Efficient Association Discovery with Keyword-based Constraints on Large Graph Data

Mo Zhou
Indiana University, USA
mozhou@cs.indiana.edu

Yifan Pan
Indiana University, USA
panyif@cs.indiana.edu

Yuqing Wu
Indiana University, USA
yuqwu@cs.indiana.edu

ABSTRACT

In many domains, such as social networks and chem-informatics, data can be represented naturally in graph model, with nodes being data entries and edges the relationships between them. We study the application requirements in these domains and find that discovering *Constrained Acyclic Paths* (CAP) is highly in demand. In this paper, we define the CAP search problem and introduce a set of quantitative metrics for describing keyword-based constraints. We propose a series of algorithms to efficiently evaluate CAP queries on large-scale graph data. Extensive experiments illustrate that our algorithms are both efficient and scalable.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval (Search process)*

General Terms

Algorithms

Keywords

Keyword-based Association Discovery, Search Algorithm, Coverage, Relevance

1. INTRODUCTION

RDF (Resource Description Framework) [13] is a W3C recommended language for describing linked data of the Semantic Web in the form of triples. RDF data can be represented by node and edge labeled graphs. The simplicity and flexibility of the graph-based data representation model facilitate the wide adoption of Semantic Web technologies in domains such as social network and cheminformatics. Applications in these domains pose challenges and opportunities for managing and searching data on the Semantic Web, as witnessed by new technologies proposed in semantic association discovery [4] and keyword search [11, 15].

The semantic association discovery problem aims to answer questions such as "what are possible relationships between X and Y" and the results are usually paths connecting the two nodes corresponding to the two entities in the graph [4]. The keyword search problem aims to answer questions such as "how do the data entities that match keywords X, Y and Z relate to each other" and the results are usually trees/sub-graphs with the labels of their nodes and edges covering all of the keywords [8, 11, 15].

Discovering acyclic paths between data entities under constraints, such as appearance of nodes, edges and patterns, and the length of paths, is at the core of many applications, such as drug discovery [7, 14] in cheminformatics and research/social networks [18]. How to express such search queries precisely and how to answer such search queries efficiently are critical problems, but are yet fully studied in the literature.

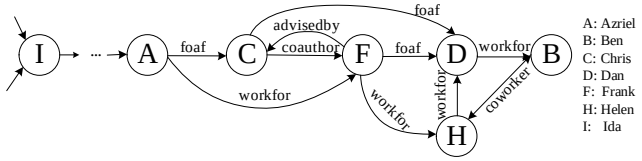

Figure 1: Graph Representation of a Sample RDF Data

EXAMPLE 1. *Figure 1 shows the graph representation of a sample RDF data representing the relationships among people in the social networks. Let's consider the following search requests:*
case1. *Find how* Azriel *connects to* Ben *through* Chris *or* Dan*;*
case2. *Find how* Azriel *connects to* Ben *through at least two people from* Chris, Dan *and* Ida *within four steps.*
case3. *Find* Azriel*'s close (within 4 steps) professional connections (e.g. relationships such as* workfor, coworker, coauthor*) to* Ben*;*
case4. *Find* Azriel*'s close semi-professional connections to* Ben *(i.e. half of the relationships in any tie should be professional);*

Query languages, such as SPARQL [16], have been proposed to query data on the Semantic Web. Later, the notions of semantic association [4], label-constraint reachability [17], and semantic keyword search [11, 15] were proposed. Extending SPARQL with regular expressions [3] enhances it with the capability to express complex patterns satisfying strictly defined constraints, e.g. cases 1 & 3. However, search requests as those in cases 2 & 4 cannot be expressed by any existing language or language extension.

Significant amount of research has been done in RDF data storage, indexing and query evaluation for answering SPARQL queries efficiently [2, 10]; however, the focus was on accelerating graph pattern matching, rather than finding arbitrary paths between data entries. Many graph searching algorithms were proposed for answering keyword searches in graph data [11, 15]; but they are searching for trees whose node/edge labels cover all keywords and cannot be applied to answering CAP queries efficiently. Traversal-based approaches such as DFS, BFS, and bidirectional search [12] were proposed for finding paths satisfying given regular expressions between two end nodes [3]. These algorithms, trailed by a filtering process, can be used to answer CAP queries in Exp. 1, but inefficiently, due to their poor scalability with respect to the size and complexity of the graph data.

CIKM'11 October 24–28,2011, Glasgow,Scotland, UK.

