Undefined 1 (2009) 1–5
IOS Press

Semantic answer graphs for keyword queries on RDF/RDFS graphs [1]

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Parthasarathy K [a,*,**], SreenivasaKumar P [b] and Dominic Damien [a]
[a] *Indian Space Research Organisation,New BEL Road,Bangalore, INDIA*
E-mail: {parthas,dominic}@isro.gov.in
[b] *Indian Institute of Technology, Madras,Chennai, INDIA*
E-mail: psk@iitm.ac.in

AbstractKeyword search is an easy way to allow inexperienced users to query an information system. It does not need the knowledge of a specific query language or underlying schema. Recently answering keyword queries on semantic data repositories has emerged as an important research topic. In particular, many efforts focus on RDF date due to wide spread adoption of RDF for the representation of both open web and enterprise data. RDF and RDFS are popular frameworks for representing data and meta-data for domain knowledge. In this article, we focus on keyword-based querying of RDF data. Current techniques adopted for supporting keyword queries on RDF data suffer from drawbacks such as inefficient path exploration, inability to exploit semantic characteristics and restriction on search within a distance neighbourhood. We highlight the drawbacks of the existing systems with different examples. We present a generic approach which adopts a pruned exploration mechanism, where component sub-graphs are formed using closely related nodes, pruned and joined using suitable hook nodes. The component sub-graphs are enlarged selectively depending upon the relative closeness of the keywords. Semantic property class/subClassOf of the RDF graph is our approach during different phases of the algorithm. A new indexing and ranking mechanism that exploits semantic relationships is also proposed. The working of the algorithm is illustrated using AIFB institute data represented as an RDF graph on keyword queries with different characteristics. The experimental results on other sets namely DBLP and LUBM and a comparative analysis with other approaches are presented. We find that with the help of techniques proposed in this paper, we can give accurate and relevant answer graphs for several keyword queries where existing techniques fail. This is mainly due to the fact that the proposed techniques has no restriction on mappings and also exploit the semantic relationships from the underlying RDF/RDFS dataset.

Keywords: RDF, RDFS, keyword search, answer graph

1. Introduction

Query processing over graph data has attracted considerable attention recently as an increasing amount of data which is available on the web, XML data sources and relational sources can be modeled in the form of graphs. RDF as a framework for web resource description appears to have gained a greater momentum on the web and an increasing collection of repositories of data are modeled using RDF framework. RDFS, the RDF schema language provides guidelines or meta data information on how to use the RDF data. RDF

[1] Footnote in title.
[*] Corresponding author. E-mail: editorial@iospress.nl.
[**] Do not use capitals for the author's surname.

data expressed in the form of triples can be represented as a directed graph structure with each triple is an edge from subject to object with the predicate as the label of the edge. Notable examples are biological and chemical databases, Web-scattered data, health-care, Personal Information Systems[18] where emails, documents and photos are merged into a single repository and launch vehicle design data where details about vehicle stages, parameters and stage sequence events are maintained. In these class of applications, raw data may not be graph structured at the first level but implicit connections will provide a graph structure. The largeness and complexity of data-sets in these domains makes their querying a challenging task. Also these data sets calls for employing semantic queries instead of simple keyword queries to closely express the information needs. For constructing queries, one has to either explore the ontology through graphical tools or one neeeds knowledge of query language like SPARQL.

As keyword queries do not require the users to know complex query language or know details regarding the underlying schema, much work has been carried out on keyword search on databases represented as a graph structure [4,6,16], tree structured data[10] and recently on RDF data represented as graphs [15,12,13]. Existing approaches for keyword queries on RDF data suffer from (i) returning incorrect answers, *i.e.,* answer graphs returned do not correspond to real subgraphs; (ii) restrictions on choice of neighbour nodes by a distance metric; (iii) inefficient path exploration and (iv) most notably does not exploit semantic characteristics of the RDF graph. Using the standard representation of RDF/RDFS data as graphs with types, classes as nodes and properties as edges, we provide a generic approach for answering keyword queries where semantic properties are exploited. Our approach blends traversal with exploiting semantics to arrive at the correct answer graphs.

Our specific contributions in this article are as follows:

- We bring out the limitations of the existing approaches using concrete examples and present an efficient generic method for generating answer graphs for keyword queries on RDF graphs that exploits semantic relationships*(e.g type/ subClassOf)*. In our approach, keywords are mapped to different types of nodes/edges, unwanted nodes are pruned during several stages and no distance limits are imposed. The distance neighbourhood restriction is implicitly managed in our approach.
- We propose a hierarchical ranking model, taking into consideration structural characteristics of RDF graph and also compactness of answer graphs from the graph perspective.
- We have also conducted performance studies on standard bench-mark data sets using various queries with different characteristics.

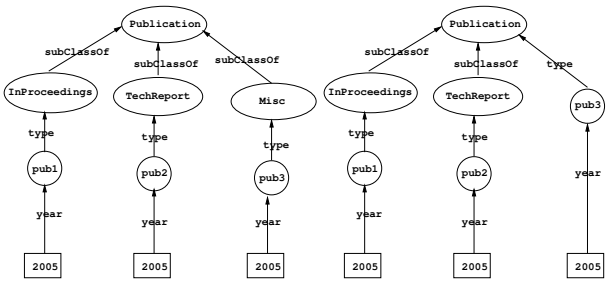
The article is organized as follows. Section 2 presents the motivation for this research activity. Section 3 describes the preliminaries of the problem. Section 4 presents the algorithm description. Section 5 presents the component expansion strategies, Section 6 presents indexing, Section 7 presents ranking, Section 8 presents the design implementation and analysis and Section 9 presents the analysis and results.

2. Motivation

The present implementations of keyword search on RDF graphs adopt the approaches from keyword search on RDBMS and XML data[4,11,6][4,11,6]. The algorithm first identifies nodes matching keywords of interest, traverses the graph for discovering possible interconnecting paths between identified parts. Candidate solutions built out of the connections found are scored and ranked. For the exploration of data graph, only substructures in the form of trees with distinct roots are computed and the root is assumed to be the answer. The algorithms uses backward search[4] and bi-directional search[16]. They work on the premise that the goal of the query is to find data entities.They typically assume a simpler data model which is a rooted directed graph without the nuances of classification and property hierarchies. Also in the existing approaches, the mapping of the keywords provided are confined only to data nodes(D-Node as defined later in section 3) of the RDF graph. In contrast, in a RDF graph search, classification and properties are important. Keywords provided might also be mapped on to edges. Often there is a need to query these data sets not only to find entities but also to identify a chain of relationship that exists in them. This calls for a need to identify meaningful graph structures with rich structural relationship such as cycles and not necessarily restricted to trees. In answering queries, we also need to exploit classification and properties of the RDF/RDFS data to arrive at the correct answer graphs.

The main inspiration for our work are [14,15], where graph based approach for answering keyword queries is presented. The limitations of the existing approaches are illustrated in the following paragraphs

- The methods adopted explores all possible paths. For example in the AIFB data set, taken from AIFB Institute, University of Karlsruhe[1] for the keyword query $\{publication, 1999\}$, all the *publication* data nodes will be brought to memory for processing. Unimportant nodes are logically pruned by means of scoring and ranking mechanism. They do not attempt to prune the search space in the earlier exploration phase of the graph search. As the input RDF graph becomes bigger, traversing through all the nodes in the exploration phase for all possible paths as adopted in [15] will lead to performance issues. An alternate approach is needed where the exploration has to start with a initial set of closely related nodes/edges to nodes mapped to keywords and then expand the node set. Also pruning unwanted nodes that cannot contribute to overall answer graph framework is essential and to be implemented in different phases of the algorithm. This is the fundamental essence of the approach adopted in this article.
- *type/subClassOf,* $subPropertyOf$ and other semantic relationships available in RDF graph framework are not being exploited suitably for keyword queries. Consider the following RDF graph fragments as shown in figure 1.



(a) *Data with all subClassOf* (b) *Data with mixed pattern*

Figure 1. *Two patterns of RDF graph fragment*

For the query$\{publication, 1999\}$, [15] does not function as it deals with RDF graphs with only data nodes. Even in improved approach of [14], the system will either return incorrect answer graphs or no answer graphs as per the scheme described in the paper.

- Existing algorithms use indexing only to identify a set of nodes containing query keywords. Standard inverted index used is inadequate to graph data since it is very difficult to identify structural relationship using inverted index. It is critical and natural to exploit indexes that provide semantic graph connectivity information to speed up searches.
- In order to limit the traversal during graph exploration step,[15] fixes a distance metric *d* to obtain a graph containing all elements related to a keyword of the query. The nodal elements are within a distance *d* to the keyword. This structure which is called the *d* neighborhood is used for further processing. In a keyword search, the major assumption is that the user is not fully aware of the underlying data model. There is a possibility that the keywords provided may be be far apart but still might have a strong connection chain. But the distance metric *d* chosen may produce a graph which is not connected. The restriction of the maximum distance between entities is to restrict exploration space and also to justify the fact that smaller distances are likely to contain the missing links the user looks for. But the user is providing the keywords without the full knowledge of the schema. So some of the keywords provided could be far apart. If *d* happens to be small then the graph constructed becomes disconnected and chained relationship links cannot be extracted. The user has to experiment with different *d* for exploring the graph. In the example illustrated in [15], the keywords *Philip X-Media publication* are closer and equal-spaced. In some scenarios,given the set of keywords, some keywords could be closer and some could be farther(from a graph perspective). In example 3 as illustrated later in the paper, the mappings to *(AIFB, journal, titles)* are not equi-spaced. The mapping to *journal, titles* are closer whereas mapping to *AIFB* is farther (d > 1). While finding out the neighbouring nodes, even though the knowledge about strength of relationship is available in terms of distance between the nodes, the closeness or the farness between the nodes is not exploited during the exploration phase.
- To improve the search performance, [14] uses summarisation technique where relations in en-

[1] http://km.aifbunikarsruhe.de/ws/eon2006/ontoeval.zip

tities of distinct types are summarised. But the summarisation which bundles all the entities of the same type into one node of its summary has the serious limitation of loosing too much information.This leads to results which are false positives and false negatives. For example, the query $\{Staab, 1999\}$to find out publications of Staab with year 1999 will include answer graph with nodes containing the keywords even though there might not be a publication of Staab in year 1999.

