iSearch: An Interpretation based Framework for Keyword Search in Relational Databases

Zhong Zeng Zhifeng Bao Tok Wang Ling Mong Li Lee
School of Computing
National University of Singapore
{zengzh, baozhife, lingtw, leeml}@comp.nus.edu.sg

ABSTRACT

Keyword search has become an effective information retrieval method for structured data. Existing works in relational database keyword search have addressed the problems of finding and evaluating candidate results. However, given that keyword queries are inherently ambiguous, it is often the case that candidate results do not match users' search intention. In this paper, we analyze the limitations of current keyword search techniques and introduce the problem of generating and ranking keyword query interpretations. We propose a novel 3-phase keyword search paradigm which consists of: (1) the ability to predict query interpretations; (2) incorporate user feedback to to remove keyword ambiguities; (3) a ranking model to evaluate a query interpretation.

1. INTRODUCTION

Inspired by the success of keyword search engine in the Web, keyword search over relational databases has emerged as a popular research topic in recent years [15]. In contrast to the traditional ways of querying a database, keyword search allows users to pose queries without having to understand databases schemas or query languages, thus providing an intuitive, convenient and effective way to explore the information in relational databases.

However, due to the inherent ambiguities of keywords [5], each keyword query generally yields a large number of results, the majority of which are not useful to the user. Various ranking strategies have been proposed to help users find the expected answers [9, 15, 4, 16, 20]. The intuitions behind these strategies include:

1. The more frequent the keywords appear in a result, the more relevant the result is;
2. The more discriminative the keywords are in the database, the more relevant the result is;
3. The smaller the size of a result is, the more relevant the result is.

KEYS'12, May 20, 2012, Scottsdale, Arizona, USA.

However, these intuitions do not always retrieve the expected answers, as we will illustrate with an example. Figure 1 shows a sample of five tables from the DBLP database [1]. The table *Paper* and *Author* contain information about academic papers and researchers respectively. Table $Conference$ stores information about conference. Tables *PaperAuthor* and *PaperCitation* are relationship tables. The former captures the $m : m$ relationships between papers and authors, while the latter Table *PaperCitation* materializes the recursive citation relationships between papers.

Suppose a user issues a keyword query *"Markov, Model"* intending to retrieve papers on *"Model"* authored by *"Markov"*. Existing keyword search techniques will return three candidate results, namely $P1$ and $P2$ from the *Paper* table and the tuple $\langle P3, A3\rangle$ from the *PaperAuthor* table since these tuples contain both keywords. Further, the first two results $P1$ and $P2$, which are often ranked before $\langle P3, A3\rangle$, do not match the user's search intention to find the papers on *"Model"* authored by *"Markov"*.

Given the inherent ambiguities in keywords, we have multiple interpretations of keyword queries and users typically do not know how to express their intentions but they can identify it when they see it [19].

In this work, we propose a framework that utilizes query interpretations to answer keyword queries in relational databases. Given a user query, we first present to the user a set of potential interpretations that could capture the search intentions. Then results are retrieved based on the user's selected interpretations. By engaging the user in the query process, the proposed approach aims to reduce query ambiguity and increase the effectiveness of the keyword search.

In order to provide an overview of keyword query interpretations, we need to address two major challenges. The *first* challenge is how to find all the possible interpretations of a keyword query. Since each keyword may appear as values of different attributes and carry different meanings, we need to find all the possible semantics of each keyword. Given the semantics of the keywords in a query, different combinations or join paths may exist to form a result that covers all the keywords. Hence, the total number of interpretations could be very large for a given query.

The *second* challenge is how to rank the interpretations. Similar to the candidate results returned by [10, 9, 16], not all the interpretations are equally useful to the users. Unlike ranking candidate results, a query interpretation consists of a set of relations instead of tuples. Hence, the ranking model in IR cannot be applied directly.

[1] http://www.informatik.uni-trier.de/~ley/db/

Paper

Pid	Title	Conid
P1	A hidden **Markov** model information retrieval system	C1
P2	Topics over time: a non **Markov** continuous-time **model** of topical trends	C2
P3	An HMM acoustic **model** incorporating various additional knowledge sources	C3
P4	ILDA: interdependent LDA model for learning latent aspects and their ratings from online product reviews	C4

PaperAuthor

Pid	Aid
P1	A1
P2	A2
P3	A3
P4	A4

Author

Aid	Name
A1	Miller
A2	Xuerui Wang
A3	**Markov**
A4	Moghuddam

Conference

Conid	Name
C1	22nd International ACM **SIGIR** Conference
C2	12th ACM SIGKDD international conference
C3	8th Conference of the International Speech Communication Association
C4	34th International ACM **SIGIR** Conference

PaperCitation

Pid	CitedPid
P2	P4

Figure 1: DBLP database sample.

