

Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

<http://www.elsevier.com/copyright>



Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Practical and effective IR-style keyword search over semantic web

Xiaomin Ning^{a,b}, Hai Jin^{b,*}, Weijia Jia^c, Pingpeng Yuan^b^a China Ship Development and Design Center, Wuhan 430064, China^b Services Computing Technology and System Laboratory, Cluster and Grid Computing Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China^c Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong

ARTICLE INFO

Article history:

Received 23 April 2008

Received in revised form 8 October 2008

Accepted 29 December 2008

Available online 10 February 2009

Keywords:

Group Steiner tree

Keyword search

RDF graph

Top-K

ABSTRACT

This paper presents a novel IR-style keyword search model for semantic web data retrieval, distinguished from current retrieval methods. In this model, an answer to a keyword query is a connected subgraph that contains all the query keywords. In addition, the answer is minimal because any proper subgraph can not be an answer to the query. We provide an approximation algorithm to retrieve these answers efficiently. A special ranking strategy is also proposed so that answers can be appropriately ordered. The experimental results over real datasets show that our model outperforms existing possible solutions with respect to effectiveness and efficiency.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Motivation

With the increasing amount of ontologies encoded in RDF/S or OWL languages available in semantic web, extensive efforts have been made towards effective retrieving facilities for the data. In this area, existing research work can be classified into two major categories: the structured query approach and the semantic search. The former is standardized by W3C which envisions that, users should be able to issue structured SQL-like expressive queries such as RQL, RDQL, or SPARQL (Prudhommeaux & Seaborne, 2008) and receive a set of triples as answers. This approach is effective, if users have a detailed knowledge of underlying schemas and ontology languages. Nevertheless, the assumed scenario is not always true in practice, since these languages are complicated even for developers of semantic web applications. An alternative approach is the so-called semantic search which usually adopts more user-friendly unstructured (Rocha, Schwabe, & Aragao, 2004) or semi-structured (Anyanwu, Maduko, & Sheth, 2005, 2003) query strategies. However, these reported approaches are still much less attractive than keyword search, which is the most effective and successful paradigm for modern information retrieval.

This paper focuses on the problem of supporting effective keyword search over semantic web data. The semantic web data model (RDF/S or OWL) conforms to a node-labeled and edge-labeled directed graph (Hayes, 2004), where nodes represent resources or literals and edges represent properties carrying heterogeneous semantics. In our proposed model, an answer to a keyword querying is a connected “minimal” subgraph, which contains all the query keywords. In a real-world semantic web application, there may exist a large number of literal nodes matching individual keywords and hence many

* Corresponding author. Tel.: +86 27 8754 3529; fax: +86 27 8755 7354.

E-mail addresses: ningxm@hust.edu.cn (X. Ning), hjin@hust.edu.cn (H. Jin), itjia@cityu.edu.hk (W. Jia), ppyuan@hust.edu.cn (P. Yuan).

answers may satisfy the query. It is a non-trivial problem how to efficiently find these answers (e.g., consider a 5-keyword query over an RDF graph with thousands of nodes). In addition, these answers should be reasonably ranked in decreasing relevance. Otherwise, users will be discouraged to apply keyword search.

To illustrate our model, suppose a simplified scientific literature knowledge base shown in Fig. 1. We assume the user issues a keyword query $Q = \{John, K6\}$ to find some useful information among the two keywords. Fig. 2 shows four possible answers (i.e., R_1 , R_2 , R_3 and C_1), which contain both keywords and convey particular semantics as well. For example, answer R_1 corresponds to paper p_4 written by author John with term K6 in its full-text. However, there exists a major difference between those answers shown in Fig. 2a and the one in Fig. 2b. The candidate answer C_1 has three literal nodes containing the same keyword K6 simultaneously. Actually, answers R_1 , R_2 and R_3 are all proper subset of C_1 . Hence, we shall remove it from the final answer results because it has redundant nodes/edges. In addition, the answers in Fig. 2a are not equally useful to users. For example, R_1 might be more desirable than R_2 and R_3 , since it represents stronger relationship between keywords John and K6.

1.2. Contributions

The key contributions of this paper are the following:

- We propose a novel IR-style keyword searching model for semantic web data retrieval. Our search scheme supports a free-from keyword querying interface, which is especially desirable as well as practical for ordinary semantic web users. This model can generate answers with explicit semantics (connected subgraphs) and return ranked top- k results effectively.
- We present an efficient searching algorithm for finding answers. The key problem is that there may be many potential answers matching a keyword querying over a large data graph. Efficiently locating these answers is a non-trivial challenge.
- We conduct experiments over real datasets. Experimental results show that our model is feasible and delivers high-quality search results.