In this article, we propose a novel approach for construction of answer graphs to keyword queries for RDF/RDFS data represented in graph format, specifically addressing the above mentioned issues. The approach is an hybrid approach that attempts to provide possible set of answers in an efficient way. For the keywords, initial sub-graphs are identified, pruned for removing unimportant nodes and then are hooked together using suitable join nodes. For each mapped node or edge for the keyword, the *type* node and a closest set of neighbouring *class* nodes are identified to form a component sub-graph using *type* and *relationship* indexes. Nodes from the sub-graphs, which cannot contribute to the overall structure of the query are removed. The modified set of sub-graphs are joined using suitable hook elements iteratively to construct the final answer graph for the keyword query.The novelty of the our approach is that we exploit the semantics of the graph at each one of the stages of the algorithm. Also the approach does not fix any distance prior. An upper bound on the distance is implicitly managed in our approach. If the sub-graphs have neighbouring class nodes on common, implying the closeness of the corresponding keywords, the pruning step maintains the existing class nodes for processing. If the keyword sub-graphs do not have class nodes in common implying that keywords are not too closer, the pruning step adds a chain of class nodes for processing. Intuitively the sub-graphs are enlarged selectively using relationship chains. The algorithm then explores the new set of sub-graphs to find suitable hook elements for joining and the build up the answer graph for the keyword query. In short, our approach constructs answer graphs by judicious expansion of additional nodes and edges, exploiting RDF graph classification and property features. We strongly believe that the framework can be generalised to exploit even other RDFS features like inverseOf,subPropertyOf.

3. Preliminaries

3.1. RDF/RDFS

An RDF/RDFS dataset is a graph composed by triples formed by subject, predicate, object in that order. The dataset can be viewed as a directed graph.

Definition 1: An RDF/RDFS graph G consists of tuple of the form (N, E, L) where

- N is a finite set of nodes which is a disjoint union of *C-Nodes representing types, EN-Nodes entities and D-Nodes data values i.e N=C-Nodes* $\uplus$ *EN-Nodes* $\uplus$ *D-Nodes.* In the RDF fragment shown in figure 2, *Person, Organization* are some of the nodes which represent *C-Nodes*, *AIFB, Efficient Algorithms* are some nodes which represent *D-Nodes* and *pub1, proj1* are some nodes which represent *EN-Nodes*.
- E is the finite set of edges connecting nodes n_1,n_2 with n_1, n_2 ϵ N. The different types of edges are *IE-Edges*(inter-entity edges), *EA-Edges*(entity-attribute edges) or *Type/SubClassOf* edges. In *Figure 1 Author, CarriesOut are examples of IE-Edges and Title, Journal are examples of EA-Edges.*
- L is a labeling function which associates a label l for an edge. *L = L(IE-Edges)* $\uplus$ *L(EA-Edges)* $\uplus$ *{Type,SubClass} where L(IE-Edges) represents labels for inter-entity edges, L(EA-Edges) represents labels for entity-attribute edges.* The following restrictions apply on *l*:
  - *l* $\in$*L(IE-Edges) if and only if* n_1,$n_2$$\in$ *EN-Nodes*
  - *l* $\in$*L(EA-Edges) if and only if* $n_1$$\in$ *EN-Nodes and* $n_2$$\in$ *D-Nodes*
  - *l = SubClassOf if and only if* n_1,$n_2$$\in$ *C-Nodes*
  - *l = Type if and only if* $n_1$$\in$ *EN-Nodes and* $n_2$$\in$ *C-Nodes.*

type and *SubClassOf* are two predefined type of edges which captures class membership of an entity and class hierarchy. In Section 3 we use a notation *CR-Node.* This is defined by the following property:

CR-Node is a C-Node which has an inter-entity relationship with another C-Node. For example the C-Node Researchgroup is a CR-Node for the C-Node Organization and vice-verse.

Figure 2 shows a fragment of RDF graph containing data taken from AIFB institute, University of Karlsruhe which will be used for illustration of the algorithm in this article. The fragment models the infor-

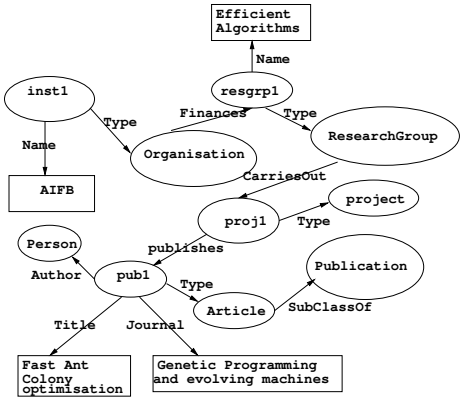


Figure 2. *RDF Graph fragment from AIFB Institute Data*

mation *related to financing of Research group by Organization AIFB, Projects carried out by the Research group and publications from those projects.* This fragment models a chain of relationship.

3.2. Chains

Relationship chain A *relationship chain RC* is a finite sequence of relations $\{RC_1, RC_2 \dots RC_n\}$ where $RC_i$is a label associated with *IE-edge or EA-edge or type/subClassOf.* For e.g in Figure 1 *{carriesout, publishes, title} is a relationship chain*

Class chain A *class chain* CC corresponding to a relationship chain *RC* is a finite sequence $\{CC_1, CC_2 \dots CC_n\}$ where

$CC_i (1 \leq i \leq n-1)$**is** a C-Node

CC_n is a D-node if $RC_n$is a IE-Edge
CC_n is a C-node if $RC_n$is a EA-Edge

For the relationship chain {*carriesout, publishes, title}, {ResearchGroup,,Project,Article,Fast ant. . .optimisation} is one class chain*
The length of a relationship chain is equal to the number of relations in the chain. For the example chain, the length is 3. The length of the class chain will be one more than relationship length.
Two relationship chains $RC_i$and RC_j are *intersecting* if *CLASSCHAIN(RC$_i$) $\cap$ CLASSCHAIN(RC$_j$) $\neq \phi$.*

C ϵ*CLASSCHAIN(RC$_i$) $\cap$ CLASSCHAIN(RC$_j$) is called intersect-node*
Two relationship chains RC_1= $\{S_1, S_2 \dots S_n\}$and RC_2 = $\{T_1, T_2 \dots T_n\}$are called similiar chains if
for all i, $1 \leq i \leq n$ S_i= $T_i$or S_iSubClassOf $T_i$or T_iSubClassof S_i

3.3. Problem statement

Problem Given the RDF/RDFS graph, we are looking at the construction of set of ranked answer graphs to the keyword query.

Queries A keyword search query *KQ* consists of a list of keywords {k_1,...k_n}. Formally a semantic query associated with the search query KQ is a triple $SQ = (KQ, \phi)$ where ϕ is a function which maps elements of *KQ* to *D-nodes, C-Nodes, IE-edges and EA-edges*. *EN-nodes* will not be used for queries as they are internal to the RDF graph. Given this list of keywords, the answer graph to the semantic query is a minimal sub-graph *A* s.t

- every $k \in K$is mapped to a vertex or an edge in *G*
- The sub-graph *A* consists only of *nodes/edges associated with keywords of KQ, C-Nodes, CR-Nodes connected by IE-edges and nodes connected by EA-edges*. The edge labels as per the defined labeling function L also form part of the answer graph
- *A* is connected i.e from every graph element of A to every other graph element of A, there exists a path.
- *A* is minimal in the sense that no sub-graph of A can be an answer to Q. If keyword node is removed, then that keyword is not matched. If a non keyword node is removed, the graph becomes disconnected.

Different mapping of the keywords will lead to multiple answer graphs. The answer graph will be used to construct SPARQL queries.

4. Algorithm Description

The term mapping step maps each keyword k_i to a set of nodes/edges of the RDF/RDFS graph using string search mechanisms. By taking one mapped node for each keyword, a node-list *NL* is formed which is an input for answer graph construction. For each node-list *NL,* an answer graph will be constructed in three steps namely *Component_subgraph_Creation, Pruning and Hooking*. This is shown in CREATE-ANSWER-GRAPH procedure.

CREATE-ANSWER-GRAPH(K, G)

1 $SG = \emptyset$ // set of set of graphs
2 **foreach** keyword $k \in K$
3 $SG = SG \cup \{$ SUBGRAPHS(k, G) $\}$
4 $CG =$ COMPUTE-SUBGRAPH-SETS-FOR-PRUNNING(SG)
5 $A = \emptyset$ // set of answer graphs
6 **foreach** graph set $C \in CG$
7 $P =$ PRUNE-GRAPHS(C) // set of graphs
8 $A = A \cup \{$ HOOK-GRAPHS(P) $\}$
9 **return** A

SUBGRAPHS(k, G)

10 $S = \emptyset$ // set of graphs for k
11 **foreach** node $n \in$ NODE-SET(k, G)
12 $S = S \cup \{$ COMPUTE-SUBGRAPH(n, G) $\}$
13 **return** S

Component Subgraph Creation Initially the component sub-graph for each element in *NL* is empty. An element from *NL* is taken and the associated node/edge category is identified. If the element is associated with a *C-Node* ,the *C-Node*, its *CR-Node* along with the associated edges are added. If it is associated with a *D-node,* then the *C-Node* for the class to which this *D-node* belongs, *CR-Node*s for the *C-Node* and associated edges are added. We restrict the *CR-Nodes* only to the nodes that are related to the *D-Node* through the corresponding *EN nodes.* If it is associated to an *IE-Edge* ,the *C-Nodes* corresponding to the *EN-Nodes* are added and if it is associated to *an EA-Edge,* a dummy node for the attribute side, *C-Node* for the class to which the entity belongs and *CR-Node*s for the *C-Node* are added. The component sub-graphs constructed as above for all node in *NL* will act as input to the *pruning* step. This step is shown as pseudo code in *Algorithm 2*. Formally, a component subgraph is formed using *C-Nodes, CR-nodes associated with C-Nodes and related edges.* The component subgraph formation for a D-Node association is shown in figure 3.