We study and tackle the problem of precisely specifying and efficiently answering CAP queries. Particularly,

• we introduce the notions of *coverage* and *relevance* for precisely describing the correlation between a resultant path and a keyword set in a CAP query (Sec. 2);

• we propose a set of efficient CAP query answering algorithms, including a family of DFS-based algorithms, which takes advantage of the keyword-based constraints to eliminate unpromising search branches (Sec. 3.1), and a novel *Search-and-Join* algorithm in which a CAP search is broken into a sequence of mini-searches and their results are *joined* to answer the CAP query and its efficiency is guaranteed by carefully designing bookkeeping and pruning (Sec. 3.2);

• we conduct extensive empirical study to understand the strengths and limits of our algorithms (Sec. 4); and

• we explore the application of CAP search queries, in stand-alone domain-specific applications, and in a new SPARQL extension, cSPARQL (Sec. 5).

2. CONSTRAINED ACYCLIC PATH DISCOVERY PROBLEM

In this section, we formally define the *Constrained Acyclic Path Discovery* problem whose applications have been illustrated in Sec. 1.

2.1 Preliminary

Let $\mathcal{L}$ be an infinite set of literals and $\mathcal{U}$ be an infinite set of URIs disjoint with $\mathcal{L}$. We represent RDF data as a node and edge labeled directed graph $G = (V, E, \lambda)$ where V is a set of nodes, $E \subset V \times V$ is a set of edges, and λ is a labeling function that maps items in $V \cup E$ into a finite set of labels and literals.

We represent a path between two nodes of graph G as a sequence of interleaving nodes and edges. Frequently, we are interested in partial paths, which we call *fragments*. In particular, we are interested in two types of fragments: e-fragment (denoted f_e) which starts and ends with an edge, and en-fragment (denoted f_{en}) which starts with an edge and ends with a node.

Given source and destination nodes, $n_s, n_d \in V$ of G, in search queries looking for paths from n_s to n_d, frequently only acyclic paths are of interest to users. In the rest of the paper, we will focus only on acyclic paths, and as the two end nodes are known, we focus on the e-fragments and use $\mathcal{F}_e(n_s, n_d)$ to represent all acyclic e-fragments from n_s to n_d in G.

To facilitate the discussion of constraints on paths, we use $nodes(f)$ ($edges(f)$) to represent the set of nodes (edges) in fragment f, and $|f|$ the length of f, defined as $|edges(f)|$. We overload the mapping function λ to map a set of nodes (edges) to their corresponding labels.

2.2 Set-based Constraints

As shown in Exp. 1, when users search for paths between a pair of nodes, it is frequently the case that the constraints are expressed in the form of a *keyword set* (denoted $\mathcal{S}$), where a keyword is a label in $\mathcal{U}$. Keyword-based constraints as discussed in [5, 17] are quite limited in terms of what can be in the keyword set and how the results are regulated by it. We generalize the keyword set to include keywords that can be mapped to labels of both nodes and edges and extend how the results are confined by the keyword set.

Definition 1. Given a finite keyword set $\mathcal{S} \subseteq \mathcal{U}$ and an e-fragment $f_e \in \mathcal{F}_e(n_s, n_d)$,

1. if $\mathcal{S} \subseteq (\lambda(nodes(f_e)) \cup \lambda(edges(f_e)))$, we say f_e satisfies *presence constraint* w.r.t. $\mathcal{S}$;
2. if $\mathcal{S} \supseteq (\lambda(nodes(f_e)) \cup \lambda(edges(f_e)))$, we say f_e satisfies *context constraint* w.r.t. $\mathcal{S}$;
3. if $\mathcal{S} \cap (\lambda(nodes(f_e)) \cup \lambda(edges(f_e))) \neq \emptyset$, we say f_e satisfies *intersection constraint* w.r.t. $\mathcal{S}$.

Using this definition, we can express the search request in Exp. 1 case1 as "find e-fragments from *Azriel* to *Ben* that satisfy the *intersection constraint* w.r.t. keyword set $\{Chris, Dan\}$".

2.3 Quantitative Metrics

Among a possible large number of resultant e-fragments of a search request, shorter ones tend to express stronger and more meaningful relationship between the two end nodes than the longer ones do [4, 11]. The *length constraint* which restricts the length of the resultant e-fragments has been studied in [3, 5], and can be used to express the search request in Exp. 1 case 2 & 3. However cases 2 & 4 require a more subtle description of the relationship between an e-fragment and a keyword set than the *all-or-nothing* set-based constraints described in Def. 1. For this purpose, we introduce quantitative metrics *coverage* and *relevance*.

