J Intell Inf Syst (2015) 44:381–395
DOI 10.1007/s10844-014-0336-1

Interactive search over Web scale RDF data using predicates as constraints

Mingyan Teng · Guangtian Zhu

Received: 1 June 2013 / Revised: 27 July 2014 / Accepted: 25 September 2014 /
Published online: 17 October 2014

Abstract RDF (Resource Description Framework) data are more and more prevalent in the applications of semantic web and web data publication. The search over Web scale RDF data is essential for users to retrieve desired information from the huge RDF datasets, which typically applied as knowledge bases supporting many advanced information seeking tasks. In this paper, we propose some techniques that allow users to interactively search over the Web scale RDF data by using keywords as well as their predicates as additional constraints. We observe that the straightforward way of keyword search over the Web scale RDF data often generates a huge number of matching sub-structures (i.e., graphs containing the query keywords) due to the ambiguity of query intention generated from a small number of query keywords, although most of them are false interpretations of the query intention. To effectively interpret the semantics of queries, we define a novel keyword query called structure-aware keyword query that utilizes the predicates of RDF triples to assist users in clarifying their query intention. The challenge of such queries is to effectively and efficiently find a proper set of predicate-keyword pairs for query interpretation, to reduce the manual cost of user feedbacks as much as possible. To verify the novel query mechanism, we implement a system, and test it over the DBPedia 3.7 dataset. Results show that, for most queries, users can often efficiently get desired results by providing a small number of simple feedbacks on the constraints of predicates automatically generated.

Keywords Interactive search · Keyword search · RDF data · Predicate constraints · Web scale · Structure-aware

M. Teng (✉)
Beijing Institute of Information and Control, Beijing, 100037, China
e-mail: tengmingyan@gmail.com

M. Teng
School of Mathematics and Physics, Bohai University, Jinzhou, 121000, China

G. Zhu
Academy of Mathematics and Systems Science, The Chinese Academy of Sciences,
Beijing, 100080, China
e-mail: zhugt@amss.ac.cn

1 Introduction

Knowledge bases extracted from the Web are becoming increasingly prevalent in our daily applications because of the advance of knowledge harvesting technologies (Weikum and Theobald 2010). Examples are Freebase (Bollacker et al. 2008), DBpedia (Auer et al. 2007), and Linked Open Data (LOD (Heath and Bizer 2011)). Many of these knowledge bases apply RDF (Resource Description Framework) as a basic data model to record the huge amount of information units. An RDF triple, typically presented as $< subject, predicate, object >$, represents a statement (*object*) of one aspect (*predicate*) of a *subject*. Because of its simplicity, the RDF model is widely used for recording the knowledge extracted from the Web or open data published from many sources. The RDF knowledge bases are able to support important applications such as semantic search (Ferré and Hermann 2011) because the high-quality information and knowledge they contain.

Recently, intensive studies have been conducted on efficient query processing over Web-scale RDF datasets. One stream of these studies is focused on SPARQL query (Huang et al. 2011; Neumann and Weikum 2010), which is a W3C standard for searching RDF data structurely. However, SPARQL query requires users to have a deep knowledge of the underlying schema (predicates and prefixes) of the desired RDF triples. It is therefore not user-friendly because general users are often ignorant to the schema of RDF datasets.

In RDF datasets, the object of an RDF triple can also be a subject of another triple. Hence, an RDF dataset can be modeled as labeled graphs, in which the nodes consist of subjects and objects, and edges are predicates (serve as edge labels). Keyword search over labeled graphs has attracted much attention because it is an easy portal for users to retrieve the desired information from labeled graphs. As introduced in the work (Bhalotia et al. 2002), given a directed graph G in which each node is associated with a plain text label, keyword search on graphs concerns with querying G by a set of keywords. The desired answer is a ranked list of subgraphs which contain all the query keywords in their node labels. Compared to the SPARQL, keyword search is more user-friendly. It is therefore widely used for many information seeking tasks on the RDF knowledge bases.

However, keyword search may not be able to capture a user's query intention accurately. The ignorance of predicates in a keyword query will generate more ambiguity for keyword search. For example, if a user presents a query "apple, snow" and hopes to find a recipe for apple snow treat, keyword search engine may incorrectly return a MacBook computer produced by Apple with Snow Leopard OS. The ambiguity of a keyword query is more prominent when it is conducted over Web-scale RDF datasets, because many RDF labels are long texts that may contain dozens of words (in contrast to the short labels used in traditional labeled graphs). As a result, it is quite common that there are a large number of sub-structures of the data graphs that contain the query keywords. However, most of them may not be the sub-structures desired by the users. This brings challenge to effectively rank sub-structures because users can only be patient to a small number of top results in web search applications.

The characteristics of RDF data model allow predicates to help to reduce the ambiguity of simple keyword queries (Pound et al. 2010). However, like SPARQL queries, predicates of RDF data are not user-friendly, and therefore they are hard to be directly applied in keyword search. In this paper, we present a structure-aware keyword search solution which allows users to leverage the discriminative functionality of predicates, through an interactive process of keyword search. Our study on this solution is focused on how to efficiently and effectively suggest predicate-keyword pairs to users so that irrelevant candidate

sub-structures can be effectively removed during a quick and easy-to-use feedback process. We propose an algorithm to efficiently propose a proper set of predicate-keyword pairs for users' feedbacks.