CREATE-COMPONENT-SUBGRAPH(ge)

1 CSG $= \{ge\}$ // component subgraph
2 **if** ge is C-NODE
3 CSG $= CSG \cup \{$CR-NODES(C-NODE)$\}$
4 **elseif** ge is D-NODE
5 CSG $= CSG \cup \{type($D-NODE$)\}$
6 CSG $= CSG \cup \{$CR-NODES($type$(C-NODE)$\}$
7 **elseif** ge is EA-EDGE
8 CSG $= CSG \cup \{type($EN-NODE(EA-EDGE$))\}$
9 CSG $= CSG \cup \{dummynode\}$
10 CSG $= CSG \cup \{$CR-NODES(C-NODE)$\}$
11 **elseif** ge is IE-EDGE
12 CSG $= CSG \cup \{type($EN-NODES(IE-EDGE$))\}$
13 CSG $= CSG \cup \{$CR-NODES(C-NODES)$\}$
14 **return** CSG

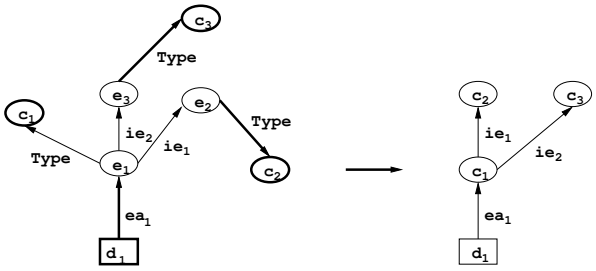


Figure 3. *Component subgraph for a D-Node association*

Pruning In this step, the algorithm prunes the loosely hanging nodes which possibly cannot be utilised for hooking. For each pairwise component subgraph $\{CC_i, CC_j\}$, common nodes are identified. Common nodes are defined by the term *similiar* nodes. This is found by the intersection of the node-list pair. Two nodes n_1, n_2 are *similiar* if they satisfy any one of the following property:

1. $n_1 = n_2$ i.e they correspond to the same nodes of the graph
2. $n_1, n_2$are related by *type/subClassOf* relationship chain

For e.g if *Publication* ϵ *CC*$_i$, *TechReport* ϵ CC$_j$*and TechReport subClassOf Publication* then

$\{Publication, Journal\} \epsilon \{CC_i\} \bigcap \{CC_j\}$

The union of all the *similiar* nodes found by considering all pairs of component sub-graphs, represents the full *C-Node* set. The *C-Nodes* obtained by pairwise intersection of component sub-graphs constitute the active C-Node set. The nodes to be pruned is the compliment of the active C-Node set with respect to the full C-Node set.

Formally the full C-Node set *FCNS* is given by $\bigcup_{i,j}\{CC_i, CC_j\}$
The active C-Node set *ACNS* is given by $\bigcup\left[\bigcap^{i,j}\right]CC_i{,}CC_j$
The pruned C-Node set *PCNS* given by $PCNS = FNCS \setminus PCNS$

The pruned nodes along with the associated edges are removed from the corresponding component sub-graph. The new list of component sub-graphs obtained will act as input to *hooking* step. For two component sub-graphs, the pruning sequence is illustrated in figure 4.

Subgraph Enlargement If the intersection of the *C-nodes* for a pair of component sub-graphs $\{CC_i, CC_j\}$ is empty, it indicates that the nodes mapped for the keywords from the RDF/RDFS graph are not close neighbours in terms of relationships and we explore for *relationship chains* connecting member elements of $\{CC_i, CC_j\}$. The sub-graphs are enlarged using *class chains* corresponding to those *relationship chains*. If multiple chains exists, the chain with the least number of C-Nodes is used. If the chain has n nodes, $\frac{n}{2}$ nodes of the chain is linked to one sub-graph and other $\frac{n}{2}$ nodes to the other subgraph with centre *C-Node* added to both. This step is shown as pseudo code in *Algorithm 3.*

PRUNE-GRAPHS(CE)

```
1  // input set of graphs for pruning
2  FNCS ← ⋃_{k,l}{CS_k, CS_l}
3  foreach pairwiseset{g, h} ∈ CE
4
5      if g ≂ h or vice-versa
6          Add keywordnodes to smaller element
7          Remove g or h
8      ICNS ← ⋂{g, h}
9      if ICNS = ∅
10         foreach pair of C − Nodes ∈ {g, h}
11             RC ← shortest RC chain {RC_1 ... RC_m}
12             CC ← ClassChain{RC}
13             Add left half of nodes of CC to g
14             Add right half of CC to h
15         ACNS = ⋃{ICNS}
16         PNCS = FNCS - ACNS
17         Strip Nodes of PrunedCNodeSet in each CS_t
18
19     return FNCS
```

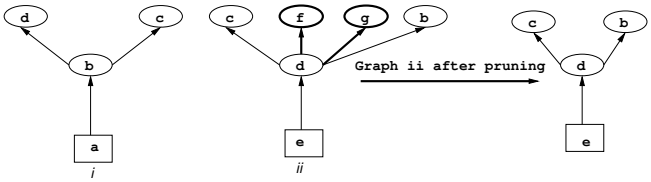


Figure 4. *Pruning illustrated for 2 component sub-graphs*

Hooking In this step, we start to explore whether a node of a component sub-graph can be hooked on to a *similiar* node of another component sub-graph. Initially any component sub-graph with lowest cardinality of *C-Nodes/CR-Nodes* is chosen for starting the hooking operations. The property of *similiar* nodes defined earlier is also used for hooking operation. Once nodes to be hooked are identified, the corresponding component sub-graphs are glued together. Nodes which are duplicates are removed and a new glued component sub-graph is used for further hooking operations. This process is iterated until no more nodes could be hooked. The final component sub-graph arrived at is analyzed for hanging nodes and they are cut off. A node n is a hanging node if it is not a keyword mapped node and it has only one edge associated with it. In figure 5, n_6, $n_7$are loosely hanging nodes and hence $\{n_6, e_6\}$and $\{n_7, e_7\}$will be pruned. The closely connected subgraph thus formed will be the answer graph for the keywords presented by the user. This step is shown as pseudo-code in *Algorithm 4.* The hooking step is illustrated in figure 6.

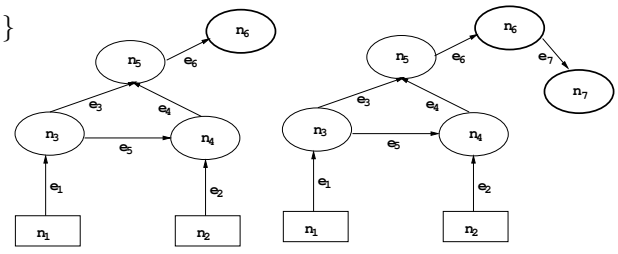


Figure 5. *Illustration of hanging nodes*

HOOK-GRAPHS(HE)

```
1  // input set of graphs for hooking
2  PRGS←{CS_{t_1} ··· CS_{t_l}}
3  while cardinality(PRGS) <> 1
4
5      g ← CS_t with lowest node cardinality
6      h ← CS_r with minimum relationship chain
7      CS_new ← merge{g,h}
8      PRGS ← merge{g,h}
9      PRGS ← PRGS ⋃ CS_new
10 remove hanging nodes of PRGS if any
11 return PRGS
```

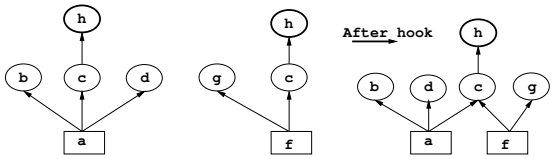


Figure 6. *Hooking illustrated*

4.1. Homomorphic graph structures

The graph structures created during the *component_subgraph_creation* phase may have some similiarity in their structures. We look for exploiting such homomorphic structures that can be merged. Two graph structures g_i, g_j for different D-Nodes are homomorphic, if

1. $C - Nodes(g_i) = C - Nodes(g_j)$ i.e the C-Nodes are same
2. $C - Nodes(g_i) = C - Nodes(g_j)$ CR-Nodes are same

C-Node,CR-Nodes are identical. The *D-Nodes* will be different. For e.g, if the keywords are *optimisation,genetic*, in the RDF graph fragment shown in figure 2, the keywords are mapped to the D-Nodes *title* and *journal*. The component sub-graphs constructed for these keywords will be homomorphic.

In a similiar manner, for EA-edges, two graph structures are homomorphic, if

1. $C - Nodes(g_i) = C - Nodes(g_j)$ i.e the C-Nodes are same
2. $C - Nodes(g_i) = C - Nodes(g_j)$ CR-Nodes are same

C-Node,CR-Nodes are identical. The *Dummy D-Nodes* will be different.

4.2. Towards optimal subgraph hooking strategies

In this section, we discuss on the strategies adopted during pruning and hooking stages of the algorithm.