Intuitively, *coverage* describes the fraction of the keyword set that appears in the label set of an e-fragment, while *relevance* describes the fraction of the labels of an e-fragment that are in the keyword set. In an RDF graph, each node has its unique label, while more than one edge may have the same label. Therefore, we refine *coverage* and *relevance* further into *node-coverage* and *node-relevance* for keyword sets to be applied only on nodes, *edge-coverage* and *edge-relevance* for edges, and use *coverage* and *relevance* for the constraints in which keywords can be mapped to both nodes and edges.

Definition 2. Given a graph G, two nodes $n_s, n_d \in V$ and a finite keyword set $\mathcal{S} \subseteq \mathcal{U}$, for an e-fragment $f_e \in \mathcal{F}_e(n_s, n_d)$[1],

$$Coverage(f_e, \mathcal{S}) = \frac{|\mathcal{S} \cap (\lambda(nodes(f_e)) \cup \lambda(edges(f_e)))|}{|\mathcal{S}|} \quad (1)$$

$$Relevance(f_e, \mathcal{S}) = \frac{|\mathcal{S} \cap \lambda(nodes(f_e))| + \Sigma_{l \in \mathcal{S}} cntE(l, f_e)}{|nodes(f_e)| + |edges(f_e)|} \quad (2)$$

The coverage and relevance for only nodes or edges can be easily inferred.

Now, taking advantage of the quantitative metrices defined above, we are able to express all constrains in Exp. 1. For example, case 4: $\{f_e \mid f_e \in \mathcal{F}_e(\text{Azriel, Ben}) \wedge EdgeRelevance(f_e, \{\text{workfor, coworker,coauthor}\}) \geq 0.5\}$.

All the constraints we defined in Def. 1 can be expressed using the quantitative metrics defined in Def. 2.

$$Presence(f_e, \mathcal{S}) \iff Coverage(f_e, \mathcal{S}) == 1 \quad (3)$$

$$Context(f_e, \mathcal{S}) \iff Relevance(f_e, \mathcal{S}) == 1 \quad (4)$$

$$Intersection(f_e, \mathcal{S}) \iff Relevance(f_e, \mathcal{S}) > 0 \quad (5)$$

Similarly, we can define the node/edge version of these functions.

2.4 Problem Definition

We define the Constrained Acyclic Path (CAP) search query:

> *A CAP search query $CAP(n_s, n_d, \tau)$ on an RDF graph G involves two end nodes $n_s, n_d \in V$, and constraint τ expressed using zero to many quantitative metrics functions that involve zero or many keyword sets, and returns the e-fragment(s) from n_s to n_d that satisfy τ.*

In this paper, we tackle the problem of efficiently evaluating CAP searches, and discuss the applications of CAP searches.

3. CAP DISCOVERY

We will first focus on the evaluation of a critical subset of CAP queries, *core CAP* queries, in which τ contains conjunctive predicates featuring only one keyword set. We use $\mathcal{S}$ to represent the

[1] $cntE(l, f_e)$ is the total numer of edges in f_e with label l

single keyword set in τ, and use τ_l, τ_c, τ_r, τ_{nc}, τ_{ec}, τ_{nr} and τ_{er} to represent the length, coverage, relevance, node/edge coverage, node/edge relevance constraints respectively, each of which is defined as an interval, for example, $\tau_l = [\tau_{l_{min}}, \tau_{l_{max}}]$.

Certainly one solution, which we call *Search-and-Filter approach* (S&F), is to first find all acyclic e-fragments in $\mathcal{F}_e(n_s, n_d)$, then eliminate those not satisfying τ. However this approach is not practically efficient because generating $\mathcal{F}_e(n_s, n_d)$ is very time and space consuming and the search cost is mostly wasted since the CAP query results are usually a very small subset of $CAP(n_s, n_d, \emptyset)$.

3.1 Constrained DFS Algorithm

Depth First Search (DFS) is a commonly adopted approach for generating paths between two nodes. In DFS, to generate an e-fragment, en-fragments are generated one step at a time. To minimize the DFS search space in answering core CAP query, we want to stop the expansion of an intermediate result f_{en} if we are certain that the expansion will not lead to any final results. The basic idea is to calculate projected value ranges of the quantitative metrics' values by considering the best and worst cases of $f_e \in \mathcal{F}_e(n_s, n_d)$ which has f_{en} as prefix. If the projected value ranges of the quantitative metrics do not overlap with those in τ, we can safely stop the expansion of f_{en}.