By using the proposed techniques, users are able to accurately and effectively search desired information from Web-scale RDF knowledge bases. The proposed solution leverages the easy-to-use keyword search approach by introducing the constraints of predicates when seeking information from RDF knowledge bases. Users are able to dynamically adjust their queries through interactively participating the feedbacks on the predicate-keyword pairs. The proposed search technique can be easily applied to applications such as semantic search and question answering (Unger et al. 2012).

The main contribution of the paper includes:

- We propose an interactive search framework that helps users to clarify their search intention through the feedbacks on the automatically generated predicate-keyword pairs. This is a large difference from the traditional SPARQL query and keyword search that have been widely used for many information seeking tasks on Web-scale RDF knowledge bases.
- To guarantee an interactive speed of efficient generation of predicate-keyword pairs for feedbacks, we propose a sub-optimal algorithm to effectively and efficiently suggest predicate-keyword pairs according to the search results. This makes the solution scalable for Web-scale RDF knowledge bases.
- We implement a prototype of the search system. Extensive experiments over real datasets demonstrate the proposed interactive search solution can be effectively and efficiently conducted over Web-scale RDF data.

The paper is organized as follows. Section 2 gives the related work study. Section 3 introduces the structure-aware keyword search solution. The strategies of interactive search by predicate-keyword pairs are given in Section 4. Experimental studies are given in Section 5, followed by the conclusions given in Section 6.

2 Related work

There are several types of accessing methods for large RDF datasets. One is a W3C standard query language called SPARQL (Pérez et al. 2006), which is a declarative language like SQL in traditional relational database systems. Studies on efficient SPARQL query processing algorithms and systems have been widely proposed (Neumann and Weikum 2010; Wilkinson et al. 2003). The major problem of SPARQL is that it is not user-friendly, given the complex query syntax and the complicated predicates and prefixes used in the query. As for this, keyword search over RDF graphs has been recently studied (Elbassuoni and Blanco 2011; He et al. 2007; Tran et al. 2009).

The goal of keyword search over RDF graphs is to find succinct local structures that contain all the query keywords. They typically ignore the predicates that serve for representing the relationships between nodes. Some of the keyword search solutions (Tran et al. 2009) try to transform a keyword query into a number of structural queries so that a SPARQL-like query can be finally formalized according to the statistical information derived from the RDF graphs. Some recent studies (Unger et al. 2012; Yahya et al. 2012) also consider to transform a natural language question to a SPARQL query, to support queries like natural language questions.

A more relevant work (Pound et al. 2010) considers to affiliate structural constraints over the query keywords. The structural constraints specify the predicates of desired results when matching local structures in RDF graphs. However, the structural constraints have to be explicitly proposed by users. To our best knowledge, techniques on how to automatically assign structural constraints over RDF data has never be studied.

3 Structure-aware keyword search

An RDF statement is a triple consisting of a subject, a predicate, and an object. It is represented as $< s, p, o >$, which corresponds to an edge from a subject s to an object o with the edge type p as a predicate. An RDF knowledge base can be abstracted as a set of graphs G. Vertices in G are partitioned into two disjoint sets: literal nodes and entity nodes (typically represented as URIs). A literal node is a node of pure textual information. It therefore can only be an object. An entity node is a node with a URI. It can be both subjects of some triples and objects of some other triples. In contrast, a literal node must be attached to one entity node. We denote the function $p(v)$ as the predicate of a literal node v. For the example shown in Fig. 1, $Paper1$ is an entity node (note that the URIs is not shown out in the figure). The node "*semantic data management*" is a literal node. Note that in our study, keywords can only appear in literal nodes because we treat the URIs of entity nodes as identifications of entities.

Given a keyword k, a literal node v is called a *spot* of the keyword k if k appears in the literal text of v. For example, in Fig. 1, nodes v_4 and v_5 are spots of the keyword "semantic". The goal of keyword search over RDF graphs is to find some local structures in the graphs such that each local structure contains spots for each keyword, and the spots are connected with each other in the local structure.

The following two intuitions motivate the definition of the structure-aware keyword search query.

Intuition 1 *Generally, users who provide the query keywords know the exact semantics of these keywords. However, they do not know how the semantics is interpreted in the huge RDF graphs because they often do not know the structure of RDF datasets in details. Simply providing more keywords for clarifying the query intention may cause many false negatives of matching local structures in the RDF graphs.*

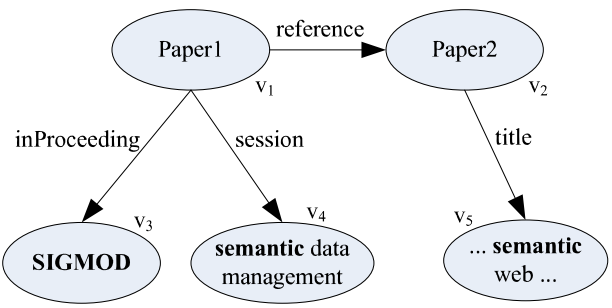

Fig. 1 Examples of RDF graph and matching trees

Intuition 2 *The desired results are local structures with a very small radius in terms of graph distance. In other words, all the query keywords must appear in literal nodes close to each other in the RDF graphs. A small radius guarantees the strong semantic correlation among the spots of all the query keywords.*

Many approaches (Bhalotia et al. 2002; He et al. 2007; Kacholia et al. 2005) of keyword search over graphs use tree-shaped local structures (in particular, Steiner trees (Bhalotia et al. 2002)) where the keyword nodes correspond to leaves, to represent the query answers. Although the edges of RDF graphs are directed, the direction (which is not specified in keyword queries) can be ignored when identifying local structures containing the spots of the query keywords. In this paper, we apply the tree structure to interpret the query results of keyword search.