In order to overcome the above challenges, we model the database schema as a Labeled Directed Graph (LDG) where each node represents a relation and each edge denotes a key-foreign key constraint between two relations. Compared to the schema graph in [10], we define the label of each edge as the key-foreign key attributes in two relations. As recursive relationships are usually common in relational databases and involve two key-foreign key constraints between two relations, we need the labels to distinguish the two different edges when finding query interpretations of a keyword query.

Next, we design a metric to measure the relevance of an interpretation to a keyword query. Since the total number of interpretations could be very large, it is important to evaluate each interpretation and rank them based on the relevance to the query. We propose three guidelines to identify the semantics which matches the user search intention for each keyword. We also design a formula to compute the confidence level of a keyword that refers to the possible semantics, as well as the relationships that link the keywords in the query.

To the best of our knowledge, this is the first work that focuses on generating and ranking keyword query interpretations. Our algorithm can be applied with existing keyword search systems to perform a 3-phase search as follows. We first generate a set of potential query interpretations from the user input query. Next, the user chooses the query interpretations that is closest to his/her search intention. Finally, we use the selected query interpretations to retrieve the results which would match the user's search intention.

The rest of the paper is organized as follows. Section 2 discusses related works. Section 3 presents preliminaries. The architecture of our framework are specified in Section 4. Section 5 presents our ranking method to evaluate query interpretations. Section 6 discusses the index techniques used to improve the efficiency. Experiment plans are presented in Section 7. In Section 8, we draw our preliminary conclusion. Section 9 is the acknowledgement.

2. RELATED WORK

Keyword search in relational databases [9, 10, 11, 17] has recently emerged as a popular research topic. Early works such as DBXplorer [17], DISCOVER [10] and BANKS [11] are systems that support free-form keyword search on relational databases.

One important issue of keyword search over relational databases is the efficiency of the algorithms. The techniques to support keyword search in relational databases can be divided into two main categories: schema graph approach and data graph approach [22]. In the schema graph approach [10, 9, 17], the relational database schema is modeled as a directed graph called schema graph where each node represents a relation and each edge represents a key-foreign key constraint. A keyword query is processed in two steps, namely candidate network (**CN**) generation and **CN** evaluation. In **CN** generation, the system constructs a set of join expressions (or CN) by traversing the schema graph. In **CN** evaluation, all the **CN**s are evaluated to produce tuple trees using SQL on RDBMSs. In DISCOVER, **CN**s are evaluated with an execution plan such that shared joining expressions are computed only once. In DISCOVER-II [9] and SPARK [16], several algorithms are proposed to get top-k query results efficiently.

In the data graph approach [11], the relational database is materialized as a data graph. Each node in the graph is a tuple and each edge between two tuples represents that they are connected by a key-foreign key constraint. Then tuple trees are directly found from the data graph with a Steiner tree algorithm. Methods to find top-k connected tuple trees have been proposed. The representative works include [6, 11, 13, 14]. [11] propose a backward search in BANKS and [13] design a bidirectional search in BANKS-II [13]. [14] propose a general framework to retrieve top-k connected trees with polynomial delay under data complexity, which is independent of the underline minimum Steiner tree algorithm. Since finding top-k connected trees is an instance of the group Steiner tree problem [7] which is NP-complete, [8] propose the distinct root semantics in BLINKS. A bi-level index was built to facilitate the computation of the shortest distances.

Besides efficiency, another important issue of keyword search over relational databases is the effectiveness. To evaluate whether a keyword search is effective or not, the best way is to see whether the user desired answers are ranked higher than other answers. Early works in schema graph approach like DBXplorer and DISCOVER simply consider the num-

ber of joins in the tuple trees. In DISCOVER-II, Hristidis et al. assumed OR semantics for answers and incorporated state-of-the-art IR relevance ranking in a straightforward manner. To overcome the side effect of overly rewarding contributions of the same keyword, Luo et al. proposed to model a joining tuple tree as a virtual document in SPARK. In data graph approach, Bhalotia et al.used PageRank style methods to assign weights to tuples and edges between tuples in BANKS. A combination of tuple weights and edge weights is used to compute the confidence of a tuple tree. In order to balance the importance of individual nodes and the structural cohesiveness of the results, [20] propose a random walk model with message passing to match the characteristics of keyword search in databases.

To date, there are very few works utilizes query interpretations in relational keyword search. The work in [5] propose to generate potential interpretations for a keyword query. The authors assume that priori access to the database content, and predict the semantics of each keyword by its position in the query. They do not consider the case where multiple paths may exist to link the keywords in the query. In contrast, we propose to generate and rank all possible query interpretations so that users can select the true intended interpretations before keyword search is performed. This approach overcomes the different interpretations that arise due to inherent keyword ambiguities. Selecting the correct interpretations early can help eliminate many results which the users are not interested in and thus increases the efficiency of the search process.

