Ranking Desired Tuples by Database Exploration

Xuedi Qin† Chengliang Chai† Yuyu Luo† Tianyu Zhao†
Nan Tang‡ Guoliang Li† Jianhua Feng† Xiang Yu† Mourad Ouzzani‡
†Department of Computer Science, Tsinghua University ‡Qatar Computing Research Institute, HBKU
{qxd17@mails., chaicl15@mails., luoyy18@mails., zhaoty17@mails., liguoliang@, fengjh@, x-yu17@mails.}tsinghua.edu.cn,
{ntang, mouzzani}@hbku.edu.qa

***Abstract*—Database exploration – the problem of finding and ranking desired tuples – is important for data discovery and analysis. Precisely specifying SQL queries is not always feasible in practice, such as "*finding and ranking off-road cars based on a combination of Price, Make, Model, Age, and Mileage.*" – not only due to the query complexity (*e.g.,* which may have many *if-then-else*, *and*, *or* and *not* logic), but also because the user typically does not have the knowledge of all data instances.**

We propose DEXPLORER, a system for interactive database exploration. DEXPLORER offers a simple and user-friendly interface which allows to: (1) confirm whether a tuple is desired or not, and (2) decide whether a tuple is more preferred than another. Behind the scenes, we jointly use multiple ML models to learn from the above two types of user feedback. Moreover, in order to effectively involve users, we carefully select the set of tuples for which we need to solicit feedback. Therefore, we devise *question selection* algorithms that consider not only the estimated benefit of each tuple, but also the possible partial orders between any two suggested tuples. Experiments on real-world datasets show that DEXPLORER is more effective than existing approaches.

I. INTRODUCTION

We study the problem of *interactive database exploration* in scenarios where a user is interested in finding and ranking some tuples from a given dataset but where it is usually hard to precisely specify the intent, This is common in practice, for both non-experts who cannot write SQL queries and scripts, and experts who are not familiar with the data. Further compounding the problem, the query intent may be complex, *e.g.,* finding tuples that satisfy a combination of many *if-then-else*, *and*, *or* and *not* conditions, and are ranked by a weighted function over multiple attributes.

Existing Works. We categorize existing works as follows.

Keyword search [15], [20], [6], [25], [23] retrieves desired tuples based on keywords. However, such queries are typically under-specified and ambiguous. Even with iterative refinement, it is impossible to formulate a keyword query that can be unambiguously translated to a (complex) query.

Query-by-example either infers an SQL query Q [22], [8] or learns a ML model M [17] over a database D using user provided examples E. Based on E, they try to infer Q or M such that $Q(D)$ or $M(D)$ is good *w.r.t.* E, where the quantification of "goodness" varies in different applications.

Preference between tuples approaches [21], [27] infer the ranking of tuples based on user provided partial orders of tuple pairs; [27] proposes an approach to find the top-1 result while [21] ranks all tuples.

Limitations of Existing Works. Let *DE* denote the general problem to discover and rank tuples, *DE-Decision* the special case of DE where users only want specific tuples (without ranking), and *DE-Ranking* another special case where users want the ranking over all tuples. Table I summarizes our comparison of different methods for data exploration.

(I) [*Supported Problems*.] Most approaches focus on *either* DE-Decision *or* DE-Ranking. Besides DEXPLORER, only SQLSynthesizer [28] can discover an SQL query with ranking, but it can only support simple (hierarchical) ranking functions such as ORDER BY attribute *year* first and then ORDER by attribute *kilometer*. No existing approach can support a combination of selection conditions and a more natural (but maybe more complicated) ranking function such as $(-0.018 \times price + 0.982 \times powerPS)$.

(II) [*Supported Special Cases*.] Even for approaches that support either DE-Decision or DE-Ranking, some of them can only handle special cases, as described in the column "Supported Special Cases" in Table I.

(III) [*Interactive vs. Static*.] Some approaches are interactive while others support static input. Note that some back-end inference algorithms can easily support user interactions, such as ML models, while it might be hard with others, such as keyword-based or SQL query inference.

The main difference between this work and those existing works is how to *holistically* solve the general DE problem, instead of treating DE-Decision and DE-Ranking separately, by minimizing the interactions with the user.

