Semantic Path Ranking Scheme for Relational Keyword Queries

Zhong Zeng[1], Zhifeng Bao[2], Gillian Dobbie[2], Mong Li Lee[1], Tok Wang Ling[1]
[1]National University of Singapore
[2]University of Tasmania
[3]University of Auckland
{*zengzh, leeml, lingtw*}*@comp.nus.edu.sg*;
zhifeng.bao@utas.edu.au; *g.dobbie@auckland.ac.nz*

Abstract. Existing works on keyword search over relational databases typically do not consider users' search intention for a query and return many answers which often overwhelm users. We observe that a database is in fact a repository of real world objects that interact with each other via relationships. In this work, we identify four types of semantic paths between objects and design an algorithm called *pathRank* to compute and rank the results of keyword queries. The answers are grouped by the types of semantic paths which reflect different query interpretations, and are annotated to facilitate user understanding.

1 Introduction

Keyword search over relational databases enables users to pose queries without learning the query languages or database schemas, and has become a popular approach to access database information [1, 5, 3, 8, 9, 2, 6, 7, 4, 11]. Existing works use a data graph where each node denotes a tuple and each undirected edge denotes a foreign key-key reference [6, 7, 11]. An answer to a keyword query is a minimal connected subgraph of tuples which contains nodes that match keywords in the query. Since the keywords in a query may match nodes which are connected by many paths in the data graph, many answers are returned, with possibly complex subgraphs whose meanings are not easy to understand.

One approach to address the above problem is to rank the query answers. The methods range from simple heuristic rules such as ranking the answers based on their sizes [5], to using the TF-IDF model [3, 8, 9], and the Random Walk model [6, 7, 11]. Another approach is to organize query answers into clusters so that users can explore the relevant clusters first [10]. However, none of these approaches consider the semantics of the answers.

We observe that a relational database is, in fact, a repository of real world objects that interact with each other via relationships. When a user searches for some target object, s/he is interested in objects that are related in some way to the target object. In this work, we utilize the compact Object-Relationship-Mixed (ORM) data graph [12] to model tuples in a relational database, and identify four types of semantic paths where a pair of nodes in the graph can be connected. These semantic paths form different interpretations of the query

answers. Based on these paths, we develop an algorithm to compute and rank the answers of keyword queries. We group the query answers by the types of semantic paths to reflect different query interpretations, and annotate each answer to facilitate user understanding. Experimental results demonstrate that our semantic path-based approach is able to rank answers that are close to users' information needs higher compared to existing ranking methods.

2 Motivating Example

Fig. 2 shows the ER diagram of the student registration database in Fig. 1. Based on the ER model, we see that the database comprises of `Student`, `Course`, `Lecturer` and `Department` objects that interact with each other via the relationships `Enrol`, `Teach`, `PreReq` and `AffiliateTo`. Suppose a user issues the query Q = {`Java DB`}. The keywords match two tuples in the Course relation, i.e., <$cs421$, DB, $l1$> and <$cs203$, $Java$, $l1$>, corresponding to course objects with identifier $cs421$ and $cs203$. Based on the ER diagram, these two objects are related via the relationships PreReq, Enrol and Teach as follows:

a. Pre-requisite of a course (Pre-Req), e.g., $cs203$ is a pre-requisite of $cs421$.
b. Students who are enrolled in both courses (Enrol), e.g., student $s1$ (`Mary Smith`) is enrolled in both courses <$cs421$, DB, $l1$> and <$cs203$, $Java$, $l1$>.
c. Lecturers who teach both `Java` and `DB` (Teach), e.g., lecturer $l1$ (`Steven Lee`) teaches both courses <$cs421$, DB, $l1$> and <$cs203$, $Java$, $l1$>.

Each of these relationships suggest objects that interest the user. We can annotate the answers by the relationships as shown in Fig. 3. We will explain the these annotations in the next Section.

Student

SID	Name
s1	Mary Smith
s2	John Depp

Enrol

TupleID	SID	Code	Grade
e1	s1	cs421	B
e2	s1	cs203	A
e3	s2	cs203	B

Lecturer

StaffID	Name
l1	Steven Lee

Department

DeptID	Name
d1	CS
d2	IS

Course

Code	Title	StaffID
cs421	DB	l1
cs203	Java	l1

PreReq

TupleID	Code	PreqCode
p1	cs421	cs203

AffiliateTo

TupleID	StaffID	DeptID
a1	l1	d1
a2	l1	d2

Fig. 1. Example student registration database

3 Preliminaries

The work in [12] utilizes database schema constraints to classify relations into object relation, relationship relation, component relation and mixed relation. An object (relationship) relation captures single-valued attributes of an object (relationship). Multivalued attributes of an object (relationship) are captured in the component relations. A mixed relation contains information of both objects and relationships, which occurs when we have a many-to-one relationship.