3. PRELIMINARIES

In this section, we define the notations and terminologies used in this work. A database D is a collection of relations. Each relation is denoted as $R(A_1, A_2, ..., A_m)$ where R is the name of the relation and $A_1, A_2, ..., A_m$ are the attributes. Each relation has one primary key and possibly some foreign keys referring to other relations. A key may comprise of one or more attributes.

DEFINITION 1. ***Labeled Directed Graph.*** *Given a relational database D, we define the schema graph of D as a Labeled Directed Graph (LDG) $G = (V, E)$, where each node $r \in V$ represents a relation in database D and each edge $e = r_1 \rightarrow r_2 \in E$ represents a foreign key-candidate key relationship between the relations represented by node r_1 and node r_2. Each edge $e = r_1 \rightarrow r_2$ has a label which comprises of the foreign key-candidate key attributes of the relations that correspond to r_1 and node r_2 respectively.*

Figure 2 shows the labeled directed graph of the sample DBLP database in Figure 1. The node *PaperCitation* corresponds to the relation *PaperCitation* in the database. There are two edges between node *PaperCitation* and node *Paper*, we use different labels to model these two different foreign key-candidate key relationships. Other edge labels are omitted for simplicity.

DEFINITION 2. ***Keyword Node.*** *Given a keyword k, a keyword node, denoted by $r_i\langle a_j, k\rangle$, is defined as the relation that at least one tuple contains the keyword k in some attribute value. r_i denotes the relation and a_j denotes the attribute whose value contain k.*

Each keyword node represents one possible meaning of the keyword. For example, $Paper\langle Title, \text{“}Markov\text{”}\rangle$ represents

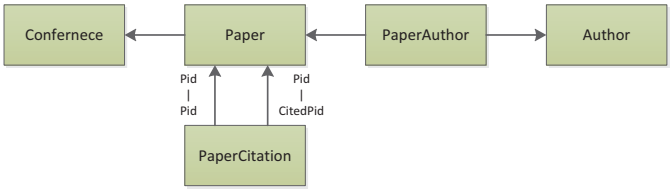

Figure 2: schema graph of the sample DBLP database.

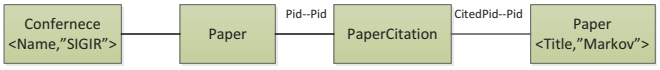

Figure 3: an example connected graph.

that keyword *"Markov"* could be a value of a paper title. In this paper, we use a keyword node to represent one semantics of the keyword.

DEFINITION 3. ***Connected Graph of Relations.*** *Given a labeled directed graph $G = (V, E)$ for a relational database, a connected graph of relations (CGR) J is an undirected graph of relations which satisfies the following two conditions:*

1. *Each node $v \in J$ corresponds to some node $v' \in G$;*
2. *Each undirected edge $(u, v) \in J$ corresponds to some directed edge $(u', v') \in G$ where u and v corresponds to the nodes u' and v' respectively.*

The size of a connected graph J is the number of relations or nodes that it involves. Figure 3 shows an example connected graph that involves two *Paper* relations one *PaperCitation* relation and one $Conference$ relation. The size of this graph is 4.

DEFINITION 4. ***Query Interpretation.*** *An interpretation of a keyword query $Q = \{k_1, k_2, ..., k_l\}$, denoted by QI, is given by a connected graph of relations J such that*

1. *All the nodes in J is comprised of a set of keyword nodes $\mathbb{R} = \{r_1\langle a_1, k_1\rangle, r_2\langle a_2, k_2\rangle, ..., r_l\langle a_l, k_l\rangle\}$ and a set of intermediate nodes;*
2. *All the nodes in $\mathbb{R}$ are connected through intermediate nodes and none of the intermediate node can be removed while all keyword nodes remain connected.*

Problem Statement. Given a keyword query Q over relational database D, we want to find the top-K interpretations with their relevant scores $\langle QI_i, score(QI_i)\rangle$, $i \in [1, K]$ such that $\forall 1 \leq i \leq j \leq K, score(QI_i) \geq score(QI_j)$.