Our Proposal. We propose DEXPLORER for with the following main features. [***Interactivity***.] DEXPLORER is designed as an interactive system, because providing a set of representative, unbiased, and sufficient samples in one shot is hard. [***Usability***.] It offers an easy-to-use interface for any user by mean of a list of tuples with two simple operations "click" and "drag", for the user to provide *true/false* tuple labels and partial orders between tuples, respectively. [***Capability***.] In order to infer (possibly) complicated query intent, we propose to jointly train several ML models instead of guessing SQL queries, along the same line of [17], because tightly specifying conditions in SQL queries in database exploration might be

[1]Chengliang Chai and Guoliang Li are the corresponding authors.

Project	Input	DE-Decision	DE-Ranking	DE	Supported Special Cases	Interactive
DISCOVER [16]	(1)	✓			only find a small number of interesting tuples	
SQLSynthesizer [28]	(2)			✓	only simple ranking function	
SQUID [8]	(2)	✓			does not support the "or" predicate	
AIDE [5]	(2)	✓				✓
Huang et al. [17]	(2)	✓			does not support the "or" predicate	✓
Chaudhuri et al. [2]	(3)		✓		only rank categorical attributes	
Qian et al. [21]	(3)		✓			✓
Xie et al. [27]	(3)		✓		only find top-1 result	✓
DEXPLORER	(1,2,3)	✓	✓	✓		✓

TABLE I
COMPARISON WITH STATE-OF-THE-ART (USER INPUT 1: KEYWORD, 2: EXAMPLES, 3: PARTIAL ORDERS)

hard. More specifically, we use random forests [19] to infer the desired tuples, and a hybrid ranking model combining LambdaMART [26] and Ranking SVM to rank tuples. [***Effectiveness***.] We devise novel question selection algorithms that jointly estimate the benefit of soliciting feedback from both tuple labeling and partial order labeling.

Contributions. Our contributions are summarized below.

(1) We formally define the problem of interactive database exploration. (Section II)

(2) We presents DEXPLORER that iteratively trains ML models and predicts on-the-fly results for interactive data exploration with the users. (Section III)

(3) We present a hybrid decision and ranking model for question selection to minimize human cost. (Section IV)

(4) We conduct extensive experiments to show the effectiveness of DEXPLORER. (Section V)

II. PROBLEM STATEMENT

Given a relational table T, the user wants to find a subset $R \subseteq T$, and tuples in R are ranked. In practice, a user's query intent might be formulated as either an SQL query, some machine learning (ML) models, or some specific logic. Generally speaking, we want to infer/learn a method/model $f()$ as the proxy of a user's query intent, where $f(T)$ returns the ranked list R' that is close to R.

Example 1: Suppose a user wants a new manual petrol car that is produced after year 2010, not provided by commercial sellers, and its brand can be either BMW with price ≤ 10000, or Volkswagen with price ≤ 8000. Moreover, assume that she wants all cars to be ranked, *e.g.,* using a weighted sum function: $-0.018 \times price + 0.982 \times powerPS$. The query intent could be expressed as Q_1 below: □

```
SELECT * FROM Car
WHERE seller != "commercial" AND year ≥ 2010
AND gearbox="manually" AND fuelType="petrol"
AND ((brand = "bmw" AND price ≤ 10000) OR
(brand = "volkswagen" AND price ≤ 8000))
ORDER BY -0.018 * price + 0.982 * powerPS DESC;
```
Q_1

Example 1 shows that a user's query intent may be: (1) ***hard to specify:*** there are complicated predicates in the WHERE clause and it is almost impossible to manually specify the weighted sum function with correct parameters in the ORDER BY clause, and (2) ***hard to infer:*** no existing work (see Table I) can effectively infer such a query, not to say more complicated cases.

User Operations. We allow two *user operations*:

(1) *true/false* labeling: the user may label a given tuple as *true*(desired) or *false*(not desired); and

(2) a partial order: given two tuples t_i and t_j, the user might tell which tuple is more preferable.

User Questions. A *question* $\mathbb{I}$ is a list of k tuples, which solicits two types of labels from a user:

(1) *decision:* the user will split $\mathbb{I}$ into three disjoint sets of tuples: $\mathbb{D}^+$ with *true* labels, $\mathbb{D}^-$ with *false* labels, and $\mathbb{D}^?$ with unknown labels. Let $\mathbb{D}$ be the set of all tuple labels, *i.e.,* $\mathbb{D} = \mathbb{D}^+ \cup \mathbb{D}^- \cup \mathbb{D}^?$.

