fastest existing algorithm takes 94 seconds. Our algorithms also ask at least 6 fewer questions than the existing algorithms for all n. For example, when $n = 500k$, they ask around 27 questions. In comparison, *ActiveRanking* asks 305 questions and *Adaptive* asks 184 questions.

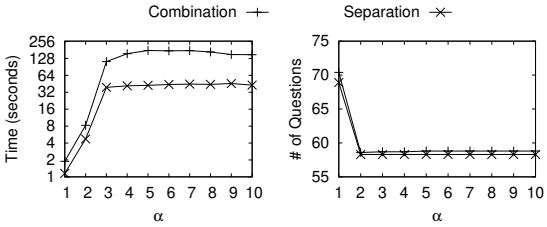


Figure 6: Update strategy

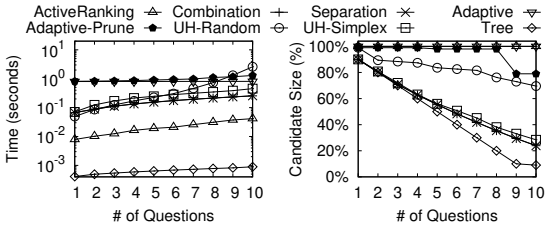


Figure 7: Synthetic (categorical only) ($c_{val} = 10$)

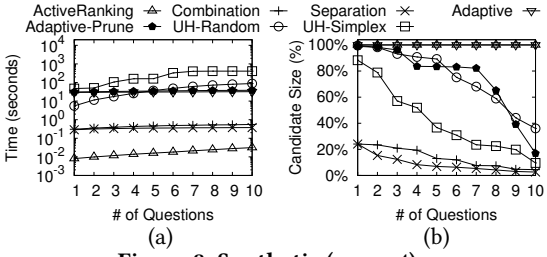


Figure 8: Synthetic ($c_{val} = 4$)

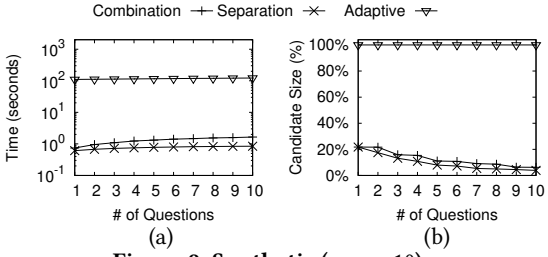


Figure 9: Synthetic ($c_{val} = 10$)