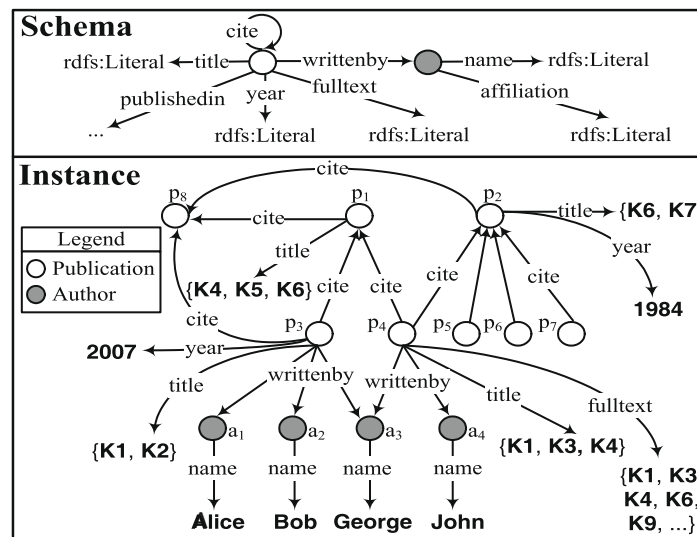


Fig. 1. An example RDF graph fragment.

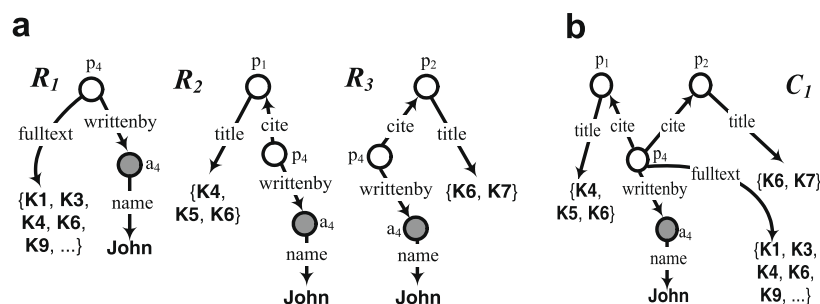


Fig. 2. (a) Answers R_1 , R_2 , R_3 to keyword query $Q = \{John, K6\}$ and (b) a candidate answer with redundant nodes.

The rest of this paper is organized as follows. Section 2 reviews related work. We define the data model, and then describe the formal query and answer semantics in Section 3. We detail our solution for keyword search in Section 4. Section 5 reports the experimental results. Finally, we conclude the paper and point out future research directions in Section 6.

2. Related work

We briefly discuss the semantic search approach. The idea of semantic search was firstly introduced by Guha, McCool, and Miller (2003). Later, Rocha et al. (2004) presented a purely entity-centric semantic search approach. It seeks to find important related entities to a given set of keywords using a spreading activation algorithm (Cohen & Kjeldsen, 1987; Crestani, 1997). The limitation is that, “an object by itself is intensely uninteresting”. To address this issue, Anyanwu et al. (2005) presented a relationship search model for querying relationships between entities. It supports reachability queries, such as “Does a semantic relationship (or a path) exist between a given entity X and a given entity Y?”. However, it imposes user-interface challenges on users, since they must preknow and then exactly specify the entities X and Y. Moreover, they do not address the problem of more complex semantic relationships (e.g., connected graphs other than individual paths) among multiple (e.g., more than 3) entities/keywords.

To the best of our knowledge, only two papers Tran, Cimiano, Rudolph, and Studer, 2007; Zhou, Wang, Xiong, Wang, and Yu, 2007, have ever tried to support free-form keyword search in semantic web. They both apply the similar method, i.e. translating keyword queries into formal logic queries. For example, Tran et al. (2007) translate keyword queries to $\mathcal{SHOIN}(\mathbf{D})$ conjunctive ones using background knowledge available in ontologies. But the effectiveness of their methods greatly depend on the precision of keyword interpretation (Tran et al., 2007) or term mapping (Zhou et al., 2007), which is still an unresolved yet difficult problem in information extraction or natural language understanding/processing (NLP).

Keyword search has also attracted a great deal of interest in the structured relational database (RDBMS) community recently. In those systems, such as BANKS (Bhalotia, Hulgeri, Nakhe, Chakrabarti, & Sudarshan, 2002), ObjectRank (Balmin, Hristidis, & Papakonstantinou, 2004), and SPARK (Luo, Lin, Wang, & Zhou, 2007), a database is viewed as a graph with tuples as nodes and foreign key references among tuples across table relations as edges. Trees connecting nodes that contain the query keywords are returned as answers to a keyword query. This model based on database graphs is similar to ours in some degree. However, RDF graphs can carry much more complicated semantics than database graphs. Therefore, keyword search poses greater challenges in semantic web than in RDBMS.