(2) *ranking:* rank (partial) pairs of tuples in $\mathbb{D}^+$, which gets a set $\mathbb{R}$ of partial orders. Implicitly, any tuple $t_i \in \mathbb{D}^+$ is more preferred than any tuple $t_j \in \mathbb{D}^-$, and there is no need to rank two tuples in $\mathbb{D}^-$, denoted by $t_x \equiv t_y$ for any $t_x, t_y \in \mathbb{D}^-$.

Example 2: The table in Figure 1 shows the labeling examples based on Q_1 in Example 1. (1) The user can *annotate* the tuples she desires or not, *e.g.,* $\mathbb{D}^+ = \{t_1, t_2, t_3, t_4\}$ and $\mathbb{D}^- = \{t_5, t_6\}$. (2) The user can specify which cars are more preferred for tuples in $\mathbb{D}^+$, either through a total order such as $(t_1 \succ t_2 \succ t_3 \succ t_4)$ or a set of partially ordered lists such as $(t_1 \succ t_2 \succ t_3)$ and $(t_1 \succ t_4)$. □

Problem Statement. Given a table T, a parameter k (*i.e.,* the number of tuples to show in each question), and a budget n for the number of questions, the problem of *interactive database exploration* (**IDE**) is to iteratively ask n questions $\{\mathbb{I}_1, \ldots, \mathbb{I}_n\}$ ($|\mathbb{I}_i| = k$ for $i \in [1, n]$) to the user, and infer a ranked set R of tuples based on the user feedback.

Two Research Challenges. There are two main research challenges to solve IDE: (1) *Answer Inference*: *how to infer R based on the user feedback for $\mathbb{I}_1, ..., \mathbb{I}_n$?* (Section III) (2) *Question Selection*: *how to select a question $\mathbb{I}_i$ in the i-th iteration?* (Section IV)

III. OVERVIEW OF DEXPLORER

A. System Overview (Figure 1)

Front-end. The user interacts with DEXPLORER in multiple iterations until the user budget is used up or the answer cannot be improved. At each iteration, the user is provided with a question I with k tuples, on which two operations are permissible: (1) "*click*" to annotate a tuple as either *true* or *false*; and (2) "*drag*" to rank a tuple higher than

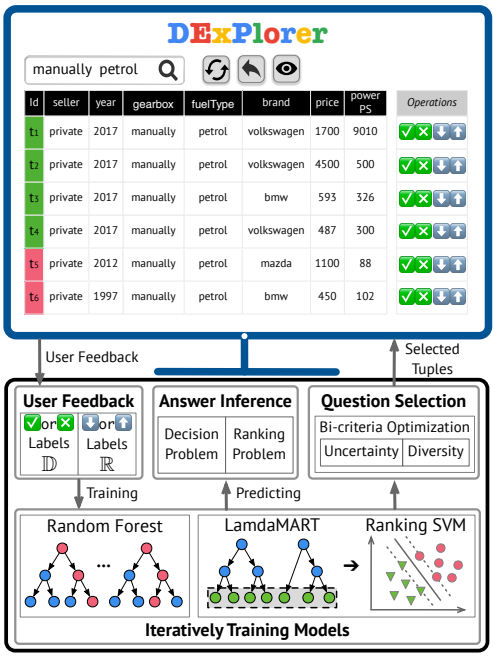

Fig. 1. An overview of DExPlorer

another. The user annotations are then transformed to a set of *true/false* labels of tuples in $\mathbb{I}$, and a set $\mathbb{R}$ of partial orders between pairs of tuples in $\mathbb{I}$.

Back-end. At the i-th iteration, the user provides a set $\mathbb{D}_i$ of *true/false* labels and a set $\mathbb{R}_i$ of partial orders. The back-end of DExPlorer needs to address two problems: answer inference and question selection.

Answer Inference. Given the user feedback from all i iterations, *i.e.,* $\{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_i\}$ and $\{\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_i\}$, we need to infer the (ranked) result R.

Question Selection. We need to select a set $\mathbb{I}_i$ with k tuples for the user to annotate in the i-th iteration.

The major challenge is that question selection for IDE is a bi-criteria problem that needs to estimate the *true/false* labeling and the partial order labeling between tuples.

Termination. The process terminates, either if the user budget is used up, or the back-end inference converges.