4. PROPOSED FRAMEWORK

In this section, we will describe our approach to answer relational keyword queries. Figure 4 shows the architecture of our proposed framework. There are 3 main phases. In the *first* phase, the system takes a keyword query as input and generates a set of possible interpretations sorted by their relevance to the query. In the *second* phase, the generated interpretations are presented to the user so that s/he can choose which interpretation(s) best match his/her

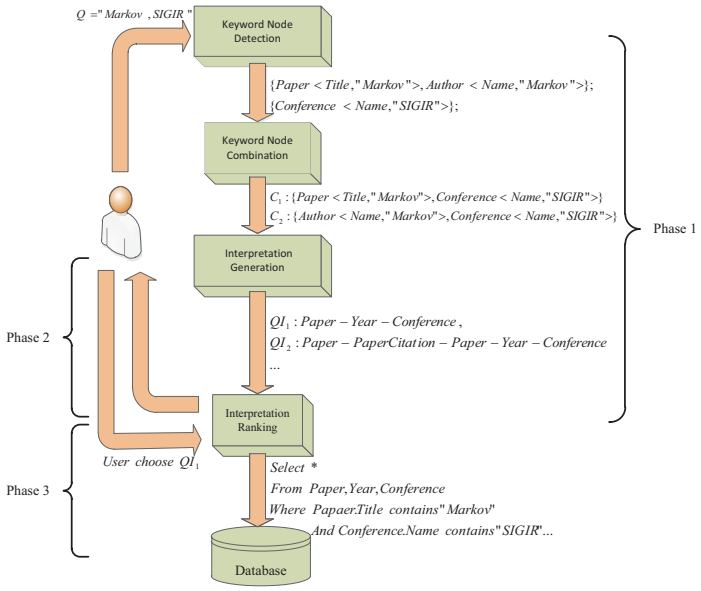

Figure 4: Framework Architecture.

search intention. In the *third* phase, the system translates the keyword query with the selected interpretations into a set of SQL queries to retrieve the results. As our major work focuses on generating and ranking query interpretations, which is the first phase, we illustrate the processes of this phase in detail followed by a running example.

Keyword Node Detection. First of all, in order to generate all possible interpretations of a query Q, we need to figure out all the semantics of each keyword in the query. Recall that a keyword usually appears in values of different attributes and carries different semantics in relational databases. Thereby, for each keyword k in the query, it is necessary to detect each relation r in the relational database to see whether k could appear in tuples from r. If a tuple of r is found to contain k in value of attribute a, we construct a keyword node $r\langle a, k\rangle$ and treat it as one possible semantics of k.

Consider the example in Figure 4 where the user issues a query $Q = \{\text{“}Markov\text{”}, \text{“}SIGIR\text{”}\}$. For keyword *“Markov”*, the detected keyword nodes are $Paper\langle Title, \text{“}Markov\text{”}\rangle$ and $Author\langle Name, \text{“}Markov\text{”}\rangle$. These two nodes indicates that keyword *“Markov”* may refer to two different semantics: (1) a value of an author name; and (2) a value of a paper title. Meanwhile, the only keyword node for keyword *“SIGIR”* (i.e. $Conference\langle Name, \text{“}SIGIR\text{”}\rangle$) denotes that *“SIGIR”* is sure to be a conference name.

It is worth noting that a keyword may appear as values of two different attributes of the same relation. In this case, we construct two keyword nodes for this keyword. Both two keyword nodes represent the same relation but labeled with different attribute names. The *Keyword Node Detection* process is usually implemented based on the inverted index supported by the major RDBMSs [1, 2, 3]. After this process, all the possible semantics of the keywords in the query are detected and grouped by each keyword.

Keyword Node Combination. After we get a set of keyword nodes for each single keyword, we need to select one keyword node from each set and combine them together. This is based on the assumption that every keyword in the query should have one and only one specific semantics, which complies with the human intuition. Take keyword query *“Markov, SIGIR”* as an example, once keyword *“Markov”* is considered as an author name, it cannot be a value of a paper title at the same time. In *Keyword Node Combination* process, we take all the sets of keyword nodes and generate all possible combinations of keyword nodes from each set.

Recall the running example in Figure 4, there are two different combinations of the given keyword nodes. The semantics of the keyword *“SIGIR”* in both keyword node combinations are as a name of a conference; while the keyword *“Markov”* refers to a paper title in the first combination and to an author name in the second one.

Note that the keyword nodes of two different keywords may be equal in a keyword node combination. This is due to two reasons. The first reason is that the two keywords appear together in a tuple of the same relation. In this case, we treat them as one “keyword phrase” and use only one keyword node to represent them. The second reason is that the two keywords appear in the same attribute values of different tuples within a relation. In this case, we need to keep two keyword nodes separately in the combination as the two keywords refer to different instances.

Interpretation Generation. Once the semantics of all the keywords in the query are determined in each keyword node combination, the next question is how to generate interpretations of the query. However, provided the semantics of each keyword, it is quite usual that there is more than one way to connect them by key-foreign key relationships, due to the many to many relationships between two relations. Based on the above observation, the *Interpretation Generation* process inputs each combination of keyword nodes and finds a set of connected graphs of relations that link all the keywords.

The existing techniques on keyword search over relational databases deploy a breadth-first graph traversal strategy on the schema graph to enumerate possible join paths [10]. We adopt this approach as well.