3. Problem definition

3.1. Graph data model

Assume there is an infinite set U (URI references), an infinite set $B = \{N_j : j \in \mathbb{N}\}$ (blank nodes), and an infinite set L (RDF literals). An 3-tuple (*subject*, *property*, *object*) $\in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple, which asserts that a resource, the *subject*, has a *property* whose value is the *object*. An RDF graph is a set of RDF triples (Hayes, 2004).

We extend RDF semantics and model the data as a weighted directed graph $G = (V, E, w)$, where $V = \{v_i : i \in \mathbb{N}\}$ is a finite set of nodes denoting subjects or objects, $E \subseteq V \times V$ is a set of edges denoting properties, and $w : E \rightarrow \mathbb{R}^+$ is an edge weight function denoting the semantic relationship strengths of properties. Smaller weights correspond to stronger relationships between connected entities. The cost of the graph G is defined as the sum of the weights of all edges, that is,

$$\text{cost}(G) = \sum_{e \in E} w(e) \quad (1)$$

A uniform edge weight function w (as in the web) might be insufficient since each edge (*property*) conveys different semantic. Edge weights can be assigned using methods in ObjectRank (Balmin et al., 2004). For our model, we first assign edge weights on the schema level (i.e., RDFS), which was also used in our previous work Ning, Jin, and Wu, 2008. Each type of relationship T is assigned a weight $w(T)$. We note that it is not the focus of this paper how to appropriately assign edge weights on the schema level. Some works Burges et al., 2005, Diligenti, Gori, and Maggini, 2005 have tried to address the problem. On the instance level, if given the source node v_i and the edge $e : v_i \rightarrow v_j$ with the edge type T , the edge weight $w(e)$ is computed as follows:

$$w(e) = w(T) \cdot \log_2(1 + \text{Degree}(v_i, T)) \quad (2)$$

where $w(T)$ is the weight of the edge typed T , and $\text{Degree}(v_i, T)$ is the number of links typed T from v_i . The equation indicates that, if a node has more incident nodes for a specific edge type, it represents a weaker relationship between them.

3.2. Query and answer semantics

Consider a query with l distinct keywords: $Q = \{k_1, \dots, k_l\}$ over an RDF data graph $G = (V, E, w)$. It tries to find possible semantic relationships connecting the l keywords. There are two possible semantics for interpreting keyword queries, i.e., *conjunctive* with “AND” semantics and *disjunctive* with “OR” semantics. This paper focuses on *conjunctive* semantics. Formally, an answer R to the query Q must be a connected subgraph of G , which additionally satisfies the following conditions:

- Every keyword is contained in at least one literal node in R .
- Answer R is minimal, that is, no proper subgraph of R can still be an answer to Q . If we remove any literal node of R , the corresponding keyword matching is lost. Additionally, if any non-literal node is removed, R becomes disconnected.

To simplify the representation, we introduce a keyword-to-node mapping function $\pi : K \rightarrow 2^V$. The mapping function gives the set of literal nodes that contain the keyword $\kappa \in K$, where K is the set of keywords that can be used for querying the data. For each keyword $k_i \in \{k_1, \dots, k_l\}$, there is a set of literal nodes denoted with V_i that contain it, namely, $\pi(k_i) = V_i \subseteq V$. As a result, there are totally l groups, i.e., V_1, \dots, V_l . So an answer is a connected subgraph which contains at least one node from each group. The subgraph must be a tree with literal nodes as leaves. Otherwise, it will have redundant nodes/edges.

As mentioned earlier, there may exist many possible answers satisfying the above requirements for a large data graph. These answers may or may not be relevant to the given query. It is observed that users usually examine only a few answer results (e.g. top-20). To avoid retrieving complicated but meaningless answers, our search model aims to find top- k minimum cost answers/subgraphs to a keyword query. The answer with a smaller cost is ranked higher. When the number of required results $k = 1$ (i.e. top-1), this problem can be modelled as the well-known finding minimum cost group Steiner tree Reich and Widmayer, 1989, a generalization of the classical NP-hard Steiner tree problem (Kou, Markowsky, & Berman, 1981). Some approximation algorithms have been proposed for the group Steiner tree problem, such as minimum spanning tree heuristic (Reich & Widmayer, 1989) and randomized algorithm (Garg, Konjevod, & Ravi, 1998). Nevertheless, our model generates the top- k answers, distinguished from the previous algorithms which are only to find one optimal/approximate solution. This is because keyword search is inherently fuzzy or imprecise. It is inappropriate or even impossible to just find one “best” answer in a large data graph.