In the following, we will focus on answer inference. The details for question selection will be discussed in Section IV.

B. Answer Inference

Next, we discuss how DExPlorer does the decision answer inference, the ranking answer inference, and then a hybrid approach of combining them for the IDE problem.

Decision Inference. Essentially, this is a binary classification problem – deciding whether a tuple is desired or not. There are several choices: decision trees (DT), random forests (RF) [19], or support vector machines (SVM) [3]. [5] uses DT for IDE-Decision. However, as observed by [17], they are not ideal to capture complex predicates. DExPlorer employs RF for three reasons:

(1) RF, a collection of many DTs, are more robust at capturing complicated cases (*e.g.,* with *if-then-else*, *and*, *or* and *not*) [24] and preventing overfitting, compared with DTs.

(2) In some cases, RF, with shallow decision trees and a few branches, are still interpretable.

(3) As will be seen shortly in question selection (Section IV), RF, which outputs a vector with 0's and 1's (each DT in an RF will output either 0 or 1 for one input), are a better choice than an SVM that gives a single value for computing diversity between tuples, an important feature for question selection.

Ranking Inference. We support linear weighted ranking functions because in many applications, especially in databases with numeric values, a weighted sum among multiple attributes is widely used to model user preferences, which is also observed by [18], [21].

The ranking problem in DExPlorer is to learn a function $g(t)$. We employ Ranking SVM [18], [21] as the basic model for two reasons: (1) The scoring function used to rank tuples can be roughly captured by a linear function, *i.e.,* $g(t) = \vec{w} \cdot \vec{t}$, where $\vec{w}$ is a weight vector quantifying the preference and importance of attributes, and $\vec{t}$ is the feature vector of tuple t. Ranking SVM is a natural choice for this case. (2) Compared with complicated model like RankNet [1], Ranking SVM can avoid overfitting by using a small number of training data. However, one drawback of using a lightweight model, such as Ranking SVM, is that it needs sophisticated feature engineering. One way to combat this is to use another model to capture more *distinguishing features*, which is inspired by a classical solution for Click-Through Rate (CTR) Prediction.

LambdaMART + Ranking SVM. Inspired by the GBDT + LR model used in many commercial IR systems [13], [14], we propose a hybrid ranking model: LambdaMART + Ranking SVM, where LambdaMART [26] is a tree structure model based on MART (Multiple Additive Regression Tree) [9] that transforms the input features.The "Lambda" in LambdaMART denotes a special value which is the negative gradient in the MART algorithm. The output of LambdaMART is then fed to Ranking SVM to infer tuple ranking.

C. Iteratively Training and Predicting

Given $\mathbb{D}_i$ for decision labels and $\mathbb{R}_i$ for partial order labels in the i-th iteration, we describe how to use them for decision and ranking training and prediction.

Training. (1) *Decision inference using RF:* we incrementally train RF by feeding new labels in $\mathbb{D}_i$, obtaining RF_i. (2) *Ranking inference using LambdaMART + Ranking SVM:* Given a set of features of ordered tuple pairs $\mathbb{R}_i$ as training examples: (2.a) We first use $\mathbb{R}_i$ to incrementally train the LambdaMART model, obtaining LM_i; (2.b) For each tuple t (whose one-hot encoding feature vector is denoted as $\vec{t}'$) appearing in $\mathbb{R}_i$, $LM_i(t)$ outputs a m-dimension transformed feature vector $(x_1, \ldots, x_m)$, where m is the number of trees in LambdaMART (see Figure 1), and x_j ($j \in [1, m]$) is the leaf node index of $\vec{t}'$ that ends up falling in the j-th

tree. (2.c) Let $\vec{\mathsf{t}}''$ be the one-hot encoding of the transformed feature $(x_1, \ldots, x_m)$. Let $\vec{\mathsf{t}}$ be the concatenation ($\oplus$) of $\vec{\mathsf{t}}'$ and $\vec{\mathsf{t}}''$. For each ordered tuple pair (t_y, t_z) in $\mathbb{R}_i$, we use the enriched features $(\vec{\mathsf{t}}_y, \vec{\mathsf{t}}_z)$ to incrementally train Ranking SVM, obtaining RS_i.

