# A Template-Based Approach to Keyword Search over Semantic Data

Paolo Cappellari[2], Roberto De Virgilio[1], and Michele Miscione[1]

[1]Department of Computing Science
University of Alberta, Edmonton, Alberta, Canada
cappellari@ualberta.ca
[2]Dipartimento di Informatica e Automazione
Universitá Roma Tre, Rome, Italy
{dvr,miscione}@dia.uniroma3.it

**Abstract.** Keyword search is receiving a lot of attention not only in Web contexts but also in the database area. It is an easy way to allow inexperienced user to query systems without the need of knowing any specific language or how data is structured. As a matter of fact, the amount of data available, in the Web as well as in other systems, is constantly increasing. And, with the improvements and the simplification of the technology, the amount of people accessing such information is growing too. Providing simple, yet effective tools that allow inexperienced users to quickly discover desired information is a big challenge in modern times. The prevalent approaches build on dedicated indexing techniques as well as search algorithms aiming at finding substructures that connect the data elements matching the keywords. In this paper, we introduce a novel keyword search paradigm for graph-structured data, focusing in particular on the RDF data model. While related techniques search the best answer trees, we propose a novel algorithm for the exploration and computation of all matching subgraphs.

**Keywords:** RDF, keyword search, semantic annotations

## 1 Introduction

Keyword-based search approaches have the huge benefit that users can ignore both the language and the structure of the data they are going to query. A keyword based search engine returns a list of candidate pages, documents or set of data that match keywords provided in input. Then a user has to dedicate her time and efforts navigating each result returned from the engine in order to discover the desired information, i.e. the answer she is looking for. Figure 1 illustrates the actual scenario of the Web. In a simple scenario the user's desired answer is contained in a single document or set of data. In a slightly more complex scenario such an answer may not be confined to a single document: it may reside on a logical connection between concepts in different documents. A user has to explore the returned results in order to discover the connection, and

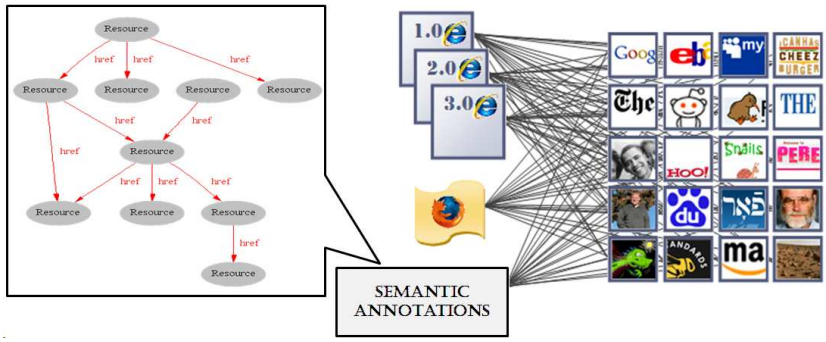


**Fig. 1.** The Syntactic and Semantic Web

then the data, she is looking for. In general, the user has to face two issues: the number of results to explore and the distance between the results containing the concepts holding the desired connection. The first issue can be mitigated by submitting a more selective query. However, it's unlikely the case that not expert users are able to submit a complex query with an high selectivity. The second issue is rather hard to address.

Therefore, attention around searching and query processing of graph-structured data continue to increase as the Web, XML documents and even relational database can be represented as a graph. In particular many efforts focus on RDF data querying, given the great momentum of *Semantic Web* in which Web pages carry information that can be read and understood by machines in a systematic way. Current approaches rely on a combination of IR and tree/graph exploration techniques whose goal is to rank results according to a relevance criterion. Keyword search on tree-structured data counts a good number of approaches already [1–6]. Examples of approaches where relational database is treated as data-structured graphs are presented in [7] and [8], while a generic approach to similarity search is presented in [9]. Then there are a number of works focusing on RDF storage and query efficiency issues like [10–12]. Simplifying, a generic approach first identifies the parts of the data structure containing the keywords of interest, possibly by using an indexing system, then explores the data structure in order to discover a connection between such identified parts. Candidate solutions, built out of found connections, are then associated with a score and ranked. In many approaches [7, 8], an exact matching between keywords and labels of data elements is performed to obtain the keyword elements. For the exploration of the data graph, the so-called distinct root assumption is employed (see [7–9]). Under this assumption, only substructures in the form of trees with distinct roots are computed and the root element is assumed to be the answer. The algorithms for finding these answer trees are *backward search* and *bidirectional search* [8]. In order to guarantee that the computed answers indeed have the best scores, both the lower bound of the computed substructures and the upper bound of the remaining candidates have to be maintained. Since book-

keeping this information is difficult and expensive, current algorithms compute the best answers only in an approximate way. Moreover, approaches aiming at returning the top-k best solutions implement pruning techniques to reduce the list of candidate solutions down to those whose score is above a threshold. Pruning techniques can have an sensible impact in both the quality of the solutions, as low scoring results are not shown or even computed, as well as on efficiency, as an early pruning reduces the space of candidate solutions to investigate.

