Interactively Discovering and Ranking Desired Tuples without Writing SQL Queries

Xuedi Qin\textsuperscript{1}, Chengliang Chai\textsuperscript{1}, Yuyu Luo\textsuperscript{1}, Nan Tang\textsuperscript{2}, Guoliang Li\textsuperscript{1}
\textsuperscript{1}Department of Computer Science, Tsinghua University \textsuperscript{2}QCRI, HBKU
\{qxdl7@mails., chailc15@mails., luoyy18@mails., liguoliang@\}tsinghua.edu.cn, ntang@hbku.edu.qa

ABSTRACT
The very first step of many data analytics is to find and (possibly) rank desired tuples, typically through writing SQL queries – this is feasible only for data experts who can write SQL queries and know the data very well. Unfortunately, in practice, the queries might be complicated (for example, "find and rank good off-road cars based on a combination of Price, Make, Model, Age, Mileage, and so on") is complicated because it contains many if-then-else, and, or and not logic) such that even data experts cannot precisely specify SQL queries; and the data might be unknown, which is common in data discovery that one tries to discover desired data from a data lake. Naturally, a system that can help users to discover and rank desired tuples without writing SQL queries is needed. We propose to demonstrate such a system, namely DExPlorer. To use DExPlorer for data exploration, the user only needs to interactively perform two simple operations over a set of system provided tuples: (1) annotate which tuples are desired (i.e., true labels) or not (i.e., false labels), and (2) annotate whether a tuple is more preferred than another one (i.e., partial orders or ranked lists). We will show that DExPlorer can find user’s desired tuples and rank them in a few interactions, even for complicated queries.

ACM Reference Format:
Xuedi Qin\textsuperscript{1}, Chengliang Chai\textsuperscript{1}, Yuyu Luo\textsuperscript{1}, Nan Tang\textsuperscript{2}, Guoliang Li\textsuperscript{1}. 2020. Interactively Discovering and Ranking Desired Tuples without Writing SQL Queries. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20), June 14–19, 2020, Portland, OR, USA. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384695

1 INTRODUCTION
This paper aims to help user discover a set of desired and ranked tuples through interactive exploration, when precise SQL queries are hard to specify. Informally speaking, given a relational table \( T \), the user wants to find the answer \( R = Q(T) \) of an unknown SQL query \( Q \), where the term “unknown” either means that the query is too hard to specify, or the data is unknown to the user. Let’s illustrate through an example why specifying such SQL queries is hard.

Example 1.1. 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 \( \leq 10000 \), or Volkswagen with price \( \leq 8000 \). Moreover, assume that she wants all cars to be ranked by a weighted sum function: \(-0.018 \times \text{price} + 0.982 \times \text{powerPS}\) (note: she does not know this function). That is, the ground truth query can be expressed as \( Q_1 \) below:

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad \text{Car} \\
\text{WHERE} & \quad \text{seller} \neq \text{"commercial" \ AND \ year} \geq 2010 \\
& \quad \text{AND} \quad \text{gearbox}=\text{"manually" \ AND } \text{fuelType}=\text{"petrol"} \\
& \quad \text{AND} \quad \left( \text{brand}=\text{"bmw" \ AND \ price} \leq 10000 \right) \text{ OR} \\
& \quad \left( \text{brand}=\text{"volkswagen" \ AND \ price} \leq 8000 \right) \\
\text{ORDER BY} & \quad -0.018 \times \text{price} + 0.982 \times \text{powerPS DESC};
\end{align*}
\]

Clearly, \( Q_1 \) is hard to specify, because: (i) there are complicated predicates in the WHERE clause, including and, or, not and range selection; and (ii) it is hard to provide the weighted sum function in the ORDER BY clause.