LEMMA 1. *Given an en-fragment f_{en} generated in the DFS of $CAP(n_s, n_d, \emptyset)$ and a keyword set $\mathcal{S}$, for any e-fragment $f_e \in \mathcal{F}_e(n_s, n_d)$ having f_{en} as prefix,*

$$\frac{|\mathcal{S} \cap (\lambda(nodes(f_{en}))|}{|\mathcal{S}|} \leq NodeCoverage(f_e) \leq \begin{cases} \frac{|\mathcal{S} \cap (\lambda(nodes(f_{en}))| + |f_e| - |f_{en}| - 1}{|\mathcal{S}|} & * \\ 1 & \textit{Otherwise} \end{cases} \quad (1)$$

$$\frac{|\mathcal{S} \cap (\lambda(nodes(f_{en}))|}{|f_e| - 1} \leq NodeRelevance(f_e) \leq \begin{cases} \frac{|\mathcal{S} \cap (\lambda(nodes(f_{en}))| + |f_e| - |f_{en}| - 1}{|f_e| - 1} & * \\ \frac{|\mathcal{S}|}{|f_e| - 1} & \textit{Otherwise} \end{cases} \quad (2)$$

$* \quad |f_e| \leq |\mathcal{S}| + (|f_{en}| - |\mathcal{S} \cap (\lambda(nodes(f_{en}))|) + 1$

We then propose two DFS-based algorithms: constraintDFS (cDFS) and enhanced-cDFS (ecDFS).

constraintDFS (cDFS) is based on the non-recursive DFS. In cDFS, we start a DFS from the source node n_s. At each step of the DFS process, we (1) identify a resultant e-fragment when the destination node n_d is reached; (2) detect loops in a fragment generated and eliminate the fragment in question; and (3) calculate projected value ranges of the quantitative metrics by applying the formulae of Lemma 1 and eliminate a fragment if its projected value ranges do not overlap with those specified in τ.

Enhanced-cDFS (ecDFS) overcomes the extra overhead brought by the cDFS algorithm, in which the projected value ranges of the quantitative metrics are computed and compared with τ for every en-fragment generated. In ecDFS, at the time when the project value ranges are estimated for an en-fragment, we also estimate, under the worst-case scenario, for how many more steps the generated en-fragments can remain promising. This allows us to skip the calculation and comparison for the en-fragments generated in these search steps, hence further improve the performance.

3.2 Localized Search and Join

Targeting an important class of CAP queries, in which the keyword-based constraints are specified on nodes, we propose the *Search&Join* (S&J) algorithm. It leverages the local information around the nodes containing the keywords to calculate more accurate projected value ranges, and thus conducts more efficient pruning.

We first introduce a few notions that are critical for this algorithm. We use $\mathcal{S}_k$ to denote a set of nodes containing keywords in $\mathcal{S}$. We define *query nodes* as $\mathcal{S}_k \cup \{n_s, n_d\}$. A *query node sequence* (QNS) is a sequence of query nodes which always starts with n_s, ends with n_d, and consists of a subset of $\mathcal{S}_k$. We are interested in a special type of e-fragment, *exclusive* e-fragment (xe-fragments), which links two query nodes but does not go through any query node. The *constrained sequence join* operation takes as input τ from a CAP query, a set of QNSs based on the keywords in τ, and sets of exclusive e-fragments between every pairs of query nodes, and computes the e-fragments that satisfy the CAP query by concatenating the xe-fragments with the guidance from the QNSs and validating the constraints τ.

The *Search & Join* (S&J) algorithm has two phases: the *search* phase takes as input the data graph G and the query $CAP(n_s, n_d, \tau)$, computes the QNSs, and issues mini-searches on pairs of query nodes to find the set of xe-fragments for each pair of query nodes; the *join* phase then produces the query results by conducting constrained sequence join on the QNSs and xe-fragment sets generated in the *search* phase.

Clearly, not all QNSs lead to valid e-fragments that satisfy τ. In addition, given a QNS qns that yields non-empty results, not all the xe-fragments between all pairs of adjacent query nodes in qns contribute to the final results. Following the "selection-push-down" scheme widely used in database system design, it is critical to minimize the cardinality of the participants of constrained sequence join. We accomplish so by
(1) identify and eliminate invalid QNSs, e.g. QNSs whose constrained sequence join result is empty; and
(2) for each QNS that may generate non-empty constrained sequence join results, identify and eliminate the xe-fragments that have no chance contributing to the results.