In this paper, we propose a novel approach to keyword search in the graph-structure data in an RDF representation. The approach aims to provide effective answers in an efficient way. The main contribution of this paper relies in how connections between identified parts of the structure containing keywords of interest are combined together. We assume here that annotations in the graph are generated out of a specific ontology domain. That means that data are instances of concepts belonging to a knowledge base schema. Therefore, the idea is to aggregate the parts of the data structures containing keywords in clusters, according to their schemas, i.e. temporarily ignoring instance values. If two instances come from a common schema then they share a *template*. Then the parts in each cluster, and the clusters too, are ranked according to a function, taking into account both structural and contextual features. Finally parts from the several clusters are composed, when possible, starting with the most relevant one in the most relevant cluster. As a result, most relevant solutions emerge early in the process because the approach tries to compose the best candidates from each cluster first. Moreover, the clustering technique allows avoiding the combination (exploration) of overlapping solutions.

The paper is organized as follows: Section 2 introduces the preliminaries of the problem and the data structures used. Section 3 describes the query processing in details. Finally Section 4 presents the related works, and Section 5 sketches conclusions and future works.

## 2 Overview

***Problem Definition.*** Formally, the problem we are trying to solve may be defined as follows. Similar to [8], given a directed graph $G = (R, P)$ where each node (resource) r ∈ R and edge (predicate) p ∈ P has a label (URI) associated with it, we are concerned with querying this graph using keywords. A keyword search query q consists of a list of $n$ keywords $(k_1, k_2, ..., k_n)$. The answer to query q is the set of paths in G where the end point of each path is a node r ∈ R that matched a user keyword based on one of the following criteria:

- There exists some keyword $k \in (k_1, k_2, ..., k_n)$ that matches label of node $r$ either lexically or on semantic query expansion.
- Node $r$ is the subject/object of the ontological triple whose predicate label matches some keyword lexically or on semantic query expansion.

***An example of reference.*** Let's consider the example of Figure 2. It illustrates an ontology about Universities composed into Departments where a Staff works.

The figure shows both the schema and a corresponding instance. Now we would process a query by submitting the keywords **University, CIV, Department, W1** that is *"all the information about the Staff W1 working in a Department CIV into a University"*.

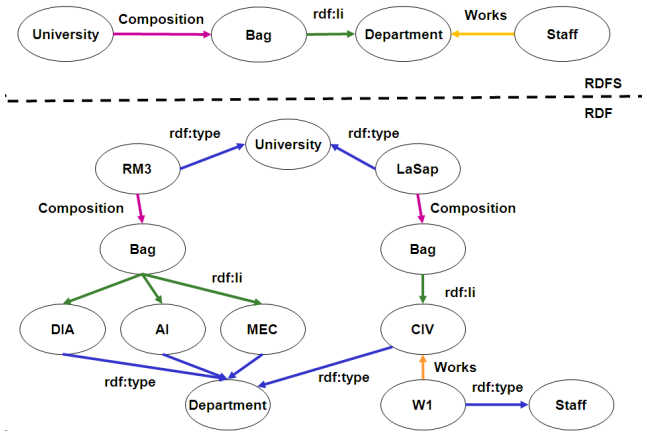


**Fig. 2.** An example of reference

***Preliminaries.*** We illustrate the basic data structures used by our approach. We model the instances into RDF documents as *informative paths* as follows.

