Using Keywords to Find the Right Path through Relational Data*

Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone

Università Roma Tre, Italy
{dvr,maccioni,torlone}@dia.uniroma3.it

Abstract. Keyword search is emerging as the standard way to access information of any kind. Following this trend, several approaches to keyword search over relational databases have been proposed recently. Most of them build tree-shaped solutions that connect tuples matching the given keywords. In this paper, we propose a novel technique to this problem that generates solutions by simply combining joining paths. In this way, the complexity of the process is reduced and the query engine of the underlying RDBMS can be fully exploited. Several experiments show a very good behavior with respect to other approaches in terms of both effectiveness and efficiency.

1 Introduction

With the size and availability of data constantly increasing, it is usually hard for users to retrieve the information they really need. In a frequent scenario, the problem is caused by the fact that the search is over structured data stored in database systems. Then, the users must be aware of the schema and know the syntax of a specific query language. For this reason, alternatives ways to access information are increasingly capturing the attention of database researchers. Among them, several approaches to keyword-based search over structured and semi-structured data have been proposed recently. Typically, keyword-based search over relational data involve the following steps: (i) selection of the tuples whose values match the input keywords, (ii) generation of tree-shaped solutions built by joining the retrieved tuples, and (iii) ranking of the solutions according to a relevance criteria. At the end, only the top-K solutions are usually returned.

In this paper, we present a novel technique for keyword-based search over relational databases that builds the best results using a path-oriented strategy. This process is inspired by an earlier work on keyword search over semantic data [3]. Unlike many current approaches that do not exploit enough the capabilities of the underlying system [9], our technique retrieves efficiently the tuples that match the given keywords by taking full advantage of the RDBMS in which the data are stored. We have tested the approach over available benchmarks and have observed a very good behavior with respect to other proposals in terms of both effectiveness and efficiency.

* Extended Abstract

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

The most prominent work in this area can be classified in *schema-based* and *schema-free* approaches. Schema-based approaches [6,8,9] are common for relational databases. Here, an answer to a query is a tree, usually called *joined tuples tree* (JTT), composed by joining tuples. A common drawback of these approaches is that the search is converted into a set of SQL statements that generate candidate answers. All queries are then executed but some of them can return empty results, leading to inefficiency, which is likely to worse with the size of the data set. Schema-free approaches (*e.g.* [7]) are more general as they operate on arbitrary graph-shaped data. A general approach searches for the top-K connected trees, each representing a *(minimal) Steiner tree* [4]. A relevant drawback is that finding a minimal Steiner tree is an NP-Hard problem [4]. Hence this methods rely on complex heuristics aimed at generating approximations of Steiner trees. Our approach tries to overcome these limitations, trading-off accuracy in top-K computation, and scaling seamlessly with the size of the input.

In the rest of the paper, we first introduce, in Section 2, a path-oriented model for relational databases. Based on this model, in Section 3 we present our technique for answering keyword search queries and illustrate, in Section 4, some experimental results.

2 Path-oriented Modeling

A relational database $\mathcal{RDB}$ can be modeled by a pair of graphs $\langle \mathcal{SG}, \mathcal{DG} \rangle$ representing the schema and the instance of $\mathcal{RDB}$, respectively, as follows.

Definition 1 (Schema Graph) *Given a relational schema* $\mathcal{RDB}$-SC $= \langle \mathcal{T}, \mathcal{A} \rangle$, *where* $\mathcal{T}$ *is a set of relation schemas* T_i *over a set of attributes* $\{T_i.A_1, \ldots, T_i.A_k\}$ *and* $\mathcal{A}$ *is the union of all attributes, a schema graph* $\mathcal{SG}$ *on* $\mathcal{RDB}$-SC *is a directed graph* $\langle V, E \rangle$ *where* $V \subseteq \{\mathcal{T} \cup \mathcal{A}\}$, $E \subseteq \{(\mathcal{T} \times \mathcal{A}) \cup (\mathcal{A} \times \mathcal{A})\}$ *and there is an edge* $(a, b) \in E$ *if (i)* $a \in T$ *and* b *is an attribute of* a, *(ii)* $a \in \mathcal{A}$ *belongs to a key of* T_i *and* b *is an attribute of* T_i, *and (iii) there is a foreign key between* a *and* b.