The *Object-Relationship-Mixed (ORM) data graph* [12] is an undirected graph $G(V, E)$. Each node $v \in V$ has an id, a $type \in \{object, relationship, mixed\}$, and a relation name $name$. A node also has a set $tupleIDs$ containing the ids of

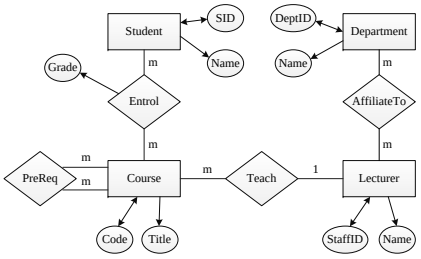


Fig. 2. ER diagram of Fig. 1

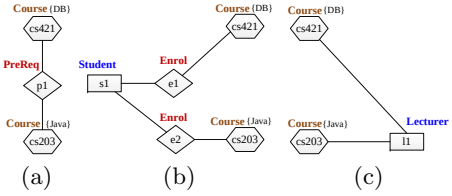


Fig. 3. Annotated answers with node types for Q = {Java DB}

tuples from an object (or relationship, or mixed) relation together with tuples from its associated component relations. Fig. 4 shows the ORM data graph of Fig. 1 comprising object nodes (rectangles), relationship nodes (diamonds) and mixed nodes (hexagons). Each node has an id and a relation name, e.g., the mixed node with id $cs421$ occurs in relation `Course`.

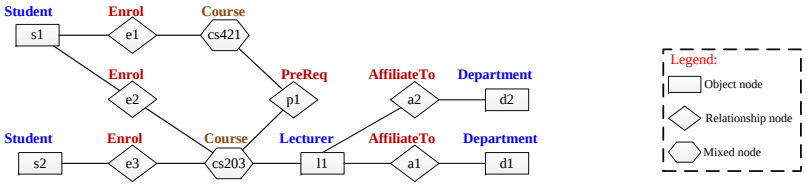


Fig. 4. ORM data graph of the database in Fig. 1

Given a keyword query $Q = \{k_1, k_2, \cdots, k_n\}$, where $k_i, i \in [1, n]$ denotes a keyword, we say k *matches* a node v in the ORM data graph if k occurs in some tuple in v. We call v a matched node for k.

We now analyze the various ways a pair of object/mixed nodes can be connected in the graph. Let u and v be two nodes in the ORM data graph G, and P be the set of paths between u and v. Each path $p \in P$ is a sequence of connected nodes $<u, \cdots, v>$. The length of a path p is given by the number of nodes in p, denoted by $|p|$. We can form a new path $p = <v_a, \cdots, v_b>$ by joining two paths p_1=$<v_a, \cdots, v_c>$ and $p_2 = <v_c, \cdots, v_b>$ over a common node v_c; we say that p can be decomposed into sub-paths p_1 and p_2. We call the paths between two object/mixed nodes *semantic paths* since they capture the semantics of objects and relationships. These paths can be classified into one of the following types:

Simple Path. A path p between u and v is a simple path if u and v are object/mixed nodes, and all the nodes in p have distinct relation names.

$$sp(u, v, p) = \begin{cases} \text{true} & \text{if } u.type = object/mixed \text{ and } v.type = object/mixed \\ & \text{and } \forall b \in p,\ b.name \text{ is distinct} \\ \text{false} & \text{otherwise} \end{cases}$$

For example, the path p=$<s2, e3, cs203, l1>$ between the two object nodes $s2$ and $l1$ in Fig. 4 is a simple path.

Recursive path. A path p between u and v is a recursive path if u and v are both object nodes or mixed nodes and have the same relation name, and all the object/mixed nodes in the path p have the same relation name as u and v.

$$rp(u,v,p) = \begin{cases} \text{true} & \text{if } (u.type = v.type = object \text{ or } u.type = v.type = mixed) \\ & \text{and } \forall b \in p \text{ such that } b.type = object/mixed, \text{we have} \\ & b.name = u.name = v.name \\ \text{false} & \text{otherwise} \end{cases}$$

For example, the path p=<$cs421, p1, cs203$> between mixed nodes $cs421$ and $cs203$ in Fig. 4 is a recursive path.