Given a QNS qns, qns is guaranteed to be invalid if there is no xe-fragment between a pair of adjacent nodes in qns, or the sum of the minimum lengths of the xe-fragments of adjacent node pairs in qns exceeds $\tau_{l_{max}}$, or the combined projected value ranges of the node coverage or relevance are guaranteed to fall outside of the ranges specified in τ. When a QNS is deemed invalid, it can be pruned immediately.

Beside generating minimal QNSs and xe-fragment sets, we also aim at exploiting minimum number of data nodes/edges in the search phrase in order to limit the search space, hence improving the performance. We accomplish this by carefully design the search steps and bookkeeping mechanism:
(1) For all node pairs that share the same starting node, one single BFS search is issued for generating the xe-fragments for all these node pairs.
(2) After each BFS expansion, the QNSs are evaluated to have the invalid ones pruned, and the newly generated intermediate e-fragments are leveraged to tighten constraints for all mini-searches.
(3) Each time, one BFS is picked to expand its intermediate e-fragments. The criteria is that we always pick the BFS search such that the expansion has the potential to prune the maximum number of invalid QNSs and restricts the search ranges of the BFSs of itself and other query nodes most sharply.

4. EXPERIMENTAL EVALUATION

We conducted extensive experiments to study the performance of our algorithms, constraintDFS (*cDFS*), enhanced-constraintDFS (*ecDFS*), and Search-and-Join(*S&J*), as well as existing Search-and-Filter algorithms based on Depth First Search (*S&F-DFS*) and

Bi-directional Search (*S&F-BIS* [12]). The experiments were carried out on a desktop PC running Red Hat 4.1.2 with dual Intel(R) Core(TM)2 2.40GHz CPU and 4GB memory.

Our experiments were conducted on two RDF datasets: *DBpedia* (1504K nodes, 5.4M edges) [1] and *Chem2Bio2RDF* (139K nodes, 1.8M edges) [7]; both have been widely used in the literature. We tested the algorithms on many randomly generated CAP queries, varying source and destination nodes, keyword sets and constraints. Same trends were observed. Due to space limitation, we will report our experimental results only on the Chem2Bio2RDF dataset, as the paths in this dataset are much longer, putting significantly more stress on our algorithms and highlighting the impact of various parameters to the algorithms, and with only queries in which the parameters on node coverage and node relevance vary [2]:

We compared the hot run of the algorithms and measured the elapsed time in *ns*. Please note that as our algorithms improve the performance over the S&F algorithms by several orders of magnitude, to better illustrate the difference, we plot the results in logarithmic scale.

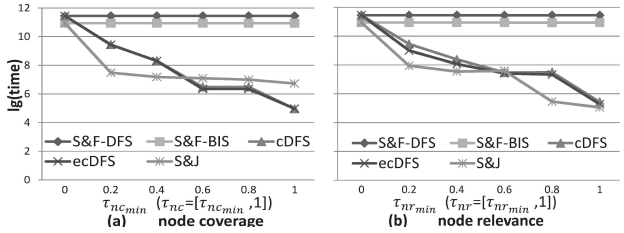

Figure 2: Performance Comparison: Varying τ_{nc} and τ_{nr}

As shown in Fig. 2(a) and 2(b), keyword-based constraints have no impact to the S&F algorithms. Our algorithms, which take advantage of such constraints, significantly outperform them, especially when the constraint is strict, e.g. $\tau_{nc_{min}}$ (or $\tau_{nr_{min}}$) is close to 1, as more intermediate results are pruned. Even when the constraint is loose, the performance of the S&J algorithm is significantly better than others, as it has much smaller search scope, thanks to the local information it takes into account. It is also worth pointing out that our DFS-based algorithms and S&J algorithm react differently when $\tau_{nc_{min}}$ changes: when the node constraint becomes tighter, e.g. closer to 1, cDFS and ecDFS are very efficient, due to their strong pruning power and small overhead; but when the constraint is relatively relaxed, e.g. < 0.5, the S&J algorithm is able to take advantage of local information around the query nodes to limit the search space and thus is much more efficient (by two orders of magnitude) than cDFS and ecDFS.

5. APPLICATIONS

CAP search can be used as a stand-alone search tool in domain-specific applications, such as drug discovery, as well as integrated into a high level query language to enhance its expressiveness.