Prediction. (1) *Decision prediction:* we use RF_i to predict each unlabeled tuple, predicting R_i as a set of desired tuples. (2) *Ranking prediction:* For each $t \in R_i$: (2.a) use the trained LM_i to get the enriched features $\vec{\mathsf{t}}$; (2.b) compute for t a score using the learned weight function $\vec{\mathsf{w}}$ in RS_i as $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}$; and (2.c) rank tuples in R_i based on their scores.

IV. Question Selection

In each user iteration, we need to select a list of k tuples from the table T as one question $\mathbb{I}$.

A. Uncertainty and Diversity

Uncertainty is the most commonly used criteria for tuple selection; it measures the confidence of the current model on evaluating a question.

Uncertainty for Decision Questions. We define the uncertainty of a tuple t as the entropy of the predicted results of all decision trees in the random forest. Suppose we have n trees, and m is the number of trees which predicted the tuple t as positive, then the uncertainty of t is defined as: $u(t) = -(e \log e + (1-e)\log(1-e))$, where $e = \frac{m}{n}$. The larger the $u(t)$, the larger the uncertainty.

Uncertainty for Ranking Questions. Given a pair of tuples t_i and t_j, the Ranking SVM model uses a hyperplane to make decisions about whether $t_i \succ t_j$. It also learns a parameter vector $\vec{\mathsf{w}}$, such that $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i$ denotes the ranking score of tuple t_i. The higher the score is, the higher the ranking t_i should be. If the pair is above the hyperplane, *i.e.,* if $\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j > 0$, then $t_i \succ t_j$, and vice versa. Thus, those pairs close to the hyperplane are the uncertain ones, *i.e.,* when $|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j|$ is close to 0. Besides selecting tuples that the decision and ranking models are uncertain about, we should also care about the diversity of the selected tuples. Consider two tuples with high uncertainty but with highly similar content, it is a waste of human efforts to label both of them because labeling only one of them can produce a model that is very likely to predict the other correctly.

Diversity. A simple way to measure diversity is to compute the string similarity of each tuple pair using some predefined function. However, such method does not consider the semantic information (dynamically) incorporated by the user feedback. Thus, we can use the feature vectors produced by RF and LambdaMART to measure the similarity of two tuples. Let $\vec{\mathsf{v}}'(t)$ be the prediction vector of all trees in RF for tuple t, where the i-th element in $\vec{\mathsf{v}}'(t)$ denotes the prediction label (1 or 0) of the i-th tree in the random forest. Let $\vec{\mathsf{v}}''(t)$ be the one-hot encoding transformed feature vector of tuple t output by the LambdaMART sub-model (*i.e.,* $\vec{\mathsf{v}}''(t) = \vec{\mathsf{t}}''$). We concatenate $\vec{\mathsf{v}}'(t)$ and $\vec{\mathsf{v}}''(t)$ as $\vec{\mathsf{v}}(t)$, which is used to compute the diversity. More specifically, we define the similarity $s(t_i, t_j)$ of tuple t_i and t_j using the Cosine Similarity: $s(t_i, t_j) = cos(\vec{\mathsf{v}}(t_i), \vec{\mathsf{v}}(t_j))$. The smaller the similarity, the larger the diversity between two tuples.

Considering both the uncertainty and diversity, we define the IDE question selection problem as follows.

Definition 1 (IDE Question Selection): Given a partially trained RF model and a hybrid ranking model, a table T, and a number k, select a set of k tuples S^* from T such that the following equation is minimized:

$$S^* = \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{t \in S} (1-u(t)) + \alpha \sum_{t_i, t_j \in S} |\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j| + \beta \sum_{t_i, t_j \in S} s(t_i, t_j) \tag{1}$$

where $u(t)$ is the normalized uncertainty of tuple t, α is a parameter to trade-off decision and ranking questions, $\vec{\mathsf{w}}$ is the weight vector output by the hybrid ranking model, β is a parameter that provides a trade-off between the uncertainty and diversity, and $s(t_i, t_j)$ is the similarity of t_i and t_j.

Theorem 1: The IDE question selection is NP-hard.

We can prove that the IDE question selection problem is NP-hard by a reduction from the MaxSumDispersion problem that is known to be NP-hard [11].