4. Our solution for keyword search

4.1. An approximate algorithm

The basic idea of our proposed algorithm is as follows. Initially, an answer tree starts with a single node in any group. Then we construct the answer tree through repeatedly adding shortest paths to nodes which groups are not covered by the current tree. After $l - 1$ iterations, the answer tree is inserted into an increasing priority queue. Later, we dequeue the tree and replace some nodes which have increasing costs. The newly obtained tree is also inserted into the queue. We first give the definition of *Shortest Path*.

Definition 1. Shortest path The shortest path between a graph $(V', E', w) \subset (V, E, w)$ and a node $v \in V \setminus V'$ is defined as $\min\{cost(u, \dots, v) | u \in V', (u, \dots, v) \text{ is a path between } u \text{ and } v\}$.

Fig. 3 details major parts of the algorithm. Inputs are an RDF graph $G = (V, E, w)$, a keyword query $Q = \{k_1, \dots, k_l\}$, and an integer parameter k which indicates that the user is only interested in top- k ranked results. We first identify l groups for individual keywords of the query $Q = \{k_1, \dots, k_l\}$ in lines 1–3. Then, we pick any group (without loss of generality, here we assume V_1) and repeatedly construct an answer tree from each node in V_1 . The initial answer tree is only a single node in V_1 (line 7). Shortest paths to nodes which groups are not covered yet, are added to the tree. The obtained tree is cleaned up by removing redundant edges and leaves, and then is inserted into the increasing priority queue *Results* (line 12). After the execution of line 13, the *Results* will have $|V_1|$ answer trees satisfying the query, where $|V_1|$ is the total number of nodes in V_1 .

For efficiency, we dequeue *Results* as the first answer result and replace it with another node in the same group (line 20). Then, we remove the original node from the tree and clean it up. Lines 23–25 are employed to prevent generating answer trees which may have been produced previously. We choose the newly produced tree with minimum cost and insert it into *Results* (line 27). The algorithm terminates if k results are obtained. Otherwise, it continues to dequeue *Results* and repeat the above processing. From Fig. 3, we can see that the answer tree cost computation is incremental.

Let us give an example shown in Figs. 4 and 5 to illustrate the execution of the algorithm. For simplicity, we assume that all edges have equal weights (i.e. 1). Fig. 4a gives an undirected graph fragment of a knowledge base. Text values are contained in literal nodes which are denoted as triangles. We ignore types of different instances (e.g. Author, Publication) which are all indicated by circles. Suppose a query $Q = \{K1, K2, K3\}$. The algorithm first identifies three groups (i.e. V_1, V_2 and V_3) for the keywords, respectively. Fig. 4b–d presents the execution process of generating one approximate result for the query. As shown in Fig. 4b, the tree is initialized to one node in group V_1 . Then the algorithm finds a shortest path to a nearest node in group V_2 . Next, it continues to find another shortest path from the tree shown in Fig. 4c to a node in V_3 which has not been covered. So one answer tree, with the total cost 7, is produced and inserted into *Results*, as shown in Fig. 4d. After generating the first result, we continue to initialize a tree to another node in V_1 and repeat the above processing. At the end of line 13, two answer trees will be put into the queue. For simplicity, the generation of the second answer tree is omitted in the figures.

Now we can dequeue *Results* and produce other results (line 14–28). The algorithm finds a shortest path to another node in V_2 , as shown in Fig. 5a. The tree becomes a candidate answer to query Q . However, we notice that it has two literal nodes containing the same keyword $K2$ and so is not “minimal”. We remove the original node, and then clean up the candidate tree

Input: RDF graph $G = (V, E, w)$, keyword query $Q = \{k_1, \dots, k_l\}$, an integer k
Output: the first k approximate results

```

1 foreach Keyword  $k_i$  do
2 | Identify group  $V_i \subseteq V$ , a set of literal nodes containing  $k_i$ 
3 end
4  $Number\_Of\_Results = 0$ ;
5  $Results = \emptyset$ ; /* An increasing priority queue */
6 foreach Node  $v_i \in V_1$  do
7 | Initialize the answer tree  $T_i^1$  to  $v_i$ ;
8 | foreach  $j \in \{2, \dots, l\}$  do
9 | | Find a shortest path  $P_i^{j-1}$  from tree  $T_i^{j-1}$  to a nearest node which group is not covered by  $T_i^{j-1}$ ;
10 | |  $T_i^j = T_i^{j-1} \cup P_i^{j-1}$ ;
11 | end
12 | Clean up tree  $T_i^l$  and insert it into priority queue  $Results$  according to  $cost(T_i^l)$ ;
13 end
14 while  $Number\_Of\_Results < k$  do
15 | Answer  $R = DeQueue(Results)$ ;
16 | Print  $R$  as one approximate result;
17 |  $Number\_Of\_Results++$ ;
18 | foreach  $j \in \{2, \dots, l\}$  do
19 | | /* Assume node  $v \in V_j$  for  $R$  */
20 | | Find a shortest path  $P_j$  from  $R$  to a nearest node  $v_j$ , where  $v_j \in V_j$  and  $v_j \neq v$ ;
21 | |  $R_j = R \cup P_j$ ;
22 | | Remove  $v$  from  $R_j$  and clean up  $R_j$ ;
23 | | if  $cost(R_j) < cost(R)$  then
24 | | | Delete  $R_j$ ;
25 | | end
26 | end
27 | Choose the  $R_j$  with the minimum  $cost(R_j)$  and insert it into  $Results$ 
28 end