Definition 1 (**Matching tree**) Given a keyword query $q = \{k_1, \cdots, k_n\}$ consisting of n keywords, a radius threshold d_{max}, if we can find an entity node v_T as a root, and a spot v_i for each query keyword k_i, such that the graph distance (the number of hops) between v_T and v_i is no more than d_{max}, then we say that the tree T rooted as v_T, with leaf nodes only consisting of v_i (note that some spots may be identical for different keywords), is a matching tree of the query q.

According to the above definition, a matching tree of a query is a tree of depth no more than d_{max}, describing a local structure. It must contain spots for each keyword. Figure 1 shows examples of matching trees for a keyword query "SIGMOD semantic". If $d_{max} = 1$, there is only one matching tree rooted at the node v_1 (with nodes v_3 and v_4 as spots of the corresponding query keywords). If $d_{max} = 2$, there will be 2 matching trees because the node v_2 can be another root node with nodes v_3 and v_5 as spots.

As stated, because of the ambiguity of the simple keyword query, a keyword query may potentially have a large number of matching trees in the huge RDF graphs, especially when only a small number of common keywords are included in the query. Our solution in this problem is to apply predicates for constraining the semantic interpretation of keywords. This leads to the concept of structure-aware spots.

Definition 2 (**Structure-aware spot**) Given a predicate set P and a keyword k, they form a pair denoted as $P : k$. A literal node v is called a structure-aware spot of the pair $P : k$ if k appears in the literal text of v, and $p(v) \notin P$.

To be a structure-aware spot of a pair $P : k$, a literal node v must first be a spot of k. Additionally, the predicate ($p(v)$) of the literal node v must not present in the predicate set P associated with the keyword k. For example, if $\{session\} : semantic$ is specified in the example of Fig. 1, node v_5 will be the only structure-aware spot for the keyword "semantic". The predicate set P is used for specifying the false interpretation of spots for the query keyword k. Note that when P is an empty set, any spot of the keyword k will be also a structure-aware spot of $P : k$.

The reason of using the predicate set P as false interpretations, instead of true ones, for defining the structure-aware spots is in two folds: 1) users are often not able to know the full set of true interpretations of the predicates. The usage of a subset of true interpretations will cause false negatives of matching trees. 2) most local structures of false interpretations come from the false interpretations of popular predicates. A small number of negative popular predicates usually help to reduce the number of matching trees significantly.

With structure-aware spots, the matching tree given in Definition 1 need to be modified as a tree connecting structure-aware spots (instead of simple spots) of all the query keywords.

Definition 3 (**Structure-aware matching tree**) A structure-aware keyword query q is specified as $q = \{P_1 : k_1, \cdots, P_n : k_n\}$, where k_i is a keyword and P_i is the set of predicates associated with k_i. Given a radius threshold d_{max}, if we can find an entity node v_T as a root, and a structure-aware spot v_i for each query pair $P_i : k_i$, such that the distance between v_T and v_i is no more than d_{max}, then we say that the tree T rooted as v_T, with leaf nodes only consisting of v_i, is a structure-aware matching tree of the query q.

The goal of structure-aware keyword search is to find all the structure-aware matching trees. In structure-aware keyword search, the matching trees can be purified by removing the false association of predicates and keywords, obtained through users' feedbacks. By pruning the false interpretation of matching trees using structure-aware spots, the number of matching trees to a given a keyword query can be significantly reduced. This is because many false interpretations of spots come from the association of keywords with a small number of popular predicates falsely interpreted.

To retrieve a list of ranked structure-aware matching trees, the relevance score of a matching tree can be evaluated by using some heuristics proposed in keyword search over graphs (He et al. 2007; Li et al. 2008). The problem of structure-aware keyword search can be simply reduced to the problem of keyword search over RDF graphs with a filtering process for the structure-aware spots as an additional operation.

4 Interactive predicate filtering

The structure-aware keyword search is a variant of keyword search over graphs. Techniques on efficient keyword search (using inverted indexes) over graphs and effective ranking models can also be applied to the problem of structure-aware keyword search. They are not the focus of the paper. In this paper, we focus on how to efficiently and effectively generate the structure-aware keyword queries through the feedbacks of users. An interactive process is designed to allow users for clarifying their query intention by providing a small number of predicate-keyword pairs to compose a structure-aware keyword query.

The interactive process must be as simple as possible to avoid too much manual effort. Moreover, the interactive process must be conducted at an interactive speed. This provides challenges for finding the optimal set of predicate-keyword pairs, which need to be efficiently detected from the search results, especially when the search results include too many matching trees. We call the interactive process as the process of interactive predicate filtering, which can be formally defined as follows:

Definition 4 (**Interactive predicate filtering**) Given a query $q_0 = \{P_1 : k_1, \cdots, P_n : k_n\}$, to find c predicate-keyword pairs forming a new query $q' = \{P'_1 : k_1, \cdots, P'_n : k_n\}$ satisfying $\sum_i |P'_i| = c$, such that the merge of q_0 and q', denoted as $q_1 = q_0 + q' = \{P_1 \bigcup P'_1 : k_1, \cdots, P_n \bigcup P'_n : k_n\}$ has the minimal number of structure-aware matching trees.