B. Question Selection Algorithms

*1) **AQS: An Approximate Algorithm**:* Inspired by [10], we present a 2-approximation algorithm, denoted by AQS, for IDE question selection problem. The algorithm first initializes an empty result set and calculates a new similarity $s'(t_i, t_j) = \frac{1-u(t_i)}{k-1} + \frac{1-u(t_j)}{k-1} + \alpha|\vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_i - \vec{\mathsf{w}} \cdot \vec{\mathsf{t}}_j| + \beta s(t_i, t_j)$ for each tuple pair (t_i, t_j). It then selects the tuple pair that has not been added to the result set and with the least s' scores, and adds the pair to the result set. It iteratively runs $\lfloor \frac{k}{2} \rfloor$ times. It adds another tuple to the result set if k is odd and the current result has $k-1$ tuples. The selected k tuples are then returned.

The time complexity of AQS is $O(\max(|T|^2 L, |T|^2 k))$, where L is the length of tuples. However, the time to compute a question can still be long when $|T|$ is large, thus the above algorithm is not efficient enough to satisfy the online question selection requirement.

*2) **IQS: An Efficient Algorithm**:* We propose an efficient algorithm IQS (Algorithm 1) to solve the IDE question, which can return approximate results of S^* in interactive time. The main idea is that to first select the tuple with the highest decision uncertainty score, and then add it to the result set if this tuple has low similarity and high ranking uncertainty with the current selected tuples.

We first sort T by descending order of $u(t)$ (line 1), and initialize the result set S as empty (line 2). Next, we iterate over each tuple t_i in T, and check whether t_i has high similarity and low ranking uncertainty with respect to the current tuples in S (line 4 - 8). If yes, we just drop it; otherwise we add t_i into S. Finally, S is returned as the selected question with k tuples (line 11).

Complexity. The time complexity for sorting T is $O(|T| \log |T|)$. The time complexity to check whether t_i can be added to the result set is $O(kL)$, where L is

Input: T, k, a weight vector $\vec{w}$, a threshold δ
Output: an approximate optimal set S

```
1  sort T by descending order of u(t), T = [t_1, t_2, ..., t_|T|];
2  initialize S ← ∅;
3  for t_i in T do
4      flag = True;
5      for t_j in |S| do
6          if α · |w · t_i − w · t_j| + β · s(t_i, t_j) > δ then
7              flag = False;
8              break;
9      if flag then S ← S ∪ {t_i};
10     if |S| = k then break;
11 return S;
```

Algorithm 1: IDE QUESTION SELECTION (IQS)

the length of tuples. The outer loop iterates up to $|T|$ times. Therefore the time complexity for the lines 3 - 8 is $O(kL|T|)$. Therefore the time complexity for Algorithm 1 is $O(\max(|T|\log|T|, kL|T|))$, which is much more efficient than AQS.

V. EXPERIMENTS

Datasets. We use two datasets: `Car` and `Pub` (for publications). The `Car` dataset is from Kaggle (https://www.kaggle.com/orgesleka/used-cars-database) with one relational table, and the `Pub` dataset is crawled from the ACM Digital Library (https://www.acm.org/publications/digital-library) and contains six relational tables. Table II gives more statistics about these two datasets.

The Ground Truth SQL Queries. We use SQL queries as the proxy for user's query intent, because providing other types of ground truth would be subjective and thus not practical. We test 10 SQL queries Q_1–Q_{10}: Q_1–Q_8 on `Car`, and Q_9, Q_{10} on `Pub`. These queries are designed to be representative and cover many different cases. Due to the space limit, we omit more descriptions of these queries. For more details, please refer to our technical report (https://github.com/Qinxuedi/dexplorer).

Environment. All experiments are conducted on a MacBook Pro with 16 GB 1600 MHz RAM and 2.5 GHz Intel Core i7 CPU, running OS X Version 10.14.5.

Parameters. We include 10 tuples in one question for each interaction with the user. We will run 20 iterations for all datasets and show the performance during this period. To bootstrap the process, we allow the user to input one keyword, *e.g.,* the user can input "bmw" for Q_1. We set α and β by making decision and ranking, and uncertainty and diversity equivalently important.

A. *Effectiveness*

Metrics. The Kendall tau distance [4] is a widely used metric for comparing two ranked lists [7], [12], with the basic idea of counting the number of pairwise disagreements between two ranked lists. Let $N = \binom{|T|}{2}$ be the total number of tuple pairs. Let KD be the "disagreement" of all tuple pairs, *i.e.,* the total number of tuple pairs whose predicted labels are different from ground truth labels. We use the normalized Kendall tau distance as the evaluation metric, denoted by $accuracy = \frac{N-KD}{N}$.