Palindrome path. A path p between u and v is a palindrome path if both u and v have the same relation name, and we can find some object/mixed node $c \in p$ such that the nodes in the paths from c to u, and c to v have the same sequence of relation names.

$$pp(u,v,p) = \begin{cases} \text{true} & \text{if } u.name = v.name \text{ and } \exists \text{ object/mixed node } c \in p \\ & \text{s.t. p can be decomposed into 2 sub-paths} \\ & p_1 =< u, b_1, \cdots, b_j, c >, p_2 =< c, b'_j, \cdots, b'_1, v > \text{where} \\ & \text{both } p_1, p_2 \text{ are simple paths, and } b_i.name = b'_i.name \\ & \quad \forall b_i \in p_1, b'_i \in p_2, 1 \leq i \leq j \\ \text{false} & \text{otherwise} \end{cases}$$

For example, the path p=<$cs203, e2, s1, e1, cs421$> between the mixed nodes $cs421$ and $cs203$ in Fig. 4 is a palindrome path as it can be decomposed into two simple sub-paths p_1=<$cs203, e2, s1$> and p_2=<$s1, e1, cs421$>.

Complex path. Any path that does not satisfy the conditions for the above three semantic path types is a complex path. A complex path is essentially a combination of simple paths and recursive paths, and has a path length $|p| \geq 3$.

The path p=<$s2, e3, cs203, p1, cs421, l1$> in Fig. 4 is a complex path as it is a combination of two simple paths and one recursive path.

4 Proposed Ranking Scheme

In this section, we describe our method called *pathRank* to compute and rank keyword query answers. We first generate Steiner trees that contain all the query keywords. Then we augment the relationship matched nodes in the Steiner trees with their associated object and mixed nodes. Finally we rank the Steiner trees based on type of semantic paths they contain. We give the highest score to simple and palindrome paths because they are more intuitive and informative. Complex paths have the lowest scores since they require more user effort to understand.

Let $Obj(k)$ and $Rel(k)$ be the sets of object and relationship nodes that match keyword k in the ORM data graph. Note that if k matches the object part of a mixed node u, then we add u to $Obj(k)$. Otherwise, if k matches the relationship part of u, we add u to $Rel(k)$.

Given two nodes u and v that match keyword k_i and k_j respectively in a Steiner tree T, we have the following cases:

a. Both $u \in Obj(k_i)$ and $v \in Obj(k_j)$. We determine the type of the path between u and v as described in Section 3.
b. Either $u \in Rel(k_i)$ or $v \in Rel(k_j)$. Without loss of generality, suppose $u \in Rel(k_i)$ and $v \in Obj(k_j)$. Let S_u be the set of object/mixed nodes that are directly connected to u, and p' be the path between v and some node $s \in S_u$ that has the highest score. Then the type of the semantic path between u and v is given by the type of path p'.
c. Both $u \in Rel(k_i)$ and $v \in Rel(k_j)$. Let S_u and S_v be the sets of object/mixed nodes that are directly connected to u and v respectively. Let p' be the path between $s \in S_u$ and $t \in S_v$ that has the highest score. Then the type of the semantic path between u and v is given by the type of path p'.

Let V be the set of matched nodes in a Steiner tree T and $C_2^{|V|}$ be the number of node pairs in V. The score of T w.r.t keyword query Q is defined as follows:

$$score(T,Q) = \begin{cases} \frac{\sum_{u,v \in V, u.id < v.id} \ pathscore(u,v,p)}{num(u,v,p) * C_2^{|V|}} & |V| > 1 \\ 1 & |V| = 1 \end{cases}$$

where $num(u,v,p)$ is the number of object/mixed nodes in the path p between nodes u and v in V, and $pathscore(u,v,p)$ is the score of the path p. Note that our proposed ranking scheme considers the semantic paths between matched nodes as well as the number of participating objects in the Steiner tree.

Algorithm 1 shows the details of *pathRank*. We first classify the matched nodes for each keyword and generate the Steiner trees (Lines 3-5). For each tree T, we check every matched node v for keyword k. If $v \in Rel(k)$, i.e., the keyword matches a relationship node or the relationship part of a mixed node, then we add the object/mixed nodes that are directly connected to v in the ORM data graph and the associated edges into T (Lines 6-11). Next, we determine the score of the path between matched nodes u and v in T (Lines 12-31).