As pointed out earlier, since some keywords could be closer and some of them could be distance apart, the algorithmic strategy should be optimal in the sense of restricting the number of *CR-Nodes* that are used to merge the sub-graphs. In our approach, we adopt a ***relationship balanced expansion*** for controlling the expansion of sub-graphs. We also use indexing which supports this expansion and which allows to determine the relationship distance. The details of indexing mechanism will be discussed in ***Section 5***.

– ***Relationship balanced expansion:*** The algorithm attempts to balance the number of CR-Nodes for each subgraph. Formally during the hooking stage, one of the the next sub-graphs to consider for hooking is the subgraph with the smallest number of CR-Nodes. Once this is identified, the second subgraph to be considered for hooking is identified based on the hook node which is chosen based on the ***Equi-spaced expansion.***

Intuitively the subgraph expansion must follow the strategy of equal-spaced expansion in each subgraph:

– ***Equi-spaced expansion****:* This strategy decides which node to hook during the hooking stage. Hooking is a logical expansion of the subgraph and the hook node is chosen in the increasing order of distance from the *CR-Nodes* of the subgraph under consideration. Formally node *h* to be hooked for subgraph $\boldsymbol{C_i}$ ***is the node with shortest path relationship chain*** $\{CR - Node_{ij}, \ldots h\}$

5. Indexing mechanisms

In order to enhance the performance of pruning and hooking steps of the algorithm and also manage relationship chains, indexes are maintained. The following indexes are maintained

Type Index For each $EN-Node$, the associated type is maintained as a list and for each $type$, the list of associated $EN - Nodes$ is maintained.

Relationship Index For each $EN - Node$, the list of associated $EN - Nodes$ to which it is related through a $IE - edge$ is maintained both as a forward and as a backward reference

SubClass Index For each $C-Node$ that has a $subClass$ relationship with other $C - Nodes$, an index list is maintained.

In figure 7, for the *R*esearcher, the EN-Nodes *r*es1 and *r*es2 will form the list entries for its *t*ype index and *R*esearcher will be a list entry for EN-Node *r*es1. *r*es2(Philip Camiano) and *i*nst(AIFB) will be entries in their respective Relationship index through the IE-edge *w*orksAt. *A*rticle and *P*ublication are maintained as part of SubClass index.

The *type index* is used to associate the type during the component_sub-graph_creation phase. The *relationship index* is used to associate the related *CR-nodes* and the *subClass index* is used to identify *Class chains* during the pruning and hooking phase.

6. Ranking

Since multiple answer graphs could be constructed through our approach, there is a need to meaningfully rank the answer graphs to identify top answers. In existing approaches of keyword search in relational sources, standard IR-ranking formula is used. In [15], the path length is used for ranking the answers. [11] provides a comprehensive ranking mechanism for graph data. Since we adopt a graph model, ranking from a graph perspective is equally important. Our approach is also similiar to [11], where we have used structural compactness as the criteria. We have also added two more factors i.e relationship relevance and node type relevance into our ranking model to take care of RDF graph attributes.

Compactness Relevance For the keywords which reflects the information needs of the user, compact answers should be preferred, i.e closer connections between the mapped nodes are preferred rather than the farther connections. From a graph perspective, this translates into structural compactness among the elements of the graph. The structural compactness of the answer graph is determined from two aspects: the compactness among the mapped nodes and the relevancy among the keyword nodes in terms of the nodes to which it is mapped. In our approach when the length of the *class chain* between the mapped nodes is larger, the compactness between the is smaller. If $n_i$and n_j are mapped nodes corresponding to keywords $k_i$and k_j we define structural compactness as follows:

$$SC(k_i, k_j \mid AG) = \begin{cases} \frac{1}{(chnl+1)^2} & if\, n_1 = n_2 \\ \frac{1}{2(chnl+1)^2} & if n_1 \neq n_2 \end{cases}$$

where *chnl = number of C-Nodes in the smallest class chain + 1*

Term Relevance Since the term mapping step uses *IR* model, the textual relevancy of the keywords with reference to the nodes mapped is one of the important attribute for ranking. For example the term *AIFB* gets mapped to node with project name *AIFB* or it gets mapped to title of a publication which has AIFB string or gets mapped to keyword of a publication which has AIFB string. The term affinity will be different for these mapping from IR. For each keyword element a matching score *TR* using standard IR approach can be computed. Combining compactness relevance and term relevance we have the following ranking formula:

$$RANKVAL(k_i, k_j \mid AG) = SC(k_i, k_j \mid AG) * (TR(k_i \mid AG) + TR(k_j \mid AG))$$

Node Relevance The node/edge to which the keyword is mapped also plays a prominent role for ranking. For example, the keyword *publications* gets mapped to the *C-Node Publication* or to a *D-Node* which has the string *publication*. The answer graph which has the class node should be ranked higher since the user intention will also be to get *publication* information rather than abstract information if the keyword specified is *publication*. *C-Nodes/IE-Edges* will have highest scores followed by *EA-Edges* followed by *D-Nodes*. The node relevance scores NR for all the mapped nodes is computed.

Relationship Relevance Since the fundamental approach to answer graph construction is identification of missing interconnection nodes, the neighbouring C-Nodes that contributes to the answer graph is an important parameter for ranking. It is computed as follows

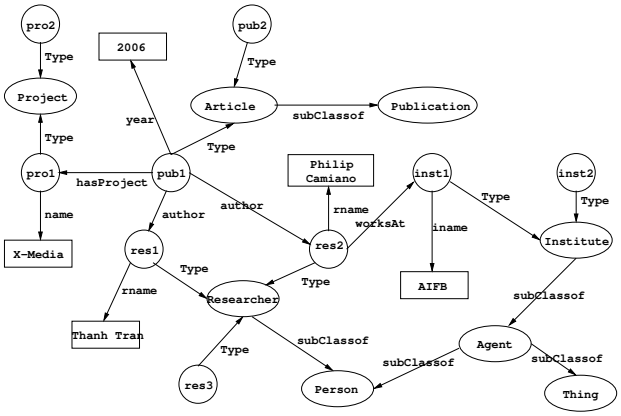


Figure 7. *RDF Example from AIFB Data*

$$RR = \frac{Number\ of\ C-Nodes\ in\ AG}{Total\ Number\ of\ C-Nodes} - \frac{Number\ of\ C-Nodes\ added\ AG}{Total\ Number\ of\ C-Nodes}$$

The overall ranking score is calculated as follows

$$\text{RANKVAL(AG)} = \sum RANKVAL(k_i, k_j \mid AG) + \text{NR(AG)} + \text{RR(AG)}\ 1 \leq i \leq j \leq n$$

7. Illustration of the algorithm for different query scenarios

The RDF fragment shown in figure 7 models the information related *to Project X-Media in institute AIFB, the researchers associated with the project and the publications.*

Example 1
Keyword Query: *X-Media Philip Publications*

This query has been taken for illustration to differentiate our approach with that of approach adopted in [15,14] as the same query has been illustrated there.

- Keyword *X-Media* will be mapped on to the nodes/edges containing the term *X-Media*
- Keyword *Philip* will be mapped on to the nodes/edges containing the term *Philip Camiano*
- Keyword *Publication* will be mapped on to the nodes/edges containing the term *Publications*

In the component sub-graph creation step, using the *C-Nodes* and *CR-Nodes* associated with the terms, the component sub-graphs as shown in figures 8a to 8c are constructed.

In the pruning step, the node *Institute* along with the edge label *WorksAt* associated with the subgraph for *Philip Camiano* is removed. *Article* node is not pruned as $\{Publication, Article\}$are similiar nodes.

In the hooking step, we start from the *Publication* node in *Figure 9* and hook it with the *Article* node in *Figure 8a*. Now in *Figure 8b* since already *Article* node is available, we choose *Article* node as it can be hooked with Article node of merged sub-graph *8a,9a*. Now there is no more component sub-graph to be considered for hooking we will have the answer graph as shown in figure 11a.
In [15], the system explores all possible nodes and edges of RDF graph . For the node *Publication,* all instances corresponding to the *Publication* are added. If the number of instances are more, then multiple edges will be added. This will lead to explosion of nodes and edges and also the need to have more traversals to find out the missing link nodes or edges. This also illustrates the scenario where the RDF semantics *type/SubClassOf* is exploited.

Example 2

For the following keyword query, the RDF data fragment as shown in figure 9 is used.

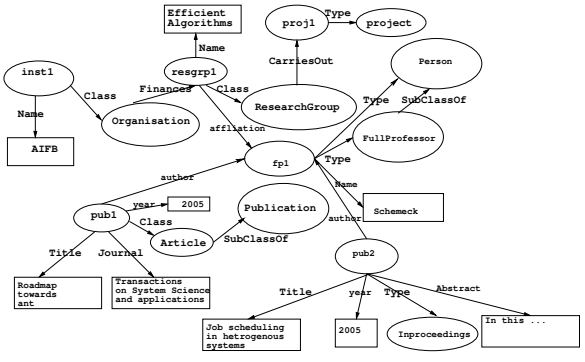
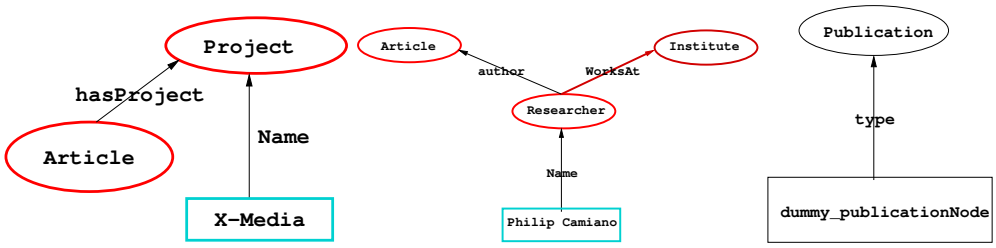


Figure 9. *RDF fragment for example 2*

Keyword Query: *abstract schmeck 2005*

The term mapping step will map the keywords in the following manner:

- keyword *abstract* will be mapped on to the nodes/edges containing the term *abstract*



(a) *X-Media* (b) *Philip Camiano* (c) *Publication*

Figure 8. *Structural Component for Example 1*

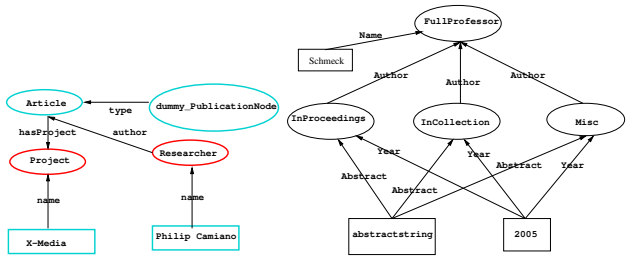
– keyword *Schmeck* will be mapped on to nodes/edges containing the term *Schmeck*
– keyword *2005* will be mapped on nodes/edges containing the term *2005*

The component sub-graph creation step will form the three components as shown in figures 10a to 10c.