Dataset	Table	#-Tuples	#-Col	#-Cat	#-Num
Car	**Car**	248419	14	10	4
Pub	**Author**	6881	3	1	2
	Institution	1453	4	2	2
	Paper	1947	3	1	2
	Conference	16	5	3	2
	Paper_Author	8615	2	0	2
	Paper_Keyword	2322	2	1	1

TABLE II
STATISTICS OF DATASETS (#-COL: #-COLUMN; #-CAT: #-CATEGORICAL; #-NUM: #-NUMERIC)

Comparisons. We test two flavors of DEXPLORER: (1) DEXPLORER-IQS uses the model in Section III-B for answer inference and IQS for question selection and (2) DEXPLORER-AQS uses the same answer inference model with DEXPLORER but different question selection algorithm AQS. We also compare DEXPLORER to (3) SQLSynthesizer [28] (denoted by SSY), which infers SQL by accepting ranked positive tuples.

Figure 2 shows the *accuracy* of Q_1-Q_{10} for DEXPLORER-IQS, DEXPLORER-AQS, SSY. Figure 2 shows that: (1) With the increasing of #-Questions, the accuracy of DEXPLORER-IQS, DEXPLORER-AQS and SSY all increase. They finally achieve an *accuracy* of 0.916, 0.968, 0.499 respectively on average for $Q_1 - Q_{10}$ after asking 20 questions. DEXPLORER-IQS behaves similarly to DEXPLORER-AQS, but DEXPLORER-AQS takes longer time (*i.e.,* more than 1 hour for $Q_1 - Q_8$), while DEXPLORER-IQS can return results in interactive time. (2) SSY supports decision using a decision tree and supports ranking by one or more attributes hierarchically. Thus, SSY achieves an *accuracy* of 1.0 on Q_1, Q_6, which contains simple decision problems and hierarchical ranking functions. But SSY behaves badly for complex decision and ranking problems, *i.e.,* $Q_2 - Q_5$, Q_7, Q_8, while DEXPLORER-IQS has a stable performance for all cases.

B. *Efficiency*

We also test the efficiency of DEXPLORER on IDE problems Q_1-Q_{10}. We repeat all experiments ten times to compute the average results. Table III shows the running time of answer inference and question selection by algorithms IQS and AQS.

We make the following observations: (1) IQS significantly outperforms AQS on all IDE problems (*i.e.,* Q_1 - Q_{10}). This observation also matches the result of the time complexity discussion in Section IV. (2) The results on answer inference and question selection of the tested algorithms do not vary a lot for different IDE problems on the same dataset. The reason is that the algorithms are independent of the complexity of the SQL queries. However, the complex SQL queries usually take more rounds to converge. (3) It takes less than 2 seconds on dataset `Car` to infer answers and select questions for the next round using IQS, which should be acceptable in practice. For the small dataset `Pub`, it only takes ∼0.1 second.