T_1: **Person**

	name	area
t_1	Watson	Database
t_2	Lenzerini	Database
t_3	Date	Database
t_4	Hunt	Inf. Systems

T_2: **Affiliated**

	professor	department
t_5	Watson	x123
t_6	Lenzerini	cs34
t_7	Date	cs34
t_8	Hunt	m111

T_3: **Department**

	id	dname	director
t_9	x123	CS	Watson
t_{10}	cs34	IE	Hunt
t_{11}	ee67	EE	Date
t_{12}	m111	ME	Hunt

T_4: **Author**

	name	publication
t_{13}	Lenzerini	Data Integration
t_{14}	Date	Foundation Matters

T_5: **Publication**

	title	year
t_{15}	Data Integration	2002
t_{16}	Foundation Matters	2002

Fig. 1. A relational database

For instance, in the relational database $\mathcal{RDB}$-$SC_1 = \langle \mathcal{T}_1, \mathcal{A}_1 \rangle$ depicted in Fig. 1 [1] we have: $\mathcal{T}_1 = \{$*Person, Affiliated, Department, Author, Publication*$\}$, and $\mathcal{A}_1 = \{$*Person.name, Person.area, Affiliated.professor*, $\ldots\}$. In the figure, underlined attributes are primary keys and we have the following foreign key constraints: *Affiliated.professor* $\xrightarrow{fk}$ *Person.name*, *Affiliated.department* $\xrightarrow{fk}$ *Department.id*, *Department.director* $\xrightarrow{fk}$ *Person.name*, *Author.name* $\xrightarrow{fk}$ *Person.name*, and

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Author.publication $\xrightarrow{fk}$ *Publication.title*. The schema graph $\mathcal{SG}_1$ on $\mathcal{RDB}$-SC_1 is depicted in Fig. 2. In a schema graph the sources represent the relations of a

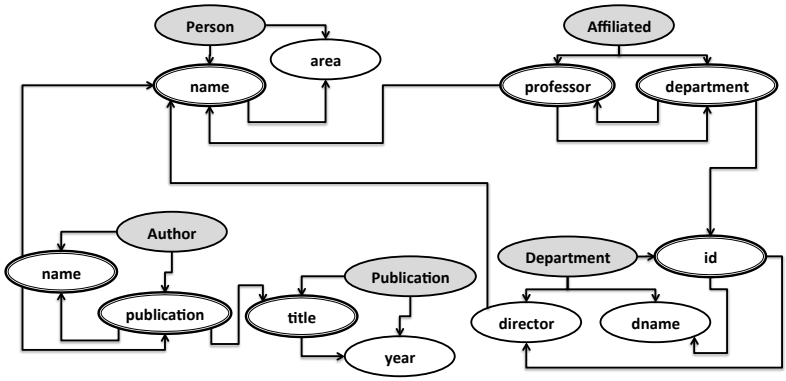


Fig. 2. The schema graph for the relational database in Fig 1.

relational database schema (grey vertices) and the so-called *Schema-Paths* track the relationships between attributes, according to primary and foreign keys.

Definition 2 (Schema-Path) *Given a schema graph $\mathcal{SG} = \{V, E\}$, a schema-path is a sequence $sp = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_f$ where $(v_i, v_{i+1}) \in E$ and v_1 is a relation node.*

In Fig. 2 a schema-path *sp* is *Affiliated* → *Affiliated.professor* → *Person.name*. We introduce an injective function called *index*, denoted by idx, that maps each tuple t of a relation for a schema $T \in \mathcal{T}$ to a *tuple-id* (tid for short) belonging to an uncountable set $\mathbb{U}$.