5 Performance Study

We implement the algorithms in Java, and carry out experiments on an Intel Core i7 3.4 GHz with 8GB RAM. We use a subset of the real world ACM Digital Library publication dataset from 1995 to 2006. There are 65,982 publications and 106,590 citations. Fig. 6 shows the ER diagram for this dataset.

We compare our semantic path ranking method (`Path`) with the following ranking schemes used in state-of-the-art relational keyword search such as Discover [5], BANKS [6] and SPARK [9]:

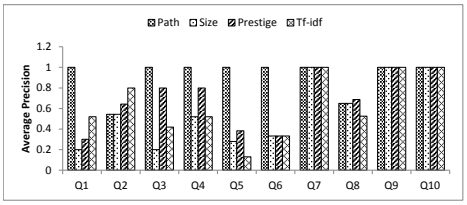
a. Number of nodes in answer (`Size`) [5].
b. Node prestige and proximity (`Prestige`) [6].
c. TF-IDF similarity between query and answer (`Tf-idf`) [3, 9].

Fig. 7 shows the keyword queries used in our experiments. We show these queries together with the ER diagram of the database to 10 users and obtain their possible search intentions. For each search intention, we generate the SQL statements to retrieve the results from the database to form the ground truth.

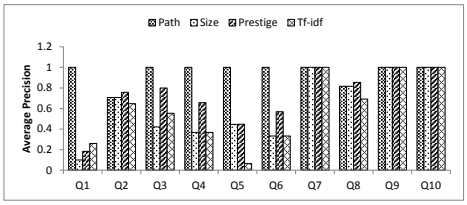
Algorithm 1: *pathRank*

Input: keyword query $Q = \{k_1, ...k_n\}$, $maxSize$, ORM data graph G
Output: answer set $Answer$

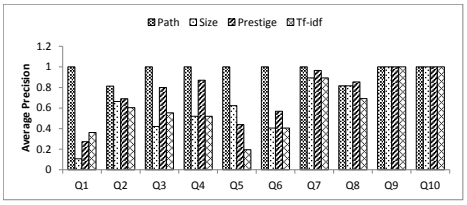
```
1  Answer ← ∅;
2  for i = 1 to n do
3  |  Let Obj(k_i) be the set of object/mixed nodes in G that match k_i;
4  |  Let Rel(k_i) be the set of relationship/mixed nodes in G that match k_i;
5  Answer = generateSteinerTree(Q, G, maxSize);
6  foreach Steiner Tree T ∈ Answer do
7  |  Let V be the set of matched nodes in T;
8  |  foreach v ∈ V do
9  |  |  if v ∈ Rel(k) then
10 |  |  |  add object/mixed nodes that are directly connected to v in G into T;
11 |  |  |  add the associated edges in G into T;
12 |  foreach u, v ∈ V do
13 |  |  S_u ← ∅; S_v ← ∅;
14 |  |  if u ∈ Obj(k_i) then
15 |  |  |  add u into S_u;
16 |  |  else if u ∈ Rel(k_i) then
17 |  |  |  add object/mixed nodes that are directly connected to u in T into S_u;
18 |  |  if v ∈ Obj(k_j) then
19 |  |  |  add v into S_v;
20 |  |  else if v ∈ Rel(k_j) then
21 |  |  |  add object/mixed nodes that are directly connected to v in T into S_v;
22 |  |  score = 0;
23 |  |  foreach s ∈ S_u, t ∈ S_v do
24 |  |  |  z = pathscore(s, t, p);
25 |  |  |  if score < z then
26 |  |  |  |  score = z;
27 |  |  pathscore(u, v, p) = score;
28 |  |  Let num be the number of object/mixed nodes between u and v;
29 |  |  T.score += pathscore(u, v, p) * num;
30 |  T.score = T.score/(|V| * (|V| − 1)/2);
31 return Sort(Answer);
```



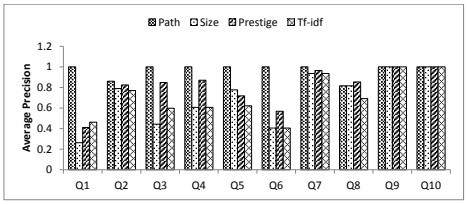
(a) Top-5 answers



(b) Top-10 answers



(a) Top-20 answers



(b) Top-50 answers

Fig. 5. Average precision of ranking schemes for top-k answers

Fig. 5 shows the average precision of four ranking methods for the queries in Table 7 when we vary k. We observe that `Path` is able to achieve a higher average precision compared to `Size`, `Prestige` and `Tf-idf` for most of the queries. In fact, the superiority of `Path` increases significantly as k decreases. All the ranking schemes are able to retrieve the relevant answers when k is equal to 50. However, `Size`, `Prestige` and `Tf-idf` start to miss relevant answers as k decreases. Note that for query $Q9$ and $Q10$, all the ranking schemes achieve an average precision of 1 because their search intentions are straightforward, and the relevant answers are ranked on the top by all the ranking schemes. Thus, we can see that `Path` is more effective than the other ranking schemes when the queries are ambiguous.

