Contents lists available at ScienceDirect

Neurocomputing

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

Semantic similarity method for keyword query system on RDF

Minho Bae [a], Sanggil Kang [b], Sangyoon Oh [a,*]

[a] Department of Computer Engineering, Ajou University, Suwon, Republic of Korea
[b] Department of Computer and Information Engineering, Inha University, Incheon, Republic of Korea

ARTICLE INFO

Article history:
Received 31 October 2013
Received in revised form
11 April 2014
Accepted 23 April 2014
Available online 6 July 2014

Keywords:
Keyword query
RDF
Semantic similarity
WordNet

ABSTRACT

Keyword query on RDF data is an effective option because it is lightweight and it is not necessary to have prior knowledge on the data schema or a formal query language such as SPARQL. However, optimizing the query processing to produce the most relevant results with only minimum computations is a challenging research issue. Current proposals suffer from several drawbacks, e.g., limited scalability, tight coupling with the existing ontology, and too many computations. To address these problems, we propose a novel approach to keyword search with automatic depth decisions using the relational and semantic similarities. Our approach uses a predicate that represents the semantic relationship between the subject and object. We take advantage of this to narrow down the target RDF data. The semantic similarity score is then calculated for objects with the same predicate. We make a linear combination of two scores to get the similarity score that is used to determine the depth of given keyword query results. We evaluate our algorithm with other approaches in terms of accuracy and query processing performance. The results of our empirical experiments show that our approach outperforms other existing approaches in terms of efficiency and query processing performance.

1. Introduction

The RDF data model is a major part of the Semantic Web architecture, and more RDF data than ever before are available to represent Web data. As a response to such popularity, there has been a recent explosion in the number of proposals for the design of an RDF data management systems that indexes and processes RDF data sets [1]. However, most of these proposals have not yet completely solved the scalability or efficiency problems inherit from the RDF graph model. As the amount of RDF data is quickly approaching the Web scale, these problems are increasing. Therefore, many of the proposals and systems limited to single-machine storage with RDBMS are not feasible to support this scale of RDF data. Although some of these proposals offer distributed storage that scale as the data size grows [2,3], the scalability issue is one of the important research challenges remaining to be solved. Query efficiency is another challenging research issue. The majority of current proposals supporting the complex SPARQL language, the W3C recommended query language for RDF [4]. However, SPARQL expressions are short for meaningful semantic queries [5] and too complex for simple information retrieval from large RDF collections.

Several recent proposals have focused on keyword query concept that does not require any knowledge about the target RDF data or prior knowledge on the SPARQL query language. As in a keyword search on the World Wide Web (WWW) through a popular search engine (e.g., Google or Bing), a keyword query on RDF data sets requires only keywords instead of triple patterns in SPARQL (i.e., a triple with variables such as $<$?x foaf:name Jon Foobar$>$) that is formalized based on the RDF structure (i.e., triples of subject, predicate, and object). A keyword query is a more intuitive way of specifying information needs. For massive-scale RDF data sets, processing a formal query such as SPARQL requires tremendous number of computations and produces large amounts of intermediate data for a join operation. As a result, a keyword query with an optimized indexing structure has become a promising design for a Web-scale RDF data management system.

However, processing a keyword query on a graph-based RDF data set is a non-trivial task. Although it is relatively easy to locate the keyword in the data set, it is a complex and complicated process to provide ranked results of the keyword query [6]. In this paper, we identify challenges that need to be tackled to use of a keyword querying paradigm for a massive-scale RDF data set. For example, how do we determine the size of the query results (i.e., sub-graphs) that will be returned to the user? In other words, how much search depth (or ranking) do we need to satisfy users? This is a necessary question to address because, unless we naively return the whole sub-graph connected to the keyword, we need a specific depth for each keyword so that the search process (i.e., query processing) can be halted at a certain point and the results returned to the users. For information retrieval on the WWW,

* Corresponding author. Tel.: +82 31 219 2633; fax: +82 31 219 1725.
E-mail address: syoh@ajou.ac.kr (S. Oh).

http://dx.doi.org/10.1016/j.neucom.2014.04.062
0925-2312/© 2014 Elsevier B.V. All rights reserved.

recalling the naïve return results from early search engines, a vast number of *url* returns is not a big problem. However, when Google introduced its pagerank algorithm [7] to rank the results from the most important first, the quality of the search results improved astonishingly. Likewise, the depth of the search is a critical factor for RDF querying. The key research challenges in this area that have to be addressed are as follows:

- How can we determine when to stop the search? What metric can we use to halt the query processing? A ranked keyword search on graph-based data including RDF data poses certain challenges. Approaches such as XRANK, XML-based keyword search [8] utilize a tree structure and the existence of the root element. However, for a keyword search on RDF data, the same strategy cannot be applied since the graph structure is more complex (e.g., it is not hierarchical data storage and there is no root). In addition, choosing the right metric for both ranking the results and eventually halting the process is another challenging research issue. Zhang et al. [9] proposed a multi-level semantic similarity method that measures the string similarity between elements of the triple, statements, and graph. While this approach is practical, we need a general method that takes advantage of the semantic relationship of the keyword that can be derived from the predicate.
- How to develop a comprehensive RDF data management system to enable an automatic depth determining algorithm for high-performance query processing at a massive-scale RDF data. There are two main problems in managing massive-scale RDF data. First, currently popular systems such as Virtuoso [2] and RDF-3x [10] are limited in scalability because they were designed to support a formal query, SPARQL, and use RDBMS for storage. Second, many approaches to RDF data management do not fully utilize indexing. They simply use an index to locate the stored vertex that contains the keyword [6].