Definition 3 (Data Graph) *Given a relational database instance $\mathcal{RDB}$-I $= \langle \mathcal{T}, \mathcal{A}, I, \mathcal{D} \rangle$, where I ($\mathcal{D}$) is the set of all tids (values) occurring in the database, a data graph $\mathcal{DG}$ on $\mathcal{RDB}$-I is a directed graph $\langle V, E \rangle$ where $V \subseteq \{\mathcal{T} \cup \mathcal{A} \cup I \cup \mathcal{D}\}$, $E \subset \{(\mathcal{T} \times \mathcal{A}) \cup (\mathcal{A} \times I) \cup (I \times \mathcal{D})\}$ and there is an edge $(v_1, v_2) \in E$ if: (i) $v_1 \in \mathcal{T}$ and v_2 is an attribute of v_1, (ii) $v_1 \in \mathcal{A}$ belongs to a key of T_i and v_2 is the tid of a tuple for T_i, and (iii) v_1 is a tid and v_2 is a value of a tuple t such that $v_1 = \text{idx}(t)$.*

Fig. 3 shows an example of data graph for the database of Fig. 1.

Definition 4 (Data-Path) *Given a data graph $\mathcal{DG} = \{V, E\}$, a data-path is a sequence $dp = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$ where $v_i \in V$, $(v_i, v_{i+1}) \in E$, v_1 is a relation node, v_2 is an attribute node, v_3 is a tuple-id and v_4 is a data value.*

In Fig. 3 a data-path dp_k is *Person* → *Person.name* → t_1 → *Watson*. A data-path $dp = T_i \rightarrow T_i.A_j \rightarrow \text{idx}(T_i.tp_s) \rightarrow dv$ can be *expanded* using a schema-path *sp* as follows.

Definition 5 (Expanded Data-Path) *Given a schema-path $sp = T_1 \rightarrow T_1.A_1 \rightarrow T_2.A_i \rightarrow T_3.A_j \rightarrow \ldots \rightarrow T_n.A_z$ and a data-path $dp = T_n \rightarrow T_n.A_z \rightarrow \text{idx}(T_n.t_s) \rightarrow v$, the expanded data-path dp' of dp through sp is the path $T_1 \rightarrow T_1.A_1 \rightarrow ?x_1 \rightarrow T_2.A_i \rightarrow ?x_2 \rightarrow T_3.A_j \rightarrow ?x_3 \rightarrow \ldots \rightarrow ?x_{n-1} \rightarrow T_n.A_z \rightarrow \text{idx}(T_n.t_s) \rightarrow v$, where each $?x_i$ is a variable ranging over the set of tids.*

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

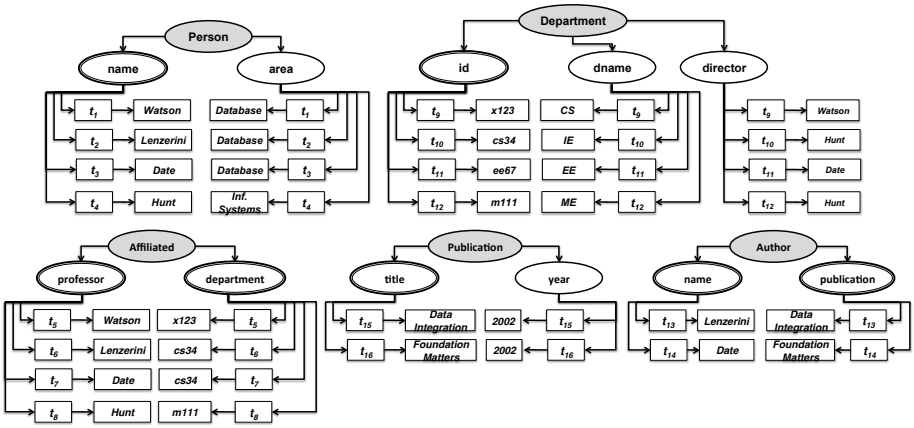


Fig. 3. The data graph for the relational database in Fig 1.