Prior Art. (1) Keyword search [5] allows the user to retrieve some desired tuples by providing some keywords. However, it is hard to capture complex SQL queries which may contain if-then-else, and, or and not logic by under-specified keywords. (2) Query-by-example [1] aims to infer user’s desired tuples by user provided examples, either desired or not desired. (3) Partial orders for tuple ranking [10] ranks tuples based on user provided partial orders for tuple pairs. (1) and (2) only discover desired tuples (denoted by decision problem), and (3) focuses on ranking all tuples (denoted by ranking problem). There are also methods that
find and rank tuples, such as SQLSynthesizer [11], but can only support simple ranking functions. That is, no existing work can support the complicated case in Example 1.1.

**Challenges.** Building a system for inferring complicated query intent faces several design choices and research challenges. (C1) [User Interface.] What operations shall we provide to the users? (C2) [SQL Queries vs. Machine Learning Models.] For the back-end inference, based on the user feedback, shall we reason about SQL queries or train ML models? (C3) [Question Selection.] One key challenge is to reduce human cost, such that the back-end engine can quickly converge, which requires algorithms to proactively select the most beneficial tuples for user feedback.

**Outline.** Section 2 overviews our proposed system, mainly to address (C1), from a design perspective. Section 3 demonstrates DEXPlorer using real-world datasets with two main goals: (G1) Easy-to-use: any user can operate on DEXPlorer with simple (click-based) operations; and (G2) Effectiveness: DEXPlorer can return good results in a few user interactions. Section 4 gives details of the back-end, mainly for (C2) and (C3), and compares with state-of-the-art solutions.

## 2 AN OVERVIEW OF DEXPLORER

Figure 1 gives an overview of DEXPlorer.

**Front-end.** It will interact with the user in multiple iterations until user budget is used up or the answer cannot be improved. At each iteration, the system provides a *question* $q_i$ with $k$ tuples, on which two operations are permissible: (1) “click” to annotate a tuple to be either true or false; and (2) “drag” to annotate that one is ranked higher than another. The answers annotated by the user are then transformed to a set $D_i$ of true/false labels of tuples in $q_i$, and a set $R_i$ of partial orders between pairs of tuples in $q_i$.

Moreover, in order to address the cold-start problem, we allow users to pose a few keyword queries to quickly get some desired tuples.

**Back-end.** In the $i$-th iteration, the user will provide a set $D_i$ of true/false labels and a set $R_i$ 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., $[D_1, D_2, \ldots, D_i]$ and $[R_1, R_2, \ldots, R_i]$, it is to infer the (ranked) result $R$.

**Question Selection.** It is to select a set $q_i$ with $k$ tuples for the user to annotate in the $i$-th iteration.

**Termination.** The entire process will terminate, when the user budget is used up, or the back-end inference will converge.

## 3 SYSTEM DEMONSTRATION

In this section, we demonstrate how DEXPlorer works (taking $Q_1$ as a running example) on the car dataset (https://www.kaggle.com/origesleka/used-cars-database).

1. **Keyword Search.** To bootstrap DEXPlorer, user can first input keywords to help DEXPlorer find her desired tuples. For example, user can type “manually petrol” in the input box in Figure 2(a) for $Q_1$, then DEXPlorer can select only manually petrol cars for her to label.

2. **Labeling Tuples.** When user inputs keywords, DEXPlorer selects tuples which are relevant to these keywords for her to label; otherwise DEXPlorer selects tuples by the question selection algorithm in Section 4.2. The selected tuples are shown to user as in Figure 2(a), then user can click the second column to annotate a tuple to be either true or false, and drag the first column of a tuple to adjust its rank. After labeling tuples as shown in Figure 2(a), the user can click the “Recommend” button, then DEXPlorer infers user’s desired tuples and ranks them by ML models in Section 4.1, and shows the desired ranked tuples to user as in Figure 2(b).

3. **Look up Recommended Results.** User can browse the desired ranked tuples recommended by DEXPlorer.