```

Fig. 3. An approximation algorithm for keyword search.

by truncating redundant nodes and corresponding edges. Since V_3 has only one node, the newly produced tree with the total cost 8 is then inserted into $Results$.

4.2. Complexity analysis

The execution time of the proposed algorithm is dominated by the time needed for the shortest path computation. We implement the shortest path problem using Fibonacci Heaps Fredman and Tarjan, 1987 in time $O(|V|\log|V| + |E|)$ for the worst case. For an l -keyword query, there is $l - 1$ iterations (lines 8–11 and lines 18–26) and so the execution time of iterations is $O(l(|V|\log|V| + |E|))$. The total execution time is $O(kl(|V|\log|V| + |E|))$ in the worst case. Hence, the algorithm is efficient in the sense that the worst case time is polynomial. Usually, the average number of keywords l is small and users are

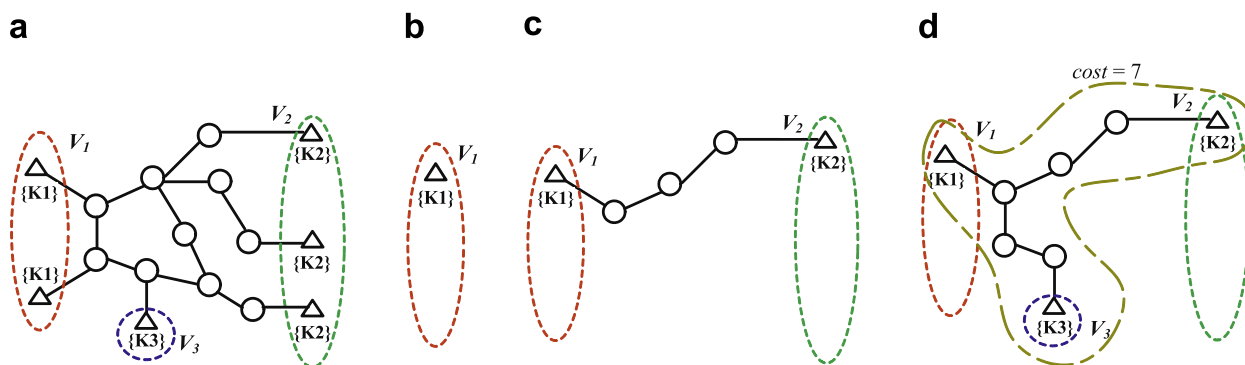


Fig. 4. The generation of one approximate result to query $Q = \{K1, K2, K3\}$.

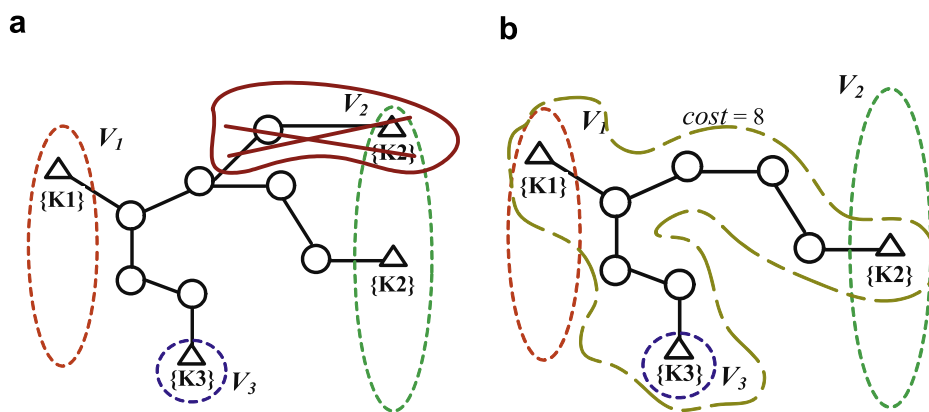


Fig. 5. The generation of another answer tree by replacing some original node with cleaning up operation.

interested in a few answers (e.g., top-10 or 20). However, if the length of the query is long (e.g., $l > 6$) and the user wants to get many more results (k is very large), this execution time may still not be acceptable. Several methods can be used to optimize the algorithm. For example, we can select the smallest group instead of any group (i.e., V_1 in the algorithm of Fig. 3), or restrict the maximum searching depths to prevent generating large yet little interesting answer trees. In addition, if the query has only two keywords, the problem can be reduced to finding Dijkstra's single-source shortest paths in time $O(k(|V|\log|V| + |E|))$.