We use the query in the running example again to illustrate how interpretation generation works. Assume that we have already known keyword *“Markov”* refers to the title of a paper and *“SIGIR”* refers to the name of a conference. Figure 4 shows two possible connected graphs of relations: one graph presents that relation $Paper$ and $Conference$ are directly connected representing a paper whose title contains *“Markov”* is accepted in conference *“SIGIR”*; the other one shows that relation $Paper$ and $Conference$ are connected through relation $PaperCite$ and $Paper$ in the meaning of a *“Markov”* paper cites another paper which is accepted in conference *“SIGIR”*.

Interpretation Ranking. In fact, the connected graphs of relations generated in the last process are only a part of query interpretations. This is because the generated query interpretations share the same semantics of all the keywords which are determined by the specific keyword node combination. To generate all possible interpretations of the keyword query, we need to undergo interpretation generation process with each possible combination of keyword nodes. Assume that the average number of keyword nodes for a keyword is x. The number of combinations would be x^l. Further assume that the average number of interpretations for each combination of keyword nodes is y. The total number of

interpretations for a keyword query Q can be as many as $y \cdot x^l$. This large number of interpretations requires an efficient interpretation generating algorithm and an effective ranking scheme so that more relevant interpretations are ranked higher than less relevant ones. Thus, the last process of our proposed framework is to compute the confidence of each query interpretation to match the search intention and output the most possible ones.

Algorithm 1: QI generator(Q, K)

```
   input  : user keyword query Q = keywords[l] and number
            of query interpretations K
   output : Top-K query interpretations of Q
1  TotalQISet ←∅;
2  cur[l]: array of cursors;
3  i ← 0;
4  foreach k_i ∈ Q do
5  |  Maintain a list L_i for keyword nodes of k_i;
6  |  cur[i] = 0;
7  while cur[l − 1] < L_{l−1}.length do
8  |  Let C = a combination of keyword nodes;
9  |  C ←∅;
10 |  for i = 0 to l − 1 do
11 |  |  if r⟨a, k_j⟩ ∈ C && r⟨a, k_i⟩ = L_i.cur[i] && k_i,k_j
   |  |  appear together then
12 |  |  |  update r⟨a, k_j⟩ to r⟨a, k_j + k_i⟩;
13 |  |  else
14 |  |  |  C.add(L_i.cur[i]);
15 |  cur[0]++;
16 |  for i = 0 to l − 2 do
17 |  |  if cur[i] = L_i.length then
18 |  |  |  cur[i] = 0;
19 |  |  |  cur[i + 1]++;
20 |  QISet = getInterpretations(C, MaxSize);
21 |  TotalQISet = TotalQISet ⋃ QISet;
22 return TotalQISet.sort(K);
```

The procedure of our proposed framework is presented in Algorithm 1, where the user input is Q and the number of candidate interpretations of Q. First it builds a list of keyword nodes for each keyword of Q (line 5) and initializes the cursor of the list (line 6). Then it generates a combination of keyword nodes for Q according to the cursors of each list (line 10-14). After that, it updates the cursors (line 15-19) and call the function $getInterpretations()$ to generate interpretations of Q (line 20). $MaxSize$ is the maximum size of query interpretations to be generated. The returned interpretations are put into $TotalQISet$. Once all the cursors are moved to the end of the lists (line 7), it finishes the loop. Last, it computes the relevance of each query interpretation and returned top-K ones to the user (line 22).

5. RANKING INTERPRETATIONS

In this section, we discuss our ranking model to predict the relevance of an interpretation to a keyword query. As mentioned above, in a database with complex schema, the total number of interpretations for some keyword queries may be very large. Therefore, it is necessary to rank them before returning them to the user for his/her to choose the intended true interpretations. Previous studies on keyword search over relational databases proposed different ranking heuristics. However, all of them focus on ranking final results instead of query interpretations. This may lead to the problem that top returned results are ranked with bias towards quite a few interpretations which are not user expected. We take a simple analysis to illustrate this problem. Assume that given a keyword query q, there are two interpretations QI_1 and QI_2 of q such that QI_1 has a size of 1 and generates 100 matching results while QI_2 has a size of 2 and generates 50 matching results. What the user expect is the 50 results generated by QI_2. However, by the approach proposed by [9], all 100 results from QI_1 will be ranked before the 50 expected results from QI_2 (we assume other ranking factors are equal). In this case, the user would find it discouraging to perform a keyword search.

To rank the query interpretations, the existing ranking strategies cannot be applied directly. This is because we only have a set of connected graphs of relations instead of query results. TF-IDF [9, 16, 15] scheme doesn't fit due to the unavailability of details of joining tuples. To solve this problem, we explore information theory and data statistics to capture the factors that reflect the relevance of interpretations to keyword queries. Similar to the interpretation generation process, our ranking strategy can also be divided into two parts: keyword semantics confidence and connected structure confidence.