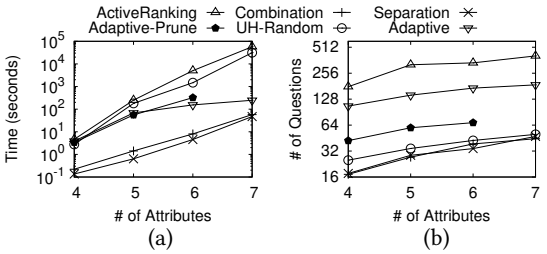


Figure 10: Vary d_{num}

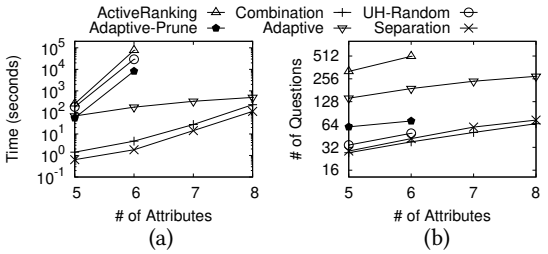


Figure 11: Vary d_{cat}

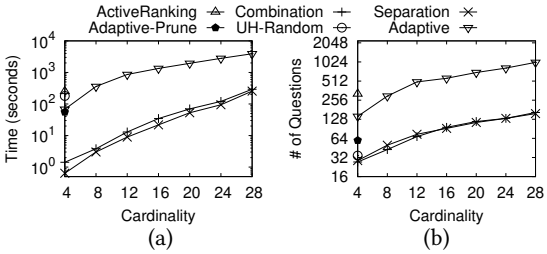


Figure 12: Vary c_{val}

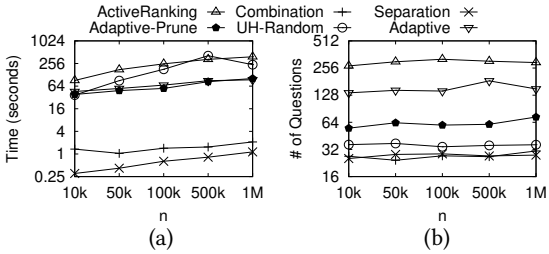


Figure 13: Vary n

6.3 Results on Real Datasets

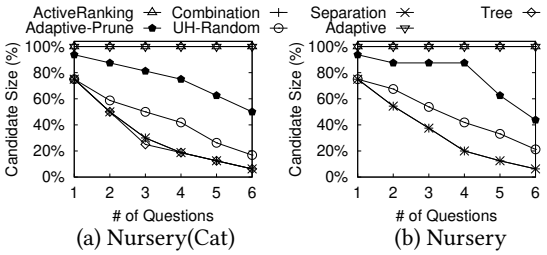
We explored how the algorithms progress during the interaction process on real datasets. We varied the number of questions we can ask and reported the execution time and the candidate size.

Nursery. We run algorithms on *Nursery* in two versions. The first one was exactly *Nursery*. The second one, namely *Nursery(Cat)*, only maintained the categorical attributes in *Nursery*, so that algorithm *Tree* can be included for comparison. The results show that all algorithms only take within 0.05 seconds to ask 6 questions for both versions, since 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 14 (a) and (b) show the candidate size of each algorithm on *Nursery(Cat)* and *Nursery*, respectively. Our algorithms are the most effective. For both datasets, after asking 6 questions, our algorithms only contain 6.25% tuples in C, while the best existing algorithm contains 17% tuples in C.

Car. Figure 15 shows the results on dataset *Car*. All algorithms only cost a few seconds. Note that *ActiveRanking* takes less time to process 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 given that 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 45% of tuples in C. Besides, since *ActiveRanking* asks more questions than our algorithm, its total execution time is longer.

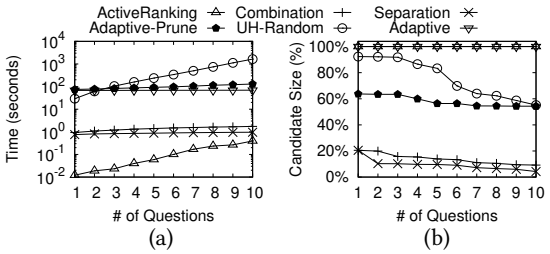
6.4 Interaction Study

Confidence Study. We studied the situation that a user might be unable to answer some questions. 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 $f(p) > f(q)$.



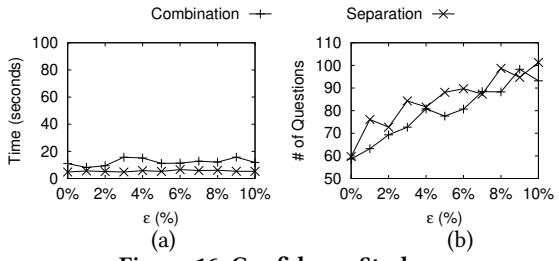
(a) Nursery(Cat) (b) Nursery

Figure 14: Nursery



(a) (b)

Figure 15: Car



(a) (b)

Figure 16: Confidence Study

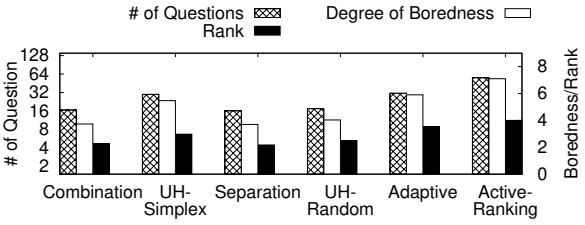


Figure 17: User Study

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 were adapted so that if the question cannot be answered, they will ask another question based on their tuple selection strategy. Figure 16 shows the performance of our algorithms by varying ϵ. The execution time remains almost unchanged and the number of questions asked slightly increases. This shows that unanswered questions affect little to our algorithms.

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 15, existing algorithms ask many questions in a long time. The user study cannot be conducted on *Car* directly. Thus, following [29, 31], 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* following [29, 31] (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 [31] 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. A higher score indicates that the participant feels more bored when s/he sees the returned used car after answering several questions. (3) *Rank*. Since participants sometimes gave the same score for different algorithms, we asked each participant to give an explicit ranking of all algorithms. Figure 17 shows the results. Our algorithms ask about $10\% - 70\%$ fewer questions than the existing algorithms. They also are the least boring and are ranked 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. This shows that our algorithms can work well in real-world scenarios.