4. **Iterate the above Process.** If user is not satisfied with current recommended tuples, she can click the “Continue Tagging” button, and DEXPlorer will select new tuples for her to label by the question selection algorithm in Section 4.2. Or she can input new keywords (e.g., “bmw”) to refine the SQL query. That is, she can iterate the above process until she is satisfied with the recommended results. Then she can download data for down-streaming applications, such as data visualization [6–8].
Figure 2: Front-end of DExPlorer

Result Analysis. Figure 2(b) shows the top-10 recommended ranked desired tuples on \( Q_1 \) for user after 3 iterations (i.e., 3 questions are answered, each question contains 10 tuples), where the top-10 tuples all satisfy her hidden decision intent and are well ranked by her ranking intent (i.e., \( Q_1 \)). The accuracy of correctly ranked tuple pairs is higher than 0.8, and the precision@50 is 0.72 after 3 iterations, meaning that 36 of real top-50 desired tuples are found in the predicted top-50 tuples, which proves that DExPlorer can efficiently find and rank the desired tuples in a few interactions.

4 BACK-END ALGORITHMS

4.1 Answer Inference

Decision Answer Inference. The decision problem is a binary classification problem - deciding whether a tuple is desired or not. There are several choices: decision tree (DT), random forests (RF), or support vector machines (SVM). The work [1] uses DT for decision problem. However, DT is not ideal to capture complex predicates. Thus we use RF for decision answer inference.

Ranking Answer Inference. DExPlorer assumes user’s ranking intent can be expressed by a weighted sum function \( f(t) = w_t \), which is a common assumption in many data exploration systems [10]. Thus DExPlorer uses Ranking SVM [4] to learn \( w \). Besides, inspired by the GBDT + LR model in many commercial IR systems [3], we develop a hybrid ranking model: LambdaMART [9] + Ranking SVM. The LambdaMART model outputs a transformed feature \( \mathbf{t}' \) for each tuple \( \mathbf{t} \), and \( \mathbf{t} = \mathbf{t}' \odot \mathbf{t}'' \) is fed to Ranking SVM.

4.2 Question Selection

In each user iteration, we need to select a list of \( k \) tuples from the table \( T \) as one question \( I \). When selecting tuples, we should consider about their uncertainty and diversity.

4.2.1 Uncertainty and Diversity. We should select tuples which the ML models are uncertain about, and these selected tuples should be as diversified as possible. Now, we define the uncertainty and diversity of tuples.

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.

Uncertainty for Ranking Questions. Given a pair of tuples \( t_i \) and \( t_j \), the Ranking SVM model learns a parameter vector \( \mathbf{w} \), and the model is uncertain about the tuple pairs whose \( |\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j| \) are close to 0.

Diversity. Let \( \mathbf{v}'(t) \) be the predicting vector of all trees in RF for tuple \( t \), and let \( \mathbf{v}''(t) \) be the transformed feature of tuple \( t \).
output by the LambdaMART model (i.e., \( \hat{\nu}''(t) = \hat{\nu}'(t) + \hat{\nu}'(t) \)). We use \( \hat{\nu}(t) = \hat{\nu}'(t) \oplus \hat{\nu}''(t) \) to compute the diversity. We define the similarity \( s \) of tuple \( t_i \) and \( t_j \) as: \( s(t_i, t_j) = \cos(\hat{\nu}(t_i), \hat{\nu}(t_j)) \).

**Definition 4.1 (Question Selection).** Given a partially trained RF model and hybrid ranking model, a table \( T \), and a number \( k \), the problem is to select a set of \( k \) tuples \( S' \) from \( T \) such that the following equation is minimized:

\[
S' = \arg \min_{S \subseteq T, |S|=k} \sum_{t \in S} (1-u(t)) + \alpha \sum_{t_i, t_j \in S} |\hat{\nu} \cdot \hat{\nu}'(t_i) - \hat{\nu} \cdot \hat{\nu}'(t_j)| + \beta \sum_{t_i, t_j \in S} s(t_i, t_j)
\]

(1)

where \( u(t) \) is the normalized uncertainty of tuple \( t \), \( \alpha \) is a parameter to trade-off decision and ranking questions, \( \hat{\nu} \) is the weight vector output by the hybrid ranking model, \( \beta \) 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 \( t_j \).

4.2.2 Algorithms. Considering both uncertainty and diversity is hard due to it is NP-hard [2]. We thus propose to first solve the question selection by only considering decision and ranking uncertainty (i.e., \( \beta = 0 \)), and then incorporate the diversity into the solution obtained in the first step.