5.1 Keyword semantics confidence

In this part, we design the metric to predict the confidence of a keyword referring to a semantics. To predict the possibility of a keyword matching some semantics, we propose 3 guidelines:

- **Guideline 1:** The more tuples match the keyword with some semantics, the higher confidence the keyword refers to that semantics;
- **Guideline 2:** The more discriminative the keyword is across the relation which the keyword refers to, the higher confidence the keyword refers to the relation;
- **Guideline 3:** The more important the relation where the keyword appears is, the higher confidence the keyword refers to the relation.

Guideline 1 indicates that a query keyword tends to refer to the semantics with many tuples that contain the keyword and match that semantics. This is inspired by the important role of data statistics in IR ranking, which usually provides a convincing way to capture human intuitions. For instance, when we consider the keyword *"Markov"* in the domain of the database in Figure 1, we tend to believe that *"Markov"* is more likely referring to a paper title than to an author name. This is because the number of papers with their titles containing *"Makrov"* is more than the number of authors with their name as *"Markov"*.

Guideline 2 indicates that a query keyword should not refer to the semantics where too many tuples of the same relation contain the keyword. This is also derived from IR ranking strategies. The reason behind is that if too many tuples contain the keyword with a particular semantics, then that keyword has very low selectivity to the query results.

Guideline 3 indicates that a query keyword tends to refer to the semantics carries important information of the databases. Obviously, all the relations are not equally important in the relational databases [21]. It is reasonable to believe that a keyword has a higher confidence to refer to the more important relations than less important relations.

By incorporating the above guidelines, we define the confidence of a keyword k referring to the value of attribute a_j

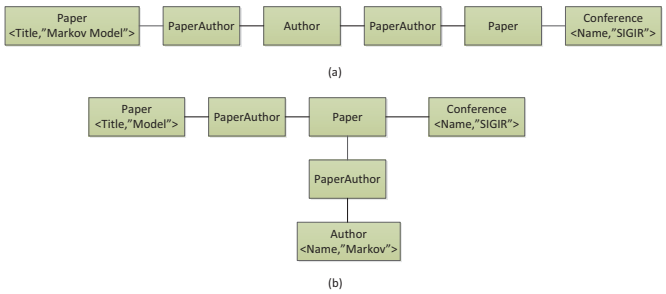

Figure 5: example interpretations for query *"Markov", "Model", "SIGIR".*

of relation R_i as:

$$Score(R_i.a_j, k) = Im(R_i) \cdot \ln \frac{N_{R_i}+1}{N_{R_i.a_j}(k)} \cdot \frac{1}{1-s_1+s_1 \cdot \frac{dl(k)}{avdl}} \cdot \frac{N_{R_i.a_j}(k)}{\sum_{\forall m,n} N_{R_m.a_n}(k)} \quad (1)$$

where $Im(R_i)$ represents the importance of relation R_i, it is set by experts or computed as [21] in advance; N_{R_i} is the number of tuples in relation R_i; $N_{R_i.a_j}(k)$ is the number of tuples in R_i that contain k in attribute a_j; $dl(k)$ represents the average size of a_j attribute values of tuples that contain k in R_i; $avdl$ represents the average size of a_j attribute values of all tuples in R_i; and s_1 is a constant.

In formula 1, to address Guideline 1, the multiplier $\frac{N_{R_i.a_j}(k)}{\sum_{\forall m,n} N_{R_m.a_n}(k)}$ computes the fraction of tuples that contain k with the specific semantics over all the tuples that contain k with all the possible semantics. The multiplier $\ln \frac{N_{R_i}+1}{N_{R_i.a_j}(k)}$ reduces the confidence of keyword k referring to the attribute a_j of relation R_i, with the increasing number of R_i tuples that contain k in a_j to address Guideline 2. Guideline 3 is simply addressed by $Im(R_i)$. The multiplier $\frac{1}{1-s_1+s_1 \cdot \frac{dl(k)}{avdl}}$ is added to penalize the attribute length since longer attribute values have a larger chance to contain the keyword.

We assume that all keywords in the keyword query Q are independent and combine them by the product of semantics confidence for each keyword.

$$Score_a(Q) = (\prod_{k_i \in Q} Score(R_m.a_n, k_i))^{\frac{1}{|Q|}} \quad (2)$$

$|Q|$ is the number of keywords in Q.