In the pruning step, the node *ResearchGroup* with relationship *affiliation* associated with subgraph for *Schmeck* gets removed. The subgraphs for *abstract* and *2005* has homomorphic structures and hence their subgraphs are merged into one. Also the subgraphs has nodes *Person* and *FullProfessor*. Since they share a *subClassOf* relationship, the node *Person* is removed and *FullProfessor* is retained.

The hooking step starts with a node in a component structure having smallest cardinality. The node *FullProfessor* in the merged component structure is hooked with *FullProfessor* node in component structure of figure 10a. Since *Person* node has already been considered and no more component structures need to be considered we get the final answer graph as shown in figure 11b

[15] does not capture the *type/subClassOf* relationship as part of the RDF graph. As such, it cannot handle the keyword scenario where the keywords either refer to the super-class or a subclass. Even in the improved version [14], RDF semantics are not exploited. We exploit this knowledge to arrive at the correct answer graph. Our approach uses schema knowledge rather than relying purely on the graph paths alone. We also exploit homomorphic structures in our algorithm.



(a) *Example 1* (b) *Example 2*

Figure 11. Answer Graph for *Examples 1 and 2*

Example 3

For the following keyword query, the RDF data fragment as shown in figure 12 is used.

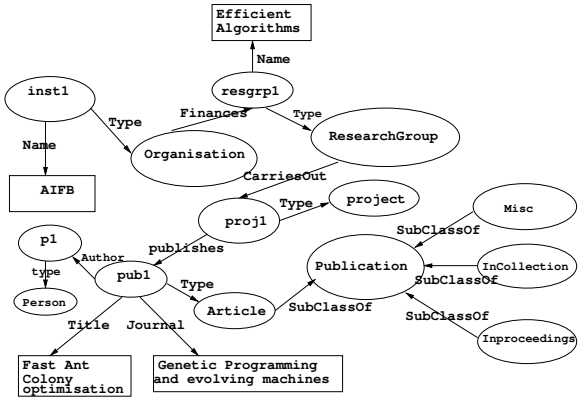
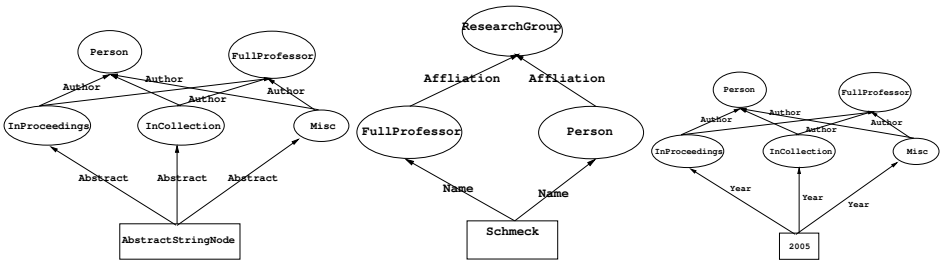


Figure 12. *RDF fragment for example 3*

Keyword Query: *AIFB journal titles*

– Keyword *AIFB* will be mapped on to the term *AIFB*
– Keyword *journal* will be mapped on to the term *journal*
– Keyword *titles* will be mapped on to the term *title*



(a) *abstract* (b) *Schemck* (c) *2005*

Figure 10. *Structural Component for Example 2*

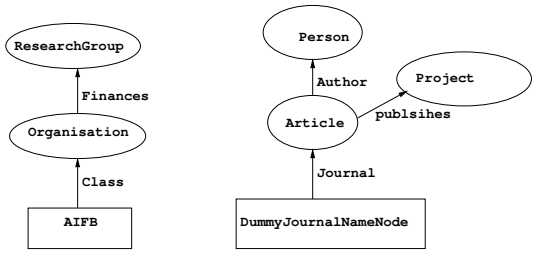
In the first step, the component subgraphs as shown in figures 13a, 13b and 14 are constructed. For *AIFB*, the *C-Node Organization* and *CR-node ResearchGroup* is added. The component subgraph as shown in figure 13a is obtained. For the term *journal* which is mapped to an entity-attribute edge, a node which represents a *dummy journal name along with C-Node Article and CR-Node Person* are added to form the component subgraph as shown in figure 13b. For the term *title* which is again mapped to an entity-attribute edge, a node which represents a dummy title, along with C-Nodes *Article, Misc, InProceedings, InCollection and CR-Nodes Person* are added to form a component subgraph as shown in figure 14.

In the pruning step, we identify homomorphic structures in the component sub-graphs. For the query, the component subgraph for *title* and *journal* provides the intersection as shown in figure 15a. The component subgraph of *journal* is a sub-graph of component subgraph of *title* except for the dummy node added. Both these sub-graphs are merged by pushing the extra dummy node to the smaller sub-graph. The intersection of node set of *title* with *AIFB* and *journal* with *AIFB* is empty. The algorithm attempts to find a relationship chain of *C-Nodes* connected by edge labels. For the query, the relationship chain as shown in **??** is obtained. The *class chain* $\{Project\}$ is added to sub-graph of *AIFB* and $\{Project, Article\}$ is added to sub-graph $\{journal, title\}$. We are left with two subgraphs corresponding to *AIFB and journal* as shown in figure 16.

In the hooking step, the *Project* node of *AIFB* component sub-graph can be hooked with the *Project* node of *Journal* sub-graph. We will have the final answer graph as shown in figure 17.

Compared to earlier approaches for graph querying and also the the approach adopted in [15] our approach has the following enhancements

- Edge mapping is allowed
- Exploits *type/subClassOf* relationship during the graph exploration phase
- Distance neighborhood is not fixed but managed during the pruning phase



(a) *AIFB* (b) *Journal*

Figure 13. Structural Component for *AIFB and Journal*

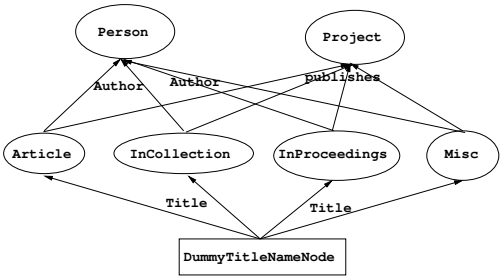
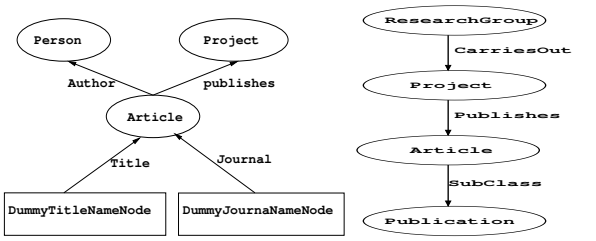
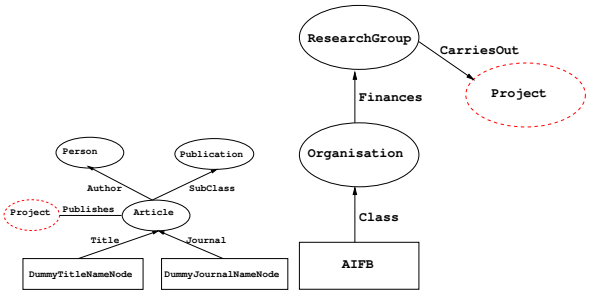


Figure 14. Structural Component for *term title*



(a) *Intersection of journal and title node* (b) *Relationship chain between Research Project and Publication*

Figure 15. *Component subgraph illustration during pruning phase*



(a) *Journal, Title* (b) *AIFB*

Figure 16. *Final Component subgraph for hooking step*

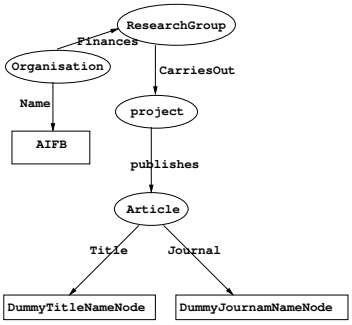


Figure 17. Final Answer Graph for the query

Example 4

figure 18 shows a fragment of RDF graph containing data taken from AIFB data set. The fragment models the information *related to association of Professors and Students with projects and events, Topics related to the project and publications related to the topics.*

Keyword Query: ***titles topic webservice student studer***

- Keyword *topic* will be mapped on to the term *topic*
- Keyword *webservice* will be mapped on to the term *webservice*
- Keyword *titles* will be mapped on to the term *title*
- Keyword *student* will be mapped on to the term *PhdStudent*
- Keyword *Studer* will be mapped on to the term *Rudi Studer*