Let us consider again the example in Fig. 3. The data-path dp = *Publication* → *Publication.title* → t_{16} → *Foundation Matters (F.M.)* can be expanded with the schema-path sp = *Author* → *Author.name* → *Author.publication* → *Publication.title* as follows:

$$Author \rightarrow Author.name \rightarrow ?x_1 \rightarrow Author.publication \rightarrow ?x_2 \rightarrow Publication.title \rightarrow t_{16} \rightarrow F.\ M.$$

This path describes the fact that the tuple with tid t_{16} has a relationship with the attribute *publication* of a tuple ($?x_2$) in the relation *Author*. Moreover, the same tuple ($?x_2$) is related to the *name* value of tuple ($?x_1$) of the same relation *Author*; for transitivity $?x_1$ is related to t_{16}, too. Referring to Fig. 3, both $?x_1$ and $?x_2$ correspond to the tuple with tid t_{14}. The path specifies that *Date* is the author of *Foundation Matters*, a publication of *2002*. Two (expanded) data-paths are correlated if they have a common node.

We adopt an Information Retrieval approach for matching values that rely on traditional full text search. In the rest of the paper we will denote the matching relationship with ≈. Intuitively, a solution to a keyword-based query Q is a set of tuples such that: (i) they match the keywords in Q and (ii) they occur in a set of expanded data-paths that, taken together, form a connected graph. More formally, a solution can be defined as follows.

Definition 6 (Solution) *Given a query Q made of a set of keywords $\{z_1, \ldots, z_n\}$, a solution S of Q is a set of expanded data-paths DP such that: (i) for each $z_i \in Q$ there exists a tuple t in DP that matches z_i* (i.e. $t \approx z_i$)*, and (ii) for each pair of expanded data-paths $dp_a, dp_b \in DP$ there is a sequence of expanded data-paths $[dp_a, dp_i, \ldots, dp_j, dp_b]$ occurring in DP such that each element of the sequence has at least a node in common with the following.*

Clearly, given a solution to a query, the answer to the final user is just the set of all tuples whose tids are contained in a solution.

Intuitively, a solution S_1 is more relevant than another solution S_2 if S_1 is more compact than S_2 since, in this case, the keywords of the query are closer between each other in S_1. This is captured by a scoring function that defines the relevance of solutions, as follows.

The size of a solution S, denoted by $\lambda(S)$, is the sum of the length of all its expanded data paths.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

Definition 7 (Ranking of solutions) *Given two solutions S_1 and S_2 of a query Q we say that S_1 is more relevant than S_2 if $\lambda(S_1) \geq \lambda(S_2)$. The ranking of a set of solutions is a function ρ such that $\rho(S_1) \geq \rho(S_2)$ if and only if S_1 is more relevant than S_2.*

Problem Statement. Given a relational database $\mathcal{RDB}$ and a keyword search based query $Q = \{z_1, z_2, \ldots, z_n\}$, where each z_i is a keyword, we aim at finding the top-K ranked solutions $S_1, S_2, \ldots, S_k$.

For example, the best solution for the query $Q_1 = \{\texttt{Matters}, \texttt{Database}\}$ is $S_1 = \{t_3, t_{14}, t_{16}\}$ as will be better clarified in the next section.

3 Path-oriented Search

Once the relational database $\mathcal{RDB} = \langle \mathcal{SG}, \mathcal{DG} \rangle$ is indexed in order to gain immediate access to the information of interest (as detailed in Section 4), the query evaluation takes place. First of all, the paths are organized in clusters.

Clustering. There are as many clusters as the number of keywords in the query Q; for each $z_i \in Q$ we retrieve all data paths dp from $\mathcal{DG}$ such that dp ends into a data value v matching z_i. Then we expand dp with each schema path $sp \in \mathcal{SG}$ ending into the attribute node of dp. Finally we insert dp and each resulting expanded data-path into the corresponding cluster. Each cluster is implemented as a priority queue where the priority decreases with the increasing length of a path. Referring to the query $Q_1 = \{\texttt{Matters}, \texttt{Database}\}$ over the relational database in Fig. 1, we obtain two clusters