5. Experiments

Section 5.1 describes the data set, evaluation metrics and setups for comparison. Section 5.2 reports experimental results.

5.1. Experimental setup

5.1.1. Data set

We use real datasets from CiteSeer (<http://www.citeseer.ist.psu.edu/oai.html>), including CiteSeer BibTex records and metadata archive oai_citeseer.tar.gz. In the experiment, we are interested in highly connected RDF graphs. Hence, we mainly choose those data in the database domain. We use keywords “data”, “database”, “SIGMOD”, “VLDB”, “ICDE”, “TKDE”, “TODS” to match the title or the booktitle or the journal fields of BibTex entries. Most publications within the database area are filtered. Next, we combine the filtered BibTex entries with the metadata archive and generate detailed information on publications. The information is then encoded into a huge RDF graph according to the schema described in Fig. 1 using the Sesame storage server. The graph contains 27 K nodes, 199 K edges, and 18 K distinct keywords. Here we do not use stemming words or stop words. In addition, we build an inverted index in which each entry is a keyword with a list of literal node IDs to quickly identify groups for keywords.

We manually chose 40 queries for evaluation. These queries include a wide variety of keywords. The maximum number of keywords is 6, and the minimum is 2. The average number of keywords is 3.7. Table 1 lists some example queries. Usually, an answer result is not simply relevant or irrelevant to a query. So we manually judged results and gave an integer relevance label ranging from 0 to 5 (i.e. 0 = “irrelevant”, 1 = “slightly relevant”, 5 = “perfectly relevant”, etc.) to each result. The human based evaluation is simple and subjective, however, it still can indicate the effect of our approach compared to other methods in a certain degree.

Table 1
Example queries.

ID	Queries
Q ₁	Index database
Q ₂	ACID transaction
Q ₃	vldb 1999
Q ₄	Jim Gray transaction
Q ₅	Query optimization performance
Q ₆	SQL programming language
Q ₇	Distributed data mining algorithm
Q ₈	Image retrieval object database
Q ₉	R-Tree indexing algorithm spatial
Q ₁₀	SIGMOD keyword search ranking database

5.1.2. Evaluation metrics and comparison setup

As far as we know, there are no published works compared to ours. Although two works Tran et al., 2007, 2007 have ever tried to support free-form keyword search in semantic web, no evaluation results or further details are given in their works. However, we still try to demonstrate the effect of our approach through comparing with two existing possible solutions: (1) *Random model*: for an l -keyword query, this method randomly picks one node from each group (hence totally l nodes), connects these nodes with a minimum tree, repeatedly obtains such k trees (if existing) and then sorts them based on their costs and (2) *Optimum model*: this method first exhaustively finds all answer trees with their optimum costs (thus an NP-hard problem) and then selects top- k minimum cost trees.

The major goal of the experiments is to show the search performance and effectiveness of our model. Search performance gives the average time cost to find top- k answers. To measure effectiveness, we adopt two popular IR metrics: (a) Precision at k that gives the fraction of results in the top- k results which are relevant. This metric measures user satisfaction with the top- k results on average; and (b) MAP (Mean Average Precision) that is the mean of average precisions of all test queries. We separately send each keyword query (totally 40 ones) to the three models, and hence there are three different result lists returned for each query.

5.2. Results

5.2.1. Search performance

Firstly, we vary the number of results k . The average time costs for three models are shown in Fig. 6. We observe that, the cost of *Optimum* is much higher (more than 10 s) than the other two. In addition, the cost of *Optimum* almost keeps unchanged (very smoothly increasing). The reason is that it always exhaustively finds all answers and then simply outputs top- k results. Thus, the search performance of *Optimum* is unacceptable even for a medium sized semantic data graph.