**Question Selection without Diversity.** We set \( \beta = 0 \) in Eq. (1) to ignore diversity:

\[
S' = \arg \min_{S \subseteq T, |S|=k} \sum_{t \in S} (1-u(t)) + \alpha \sum_{t_i, t_j \in S} |\hat{\nu} \cdot \hat{\nu}'(t_i) - \hat{\nu} \cdot \hat{\nu}'(t_j)|
\]

(2)

To better illustrate the optimization problem, we first denote \( S = \{t_1, t_2, ..., t_k\} \), where \( \hat{\nu} \cdot \hat{\nu}'(t_i) \leq \hat{\nu} \cdot \hat{\nu}'(t_j) \) iff \( i \leq j \), then we expand the second term of Eq. 2 to Eq. 3. Thus we have:

\[
S' = \arg \min_{S \subseteq T, |S|=k} \sum_{i=1}^k (1-u(t_i)) + \alpha \sum_{i=1}^k ((i-1)\hat{\nu} \cdot \hat{\nu}'(t_i) - (k-i)\hat{\nu} \cdot \hat{\nu}'(t_i))
\]

\[
= \arg \min_{S \subseteq T, |S|=k} \sum_{i=1}^k (1-u(t_i)) + \alpha \sum_{i=1}^k (2i-k-1)\hat{\nu} \cdot \hat{\nu}'(t_i)
\]

(3)

We sort \( T \) by \( \hat{\nu} \cdot \hat{\nu}' \) in ascending order and obtain a sorted list \( T = [t_1, t_2, ..., t_T] \). We use \( T_m \) to denote the prefix of list \( T \) with length \( m \), i.e., \( T_m = [t_1, t_2, ..., t_m] \). We define \( S(m, n) = \arg \min_{S \subseteq T_m, |S|=n} \sum_{i=1}^n (1-u(t_i)) + \alpha \sum_{i=1}^n (2i-k-1)\hat{\nu} \cdot \hat{\nu}'(t_i) \) and \( F(m, n) \) denotes the corresponding optimal value. We can see that \( S(|T|, k) \) is the optimal solution of Eq. 2 and \( F(|T|, k) \) is the corresponding optimal value. Then we have:

\[
F(m, n) = \min(F(m-1, n), F(m-1, n-1) + \phi(m, n))
\]

(4)

where \( \phi(m, n) = (1-u(t_m)) + \alpha (2n-k-1)\hat{\nu} w_m \). Thus we devise a dynamic programming algorithm to find the optimal solution for Eq. 2.

**Question Selection with Diversity.** When considering both uncertainty and diversity, we optimally choose tuples with high uncertainty by the above dynamic programming algorithm, but when adding a tuple \( t \) to the result set \( S \), we check whether \( t \) has a high similarity with existing selected tuples. If yes, we just drop it; else, we include it to the result.

4.2.3 Comparison. We compare our method with state-of-the-art solution for discovering and ranking desired tuples: SQLSynthesizer [11]. Since SQLSynthesizer can only support simple (hierarchical) ranking function, we also test another SQL query \( Q_2 \) by replacing the ORDER BY clause in \( Q_1 \) with “ORDER BY year, -kilometer, -price, powerPS DESC”. The accuracy of correctly ranked tuple pairs of \( Q_1 \) and \( Q_2 \) are shown in Figure 3(a) and Figure 3(b) respectively.

From Figure 3, we can know SQLSynthesizer only supports simple (hierarchical) ranking function (i.e., \( Q_1 \)), and has a poor performance on complex ranking function (i.e., \( Q_2 \)), while DExPlorer supports both simple and complex ranking functions well. Also, DExPlorer performs better than SQLSynthesizer on \( Q_2 \), which is due to RF can better capture user’s complex query intents, and the effectiveness of the question selection algorithm in DExPlorer.

**Acknowledgement.** This work was supported by NSF of China (61925205, 61632016, 61521002), Huawei, and TAL Education Group.

**REFERENCES**