$$cl_{Matters} : \begin{pmatrix} dp'_1 : \texttt{Author} \rightarrow \texttt{Author.publication} \rightarrow \texttt{t}_{14} \rightarrow F.M. \\ dp'_2 : \texttt{Publication} \rightarrow \texttt{Publication.title.} \rightarrow \texttt{t}_{16} \rightarrow F.M. \\ dp'_3 : \texttt{Author} \rightarrow \texttt{Author.name} \rightarrow ?x_1 \rightarrow \texttt{Author.publication} \rightarrow \texttt{t}_{14} \rightarrow F.M. \\ dp'_4 : \texttt{Author} \rightarrow \texttt{Author.publication} \rightarrow ?x_2 \rightarrow \texttt{Publication.title} \rightarrow \texttt{t}_{16} \rightarrow F.M. \\ \ldots \end{pmatrix}$$

$$cl_{Database} : \begin{pmatrix} dp'_5 : \texttt{Person} \rightarrow \texttt{Person.area} \rightarrow \texttt{t}_1 \rightarrow Db \\ dp'_6 : \texttt{Person} \rightarrow \texttt{Person.area} \rightarrow \texttt{t}_2 \rightarrow Db \\ dp'_7 : \texttt{Person} \rightarrow \texttt{Person.area} \rightarrow \texttt{t}_3 \rightarrow Db \\ \ldots \\ dp'_8 : \texttt{Author} \rightarrow \texttt{Author.name} \rightarrow ?x_3 \rightarrow \texttt{Person.name} \rightarrow ?x_4 \rightarrow \texttt{Person.area} \rightarrow \texttt{t}_3 \rightarrow Db \\ \ldots \end{pmatrix}$$

Building. Once we have computed the expanded data-paths organized in $\mathcal{CL}$, we combine them to provide the best k solutions to Q. The building algorithm (Alg. 1) is an incremental process. In the following we refer to data-paths including also the expanded paths. Every step starts dequeuing the best paths (*i.e.* the shortest ones) from each cluster into a set DP (`dequeueTop`). Then, the procedure `combinations` generates all the possible combinations of paths within DP in order to find all connected graphs representing candidates of solutions. This is done by taking exactly one data-path from each cluster. For instance referring to our example, at the first running of the algorithm we have to combine dp'_1, dp'_2 from $cl_{Matters}$ with dp'_5, dp'_6, dp'_7 from $cl_{Database}$; we obtain six pairs, *e.g.* $\{dp'_1, dp'_5\}$, $\{dp'_1, dp'_6\}$ and so on. A combination c represents a solution if and only if there exists a *center* in c, that is, a tuple(-id) from which it is possible to reach the tuples matching all the keywords in Q. This evaluation is performed by the procedure `backward_exploration`.

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

Algorithm 1: Building

Input : The set of clusters $\mathcal{CL}$, a query Q, the number k.
Output: The set of solutions $\mathcal{S}$.

```
1  finished ← false;
2  S ← ∅;
3  while ¬finished do
4      DP ← ∅;
5      foreach cl_i ∈ CL do
6          DP ← DP ∪ dequeueTop(cl_i);
7      if CL = ∅ then finished ← true;
8      else
9          // we re-enqueue last extracted dp in the corresponding
             empty cl_i for combining such dp with non empty clusters'
             data-paths
10         foreach cl_i ∈ CL: cl_i = ∅ do
11             foreach dp ∈ DP: dp.vf = cl_i.keyword do
12                 cl_i.enqueue(dp);
13     C ← combinations(DP );
14     foreach c ∈ C do
15         P ← ∅;
16         foreach dp ∈ c do
17             is_sol ← backward_exploration(dp, Q, P);
18         if is_sol then
19             S.enqueue(P.keys) ;      // The priority is score(P.keys)
20             if |S| = k then finished ← true;
21 return S;
```