Motivated by these problems, we propose a novel automatic depth search algorithm for an RDF data management system. Our main contributions to these challenges are as follows:

- **Automatic *k*-depth decision for a keyword search based on relational and semantic similarities.** Our algorithm automatically determines the depth of the search by calculating the similarity between the given keywords and vertices in the RDF graph (i.e., RDF data set). In the query processing algorithm, we first search the keyword in the RDF data set using an index. Since the predicate represents the semantic relationship between a subject and object, we take advantage of this to narrow down the target RDF data. The similarity score is then calculated for objects with the same predicate using the keyword vertex. We use the word distance for the relational similarity and measure the semantic similarity using WordNet [11] value to determine the semantic similarity score of the object to the given keyword.
- **A novel scalable and high-performance structure indexing.** This algorithm is based on the indexing structure described in our previous work [12], which explored how to index massive-scale RDF data sets for formal queries such as SPARQL and keyword queries. In particular, we extend the indexing structure for a keyword query and modify the keyword querying algorithm to adapt the automatic *k-depth* using the similarity.

To empirically verify the effectiveness of our approach, we conducted experiments on an RDF data management system based on our indexing and querying algorithm. In these experiments, we verified the effectiveness of our algorithm and demonstrate the process of computing the relational and semantic similarity. Also, we compared our approach with a naïve keyword search approach in terms of both query response time and accuracy.

We proceed as follows. In Section 2, we present related researches of approaches that support a keyword query on an RDF data set, and algorithms for calculating the similarity in structured data. In Section 3, we present our approach for an efficient RDF keyword query processing with an automatic *k-depth* decision. In Section 4, we present our empirical experiment results to demonstrate the effectiveness of our automatic *k-depth* algorithm for keyword queries on RDF data sets. Finally, we provide some concluding remarks and describe some future research directions in Section 5.

2. Related work

There have been a lot of proposals that provide a top-*k* keyword query to structured or semi-structured data. In He et al. [6], BLINKs, a bi-level indexing and query processing scheme was proposed to search the top-*k* keyword of a data graph. In addition, the authors introduced a bi-level indexing to reduce the abundant memory usage of previous approaches and improve the search performance. For this, the data graph is partitioned into a sub-graph (or block) rather than naively storing whole graphs and the shortest path. A keyword list is built first. The graph is then partitioned into sub-graphs using each keyword in the list and the breadth first search (BFS) algorithm. Since it only searches the sub-graph of the keyword, the overall query processing performance is improved.

The keyword search system over structured data by Elbassuoni and Blanco [13] directly retrieves results for the keyword query. Using a keyword phrase, the authors use a sub-graph retrieval algorithm to return a set of sub-graphs that match the query keywords using their ranking model. To do so, for all triple data, they maintain a list of keywords derived from the subject- and object-associated predicate. They then create an inverted index for each keyword query with a list of corresponding triples so that they can join these data to obtain the sub-graphs of all keywords in the query by adapting the backtracking algorithm [14]. Considering and maintaining keywords from all three factors (subject, predicate, and object) requires time, memory, and a well-designed schema rather than only the literal values of the objects.

In Zhang et al. [9], various methods for dealing with the similarity in RDF graph matching was proposed. They use four methods and provided a formula for their integration. The first method measures the level. If the node is closer, it has a better similarity. The second method measures a string similarity between objects using WordNet. Finally, the third and fourth methods measure the RDF data statement similarity and RDF graph structure similarity, respectively.

An algorithm for the top-*k* exploration of sub-graphs to retrieve the top-*k* most relevant structured queries was proposed in Tran et al. [15]. In this proposal, a keyword index is used for elements, and a structure index is used for an RDF data graph. The keywords in the query are translated into expressive formal queries. First, instead of mapping the keywords to the data tuples, they are mapped to the elements in the data graph using a keyword index. Next, with each mapping of such keyword elements, the summary graph (i.e., structure index of the original RDF data graph) is explored to search for the augmented query graph (substructure) connecting them. From these sub-graphs, conjunctive queries are created by mapping the edges with predicates, and the vertices with variables or literal values. Their system then generates the top-*k* queries using a scoring function, instead of computing answers for the keywords. Finally, users need to select their appropriate queries from these proposed structure queries to find