In cheminformatics, a drug could affect a disease by affecting proteins and genes in a treatment process. By examining the paths from a drug to a disease, domain experts can assess the effectiveness of the drug before conducting chemical experiments [7]. Existing path discovery tools on Chem2Bio2RDF [7] can only find all paths between drug and disease, leaving the domain experts doing the filtering manually or relying on other tools to do so, rendering them impractical in dealing with the large data set and the subtle constraints demanded by the domain experts. Based on the research presented in this paper, we developed a tool that features CAP search queries [20], which enables the domain experts to compose CAP search queries, execute it, get results instantly, and adjust the constraints to alter the results as the research leads them.

[2] queries in each category shares the same $\tau_{l_{max}}$ and $\mathcal{S}$

If a drug is considered to be effective in treating certain disease, its side-effect should also be considered [14]. As such data entries do not appear directly on the paths between drug and disease, more complex pattern matching is required in addition to path finding to integrate constraints about side-effects into the CAP search queries for drug discovery. We propose cSPARQL to integrate the CAP search into the structured search of SPARQL [16] by introducing (1) *path variables* for expressing arbitrary e-fragments in a graph pattern; and (2) a set of quantitative metrics functions as defined in Sec. 2 for specifying the length and keyword-based constraints. More details about cSPARQL can be found in [19].

6. SUMMARY AND FUTURE WORK

In this paper we identify the problem of discovering acyclic paths between two given nodes in a directed graph under keyword-based constraints (CAP). We introduce the notions of coverage and relevance for specifying subtle relationship between paths and the keyword set. We propose algorithms, including cDFS, ecDFS and S&J, to efficiently evaluate CAP queries. Our empirical evaluation proved that our algorithms outperform existing Search-and-Filter algorithms using both DFS and bidirectional search and improve the performance by several orders of magnitude. We further discuss the applications of CAP queries and propose cSPARQL, an extension of SPARQL, to integrate CAP queries and the structured search on graph data.

7. REFERENCES

[1] http://wiki.dbpedia.org

[2] D. Abadi, *et al.* SW-Store: a Vertically Partitioned DBMS for Semantic Web Data Management. In *VLDBj*, 2009.

[3] F. Alkhateeb, *et al.* Constrained Regular Expressions in SPARQL. In *SWWS*, 2008.

[4] K. Anyanwu, *et al.* ρ-Queries: Enabling Querying for Semantic Associations on the Semantic Web. In *WWW*, 2003.

[5] K. Anyanwu, *et al.* SPARQ2L: Towards Support for Subgraph Extraction Queries in RDF Databases. In *WWW*, 2007.

[6] K. Anyanwu, *et al.* Structure Discovery Queries in Disk-Based Semantic Web Databases. In *DOI*, 2008.

[7] C. Bin, *et al.* Chem2Bio2RDF: a Semantic Framework for Linking and Data Mining Chemogenomic and Systems Chemical Biology Data. In *BMC Bioinformatics*, 2010.

[8] G. Bhalotia, *et al.* Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, 2002.

[9] B. Dalvi, *et al.* Keyword Search on External Memory Data Graphs. In *VLDB Endowment*, 2008.

[10] G. Fletcher, *et al.* Scalable Indexing of RDF Graphs for Efficient Join Processing. In *CIKM*, 2009.

[11] H. He, *et al.* BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD*, 2007.

[12] V. Kacholia, *et al.* Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB*, 2005.

[13] K. Kochut, *et al.* SPARQLeR: Extended SPARQL for Semantic Association Discovery. In *ESWC*, 2007.

[14] S. Lee, *et al.* Building the Process-drug-side Effect Network to Discover the Relationship Between Biological Processes and Side Effects. In *BMC Bioinformatics*, 2011.

[15] G. Li, *et al.* EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *SIGMOD*, 2008.

[16] E. Prud'hommeaux, *et al.* SPARQL Query Language for RDF. In *W3C Recommendation*, 2008.

[17] J. Ruoming, *et al.* Computing Label Constraint Reachability in Graph Databases. In *SIGMOD*, 2010.

[18] L. Zhang, *et al.* An Enhanced Model for Searching in Semantic Portals. In *WWW*, 2005.

[19] M. Zhou, *et al.* Efficient Association Discovery with Keyword-based Constraints on Large Graph Data. *Tech. Report, IUB*, 2011.

[20] M. Zhou, *et al.* Conkar: Constraint Keyword-based Association Discovery. *CIKM Demo*, 2011.