In the Component_subgraph_Creation step, using the Type nodes and relationship nodes associated with the terms, the component sub-graphs as shown in figures 19 to 21 are constructed. For the term *Rudi Studer*, the *C-Node FullProfessor*, *CR-nodes PhDStudent ,Project and Event and edges Supervises ,WorksAt and ChairOf* are used to form the component subgraph. For the term *PhdStudent,* which refers to a *C-Node*, the *CR-Nodes FullProfessor,Project, InProceedings along with the respective edges are added to form its component subgraph.* For the term *title* which is again mapped to an entity-attribute edge, a dummy title node along with C-Node *InProceedings and CR-Nodes FullProfessor,PhdStudent and ResearchTopic are added to form its component subgraph as shown in figure 20*. For the term *webservice* which is mapped on to the *D-Node 'Semantic Web Service',* the *C-Node ResearchTopic along with CR-Node InProceedings are added to form the subgraph.* For the term *topic* the *C-Node topic and the CR-Node InProceedings is used to form its subgraph.*

In pruning step, we take the pairwise intersection of the nodes of the components to prune hanging nodes. For the sub-graphs constructed, the *Node Event along with the associated edge ChairOf* will be pruned. The component subgraph of *webservice* is identical to the component subgraph of *Topic* except for the additional *D-Node an*d the node *ResearchTopic*. But *ResearchTopic and Topic are similiar nodes(Type/SubClassOf relationship).* Both these sub-graphs are superimposed by pushing the *D-Node* to the smaller subgraph and retaining the node *ResearchTopic*.

In the hooking step, the *InProceedings* node of *webservice* subgraph is hooked with the *InProceedings* node of *Title* subgraph and merged. In the next hooking operation, the *Phdstudent node of previous merged subgraph is hooked with PhdStudent node of PhdStudent subgraph*. The two sub-graphs formed are shown in figure 22. Finally in the last hooking, *FullProfessor node of the previous merged subgraph will be hooked with the FullProfessor node of Rudi Studer subgraph.*

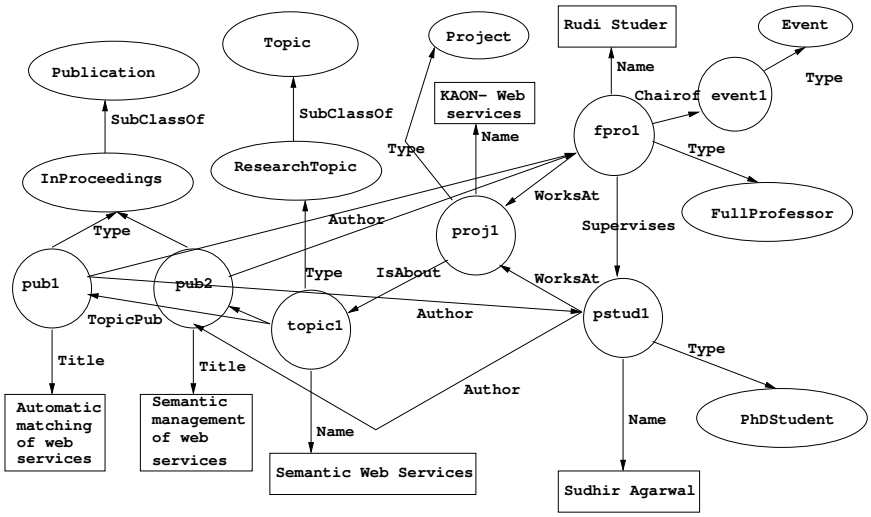
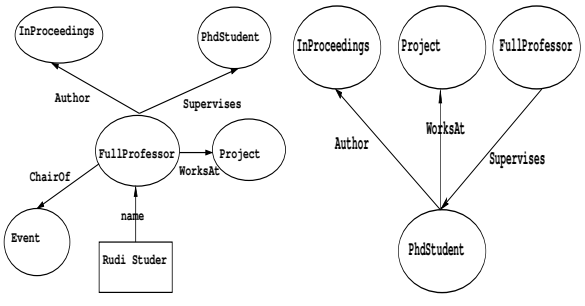


Figure 18. *RDF Graph fragment from AIFB Institute Data*

There is no more component to be considered and the final merged subgraph will be the answer graph as shown in figure 23.

Compared to earlier approaches adopted for keyword queries on graphs and also to the approach adopted in [15], our approach has the following enhancements

- Edge mapping is allowed
- Exploits graph semantics(*type/subClassOf relationship*) during the graph exploration phase
- The final answer graph can also have loops or cycles
- Uses graph similarity pattern for merging similiar graphs.



(a) *Rudi Studer* (b) *Phdstudent*

Figure 19. Component subgraph for *Rudi Studer,Phdstudent*

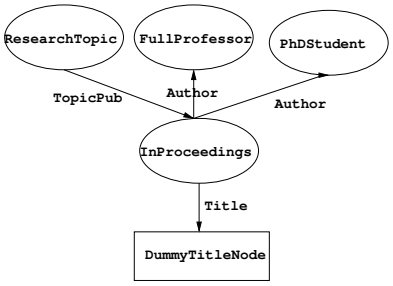


Figure 20. Component subgraph for *title*

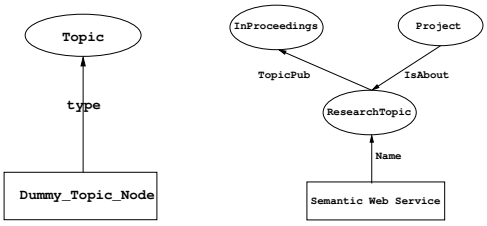
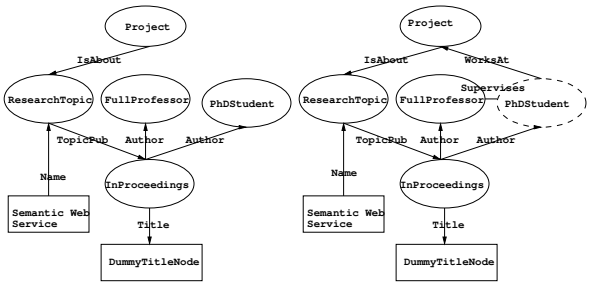


Figure 21. Component sub-graphs for *topic,webservice*



(a) *sub-graph* (b) *sub-graph*

Figure 22. *Component sub-graphs formed during hooking*

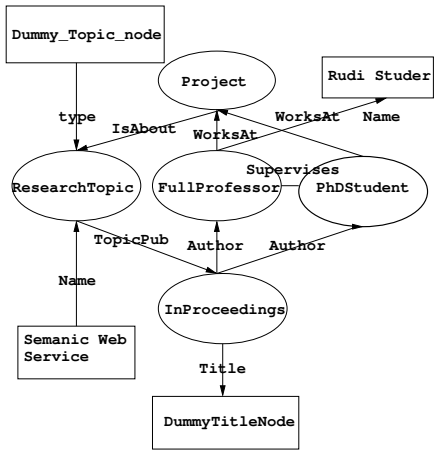


Figure 23. Final answer graph for the query

8. Design, Implementation and Analysis

8.1. Design and implementation

One of the approaches for implementing keyword search on RDF graphs is to represent it as a memory resident graph using JENA or SESAME to which graph traversal algorithms can be applied. For finding relationship chains, path algorithms can be used. The issues with this approach are if the data graph is large, the memory requirements will be high and exploring relationship chains also will be time consuming.

In our approach, we leverage on the existing RDF data store technology Allegro RDF Suite and develop a query processing layer which performs some of the computation and relegates specific portions of the computation to the data store layer. The first phase of the algorithm i.e *Component Sub-graph Creation* uses the data store layer. A query to the data store layer which exploits *rdf;type* statement is used to identify the *C-Nodes*. We use the *p*redicate and *o*bject indices of the Allegro Suite to identify associated triples in a faster time. the free-text indices are also used to identify the multiple triples which are matched. Based on the mapping of keyword to node or an edge and the type of the node or edge corresponding to the triple, component sub-graph is constructed. *(*C-Node), *CR-Nodes* and *(*SubClass) chains are identified from the data store using *(*type index), *relational index* and *(*SubClass index. The component sub-graphs relevant to the keywords are only maintained in memory. Much of the processing is done in the pruning and hooking phase where graph techniques and index structures as mentioned the section 5 are used. In the pruning stage, irrelevant nodes and edges are renewed from the component sub-graphs constructed. Homomorphic features of the sub-graphs are exploited to merge component sub-graphs. If there are no common nodes between the component sub-graphs, the relationship chains corresponding to the C-Nodes of the component sub-graphs are identified and the chain with the minimum length is used for linking the component sub-graphs.The hooking phase implements merger of component sub-graphs using *e*qui-spaced and *r*elationship-based expansion. The algorithmic framework for keyword search on RDF data is implemented using Allegro RDF Suite framework and Java.

8.2. Experiments

The approach presented has been implemented using Java. For storing of the RDF data AllegroGraph RDF data store was used. We used three data sets AIFB, DBLP and LUBM(50). The experiments was conducted on a Intel Xeon 2.6 Ghz machine under 64 bit environment with 6 GB memory. The number of triples loaded was varied from a few thousand triples in AIFB to around 5M triples from DBLP data set. The load time is shown in figure 24a.

During the index creation phase, the system creates type index, relationship index and class chains and also removes redundant type declarations. The index creation time for DBLP data set of 2M triples is shown in figure 24b.

The system has been tested on the following set of queries in the AIFB database:

1. *{smart-web, Cimiano, Publication}*
2. *{abstract, Staab, 2005}*

3. *{AIFB, journal, titles}*
4. *{titles, topic, web-service,student,studer}*
5. *{Publication,staab, 2013}*
6. *{title,scheduling, 2005, InProceedings}*
7. *{title,scheduling, 2005, Article}*

The set of queries illustrate different keyword mappings to different types of nodes and edges of the RDF graph.