**Definition 1 (Informative Path)** *Given an RDF graph G (R,P) and a set of keywords* $k_1, ..., k_n$*, an informative path pt is a vector* $r_1 - p_1 - r_2 - p_2 - ... - p_{n-1} - r_n$ *where each* $r_i$ *is a resource* $\in R$*, each* $p_i$ *is a predicate* $\in P$*,* $r_i$ *and/or* $p_i$ *match one or more keywords* $k_i$*, and* $r_1$ *is a root node, i.e. a resource without incoming edges (predicates).*

For instance `W1-Works-CIV` is an informative path $pt_k$. We use the notation $pos_{pt_k}(r_i)$ ($pos_{pt_k}(p_j)$) to indicate the position of the resource $r_i(p_j)$ into the vector $pt_k$. For example $pos_{pt_k}(W1)$ returns 1. We compute the informative paths from root nodes because they allow to reach any node in the graph. In case a root node is not present, a fictitious one can be added. Having the information to navigate from the roots to nodes matching keywords is at the basis of our approach to solutions discovery. Then we call *template* the schema associated to an informative path.

**Definition 2 (Template)** *Given an informative path pt, we associate a template t to pt replacing each* $r_i \in pt$ *with the wild card #*

For instance the template associated to $pt_k$ is `#-Works-#`. Then we introduce two basic notions as follows.

**Definition 3 (Subsumption)** *Given two informative paths $pt_1$ and $pt_2$, we say that $pt_1$ is subsumed by $pt_2$, denoted by $pt_1 \lhd pt_2$, if $\forall r_i, p_j \in pt_1$ then $\exists r_m, p_n \in pt_2$ such that $r_i = r_m$ and $p_j = p_n$, and $pos_{pt_1}(r_i) = pos_{pt_2}(r_m)$ and $pos_{pt_1}(p_j) = pos_{pt_2}(p_n)$*

**Definition 4 (Graft)** *Given two informative paths $pt_1$ and $pt_2$, there is a graft between $pt_1$ and $pt_2$, denoted by $pt_1 \leftrightarrow pt_2$, if $\exists r_i \in pt_1(pt_2)$ such that $r_i \in pt_2(pt_1)$*

***An Architecture of Reference.*** A flexible architecture of the system was design, as shown in Figure 3. It serves as a logical view of how the system looks like. This is a typical use scenario of the system:

1. The RDF Parser takes as input a collection of RDF Documents and parses them into triples. Here we use the Jena framework (`http://jena.sourceforge.net/`);
2. The Indexer builds an index on top of the triple collections to achieve structural information useful for the query process. Here the indexing is supported by Lucene (`http://lucene.apache.org/`) and WordNet (`http://wordnet.princeton.edu`) to allow query expansion;
3. A user performs a query through a GUI helper, handling events and the query itself;
4. The parsed query is given to the Searcher for processing;
5. The Searcher processes the query over the Indexed Resource Base and returns the search result to the caller. It communicates with the Indexer to extract the instances matching input keywords, group them into clusters and compose elements from clusters into the final solutions (i.e. subgraph structures). Each structure (i.e. instance, cluster and solution) are ordered by a ranking function. Here we adopt the exhaustivity of the result as scoring function that is the number of matched keywords with respect to the number of submitted keywords. Although a ranking function is a relevant aspect in the framework, in this paper we focus on the composition of the solutions.

The approach is composed of two main phases: an off-line one where documents of interest are indexed in order to have immediate access to nodes (steps 1 and 2), and an on-line one where the query evaluation takes place (steps 3 to 5). In the next section we illustrate the query processing.

## 3 Query Processing

The approach is composed of two main phases: the *off-line indexing* where documents of interest are indexed in order to have immediate access to nodes, and the *keyword processing* (on-the-fly) where the query evaluation takes place.

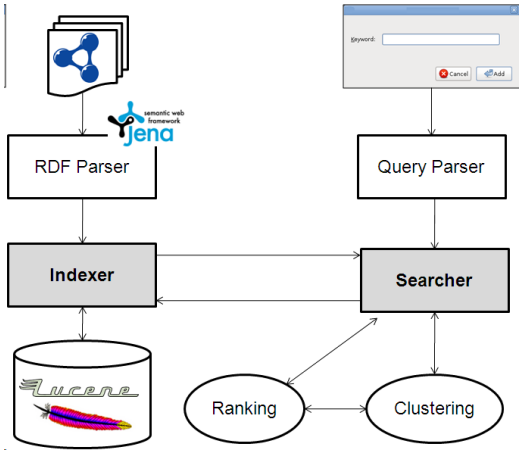


**Fig. 3.** An Architecture of Reference

***Off-line indexing*** During this phase, an index structure is built and incrementally updated while documents of interest are loaded or modified. Here the indexing is supported by Lucene (`http://lucene.apache.org/`) with WordNet (`http://wordnet.princeton.edu`) for query expansion. With the indexing the information in the graph is augmented by: identifying root nodes in the graph and storing in each node the paths to reach such node from the roots. Each path is computed by implementing the breadth-first search (BFS) algorithm. Although this process could be expensive, let us remark that: (i) it is an incremental process, so its cost dramatically reduces once the system is loaded, and (ii) the index drastically speeds up the on-the-fly query evaluation. We don't have to navigate the graph at runtime and we have immediate access to the path root-matching node that is the basis for our clustering and combining techniques.