According to above definition, the merge of a query component $P_i : k_i$ with a predicate-keyword pair $p : k_i$ will generate an updated query component $P_i \bigcup \{p\} : k_i$, which happens when a predicate-keyword pair $p : k_i$ is picked through the user's feedbacks. When a pure

keyword query (of n terms) is first generated by a user, $P_i = \emptyset$, for $i = 1, \ldots, n$. The more the predicate-keyword pairs for a keyword k_i selected by the user during the feedback process, the more predicates in the set P_i, and therefore the less ambiguity the spots of keyword k_i have. After selecting a small number of frequent predicate-keyword pairs of false semantic interpretation of spots, the number of structure-aware matching trees (search results) will be significantly reduced. Through several rounds of feedbacks, the query intention of users will be more and more clear.

With the process of interactive predicate filtering, the framework of structure-aware keyword search over RDF graphs can be illustrated in Fig. 2. The initial list of matching trees of a keyword query can be found from inverted indexes of the RDF graphs using techniques such as (Li et al. 2008). As intermediate results, the matching trees will be partially ordered (depending on the number of matching trees found), with some top results shown out to the user. In the meanwhile, the component of the predicate-keyword pair suggestion automatically finds c predicate-keyword pairs for users' feedbacks. After taking the feedbacks from users, the structure-aware keyword query is updated. The component of interactive predicate filtering then removes those false interpretations of matching trees derived from the user's feedbacks. After this, a new round of feedbacks will be triggered. This loop continues until the user does not provide feedbacks any more, which usually happens when he finds ideal interpretations of the query from a limited number of structure-aware matching trees.

Through experiments over large scale real life datasets, we observe that for many non structure-aware queries, the number of distinct matching trees can be as large as millions to billions when $d_{max} \geq 2$. It is prohibitive to evaluate the relevance scores of so many matching trees one by one. The process of interactive predicate filtering is necessary to prune most of those matching trees before ranking them, because most of them are false interpretations of the query. Fortunately, we find that although the number of matching trees is often very huge, the number of distinct roots for those trees is however typically not too large (typically tens to thousands). This is because in many cases, for each root node, we can find a large number of spots (whose distance to the root is no more than d_{max}) of each query keyword. The combination of these spots generates a huge number of distinct matching trees. Another observation is that, although there are often a large number of spots in the matching trees for each query keyword, the predicates for the spots of many non

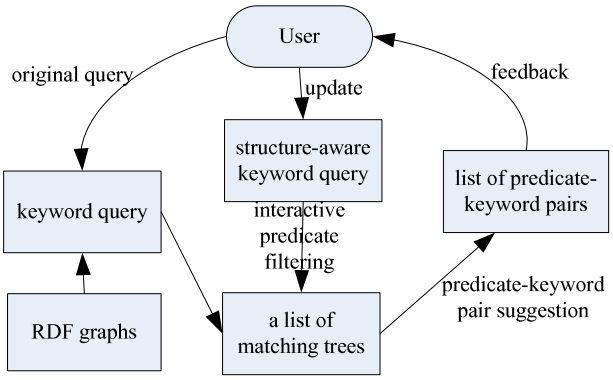

Fig. 2 The framework of structure-aware keyword search. Note that, the original query is issued by the user only by once

popular query terms, however, are often limited to a small number. This allows us to be able to efficiently find an effective set of predicate-keyword pairs for predicate filtering.

Given a query $q = \{P_1 : k_1, \cdots, P_n : k_n\}$, the number of structure-aware matching trees that can be pruned (also denoted as the pruning power) by the query, denoted as $C(q)$, can be computed without enumerating all the matching trees using the following formula:

$$C(q) = \sum_r \left(\prod_i C_r(k_i) - \prod_i C_r(P_i : k_i) \right) \tag{1}$$

where r is a distinct root of for a number of structure-aware matching trees with the same root r, $C_r(k_i)$ is the number of spots of k_i rooted at r, and $C_r(P_i : k_i)$ is the number of structure-aware spots of k_i rooted at r, restricted by predicates in P_i. The component $\prod_i C_r(k_i)$ is the number of all structure-aware matching trees rooted at r, which come from the combinations of spots for each keyword.

Lemma 1 *For two structure-aware keyword queries q_1 and q_2, we have $C(q_1 + q_2) \leq C(q_1) + C(q_2)$.*

Proof Let $q_1 = \{P_{1,1} : k_1, \cdots, P_{n,1} : k_n\}$, $q_2 = \{P_{1,2} : k_1, \cdots, P_{n,2} : k_n\}$. To prove Lemma 1, we need to prove that $\prod C_r(k_i) - \prod C_r(P_{i,1} \bigcup P_{i,2} : k_i) \leq (\prod C_r(k_i) - \prod C_r(P_{i,1} : k_i)) + (\prod C_r(k_i) - \prod C_r(P_{i,2} : k_i))$, i.e., to prove $\prod C_r(k_i) + \prod C_r(P_{i,1} \bigcup P_{i,2} : k_i) \geq \prod C_r(P_{i,1} : k_i) + \prod C_r(P_{i,2} : k_i)$ for any given r. Let $a_i = C_r(k_i)$, $b_i = C_r(P_{i,1} \bigcup P_{i,2} : k_i)$, $c_i = C_r(P_{i,1} : k_i)$, and $d_i = C_r(P_{i,2} : k_i)$. According to the definition of $C_r()$, we have $a_i + b_i = c_i + d_i$, and $0 \leq b_i \leq c_i, d_i \leq a_i$. Let $m_i = (a_i + b_i)/2 = (c_i + d_i)/2$, $x_i = (a_i - b_i)/2$, $y_i = |c_i - d_i|/2$. Obviously, $x_i \geq y_i$. Then, $\prod a_i + \prod b_i = \prod(m_i + x_i) + \prod(m_i - x_i)$, and $\prod c_i + \prod d_i = \prod(m_i + y_i) + \prod(m_i - y_i)$. When conducting polynomial expansion for $\prod(m_i + x_i) + \prod(m_i - x_i)$ and $\prod(m_i + y_i) + \prod(m_i - y_i)$ respectively, it can be easily found that $\prod(m_i + x_i) + \prod(m_i - x_i) \geq \prod(m_i + y_i) + \prod(m_i - y_i)$. Therefore, we have $\prod a_i + \prod b_i \geq \prod c_i + \prod d_i$, and Lemma 1 can then be proved. □