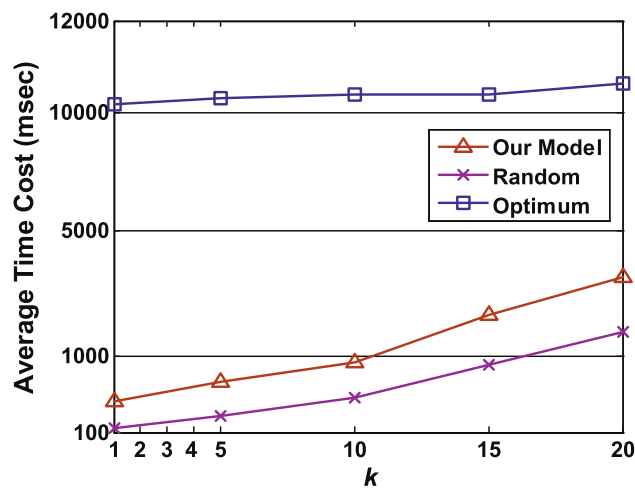


Fig. 6. Effect of the number of results, k .

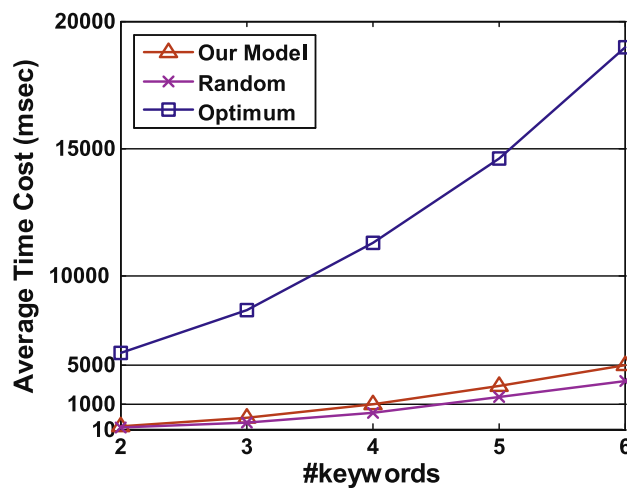


Fig. 7. Effect of the number of keywords, l .

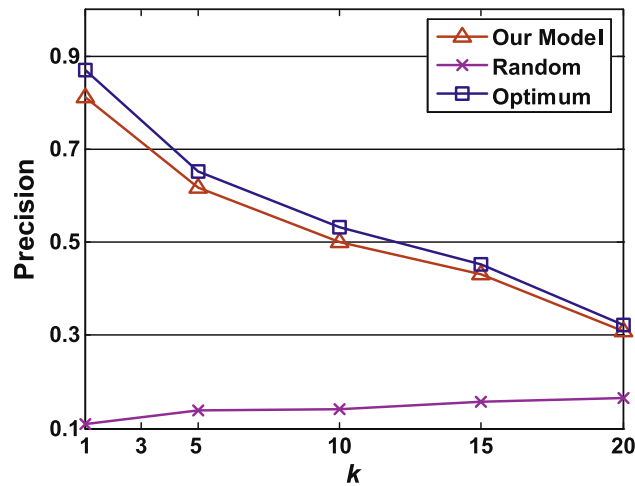


Fig. 8. Evaluation of precision at k .

Table 2

Mean average precision (MAP).

	Random	Optimum	Our model
MAP	0.139	0.552	0.519

The search cost of our model is slightly higher than that of *Random*. When $k = 10$, the average time costs of our model and *Random* are 0.93 and 0.61 s, respectively. This is because that *Random* generates results randomly but depresses the quality of results significantly, as shown later.

Next, we fix $k = 10$ and vary the number of keywords l from 2 to 6. The average time costs are presented in Fig. 7. The costs increase with more keywords, although all models are more responsive for less keywords. We also find that the cost of *Optimum* is highest among all three models. When $l = 6$, the average time cost of our model reaches 5 s. However, if we notice that l is usually small (2.35 on the average in web Oyama, 2002), the search performance of our model is still acceptable in real-world applications.

5.2.2. Effectiveness

Fig. 8 reports averaged Precision@ k for all models. For a given query, Precision@ k gives the fraction of results in the top- k results that are relevant to the query. The relevance value ranges from 0 to 5. We averaged relevance values for 40 test queries. Fig. 8 shows that our model can satisfy user search intension with the top- k results on average, almost as well as *Optimum*. In particular, we notice that our model has averaged Precision@1 of 81%. This is very useful when searching over large scale semantic web data. In contrast, the quality of *Random* is quite low (only about 10%) compared to the other two models.

Table 2 reports the results in terms of MAP. From the table, we find that our model as well *Optimum* performs significantly better than *Random*, similar to the results in Fig. 8. The MAP of *Random* is just 0.139.

From all the above the experimental results, we observe that our proposed model is effective in keyword searching. In addition, its search performance is much better than *Optimum*. Our model gives a good balance between the effectiveness and efficiency. Thus, we can draw the conclusion that our model achieves better overall performance compared to *Random* and *Optimum*.