5.2 Connected structure confidence

To measure the relevance of a connected graph of relations, the existing techniques simply consider the number of joins in the graph. The reason behind is that smaller number of joins indicates that query keywords are connected in a more intuitive way. For ranking a query interpretation, another factor need to be considered is the distribution of the query keywords. A dense distribution implies that the query keywords are close to each other in semantics while a sparse distribution tends to indicate that keywords are connected by "faraway" relationships. We illustrate the importance of this factor in the following example. Consider the user issues a keyword query *"Markov", "Model", "SIGIR".* Two possible query interpretations are shown in Figure 5. From these two interpretations, we can see that both of two query interpretations consist of same relations and have the same size. The only difference is that in interpretation (a), all three keywords are represented by two keyword nodes while in interpretation (b), each keyword corresponds to one keyword node respectively. From the prospect of semantics, intuitively, the first interpretation will be more likely useful to the user as the keywords together are more cohesive to represent the search intention.

DEFINITION 5. ***(Cohesiveness)*** *To capture the distribution characters of keyword in a query interpretation, we define the keyword cohesiveness in query interpretation QI as:*

$$Coh(QI) = \frac{\sum_{\forall k_i,k_j \in Q} \sqrt{distance(QI,k_i,k_j)}}{size(QI) \cdot |Q|} \quad (3)$$

where $distance(QI, k_i, k_j)$ is the number of joins in the path between keyword nodes that correspond to k_i and k_j; and $Size(QI)$ represents the size of QI. The smaller value of $Coh(QI)$, the more cohesive the keywords are in QI. According to the definition, the cohesiveness of QI_a is $\frac{\sqrt{5}+\sqrt{5}+0}{6*3} = 0.25$ and the cohesiveness of QI_b is $\frac{\sqrt{4}+\sqrt{3}+\sqrt{3}}{6*3} = 0.30$.

To sum up, we draw two intuitions to evaluate the connected structure confidence of an interpretation.

1. The larger size of the interpretation, the less useful it tends to be;
2. The more cohesive the keywords are in the query interpretation, the more useful it tends to be.

Based on the above intuitions, we define the connected structure confidence of an interpretation QI as:

$$Score_b(QI) = \sqrt{1-(\frac{size(QI)-1}{MaxSize})^2} \cdot e^{-Coh(QI)} \quad (4)$$

where $size(QI)$ is the size of interpretation QI, $MaxSize$ is the maximum size of the interpretations, which is set as a threshold.

Finally, the confidence of a query interpretation is obtained by combining both two types of confidence. To calculate top-K interpretations of a keyword query, we generate all possible interpretations and rank them based on this confidence of relevance.

$$Score(QI) = Score_a(Q) \cdot Score_b(QI) \quad (5)$$

6. INDEX CONSTRUCTION

In this section, we explore the index techniques to speed up the query interpretation generating and ranking. Current major RDBMSs support an inverted index called full-text index which gets some keywords and the attribute names from the users as input, and returns all the tuples whose attribute values contain the keywords. However, for query interpretation generating and ranking, what we need is the statistics of tuples instead of tuples themselves. Thereby, we scan the database and build two indices in order to speed up our computation.

The first index built is called keyword node list, which retrieve the a list of keyword nodes of the input keyword. In particular, we have designed and evaluated three candidates for the inverted list: (1) **KNode**, the most basic index which

stores only keyword nodes; (2) **KNodeNum**, which stores an extra integer to denote the number of tuples that contain the keyword compared to **KNode**; (3) **KNodeNumLen**, which extra stores a factor $dl(k)$ (in Formula 1) associated with this keyword node compared to **KNodeNum**. **KNodeNumLen** provides the most efficient computation of keyword semantics confidence, as it costs the least index lookup time; In contrast **KNode** and **KNodeNum** need extra lookup of the full-text index to compute $dl(k)$. Given a keyword k, the keyword node list returns a set of nodes, each of which has some tuple that contains the input keyword and is in form of a tuple $\langle r\langle a, k\rangle, Num, dl(k)\rangle$. Each term in the tuple has been explained as above.

The second index built is called path table, which stores paths that connect each two input keyword nodes in the schema graph. In query interpretation generation phase, we use the path table index to find query interpretations of a keyword query. As duplicate relations could appear in the paths between two input nodes, the number of paths would be very large (data bound). To avoid this problem, we only stores the paths with size smaller than $MaxSize$. Since the number of nodes in schema graphs is usually quite small (e.g. 5 in our example and 8 in TPC-H[2]), the size of the path table is reasonable. Given two keyword nodes in the schema graph, the path table returns a set of paths that connect the two input nodes.

7. EXPERIMENT PLAN

To evaluate the effectiveness and efficiency of our proposed framework, we design extensive experiments on large scale real world datasets.

Data Set. We use two real world datasets: the Internet Movie Database(**IMDB**[3]) 90**MB**, where around $200,000$ movies of recent years are selected in our dataset. Each movie contains information like title, rating, director, cast, etc. (2) **DBLP** 520**MB**, which contains publications since 1990.