Let F be the set of all predicate-keyword pairs detected from the existing list of structure-aware matching trees. According to the Definition 4, given a query q_0, the goal of interactive predicate filtering is to find c pairs (forming a new query q') from F such that they $(q_0 + q')$ can maximize the number of structure-aware matching trees that can be pruned, if these c pairs are all taken by the user. The straightforward way of finding the optimal c predicate-keyword pairs requires to generate all possible c-combinations from the set F, and compute the pruning power of each combination to find the optimal one. Therefore, there will be $\binom{|F|}{c}$ times of computing the pruning powers. However, computing the pruning power based on the (1) is an expensive process. The total cost of interactive predicate filtering will be extremely expensive when $|F|$ is large (which happens in many cases), which may lead to the predicate filtering cannot be interacitvely processed. When an interactive speed of predicate filtering cannot be guaranteed (for example, by an estimator according to $|F|$), we will apply the following strategy which leads to a sub-optimal set of predicate-keyword pairs for pruning the structure-aware matching trees.

As stated in Algorithm 1, the queries formed from each predicate-keyword pair are first computed and ranked as queries in L in a descending order of their pruning power (lines 3–7). Instead of finding the optimal combination of c predicate-keyword pairs straightforwardly, we apply a greedy strategy which finds the sub-optimal combinations of c'

predicate-keyword pairs incrementally, with c' increased from 2 to c (lines 13–37). For each c', instead of get a ranking list of queries from all combinations of c' predicate-keyword pairs, we only find the top-m (in term of pruning powers) queries stored in H_2. This allows us to apply an upper bounding technique to prune most of combinations that satisfy $C(L[i]) + C(q) \leq H_2.min$ (line 21 to line 25). The top-m list of c' is computed incrementally, i.e., it expands the top-m list of $c' - 1$ by one predicate-keyword pair heuristically (lines 17–30). Compared to the optimal solution where all combinations of $c' - 1$ predicate-keyword pairs are need to be expanded, such an approximation (by only expanding top-m combinations in the previous step) often allows us to get a sub-optimal solution very efficiently. According to Algorithm 1, there will be $O(cm^2)$ times of computing the pruning powers (note that an early abandoning strategy is applied), much

Algorithm 1: A ranking algorithm for predicate-keyword pairs

Input: q_0, the current query
Input: R, the set of distinct roots of matching trees
Output: a combination of c predicate-keyword pairs to form q' in the Definition 4

```
1  compute F from R;
2  let L be an list of queries;
3  foreach i = 1 to |F| do
4  |   suppose F[i] = p : k_j be a predicate-keyword pair, let q = {k_1,...,{p} : k_j,...,k_n};
5  |   insert < q, C(q) > into L;
6  end
7  Reorder queries in L by their pruning powers in an descending order;
8  let H_1 = ∅ be a heap recording top-m queries in terms of their pruning powers;
9  foreach i = 1 to m do
10 |   //to initialize H_1;
11 |   push < L[i], C(L[i]) > into H_1;
12 end
13 foreach c' = 2 to c do
14 |   let H_2 = ∅ also be a heap recording top-m queries in terms of their pruning powers;
15 |   while H_1 ≠ ∅ do
16 |   |   pop the maximum out of H_1 as q;
17 |   |   foreach i = 1 to |F| do
18 |   |   |   if L[i] ⊈ q then
19 |   |   |   |   //L[i] ⊈ q if and only if L[i] + q ≠ q;
20 |   |   |   |   let q_1 = L[i] + q;
21 |   |   |   |   if C(L[i]) + C(q) ≤ H_2.min then
22 |   |   |   |   |   //H_2.min: the minimal pruning power of elements in H_2;
23 |   |   |   |   |   break;
24 |   |   |   |   |   //early abandoning, because C(q_1) ≤ C(L[i]) + C(q) according to Lemma 1
25 |   |   |   |   end
26 |   |   |   |   else if C(q_1) > H_2.min then
27 |   |   |   |   |   push < q_1, C(q_1) > into H_2;
28 |   |   |   |   end
29 |   |   |   end
30 |   |   end
31 |   end
32 |   while H_2 ≠ ∅ do
33 |   |   pop the minimum out of H_2 as q;
34 |   |   push < q, C(q) > into H_1;
35 |   |   //copy H_2 to H_1
36 |   end
37 end
38 pop the maximum out of H_1 as q_1;
39 return c predicate-keyword pairs (to form q') such that q_1 = q_0 + q';
```

more efficient than the solution to find the optimal combination whose complexity is $\binom{|F|}{c}$, because $m \ll |F|$. Although the returned set of predicate-keyword pairs may not be optimal, as will be shown in the experimental results, it is often able to find a set of predicate-keyword pairs very effectively.