6.5 Summary

The experiments showed the superiority of our algorithms over the best-known existing ones. (1) We are both effective and efficient. Our algorithms achieve orders of improvement on execution time and ask much fewer questions (e.g., they ask 30% fewer questions and run one order of magnitude faster than the existing algorithms on the dataset with $d_{num} = 2$ and $d_{cat} = 2$). (2) Our algorithms scale well in terms of d_{cat}, d_{num}, c_{val} and n (e.g., *Separation* asks 59 questions in 15 seconds on the dataset with $d_{num} = 3$ and $d_{cat} = 4$, while *Adaptive* asks 238 questions in 329 seconds). (3) Our algorithms are robust (e.g., in our user study, even if users can provide inconsistent feedbacks during the interaction, our algorithms consistently outperform existing ones by achieving the least degree of boredness). (4) The pruning strategies are useful (e.g., we reduce the candidate size to 13% after asking 5 questions on dataset *Car*, while the existing algorithms can only reduce to 56%). In summary, *Tree* asks the least number of questions in the shortest time for the special case of ISM, while *Combination* and *Separation* run the fastest and ask the fewest questions for the general case of ISM.

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. As for future work, we are looking forward to enhancing the robustness of the algorithms, i.e., considering the case that users may make mistakes when answering questions.

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. 2022. https://repository.up.ac.za/bitstream/handle/2263/27411/04chapter4.pdf?sequence=5&isAllowed=y

[3] Anonymous Author(s). 2022. *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] Wayne D Blizard. 1990. Negative membership. *Notre Dame Journal of formal logic* 31, 3 (1990), 346–368.

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

[9] 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.

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

[11] James Dyer and Rakesh Sarin. 1979. Measurable Multiattribute Value Functions. *Operations Research* 27 (08 1979), 810–822.

[12] 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.

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

[14] 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.

[15] 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.

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

[17] Ralph Keeney, Howard Raiffa, and David Rajala. 1979. Decisions with Multiple Objectives: Preferences and Value Trade-Offs. *Systems, Man and Cybernetics, IEEE Transactions on* 9 (08 1979), 403 – 403.

[18] 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.

[19] Alchemer LLC. 2022. https://www.alchemer.com/resources/blog/how-many-survey-questions/

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] QuestionPro. 2022. https://www.questionpro.com/blog/optimal-number-of-survey-questions/

[28] 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.

[29] 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 pages.

[30] 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.

[31] 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.

[32] 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.

[33] 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 show how to utilize the linear programming algorithm (LP) to identify untight lower bounds. We define a variable w to be the objective function. Assume that there are r other lower bounds $u_{num} \cdot x_i$ for $g(L)$, 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 of Lemma 5.5 (i.e., the sufficient condition for identifying the untight lower bound). 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 [10]. Specifically, suppose $\mathcal{R}$ has v vertices u_{vi}, where $i \in [1, \mathrm{v}]$. Each u_{num} in $\mathcal{R}$ can be represented by the vertices of $\mathcal{R}$, i.e., $u_{num} = \sum_{i=1}^{\mathrm{v}} t_i u_{vi}$, where t_i is a positive real number such that $\sum_{i=1}^{\mathrm{v}} t_i = 1$.

We compare 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 $i \in [1, \mathrm{v}]$, satisfy $u_{vi} \cdot (x' - x) > 0$, . If yes, $\forall u_{num} \in \mathcal{R}$, we have

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

A.1.2 Upper Bounds. In the following, we first show the 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 $g(L)$ if $\forall u_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot y'$ for $g(L)$ such that $u_{num} \cdot (y' - y) < 0$.

Intuitively, if $u_{num} \cdot y$ is untight, $\forall u_{num} \in \mathcal{R}$, there exists an upper bound $u_{num} \cdot y'$ which bounds $g(L)$ more tightly than $u_{num} \cdot y$. We utilize the LP algorithm to check whether an upper bound $u_{num} \cdot y$ is untight. Specifically, we define a variable w to be the objective function. Assume that there are r other upper bounds $u_{num} \cdot y_i$ for $g(L)$, 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 $g(L)$ if there exists an upper bound $u_{num} \cdot y'$ for $g(L)$ 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 $g(L)$ such that $\forall u_{num} \in \mathcal{R}$, $u_{num} \cdot (y' - y) < 0$, i.e., $\forall u_{num} \in \mathcal{R}$, $g(L) < u_{num} \cdot y' < u_{num} \cdot y$. This implies that $\forall u_{num} \in \mathcal{R}$, upper bound $u_{num} \cdot y'$ always bounds $g(L)$ more tightly than $u_{num} \cdot y$. □