***Keywords processing.*** Given a keyword the index structure allows immediate access to the node with such a keyword. Lucene index returns all the informative paths from roots to the nodes matching one of the specified keywords. So we result a list of informative paths ordered by the length. Referring to our example we obtain the following list:

```
c) [RM3-Composition-Bag-rdf:li-DIA-type-Department]
d) [RM3-Composition-Bag-rdf:li-AI-type-Department]
e) [RM3-Composition-Bag-rdf:li-MEC-type-Department]
f) [LaSap-Composition-Bag-rdf:li-CIV-type-Department]
z) [LaSap-Composition-Bag-rdf:li-CIV]
g) [W1-Works-CIV-type-Department]
a) [RM3-type-University]
b) [LaSap-type-University]
y) [W1-Works-CIV]
```

For instance $y$) is subsumed by $g$) and there is a graft between them in nodes $W1$ and $CIV$. Now the goal is to "extract" the schema behind the paths so

that we can cluster paths according to the discovered schema(s). A cluster is represented by a template $t$. So it is a set of informative paths that share the same template $t$. Such templates are the attempt of identifying and giving values to a structure in the information graph that is not explicitly provided with the query. Such a structure is a recurrent pattern in an anonymous path that we refer to as schema. Therefore given the list of informative paths $PT$, the set of clusters $CL$ is computed in the following way

while $PT$ is not empty, we extract $pt$ from $PT$.

- if $\exists CL_i \in CL$ such that we can associate the representative template $t$ of $CL_i$ to $pt$, then
  - if $\nexists pt' \in CL_i$ such that $pt \lhd pt'$ then insert $pt$ into $CL_i$. We order the elements of $CL_i$ with respect to the exhaustivity.
- else a new cluster $CL_j$ with the representative template associated to $pt$ is built and we insert $pt$ into $CL_j$, $CL_j$ into $CL$.

In this process we need to order both paths and clusters with respect to a scoring function. So we define the *rank* of an element (path or cluster) with respect to the query $Q = k_1, k_2, \ldots, k_n$ as

$$R(e,Q) = \sum_{k \in Q} weight(k,Q) \cdot weight(k,e) \; ; \; weight(k,e) = \frac{weight_{ctx}(k,e)}{weight_{str}(k,e)}$$

where $weight(k,Q)$ is the weight associated with each keyword $k$ with respect to the query $Q$ and where $weight(k,e)$ is the weight associated with each keyword $k$ and $e$ where is a path, when ranking paths, or a cluster, when ranking clusters. We assume all the keywords in the input query have the same weight, i.e. from the user point of view all the keywords have the same relevance. The weight $weight(k,e)$ is the ratio between the *contextual* and the *structural* weights of $k$ when considering $e$. When ranking paths the structural weight represents a measure of the proximity of $k$ with the other keywords whereas the contextual weight represents the relevance of $k$ with respect to the other keywords, according to the following:

$$weight_{str}(k,pt) = \frac{\sum_{k_i \in pt, k_i \neq k} d_{pt}(k,k_i)}{dl} \; ; \; weight_{ctx}(k,pt) = \frac{1+ln(1+ln(tf))}{(1-s)+s \cdot \frac{dl}{avg(dl)}} \cdot (1 + ln(\frac{N}{df+1}))$$

In the first formula, $pt$ is a path, $d_{pt}(k,k_i)$ is the distance between two keywords and $dl$ is the number of terms in the path $pt$. In the second formula, $tf$ is the number of times the keyword $k$ occurs in the path $pt$, $df$ is the number of paths where $k$ occurs, $N$ is the number of path containings at least one of the input keywords; $dl$ has the same meaning as in the first formula and $avg(dl)$ is the mean of all the $dl$; finally, $s$ is a constant, usually 0.2 [5]. The logarithm function is used, twice in one case, to smooth the values in precense of large numbers of terms and keywords. Last formula can be read as a trasposition of one of the most widely used weighting method in IR , where: the first factor is the normalized term frequency, the second factor is the so called inverse document frequency and the denominator is the normilzed document lenght [13, 14].