5 Experimental results

To test the performance (both efficiency and effectiveness) of the proposed algorithm, we implement an interface that is shown in Fig. 3. We allow users to submit any keyword query to the system. The top results of the structure-aware matching trees will be shown out to the user. When the number of structure-aware matching trees is very large, to guarantee the search performance, only part of them will be evaluated for computing the relevance scores of the matching trees. For each query, a number of predicate-keyword pairs will be presented to the users for feedbacks. Those pairs are selected by default. However, users can deselect some of them which interpret the semantics of keywords correctly. Once the user submits the picked predicate-keyword pairs for feedbacks, the results of the structure-aware matching trees will be updated. The number of matching trees will be shown out so that the users can experience the pruning power of the predicate filtering process. In addition to the updated results, a new set of predicate-keyword pairs will be generated for the next round of feedbacks from the user.

Experiments are run on a server with a 4-core processor of 2.40 GHz, 16GB memory, 2TB disk. The DBPedia 3.7 dataset[1] (contains 3.64 million entities) is applied as the KB in our experiments. We encode the dataset using a word dictionary. We apply the query set of INEX 2009 entity track[2] that contains 55 query topics. Stop words are removed from the original topics.

To test the effectiveness of the proposed structure-aware keyword search solution, we compare the proposed solution (labeled as *filter* in Fig. 4) with two existing solutions of keyword search over RDF graphs (labeled as *keyword* for a pure keyword search solution over RDF data (Tran et al. 2009) and *NLQ* for a nature language question solution (Yahya et al. 2012) respectively in Fig. 4). For each solution, we test the precision of top-k retrieved results that belong to the ground truth provided by the test set of INEX 2009 entity track. We adjust the parameter k used in picking top-k results, and compute the average precision of all the queries. The results for the three compared solutions are shown in Fig. 4. It can be easily found that the precision of the proposed solution (*filter*) is much better than those of the other two baselines (*keyword* and *NLQ*) for all settings of the query parameter k. This is because the proposed solution is able to more accurately capture the user query intention through the feedback process.

Note that, for the *filter* solution in the above test, we allow users to provide at most 3 rounds of feedbacks. The results of *filter* is based on an average of results from five users participating in the test.

To evaluate the proposed ranking algorithm for predicate-keyword pairs (Algorithm 1), we use two baselines for predicate-keyword pairs' suggestion. The first baseline (shorted as B_1) simply ranks predicate-keyword pairs based on their frequency in the searching results.