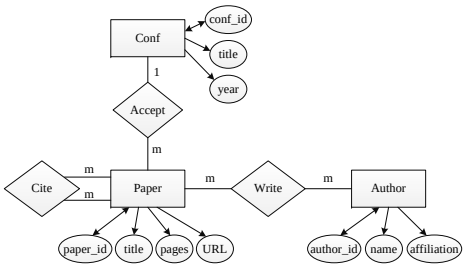


Fig. 6. ER diagram of the ACM Digital Library dataset

Queries	
Q1	Streaming QSplat hierarchical wavelets
Q2	Texture synthesis painting
Q3	lambda calculus resource usage
Q4	Jeffrey Naughton David DeWitt
Q5	Gray Alexander
Q6	Alla Sheffer Hugues Hoppe
Q7	Brad Calder Timothy Sherwood
Q8	Yannis Papakonstantinou
Q9	Jagadish query
Q10	Stonebraker SIGMOD

Fig. 7. Queries used

Table 1 shows two sample answers for query $Q3$ and their rankings by the different schemes. The keywords in this query match two paper titles. The first answer indicates that these two papers are related via 2 Paper-Cite relationships, while the second answer indicates that these two papers are related via a third paper published in the same conference and cites one of these papers. We observe that although the second answer is complex and not easy to understand, it is ranked higher by `Size` and `Prestige`. `Tf-idf` gives the similarly ranks to these two answers without considering the type of the semantic paths. In contrast, `Path` ranks the first answer much higher because it contains a recursive path while the second answer contains a complex path. Fig. 8 shows the corresponding annotated answers output by our approach to facilitate user understanding.

	Sample query answers	`Size[5]`	`Prestige[6]`	`Tf-idf[3,9]`	`Path[Ours]`
(a)	p1: `Resource usage` analysis p2: Once upon a type p3: A call-by-need `lambda calculus`	9	7	10	4
(b)	c1: POPL'04 p4: Channel dependent types for higher-order... p1: `Resource usage` analysis p5: ... evaluation for typed `lambda calculus`...	2	5	11	21

Table 1. Ranking of two sample answers for query $Q3$

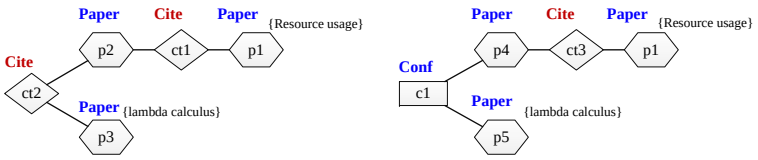


Fig. 8. Annotated answers for query $Q4$

6 Conclusion

In this paper, we have proposed a semantic approach to help users find informative answers in relational keyword search. This is achieved by capturing the semantics of objects and relationships in the database with the ORM data graph. We examined how objects are related and identified four types of semantic paths between object/mixed nodes. Based on these semantic paths, we have developed an algorithm to compute and rank the answers of keyword queries. We group the answers by the types of semantic paths and annotate them to facilitate user understanding. Experimental results on a real world dataset demonstrated the effectiveness of our path-based approach.

References

1. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.
2. S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, 2011.
3. V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, 2003.
4. V. Hristidis, H. Hwang, and Y. Papakonstantinou. Authority-based keyword search in databases. *ACM Trans. Database Syst.*, 2008.
5. V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *VLDB*, 2002.
6. A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.
7. V. Kacholia, S. Pandit, and S. Chakrabarti. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.
8. F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.
9. Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, 2007.
10. Z. Peng, J. Zhang, S. Wang, and L. Qin. TreeCluster: Clustering results of keyword search over databases. In *WAIM*, 2006.
11. X. Yu and H. Shi. CI-Rank: Ranking keyword search results based on collective importance. In *ICDE*, 2012.
12. Z. Zeng, Z. Bao, M. L. Lee, and T. W. Ling. A semantic approach to keyword search over relational databases. In *ER*, 2013.