Intuitively the best solution contains tuples strictly correlated, *e.g.* tuples of the same table. In our representation such kind of solution corresponds to a set of data-paths with the shortest length, *i.e.* a connected graph with the smallest diameter. The building process follows this direction: first of all we try to compose the shortest paths from each cluster, that are good candidates to be the best solutions. If we didn't find k solutions, we try with longer data-paths that would bring to new tuples and potentially to new interconnections and other solutions. Of course, the solutions generated later will present lower score, since the diameter of the corresponding graph of data-paths increases. The combinations produced in the first running of the building are discharged since the pairs of data-paths do not present any intersection. Selecting longer data-paths, we obtain a valid combination $c' = \{dp'_8, dp'_4\}$. Now we have to verify if c' is a solution to include in the set $\mathcal{S}$. Such evaluation explores a data-path in backward: in other terms we follow the foreign key constraints contrariwise. We have two cases: (i) two data-paths correspond to the same table or (ii) two data-paths correspond to two tables linked by a foreign key constraint. In the former case, we instantiate the variables of extended data-paths by extracting the data value associated to the attribute a of the same tuple (*i.e.* it is a simple projection $\pi_a(\sigma_{tid=t}(T))$). In the latter case we follow the key constraint and we need to extract from the new relation T the tuple having the data value v associated to the attribute a (*i.e.* it is a simple selection $\sigma_{a=v}(T)$). The algorithm ends when k solutions are found or when $\mathcal{CL}$ is empty (i.e. we have no more data-paths to combine). We remark that in the building process we exploit operations of

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,

selection (σ) and *projection* (π) on limited groups of tuples (often only one) and we never utilize join ($\bowtie$) operations. Let us consider the exploration of the

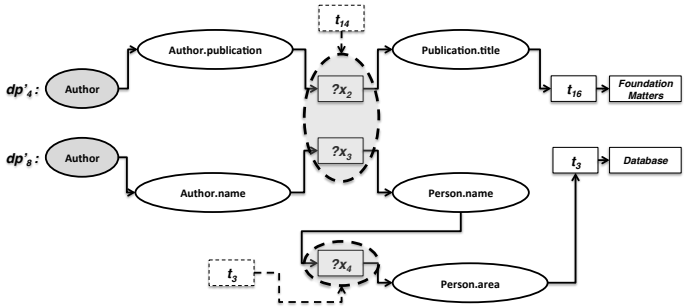


Fig. 4. The building process

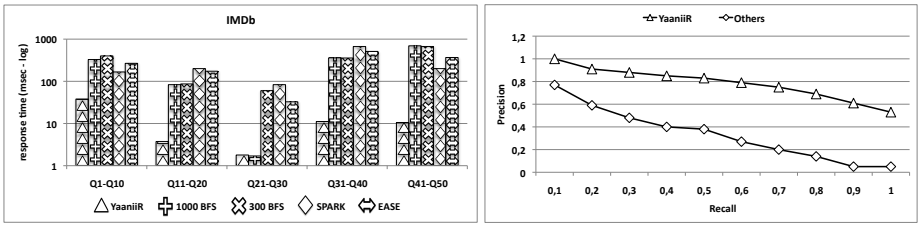
combination $\{dp'_8, dp'_4\}$ of our example, as shown in Fig. 4. In this case the backward exploration starts from analyzing the variable $?x_4$ in the data-path dp'_8. In this case we have a simple projection $\pi_{name}(\sigma_{tid=t_3}(Person))$ to assign the value t_3 to $?x_4$. Following the result of this projection (*i.e.* the value *Date*) allows to process the selection $\sigma_{name='Date'}(Author)$ that assigns the value t_{14} to $?x_3$. Similarly in dp'_4 the selection $\sigma_{publication='Foundation\ Matters'}(Author)$ assigns the same value t_{14} to $?x_2$. Since both $?x_2$ and $?x_3$ present the same value, they represent a node in common between dp'_4 and dp'_8; this means that t_{14} is the center of the graph $\{dp'_8, dp'_4\}$, able to reach all the tuples matching the keywords of the query (*i.e.* t_3 and t_{16}). In this case from the data-paths we extract all tuple-ids (*i.e.* t_3, t_{14}, t_{16}) that represents the first solution to provide.