[1] http://dbpedia.org

[2] http://www.inex.otago.ac.nz/tracks/entity-ranking/entity-ranking.asp

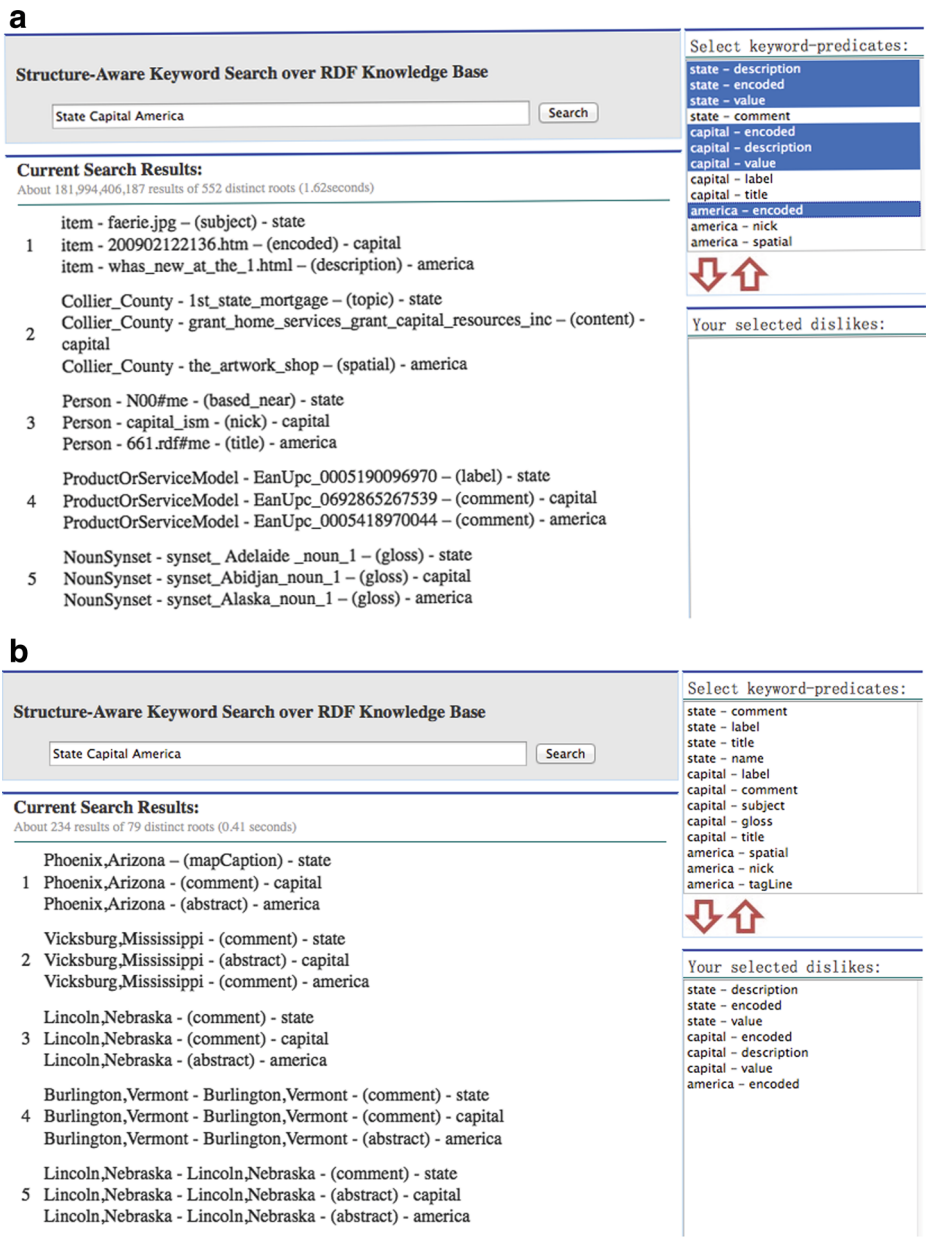

Fig. 3 Snapshots of the search interface

The second baseline (shorted as B_2) exactly computes the optimal c predicate-keyword pairs maximizing the number of structure-aware matching trees that can be pruned. We compare our solution with the two alternatives using two measures: 1) the filtering ratio (percentage of remaining results after feedbacks) and 2) the elapsing time of each feedback process. By default, we set the query parameter $d_{max} = 2$ for all keyword queries.

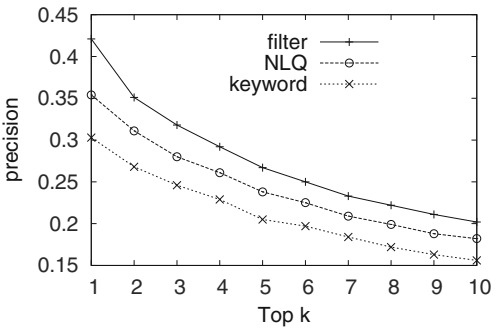

Fig. 4 The precision of *top*-k results

A subset of 18 queries that have the top numbers of matching trees are selected from the overall query set. Before testing, for each query, we manually select all the qualified predicate-keyword pairs as the ground truth. They will be deselected by the user once appearing in the feedback interface. This guarantees that the three testing approaches use the same rule for filtering structure-aware matching trees. For each query topic in the results of Figs. 5 and 6, the red (left) one stands for B_1; the blue (right) one stands for B_2; and the black (middle) one stands for the proposed algorithm (shorted as A_1).

Figure 5 shows the results of the percentage of remaining results after the first three rounds of feedbacks. Squares, crosses and circles represent the first, the second and the third round of feedbacks respectively. We can see that, for all the approaches, the more rounds of feedbacks applied, the less of the remaining results left. Comparatively, the pruning power of A_1 and B_2 is much better than that of B_1 for almost every round of a query. This is reasonable because both A_1 and B_2 try to maximize the number of structure-aware matching trees that can be pruned by selecting all the predicate-keyword pairs shown out in the query interface. The pruning power of B_2 is slightly better than that of A_1 in many cases. This is because B_2 is an optimal solution (although expensive) for suggesting predicate-keyword pairs. However, the pruning performance of A_1 is not surpassed by B_2 too much, in some cases (e.g., $Q2$, $Q4$, $Q6$, $Q17$), it even performs better than B_2. This is also reasonable, because not all the predicate-keyword pairs will be selected for filtering during the feedbacks. In general, the results of Fig. 5 show that the proposed algorithm (A_1) is a good approximation of B_2. It can significantly reduce the number of structure-aware matching trees through a small number of feedbacks for picking predicate-keyword pairs to be filtered.

Figure 6 shows the time cost for generating the contents of interface, majorly for creating the list of top predicate-keyword pairs (in our implementation, we set $c = 12$ in the Algorithm 1). A square represents the end-to-end time from receiving a keyword query to

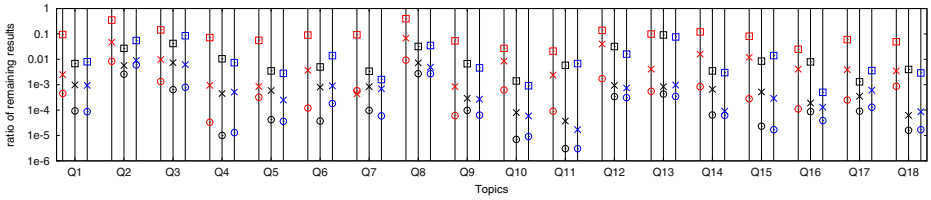

Fig. 5 Comparison of the pruning power of the 3 approaches for the first three rounds of feedbacks

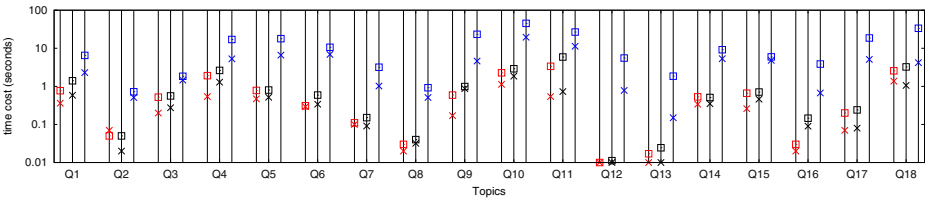

Fig. 6 Comparison of the efficiency of the 3 approaches for generating feedback interfaces

generating the interface for the first round of feedback. A cross represents the end-to-end time from receiving the first feedback to generating the interface for the second round of feedback. We do not show the results for the other rounds of feedbacks as they are typically faster than the first round. According to the results of Fig. 6, the performance of B_2 is the worst among the three approaches. B_2 often costs more than 4 s, which are not tolerable for many interactive information seeking tasks. Comparatively, B_1 and A_1 are much faster. Almost all the cases of A_1 take less than 3 s (with only one exception $Q11$). B_1 is faster than A_1 because it is a simple implementation with very poor pruning power of matching results (shown out in Fig. 5). In general, the results of Fig. 6 show that the proposed algorithm (A_1 for Algorithm 1) is a very efficient solution for the interactive keyword search task. It can be easy applied to large scale RDF knowledge bases such as DBPedia containing millions of entities.