Query Set. We set 18 queries for each of the datasets. Based on the number of keywords in the queries, we divide these queries into three clusters: short queries with 2 or 3 keywords, medium queries with 4 or 5 keywords and long queries with 6 keywords. For each cluster of the queries, the selectivity and the ambiguity of the keywords are chosen to range a lot.

Processing Time. To evaluate the efficiency of the query interpretation generating and ranking, we run our algorithm 10 times and collect the average processing time on hot cache for each test query. Then we evaluate the same keyword query on the existing keyword search engine 10 times and collect the average processing time. We compare the two processing time. If the interpretation proceeding time is much shorter than the result processing time, then our framework is proved to be efficient.

Quality of the Query Interpretations. In order to measure the quality of the query interpretations returned by our approach, we select the queries with great keyword ambiguities to conduct a user study. To conduct a fair evaluation of our approach, we are aware of two things. *First*, we invite both experts and novices in participating the task of scoring the query interpretation. For DBLP, we ask three CS research students and three undergraduates in other faculties; for IMDB, we ask three movie fans and three non-fans. The participants are shown the matching results of each query, the best 5 query interpretations together with the corresponding sample query result. *Second*, the participants are asked to score the quality of each query interpretation by using the Cumulated Gain-based evaluation (CG) metric [12] (from 0 to 5 points, 5 means best while 0 means worst). In contrast to traditional metrics like precision and recall which adopts a binary judgement (yes or no), CG is aware of the fact that all results are not of equal relevance to user. We calculate the average scores for best-3 and best-5 query interpretations. To evaluate the quality of the query interpretations, we compare the average scores for best-3 interpretations with the scores raised by the participants. The higher the average scores are, the higher the quality is. To evaluate the effect of our ranking scheme, we compare the two set of average scores. If the average scores for best-5 is higher than the average scores for best-3, then we can conclude that our proposed ranking scheme is effective.

[2]http://www.tpc.org/tpch/

[3]http://www.imdb.com/interfaces

8. CONCLUSION

Query interpretation is a critical issue in keyword search over relational databases that has yet received very little attention. Existing works either focus on ranking results directly based on an intuitive model, or predict query interpretations using keyword position information and external knowledge, making users difficult to find desired results if the expected query interpretations failed to be detected. In this paper, we proposed a 3 phase keyword search paradigm that focused on query interpretation generating and ranking. We modeled the databases schema as a labeled directed graph and presented the framework to generate all possible query interpretations for a keyword query. Then we analyzed the challenges to rank all the interpretations and proposed some guidelines to compute the confidence of an interpretation to be the user desired. On the basis of that, we designed a ranking model to incorporated those guidelines. We designed extensive experiments and the preliminary experimental results shew that our proposed keyword query interpretations can help users perform an effective, efficient keyword search.

9. ACKNOWLEDGMENTS

Zhifeng Bao was partially supported by the Singapore Ministry of Education Grant No. R252-000-394-112 under the project name of UTab.

10. REFERENCES

[1] Db2 text information extender. `http://www.ibm.com/developerworks/data/tutorials/dm-0810shettar/index.html`.

[2] Microsoft sql server 2008 r2. `http://msdn.microsoft.com/en-us/library/ms142571.aspx`.

[3] Mysql. `http://dev.mysql.com/doc/refman/5.5/en/fulltext-search.html`.

[4] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: authority-based keyword search in databases. VLDB '04, pages 564–575.

[5] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, and Y. Velegrakis. Keyword search

over relational databases: a metadata approach. SIGMOD '11, pages 565–576.

[6] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. ICDE'07, pages 836–845.

[7] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, pages 195–207, 1972.

[8] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. SIGMOD '07, pages 305–316.

[9] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. VLDB '2003.

[10] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. VLDB '02.

[11] A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using banks. ICDE '02.

[12] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 2002.

[13] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. VLDB '05, pages 505–516.

[14] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. PODS '06, pages 173–182.

[15] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. SIGMOD '06.

[16] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. SIGMOD '07.

[17] A. Sanjay, C. Surajit, and D. Gautam. Dbxplorer: A system for keyword-based search over relational databases. ICDE '02.

[18] A. Termehchy and M. Winslett. Using structural information in xml keyword search effectively. *ACM Trans. Database Syst.*, 36(1):4:1–4:39, Mar. 2011.

[19] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. SIGMOD '07.

[20] Y. Xiaohui and S. Huxia. Ranking keyword search results based on collective importance. VLDB '12.

[21] X. Yang, C. M. Procopiuc, and D. Srivastava. Summarizing relational databases. *Proc. VLDB Endow.*, 2(1):634–645, Aug. 2009.

[22] J. X. Yu, L. Qin, and L. Chang. Keyword search in relational databases: A survey. *IEEE Data Eng. Bull.*, pages 67–78, 2010.