The queries as shown in table 1 are used for keyword search on DBLP data set with 2M and 5M triples:

The number of keyword nodes mapped for 2M, 5M triples in DBLP data set is shown in table 2

Query Ref	No. of nodes(2M)	No. of nodes(5M)
Q1	(38634,5036)	(50223,8912)
Q2	(4,3016)	(8,5743)
Q3	(2,3370,38634)	(2,5743,50223)
Q4	(227,5210,1888)	(368,9671,3858)
Q5	(2,24,3150,302)	(6,28,3872,434)
Q6	(1,2,1932,3150)	(1,2,5860,3872)
Q7	(1,2)	(1,4)
Q8	(1,3794)	(1,7348)
Q9	(36,794)	(84,1686)
Q10	(6,320)	(18,446)
Q11	(4,4,3016)	(4,8,5743)

Table 2
Number of nodes mapped for DBLP Queries

The query time for running the queries Q1-Q10 is shown in figure 25.

8.3. Answer graph to Query conversion

The final answer graph is a graphical representation of a SPARQL query. For each concept node in the query subgraph or edge between the nodes, a triple pattern expression of SPARQL is generated. In the first case it specifies the node type, in the second case it specifies the relation between the nodes. Finally for each search term, a filter expression is added. For the query {X-Media, Philip, Publication} the SPARQL query is as follows:

*SELECT * where {?dummy-publicationNode rdf:type aifb:Article.*

?project rdf:type aifb:Project ?researcher rdf:type aifb:Researcher

?researcher aifb:author ?dummy-publicationNode. ?dummy-publicationNode aifb:hasProject ?project.

*?Researcher aifb:name ?term1 FILTER regex(?term1, ".*X-Media.*").*

*?project aifb:name ?term2 FILTER regex(?term2, .*X-Media.*").*

8.4. Evaluation and Analysis

The queries chosen for the AIFB data set illustrates a mixture of queries where the keywords are mapped to nodes and edges of different categories.

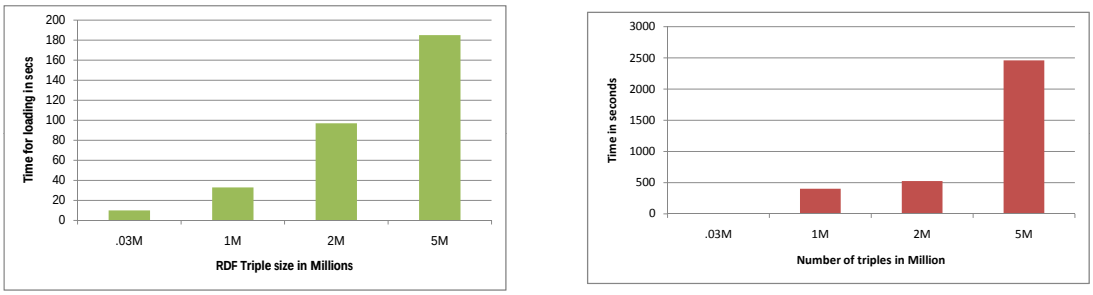
In our approach, keywords get mapped to different nodes and edges of the RDF graph. For example, if we consider Q3 in table 1, the keywords *(*adaptive, algorithm) gets mapped to D-Node representing *T*itle or *A*bstract. In addition, the mapping can be to multiple D-Nodes. The component sub-graphs with respect to the mapped D-Nodes will be grouped to form different clusters. The sub-graphs in the same cluster are identical. We bring one such component sub-graph element from each cluster into memory for processing. Depending on the associated EN-Node of the mapped D-Node and the relationship and sub-class chain, the component sub-graph has full set of Cr-Nodes associated or a partial set of CR-Nodes. The approach thus helps us to maintain minimum component sub-graphs in memory for processing during pruning and hooking stage.

In the query *{smart-web,Camiano,Publication}*, the semantic relationship *type/subClassOf* is exploited for finding the answer graph. Even though the triples are represented using keywords *{InProceedings, Misc, InCollection}* and not through keyword *P*ublication, the *s*ubClassOf property is exploited for answering the keyword query. For the query $\{abstract, Staab, 2005\}$, the term *abstract* is mapped on to an edge of an RDF graph. In the query $\{AIFB, journal, title\}$, component sub-graphs of $[AIFB, journal/title]$ do not have any *CR-Nodes* in common and hence sub-graph enlargement step is performed for answer graph construction. The query *{Publication,staab,2013}* is interesting. As per the data, there are no *publications* of *Staab* in year *2013*, and we do not return any answer graph. In [14] approach, answer graphs will still be constructed as only summary information is maintained. If we consider the query $\{title, scheduling, InProceedings, 2005\}$, the answer graph will directly represents the answer for *(*titles with keword scheduling in Publications of type InProceedings published in year 2005) whereas the query $\{title, scheduling, Article, 2005\}$, no answer graph wiil be returned as results combining such data combination does not exist in the RDF graph. Thus we are able to address different class of keyword queries.

Q1	*algorithm 1999*	*Q9*	*kim statistical*	*Q17*	*1999 243-290*
Q2	*wang database*	*Q10*	*liu carrier*	*Q18*	*multiple instance learning, 1999*
Q3	*arumugam adaptive algorithm*	*Q11*	*publication wang 1999*	*Q19*	*1998 IEEE*
Q4	*verma dynamic optimal*	*Q12*	*ai 1999*	*Q20*	*1999 acm*
Q5	*William zander performance improving*	*Q13*	*ir 1999*	*Q21*	*guassian weighted histogram IEEE*
Q6	*chuni yang software performance*	*Q14*	*machine learning, 1999*	*Q22*	*computer graphic springer*
Q7	*Lorenz cached*	*Q15*	*web services, 1999*	*Q23*	*guassian weighted histogram IEEE*
Q8	*lam optimization*	*Q16*	*1999 171-183*	*Q24*	*1999 springer*

Table 1

DBLP Queries



(a) *Load time* (b) *Index time*

Figure 24. *Timings for load, index operations*

To assess the scalability of our approach, we have run the queries from table 1. These queries are similiar to the queries used for evaluation in [6,14]. The environment used is also equivalent to what was used in those approaches. Strict performance comparison cannot be done. as [14] approach is based on summarization technique, [6] uses general data graphs and [15] maps keywords only to data nodes.[14] uses summarization approach and hence performance is better for queries on RDF graphs with regular structure as in DBLP. With respect to [6], based on the data provided in their paper, the nodes mapped to keywords for *query Q1*, in DBLP data set, the number of nodes mapped to keywords is $(1299, 941)$ whereas the number of nodes mapped in our case is $(50233, 8912)$ *with 5M triple data set*. The node mappings in our case are considerably large. Taking this into consideration, our performance for (2M,5M) triples is comparable with [6]$\left(2 \times 10^4, 1 \times 10^4\right)$. To evaluate the performance of queries that exploit *type/SubClassOf* relationship in larger data sets, extra triples as specified below were added to DBLP data set(2M,5M).

- *(Inproceedings subClassOf Publication)*
- (Misc subClassOf Publication)
- (InCollection subClassOf Publication)
- (Article subClassOf Publication)

Triples of the form $pub_i rdf\ typePublication$ were modified with *Publication* replaced randomly by the types: *{InProceedings, Misc,InCollection}*. The query *Q11* was run and the timings are *(3233, 1877)*. In order to validate our method further, experiments were conducted on the queries shown in from the LUBM(50)[5], a synthetic benchmark. The queries chosen for the experiments are shown in table 3. The total number of triples, load time and index time are shown in table 4. The timings of the four LUBM queries are shown in figure 26.

Though our emphasis in this paper is not ranking, we would just like to illustrate our ranking mechanism. For example, if we consider Q3 in table 1, the key-

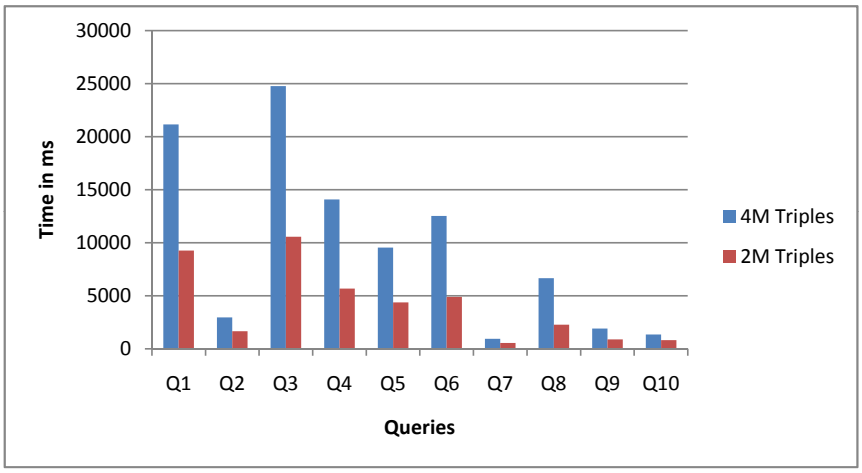


Figure 25. *Timings for query processing for 2M,5M triples*

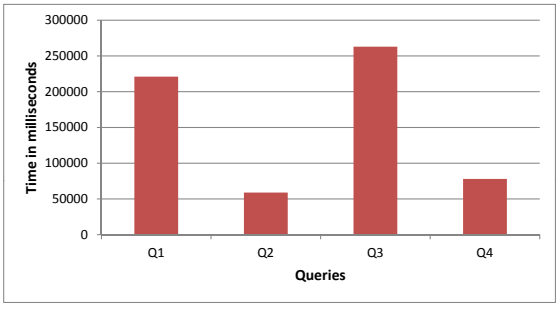
words (adaptive, algorithm) gets mapped to D-Node representing *T*itle or *A*bstract. In addition, the mapping can be to multiple D-Nodes as there will be several D-Nodes containing these keywords. The rank factor *Term Relevance* differentiates the affinity of different D-Node mappings and plays a key role for providing different rank values. Also there could be a mapping of a keyword to a D-Node or a C-Node. In Q1 of AIFB data set or Q11 in DBLP data set, *P*ublication gets mapped to a C-Node and may be to some D-Node. In this scenario, Node Relevance rank factor is high for C-Node mapping and hence ranked higher relative to a D-Node mapping. In case the component sub-graphs are enlarged for certain mappings, relationship relevance factor will be the key factor. Sub-graphs which have shorter relationship chain has a larger relationship relevance factor. Thus our hierarchical ranking scheme enables to do correct ranking of answer graphs.