We further test the performance of the proposed ranking algorithm for predicate-keyword pairs (Algorithm 1) under different parameter settings. We choose two important parameters of Algorithm 1 for testing: c and m. We test the impacts of the parameter c on the ratio of remaining results simply for the first round of feedbacks. We also test its impacts on the time cost for generating the interface of the first round feedback. Table 1 show the results when the parameter c is set from 6 to 18. Note that the results are an average over all the queries. As we can see from the results, the enlargement of the parameter c increases the pruning power (with ratio of remaining results drops) of the algorithm. However, larger computational cost are desired when the parameter c is enlarged. Obviously, there is a trade-off between the pruning power and the time cost when setting the parameter c. We set $c = 12$ by default for the other tests, because practically it is convenient for users to pick predicate-keyword pairs from such a number of choices.

Similarly, we also test the impacts of the parameter m in Algorithm 1 by adjusting its value from 10 to 50. The results are shown in Table 2. We can observe from the results that, the enlargement of the parameter m increases the pruning power of the algorithm. However, this is achieved by paying a larger computational cost. We set $m = 20$ by default for the other tests.

Table 1 The impacts of the parameter c

c	6	8	10	**12**	14	16	18
Ratio of remaining results	0.0673	0.0252	0.0114	**0.0079**	0.0046	0.0029	0.0013
Time cost (seconds)	0.22	0.27	0.62	**0.88**	1.15	1.40	1.52

The bold emphasis indicates a default value of c

Table 2 The impacts of the parameter *m*

m	10	15	**20**	30	40	50
Ratio of remaining results	0.095	0.0082	**0.0079**	0.0071	0.0066	0.0059
Rime cost (seconds)	0.42	0.74	**0.88**	1.18	1.29	1.52

The bold emphasis indicates a default value of *m*

6 Conclusion

In this paper, we propose an effective keyword search solution to answer keyword queries over large-scale RDF knowledge bases. Our method takes the advantage of keyword search (easy-to-use) and that of SPARQL query (better semantic discrimination) by associating predicates with keywords. This is achieved by allowing users to select predicate-keyword pairs for filtering the false interpretations of spots of query keywords. To make the feedback process more effective (i.e., avoiding too much manual effort) and efficient (at an interactive speed), we adopt a greedy approach to find a sub-optimal set of predicate-keyword pairs for user to provide feedbacks. By applying the proposed algorithm for interactive predicates filtering, we conduct extensive experiments to show that the proposed solution is able to support structure-aware keyword search over large-scale RDF data efficiently and effectively.

References

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G. (2007). Dbpedia: a nucleus for a web of open data. In *ISWC/ASWC* (pp. 722–735).

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S. (2002). Keyword searching and browsing in databases using banks. In *ICDE* (pp. 431–440).

Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD conference* (pp. 1247–1250).

Elbassuoni, S., & Blanco, R. (2011). Keyword search over rdf graphs. In *CIKM* (pp. 237–242).

Ferré, S., & Hermann, A. (2011). Semantic search: reconciling expressive querying and exploratory search. In *International semantic web conference* (Vol. 1, pp. 177–192).

He, H., Wang, H., Yang, J., Yu, P.S. (2007). Blinks: ranked keyword searches on graphs. In *SIGMOD conference* (pp. 305–316).

Heath, T., & Bizer, C. (2011). Linked data: evolving the web into a global data space. Synthesis lectures on the semantic web. Morgan & Claypool Publishers.

Huang, J., Abadi, D.J., Ren, K. (2011). Scalable sparql querying of large rdf graphs. *PVLDB*, *4*(11), 1123–1134.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *VLDB* (pp. 505–516).

Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L. (2008). Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD conference* (pp. 903–914).

Neumann, T., & Weikum, G. (2010). The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, *19*(1), 91–113.

Pérez, J., Arenas, M., Gutierrez, C. (2006). Semantics and complexity of sparql. In *International semantic web conference* (pp. 30–43).

Pound, J., Ilyas, I.F., Weddell, G.E. (2010). Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *SIGMOD conference* (pp. 423–434).

Tran, T., Wang, H., Rudolph, S., Cimiano, P. (2009). Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE* (pp 405–416).

Unger, C., Bühmann, L., Lehmann, J., Ngomo, A.C.N., Gerber, D., Cimiano, P. (2012). Template-based question answering over rdf data. In *WWW* (pp. 639–648).

Weikum, G., & Theobald, M. (2010). From information to knowledge: harvesting entities and relationships from web sources. In *PODS* (pp. 65–76).

Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D. (2003). Efficient rdf storage and retrieval in jena2. In *SWDB* (pp. 131–150).

Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., Weikum, G. (2012). Deep answers for naturally asked questions on the web of data. In *WWW (Companion Volume)* (pp. 445–449).