When ranking clusters we define the structural and contextual weights of a keyword as the mean of its correspondent weights respect to the paths contained in the cluster $CL$, as shown in the formulas below:

$$weight_{str}(k, CL) = \frac{\sum_{pt \in CL} weight_{str}(k,pt)}{|CL|} \; ; \; weight_{ctx}(k, CL) = \frac{\sum_{pt \in CL} weight_{ctx}(k,pt)}{|CL|}$$

In the following the resulting ordered clusters set computed respect to the reference example.

```
CL3: [#-Works-#-type-#] { (g) }
CL2: [#-Composition-#-rdf:li-#-type-#] { (f) , (c,d,e) }
CL1: [#-type-#] { (a,b) }
```

The final step is combining the paths from different clusters. A solution is a set of informative paths that present a graft in pairs. So computing a solution an informative path $pt$ can be included into a solution $S$ if $\exists pt' \in S$ such that $pt$ and $pt'$ present a graft. Paths in clusters are combined, when possible, starting from the most relevant path in the most relevant cluster: we combine the best candidate paths of each cluster to compose a single solution. Including an informative path into a solution, we delete it from the starting cluster. Therefore the solutions generation ends when the set $CL$ is empty. Because of the ranking on both the clusters and the paths in each cluster, solutions will come out from the process starting with the most relevant one, then descending to the least relevant. This allows for optimizations like: cutting the answering process if relevance decrease under a specific threshold, or returning the user with an initial set of answers she can start exploring while elaboration of remaining ones is still undergoing. More important, the clustering and the combining techniques guarantee the combination of paths only for instance data belonging to different schemas, thus avoiding overlapping. Overlapping is undesired and time consuming. At the end we result the following solutions (also depicted in Figure 4) over the reference example: S1: { (g) , (f) } and S2: { (c,d,e) , (a,b) }.

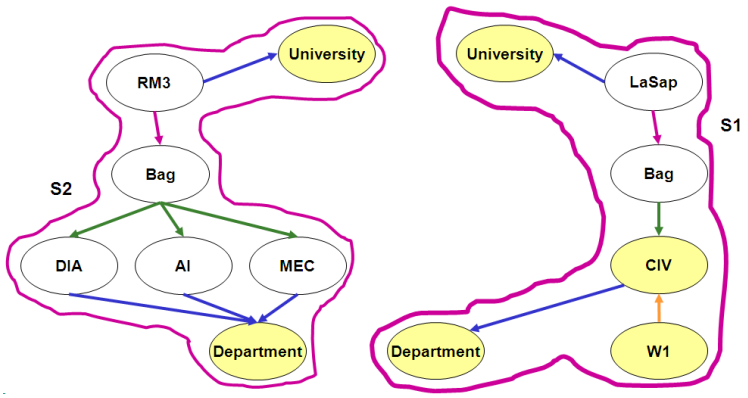


**Fig. 4.** Final Solutions

## 4 Related Works

There is a broad literature on information search. Natural language interfaces to database systems have been investigated in several works. Approaches addressing relational database systems are: BANKS [7], DISCOVER [4], DBXplorer [1] and Hristidis et al. [15]. Simplifying, a database is represented as a graph where tuples are the nodes and foreign keys. Then they generate tuple trees from multiple tables as answers. Finally, IR-style techniques are applied to returned results in order to rank them according to a scoring function. Another work on relational database is [5]. Here authors address that strictly focus on search effectiveness while ignoring efficiency. Authors' claim is that effectiveness is as important as efficiency. Approaches have been proposed for keyword search over XML in [2, 3, 16, 17]. With respect to search on RDF data, search on XML data is a similar but simpler problem. The tree structure guarantee each node to have a single incoming path: this allows the implementation of a number of optimizations. And such optimization cannot be easily applied in general graphs. For instance in XRANK [3] an indexing solution is defined that allows the evaluation of a keyword query without requiring tree traversal. Then there are works specifically addressing query processing efficiency over graph-structured data and RDF storage [8, 10–12]. In BLINKS [8] a scoring function is defined to find the top-k most relevant queries, a good part of the contribution relies on the novel indexing structure. Authors present a bi-level indexing structure that allows for early pruning that accelerates the search. In [12] authors propose a slightly different approach. While still addressing performance issues, the approach first compute queries from keywords then asks the user to choose the most appropriate query. Computation of queries is based on the exploration of the top-k matching subgraphs, exploiting an off-line built index structure and a variant of backward search where expansion (exploration) is driven by a cost function.