Query Ref	Query Keywords
Q1	Publication2, lecturer6
Q2	Researcher5, FullProfessor9, Publication8
Q3	FullProfessor9, GraduateStudent0, Publication18, Lecturer6
Q4	Department0, GraduateStudent1, Publication8, AssociateProfessor0

Table 3
LUBM Queries

No. of tripless	Load time in secs	Index Time in secs
4646923	96	1830

Table 4
LUBM Load and Index timings



9. Related Work

In this technical article, we have addressed the issue of answer graph construction for keyword queries on RDF/RDFS data through a concrete graph exploration algorithm . We have tried to improve the graph exploration through an alternative approach by adopting pruning and hooking. as compared to [15, 12]. Our approach exploits RDF graph semantics like Type/SubClass relationship. We also extended our earlier approach [8] to remove the distance neighborhood restriction in our approach. We have also proposed a formal indexing scheme which exploits RDF semantics. In order to restrict index size to a comparable level we have proposed a two tier indexing scheme using graph partition techniques similiar to [6]

Keyword search on structured data has been extensively investigated in recent years under different contexts. Earlier approaches BANKS [4,6,16] addresses keyword search in the context of providing those facilities for databases. Exact matches between keywords and labels of data elements were done. Also substructures in the form of trees were constructed and the root element is assumed to be the answer. [4] uses backward search algorithm. In order to improve the search by limiting the nodes to be visited, [16] proposed bi-directional search algorithm where the exploration is through both backward and forward edge. The idea is to reach the root element faster through this approach. [6] also adopts distinct root semantics but improves the efficiency of the search using partitioning, balanced cost strategy and indexing to support forward jumps. These methods however do not exploit the schema knowledge for processing queries. The graphs are schema-less.

[15,12,9] presents different approaches for adapting keyword queries against different knowledge repositories represented in the form of graphs. [9] presents a system called SemSearch which follows a template based approach. In SemSearch, keywords are first interpreted as instances, concepts or properties. Even if two keywords are given as input nine templates are maintained and templates fix the structure. Also it is assumed that direct connections exists between entities denoted by keywords. For higher order keyword queries, heuristics are applied. In contrast to SemSearch, [15] presents a generic graph based approach to explore the connections between terms mapped to keywords of the query using knowledge available in ontology. A three step process consisting of term mapping, connection exploration and DL query construction is used. The exploration is restricted to connections where an instance is related to a concept by an *is-a* relation and two instances are related by object and data properties.The exploration builds a graph connecting a term element with all its neighbours within a specified range *d.* The process of exploration relies mainly on assertional knowledge resulting in a large number of paths that need to be processed. The graph does not model type/subClassOf of relationship. Ranking is based on path length. [12] also adopts three step process: term mapping, query graph construction and query ranking. Term mapping maps keywords into different terms using techniques like stemming, Edit-distance and sub-string matching. For each grouping of terms different query sets are constructed by enumerating all possible combinations from different sense of terms. From each query set a query graph is derived. A probabilistic ranking model is adopted for ranking the query graphs. In this system also the knowledge features and pruning mechanisms are not exploited during the exploration phase. In another interesting paper [11] which is also a motivation for this work, the authors have modeled unstructured, semi-structured and structured data as graphs and propose an efficient keyword search method EASE to adaptively process keyword queries over heterogeneous data. The data set is implemented as graphs and the search is modeled as *r-Radius Steiner Graph problem i.e to identify all all r-radius Steiner graphs which contain all the keywords.* In this paper also radius is fixed. The argument proposed is that graphs with larger diameter are not so meaningful and relevant. In[7] the authors present an approximation algorithm STAR for relationship queries. The problem is modeled as computing *k lowest-cost Steiner trees.* During the process taxonomic relationship *type or subclass* relationship is explored to construct first a tree and then improving the tree by scanning and pruning the neighborhood. In this paper, the graph is an entity relationship graph and only *type/subclass* relationship is exploited for answer tree construction. In [14], the authors improve upon their earlier work[15]. The keywords can also be mapped to edges. Complex data structures are introduced to keep track of the explored paths. To reduce the search space, a strategy for graph summarisation is employed. Exploration is restricted to a summary containing only the necessary elements.

In our approach ,we have adopted a different strategy for the exploration phase. We construct a hybrid graph from the original graph, which preserves as much connectivity information as possible. We create

fragments of closely related nodes and edges depending on the node or edge mapped to the keyword using type and relationship schema attributes and then prune unwanted nodes and edges. We also adopt a guided exploration strategy which exploits other knowledge characteristics(*type/subClassOf relationship*). Also we do not impose any distance metric. The distance factor is implicitly managed in the pruning phase depending on the closeness of the relationship nodes found out during the component subgraph creation phase. We have also presented a ranking scheme for the answer graphs in line with our algorithm.

10. Conclusions and Future Work

We have presented a concrete algorithm for answer graph construction given a set of keywords and a RDF/RDFS repository represented as a graph. We have illustrated the approach on a set of sample queries with different keyword mappings and demonstrated how semantic characteristics like *type/subClassOf* can be exploited. We have also presented the experimental results on benchmark sets AIFB,DBLP and,LUBM. We could demonstrate that correct answer graphs can be constructed for several queries with different characteristics and different mappings*(nodes, edges)*.

References

[1] Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist Modeling in RDF, RDFS and OWL*. Morgan Kaufmann Publishers, Reading, Massachusetts, 2008.

[2] B.Kimfield and Y.Sagir. Finding and approximating top-k answers in keyword proximity search. In *PODS 2006*, pages 173–182. ACM, 2006.

[3] Shady Elbassuoni and Roi Blanco. Keyword search over rdf graphs. In *CIKM*, pages 237–242. IEEE, 2011.

[4] G.Bhalotia, A.Hulgeri, C.Nakhe, S.Chakraborti, and S.Sudharshan. Keyword searching and browsing in database using banks. In *ICDE 2002*, pages 431–440. ACM, 2002.

[5] Yuanbo Guo, Zhengxiangpan, and Jeff Heflin. LUBM: A benchmark for owl knowledge base systemss.

[6] H.He, H.Wang, J.Yang, and P.S.Yu. Blinks: Ranked keyword searches on graphs. In *SIGMOD Conference 2007*, pages 305–316. ACM, 2007.

[7] Gjergji Kasneci, Maya Ramanath, Mauro Sozio, Fabian Suchanek, and Gerhard Weikum. Star: Steiner tree approximation in relationship graphs. In *25th IEEE International Conference on Data Engineering, ICDE 2009*, pages 868–879. IEEE, 2009.

[8] K.Parthasarathy, P.SreenivasaKumar, and Dominic Damien. Answer graph construction for keyword search on graph structured (rdf) data. In *Accepted in International Conference on Knowledge Discovery and Information Retrieval(KDIR) 2010*. INSTICC, Oct 2010.

[9] Lei.Y, Uren.V, and Molta.E. Semsearch: A search engine for the semantic web. In *15th International Conference on Knowledge Engineering and Knowledge Management(EKAW),(2006)*, pages 238–245, 2006.

[10] L.Guo, F.Shao, C.Botev, and J.Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD Conference 2003*, pages 16–27. ACM, 2003.

[11] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD 2008*, pages 1452–1455. ACM, 2008.

[12] Q.Zhou, C.Wang, M.Xiong, H.Wang, and Y.Yu. Spark: Adapting keyword query to semantic search. In *ISWC/ASWC,2007*, pages 694–707. SWSA, 2007.

[13] Sandhya Revuri, Sujatha Upadhyaya, and P.SreenivasaKumar. Using domain ontologies for efficient information retrieval. In *International Conference on Management of Data COMAD 2006*, pages 84–89. CSI, Dec 2006.

[14] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Camiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE,2009*, pages 405–416. IEEE, 2009.

[15] T.Tran, P.Camiano, S.Rudolph, and R.Studer. Ontology based interpretation of keywords for semantic search. In *ISWC/ASWC,2007*, pages 523–536. SWSA, 2007.

[16] V.Kacholia, S.Pandit, S.Chakraborti, S.Sudharshan, R.Desai, and H.Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB 2005*, pages 505–516. VLDB, 2005.

[17] Haofen Wang, Kang Zhang, Qiaoling Liu, Thanh Tran, and Yong Yu. Q2semantic: A lightweight keyword interface to semantic search. In *ESWC*, pages 584–598. IEEE, 2008.

[18] Y.Cai, X.Dong, A.Halevy, J.Liu, and J.Madhavan. Personal information management with semex. In *SIGMOD 2005*, pages 921–923. ACM, 2005.

[19] Y.Sure, S.Bloehdorn, P.Haase, J.Hartmann, and D.Oberle. The swrc ontology - semantic web for research communities. In *In Proceedings of the 12th Portuguese Conference on AI(EPIA 2005)*, pages 218–231. ECCAI, 2005.