6. Conclusion and future work

Existing semantic web data retrieval methods can be classified into two major categories: the structured query approach and the semantic search. However, it is still very desirable to support flexible keyword search over semantic web, since ordinary users usually do not understand the underlying data structure. Moreover, they have been accustomed to traditional keyword search for years. This paper presents a novel IR-style keyword search model for semantic web data retrieval. In our model, an answer to a query is defined as a minimal connected subgraph that contains all the query keywords. We provide an approximation algorithm to efficiently retrieve these answers. A special ranking mechanism is also developed. The experimental results show that our model performs almost as well as *Optimum*, and significantly exceeds *Random* in terms of Precision@ k and MAP. The search performance of our model is slightly lower than that of *Random*, but outperforms *Optimum* by about an order of magnitude (0.93 vs. 10 s). Hence, our model has better overall performance, compared to *Random* and *Optimum*.

Future research includes several directions. First, we are trying to incorporate node weights into our proposed model. Another promising direction is to further optimize the approximation algorithm, such as graph connectivity information pre-computing and indexing.

Acknowledgements

This work is supported by the National 973 Key Basic Research Program under Grant No. 2003CB317003, and the CityU Strategic Research Grants 7002102 and 7002214.

References

- Anyanwu, K., Maduko, A., & Sheth, A. (2005). SemRank: Ranking complex semantic relationship search results on the semantic web. In *Proceedings of the 14th international conference on world wide web (WWW), Chiba, Japan* (pp. 117–127).
- Balmin, A., Hristidis, V., & Papakonstantinou, Y. (2004). ObjectRank: Authority-based keyword search in databases. In *Proceedings of the 30th international conference on very large data bases (VLDB), Toronto, Canada* (pp. 564–575).
- Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., & Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *Proceedings of the 18th international conference on data engineering (ICDE), San Jose, USA* (pp. 431–440).
- Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., et al. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on machine learning (ICML), Bonn, Germany* (pp. 89–96).
- Cohen, P., & Kjeldsen, R. (1987). Information retrieval by constrained spreading activation on semantic networks. *Information Processing and Management*, 23(4), 255–268.
- Crestani, F. (1997). Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6), 453–482.
- Diligenti, M., Gori, M., & Maggini, M. (2005). Learning web page scores by error back-propagation. In *Proceedings of the 9th international joint conference on artificial intelligence (IJCAI), Edinburgh, Scotland* (pp. 684–689).
- Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3), 596–615.
- Garg, N., Konjevod, G., & Ravi, R. (1998). A polylogarithmic approximation algorithm for the group Steiner tree problem. In *Proceedings of the 9th annual ACM-SIAM symposium on discrete algorithms (SODA), San Francisco, California* (pp. 253–259).
- Guha, R., McCool, R., & Miller, E. (2003). Semantic search. In *Proceedings of the 12th international conference on world wide web (WWW), Budapest, Hungary* (pp. 700–709).
- Hayes, P. (2004). RDF semantics. <<http://www.w3.org/TR/rdf-mt/>>.
- Kou, L., Markowsky, G., & Berman, L. (1981). A fast algorithm for Steiner trees. *Acta Informatica*, 15, 141–145.
- Luo, Y., Lin, X., Wang, W., & Zhou, X. (2007). SPARK: Top-k keyword query in relational databases. In *Proceedings of the 26th ACM SIGMOD international conference on management of data (SIGMOD), Beijing, China* (pp. 115–126).
- Ning, X., Jin, H., & Wu, H. (2008). RSS: A framework enabling ranked search on the semantic web. *Information Processing and Management*, 44(2), 893–909.
- Oyama, S. (2002). *Query refinement for domain-specific web search*. Ph.D. Thesis, Kyoto University.
- Prudhommeaux, E., & Seaborne, A. (2008). SPARQL, W3C recommendation. <<http://www.w3.org/TR/rdf-sparql-query/>>.
- Reich, G., & Widmayer, P. (1989). Beyond Steiners problem: A VLSI oriented generalization. In *Proceedings of the 15th international workshop on graph-theoretic concepts in computer science*.
- Rocha, C., Schwabe, D., & Aragao, M. (2004). A hybrid approach for searching in the semantic web. In *Proceedings of the 13th international conference on world wide web (WWW), New York, USA* (pp. 374–383).
- Tran, T., Cimiano, P., Rudolph, S., & Studer, R. (2007). Ontology-based interpretation of keywords for semantic search. In *Proceedings of the 6th international semantic web conference (ISWC), Busan, Korea* (pp. 523–536).
- Zhou, Q., Wang, C., Xiong, M., Wang, H., & Yu, Y. (2007). SPARK: Adapting keyword query to semantic search. In *Proceedings of the sixth international semantic web conference (ISWC), Busan, Korea* (pp. 694–707).