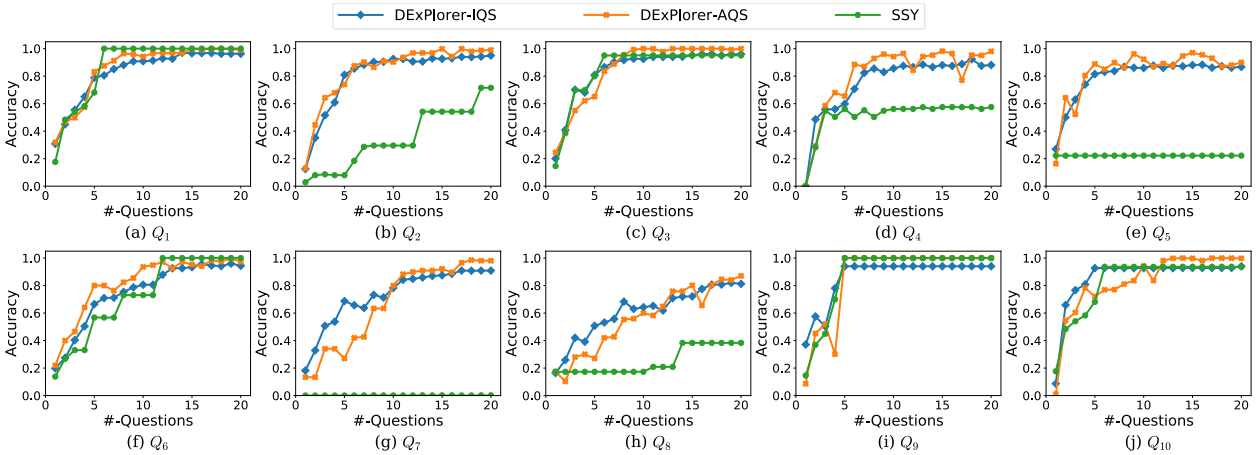

Fig. 2. Effectiveness Study for **IDE** Problem (x-axis: #-Questions; y-axis: Accuracy)

Dataset		Car								Pub	
Qid		Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
IQS	AI (sec)	1.02	0.97	0.98	1.02	1.12	1.14	1.19	1.26	0.04	0.08
	QS (sec)	0.59	0.63	0.60	0.65	0.55	0.60	0.86	0.98	0.02	0.04
AQS	AI (sec)	1.03	1.04	0.88	1.01	1.05	1.12	1.18	1.20	0.05	0.08
	QS (sec)	4178.91	4081.05	4010.61	3969.71	3990.66	4215.86	4072.40	4326.84	5.57	18.61

TABLE III
EFFICIENCY STUDY FOR **IDE** PROBLEM (AI: ANSWER INFERENCE; QS: QUESTION SELECTION)

VI. CONCLUSION

This paper presented DEXPLORER, an interactive database exploration system. DEXPLORER is based on a well-performed answer inference model that selects a set of tuples to be annotated for relevance and partially ranked for preference by the user. It offers a user-friendly front-end that allows the user to select and rank tuples. We have proved that the optimal question selection problem is NP-hard and proposed an efficient and effective algorithm. Extensive experiments show the effectiveness of DEXPLORER.

Acknowledgement. This work was supported by NSF of China (61925205, 61632016), Huawei, and TAL Education.

REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.
[2] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.
[3] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
[4] P. Diaconis. Group representations in probability and statistics. *Ims Lecture Notes–monograph*, 72(2):7–108, 1988.
[5] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, pages 517–528, 2014.
[6] J. Du, L. Zheng, J. He, J. Rong, H. Wang, and Y. Zhang. An interactive network for end-to-end review helpfulness modeling. *Data Sci. Eng.*, 5(3):261–279, 2020.
[7] Dwork, Cynthia, Kumar, Ravi, Naor, Moni, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.
[8] A. Fariha and A. Meliou. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *PVLDB*, 12(11):1262–1275, 2019.
[9] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
[10] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.
[11] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations research letters*, 21(3):133–137, 1997.
[12] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526. ACM, 2002.
[13] K. Hazelwood and et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *HPCA*, 2018.
[14] X. He and et al. Practical lessons from predicting clicks on ads at facebook. In *ADKDD*, pages 5:1–5:9, 2014.
[15] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.
[16] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.
[17] E. Huang, L. Peng, L. D. Palma, A. Abdelkafi, A. Liu, and Y. Diao. Optimization for active learning-based interactive database exploration. *PVLDB*, 12(1):71–84, 2018.
[18] T. Joachims. Training linear svms in linear time. In *SIGKDD*, pages 217–226, 2006.
[19] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.
[20] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.
[21] L. Qian, J. Gao, and H. Jagadish. Learning user preferences by adaptive pairwise comparison. *PVLDB*, 8(11):1322–1333, 2015.
[22] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *SIGMOD*, pages 493–504, 2014.
[23] L. F. Sikos and D. Philp. Provenance-aware knowledge representation: A survey of data models and contextualized knowledge graphs. *Data Sci. Eng.*, 5(3):293–316, 2020.
[24] R. Singh, V. V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, and N. Tang. Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202, 2017.
[25] P. Tong, Q. Zhang, and J. Yao. Leveraging domain context for question answering over knowledge graph. *Data Sci. Eng.*, 4(4):323–335, 2019.
[26] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.
[27] M. Xie, T. Chen, and R. C.-W. Wong. Findyourfavorite: An interactive system for finding the user's favorite tuple in the database. In *SIGMOD*, pages 2017–2020, 2019.
[28] S. Zhang and Y. Sun. Automatically synthesizing sql queries from input-output examples. In *ASE*, pages 224–234, 2013.