4 Experimental Results

We implemented our approach in YaaniiR, a Java system for keyword search over relational databases. In our experiments we used the benchmark provided by Coffman et al. [2]. We employ three datasets, IMDb, Wikipedia, and Mondial. For each dataset, we run the set of 50 queries provided in [2]. Experiments were conducted on a dual core 2.66GHz Intel Xeon, running Linux RedHat, with 4 GB of memory, and we used PostgreSQL 9.1 as RDBMS.

Performance. Our algorithms have been implemented in terms of PL/pgSQL procedures. Such procedures exploit a simple index based on two tables: SG(*attribute,path*) and DG(*value,path*). The former stores all schema-paths while the latter all data-paths. Such index is built efficientlty: from few milliseconds on Mondial to a couple of minutes on IMDb and Wikipedia. We compared YaaniiR with the most related approaches: SPARK [8], EASE [7], and the best performing techniques based on graph indexing, i.e. 1000 BFS and 300 BFS that are two configurations of Blinks [5]. For each dataset, we group the queries in five sets: each set is homogeneous with respect to the complexity of the queries (e.g. number of keywords, number of results and so on). For instance referring to IMDb, the first set (i.e. Q1-Q10) searches information about the actors (providing the name as input). The other sets combine actors, movie and characters. For

each set, we ran the queries ten times and measured the average response time (i.e. for computing the top-10 answers). The query response times are shown in Fig. 5 (in *ms* and logarithmic scale). Due to space constraints, in the figure. we report times only on IMDb and Wikipedia, since their much larger size poses more challenges. However the performance on Mondial follows a similar trend. In general Spark and EASE are comparable with Blinks. Our system per-



(a) (b)

Fig. 5. (a) Response Times on IMDb and (b) Precision-Recall interpolation

forms consistently better (in any strategy) for most of the queries, significantly outperforming the others in some cases (e.g. sets Q21-Q30 or Q31-Q40). This is due to the greatly reduced (time) complexity of the overall process with respect to the others spending much time traversing large amount of tuples.

Effectiveness. We have also evaluated the effectiveness of results. We measured the interpolation between precision and recall to find the top-10 solutions, on the queries on all datasets. We calculate the top-10 interpolated precision curve averaged over the systems: Fig. 5.(b) shows the results. As to be expected, the precision of the other systems dramatically decreases for large values of recall. On the contrary our strategies keeps values on the range [0.4,0.8].

References

1. Bergamaschi, S., Domnori, E., Guerra, F., Lado, R.T., Velegrakis, Y.: Keyword search over rel. databases: a metadata approach. In: SIGMOD. pp. 565–576 (2011)
2. Coffman, J., Weaver, A.C.: A framework for evaluating database keyword search strategies. In: CIKM. pp. 729–738 (2010)
3. De Virgilio, R., Cappellari, P., Miscione, M.: Cluster-based exploration for effective keyword search over semantic datasets. In: ER. pp. 205–218 (2009)
4. Garey, M.R., Graham, R.L., Johnson, D.S.: The complexity of computing Steiner minimal trees. SIAM Journal on Applied Mathematics 32(4), 835–859 (1977)
5. He, H., Wang, H., Yang, J., Yu, P.S.: Blinks: ranked keyword searches on graphs. In: SIGMOD. pp. 305–316 (2007)
6. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. In: VLDB. pp. 850–861 (2003)
7. Li, G., et al.: Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD. pp. 903–914 (2008)
8. Luo, Y., Lin, X., Wang, W., Zhou, X.: Spark: top-k keyword query in relational databases. In: SIGMOD. pp. 115–126 (2007)
9. Qin, L., Yu, J.X., Chang, L.: Keyword search in databases: the power of rdbms. In: SIGMOD. pp. 681–694 (2009)

Nicola Ferro and Letizia Tanca (Eds.): SEBD 2012, Edizioni Libreria Progetto, Padova, Italia
ISBN: 978-88-96477-23-6,