## 5 Conclusion and Future Works

We presented a full-text search index for ontology triples that provides matching capabilities based on semantic and morphological expansion of terms used for indexing the triple. Given a set of text matches, we propose a method to construct the set of answer paths by a template based clustering technique. In this paper the paths retrieved by the system are ordered with respect to a trivial measurement, that is the exhaustivity. Hence, it could potentially lead to information overload. Semantic association ranking metrics could be used to present only paths most relevant to users context. Future works concern a sophisticated ranking function, and using it a set of experimental results over (very) large datasets. DBpedia, Yago are recent efforts to generate semantic metadata by extracting structured information form the Web (Wikipedia). A keyword or natural language search interface to such knowledge bases would prove immensely useful as the end user need not be aware of the structure of the information. It will be worthwhile to build a search interface that accepts queries in natural language.

## References


1. Agrawal, S., Chaudhuri, S., Das, G.: Dbxplorer: Enabling Keyword Search over Relational Databases. In: Franklin, M.J., Moon, B., Ailamaki, A. (eds.): SIGMOD, pp. 627–627, ACM (2002)
2. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: Xsearch: A Semantic Search Engine for XML. In: VLDB, pp. 45–56 (2003)
3. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: Xrank: Ranked Keyword Search over XML Documents. In: Halevy, A.Y., Ives, Z.G., Doan, A. (eds.): SIGMOD, pp. 16–27. ACM (2003)
4. Hristidis, V., Papakonstantinou, Y.: Discover: Keyword Search in Relational Databases. In: VLDB, pp. 670–681. Morgan Kaufmann (2002)
5. Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective Keyword Search in Relational Databases. In: Chaudhuri, S., Hristidis, V., Polyzotis, N. (eds.): SIGMOD, pp. 563–574. ACM (2006)
6. Kimelfeld, B., Sagiv, Y.: Finding and Approximating Top-k Answers in Keyword Proximity Search. In: Vansummeren, S. (ed.): PODS. , pp. 173–182. ACM (2006)
7. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword Searching and Browsing in Databases Using Banks. In: ICDE, pp. 431–440. IEEE Computer Society (2002)
8. He, H., Wang, H., Yang, J., Yu, P.S.: Blinks: Ranked Keyword Searches on Graphs. In: Chan, C.Y., Ooi, B.C., Zhou, A. (eds.): SIGMOD, pp. 305–316. ACM (2007)
9. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional Expansion for Keyword Search on Graph Databases. In: VLDB, pp. 505–516. (2005)
10. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An Efficient SQL-based RDF Querying Scheme. In: VLDB, pp. 1216–1227. (2005)
11. Abadi, D.J., 0002, A.M., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: Koch, C., Gehrke, J., Garofalakis, M.N., Srivastava, D., Aberer, K., Deshpande, A., Florescu, D., Chan, C.Y., Ganti, V., Kanne, C.C., Klas, W., Neuhold, E.J. (eds.): VLDB. , pp. 411–422. ACM (2007)
12. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k Exploration of Query Graph Candidates For Efficient Keyword Search on RDF. In: ICDE, pp. to appear, IEEE Computer Society (2009)
13. Singhal, A., Buckley, C., Mitra, M.: Pivoted Document Length Normalization. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Experimental Studies, pp. 21–29. ACM (1996)
14. Singhal, A.: Modern Information Retrieval: A Brief Overview. IEEE Data Engineering Bulletin 24(4), pp. 35–43. (2001)
15. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient IR-Style Keyword Search over Relational Databases. In: VLDB, pp. 850–861. (2003)
16. Hristidis, V., Papakonstantinou, Y., Balmin, A.: Keyword Proximity Search on XML Graphs. In: Dayal, U., Ramamritham, K., Vijayaraman, T.M. (eds.): ICDE, pp. 367–378. IEEE Computer Society (2003)
17. Kaushik, R., Krishnamurthy, R., Naughton, J.F., Ramakrishnan, R.: On the Integration of Structure Indexes and Inverted Lists. In: Weikum, G., König, A.C., Deßloch, S. (eds.): SIGMOD, pp. 779–790. ACM (2004)