Suppose $\mathcal{R}$ has v vertices u_{vi}, where $i \in [1, \mathrm{v}]$. Since $\mathcal{R}$ is a polyhedron, each u_{num} in $\mathcal{R}$ can be represented as $u_{num} = \sum_{i=1}^{\mathrm{v}} t_i u_{vi}$, where t_i is a positive real number such that $\sum_{i=1}^{\mathrm{v}} t_i = 1$ [10].

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 $i \in [1, \mathrm{v}]$, satisfy $u_{vi} \cdot (y' - y) < 0$, . If yes, $\forall u_{num} \in \mathcal{R}$, we have

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

A.2 Update

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(15) Suppose $L_1 \oplus 2L2 = \overline{L_3}$.

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

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

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

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.

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

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 tuple pair $\langle q, p \rangle$. Then, we check whether p can be pruned because of q based on the lower bounds in v. Specifically, suppose there are 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 $\mathcal{R}_i$, we build a hyperplane h_i which passes through the origin with its unit norm in the same direction as vector $q_{num} - p_{num} + x_i$; 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 $O(v_i)$ time to check whether all vertices in $\mathcal{R}_i$ are in h^+_i. The total time complexity is $O(\sum_{i=1}^{r} v_i)$.

Appendix C PROOFS

Proof of Lemma 3.2. Consider two tuple sets $\mathcal{D} = \{p_1, p_2, ..., p_n\}$ and $\mathcal{D}' = \{p'_1, p'_2, ..., p'_n\}$, where $\mathcal{D}'$ is a numerical rescaling of $\mathcal{D}$, i.e., (1) for the numerical attributes, there exist positive real numbers $\lambda_1, \lambda_2, ..., \lambda_{d_{num}}$ such that $p'_{k\ num}[j] = \lambda_j p_{k\ num}[j], \forall k \in [1, n]$ and $\forall j \in [1, d_{num}]$; and (2) for the categorical attributes, $p'_{k\ cat}[i] = p_{k\ cat}[i]$, $\forall k \in [1, n]$ and $\forall i \in [1, d_{cat}]$.

Denote $U = \sum_{j=1}^{d_{num}} \frac{u_{num}[j]}{\lambda_j}$. For any utility function f, there exists a corresponding utility function f' such that (1) $\forall j \in [1, d_{num}]$, $u'_{num}[j] = \frac{u_{num}[j]}{U\lambda_j}$; and (2) $\forall i \in [1, d_{cat}]$, $u'_{cat}[i] = \frac{u_{cat}[i]}{U}$. Since $u_{num}[j] \geq 0$ and $\lambda_j > 0$, $u'_{num}[j] \geq 0$. 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 $p, q \in \mathcal{D}$ such that $f(p) > f(q)$. We have the conclusion as follows.

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

Therefore, for any utility function f, there exists a corresponding utility function f' such that $\forall p_k, p_l \in \mathcal{D}$, 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 3.4. 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 [31], 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 (1) $S_l > S_k$, i.e., $\sum_{c \in S_l} g(c) > \sum_{c \in S_k} g(c)$, and (2) 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, i.e., $\sum_{c \in P_1} g(c) > \sum_{c \in P_2} g(c)$. 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 Corollary 4.7. As shown in Theorem 3.4, if all the tuples differ in categorical attributes, $n = O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$. The lower bound of the number of questions asked is $\Omega(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$. Since algorithm *Tree* determines the user's favorite point by asking $O(s_1 + \prod_{i=2}^{d_{cat}} s_i^2)$ questions, it is asymptotically optimal in terms of the number of questions asked. □

Proof of Lemma 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 $g(L)$ such that $\forall u_{num} \in \mathcal{R}, u_{num} \cdot (x' - x) > 0$, i.e., $\forall u_{num} \in \mathcal{R}$, $g(L) > u_{num} \cdot x' > u_{num} \cdot x$. This implies that $\forall u_{num} \in \mathcal{R}$, lower bound $u_{num} \cdot x'$ always bounds $g(L)$ 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 $|\mathcal{D}| = n$, there must be only one tuple left in C after $O(n)$ rounds. □