JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 923-934 (2019)
DOI: 10.6688/JISE.201907_35(4).0013

A Methodology for Indexing Temporal RDF Data*

PING ZHAO AND LI YAN+
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, 211106 P.R. China

Time information widely exists in various real-world applications and RDF (Resource Description Framework) has been using as a data model and representation format for semantic data processing. With the rapid growth of RDF data with time features, efficient and effective management of temporal RDF data is an important task. For this purpose, we propose to index temporal RDF data in the paper. We propose a novel temporal RDF data index structure. We present its two levels of indexes: the first one is a global index for time information of RDF triples and the second is a local index for non-time information of RDF triples. We exploit efficient algorithms to build the global index and the local index. We evaluate our index approach from index performances and querying performances. The experimental results show that our index approach is scalable and efficient.

Keywords: RDF, temporal RDF, index structure, K-D tree index, bitmap index

1. INTRODUCTION

The resource description framework (RDF) [1] is a metadata model recommended by the W3C (World Wide Web Consortium) for building an infrastructure of machine-readable semantics for resources on the Web. Currently, many applications have started using RDF as a data model and representation format for semantic data processing. For example, Bio2RDF [2] uses RDF model to build a life sciences network; Lexvo [4] takes language and literature into semantic network; LinkedMDB [5] releases the opening data to build the first movie semantic web. Furthermore, the Linked Open Data (LOD) [6] provides one large RDF data cloud by linking web data from a diverse set of domains like DBpedia [3] from Wikipedia. With the increasing amount of RDF data available, efficient management of RDF data is of crucial importance [7, 28, 29]. Among the issues of RDF data management, indexing plays an important role in RDF data query and it is especially true for querying large-scaled RDF data [8-17].

Time information widely exists in various applications such as sensor network [14], biology [15], industry [16] and so on. Typically, some data may be available or correct at some time and some historical data need to be recorded. So, it is necessary to manage temporal data effectively. A core issue in temporal data management is to establish a suitable temporal data model [17]. Many temporal data models, for example, temporal database model and temporal ER/EER model [18], have been proposed. More recently, with the wide utilization of XML (eXtensible Markup Language) for data representation and exchange on the Web, temporal XML model has been proposed as well [19]. Tem-

Received January 26, 2018; revised August 20 & October 24, 2018; accepted November 15, 2018.
Communicated by Wei-Shinn Ku.
+ Corresponding author, e-mail: yanli@nuaa.edu.cn.
* This work was supported in part by the National Natural Science Foundation of China (61772269 and 61370075).

poral XML provides an infrastructure of managing temporal data on the Web.

Generally, RDF can only reflect that a majority of facts is static in the real world. In order to model the facts that it exists only for a limited time, several efforts have been made to process temporal information with RDF. Temporal RDF model is first proposed in [20], in which a syntax and semantics for temporal RDF are provided. Several issues such as *reasoning* [21] and *query* [22] are investigated in the context of temporal RDF. We argue that efficient queries of large-scaled RDF data rely on RDF index. Indexing temporal RDF data is not carefully considered yet. So, we investigate the issue of temporal RDF index in this paper. Temporal RDF index can play an important role in RDF data management and especially improve RDF query efficiency.

In this paper, we propose a two-level temporal RDF index. First, we apply K-D tree index as the global index to index temporal information about RDF data. Second, we apply composition bitmap index as a local index to index RDF triple information. The main contributions of this paper are summarized as follows:

(1) We propose a temporal RDF data model with transaction time instead of the existing temporal RDF data models with valid time.
(2) On the basis of the temporal RDF data model proposed in the paper, we propose an index model for the temporal RDF data model. To the best of our knowledge, this is the first effort to construct index directly against the temporal RDF triple model.
(3) We develop an efficient algorithm for constructing temporal RDF indexes. We further evaluate the performances of our index approach with experiments by constructing the synthetic temporal RDF datasets from the traditional benchmark LUBM.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of related work in temporal RDF model and classical RDF index. Section 3 proposes our scheme about temporal RDF index and the index algorithm. Section 4 presents the experimental evaluations of our approach. Section 5 concludes this paper and sketches our future work.

2. RELATED WORK

2.1 Index for RDF Data

In order to improve RDF query efficiency, several index schemes about RDF are proposed for different application scenarios. Basically, we can identify two categories of RDF data index models: one is based on the triples and the other is based on the RDF graph. The former constructs the undifferentiated index structure about three triple elements. The latter builds the indexes according to the features of RDF graph.

In [8], the sextuple indexing model is proposed, which extends the method of vertical partitioning and treats subjects, predicates, and objects equally. The bitmap index for triples is proposed in [9], which contains three composite bitmap indexes. The keys of those composite bitmap are composite values of *predicate-subject*, *predicate-object*, and *subject-object*. In [10], Up and Down documents are introduced to store temporal RDF data. With respect to an object element of triples, the Up document stores the data group that has a predicate as a key and subjects as its values (*e.g.*, {"$predicate_1$":[$subject_1$, *sub-*

ject$_2$, ..., *subject*$_n$]}, ..., {"*predicate*$_n$":[subject$_{n+1}$, subject$_{n+2}$, ..., *subject*$_m$]}). With respect to a subject element of triples, the Down document stores objects (*e.g.*, [*object*$_1$, *object*$_2$, ..., *object*$_n$]).

In [11], the RG-index is proposed, which can improve the filtering power of triples. The RG-index can index graph patterns in RDF graph. In [12], RDF graph is mapped to an index structure, where the index nodes are a collection of the nodes of the original RDF graph and the index edges are those of equivalent types. Based on RDF graph, indexed graphs are generated in [13]. The created index graph is a summary of the paths derived from the corresponding data graph. So, the index graph is data-oriented structure and the same path elements are placed in an element group.

We summarize current major RDF indexes as follows. The index based on RDF triples is suitable for single element query, whose structure is document and whose storage space is relatively large. The index based on RDF graph is suitable for path query, whose structure is summary graph and whose storage space is relatively small.

2.2 Temporal Data Modeling in RDF

Temporal data modeling has been investigated in the context of databases (*e.g.*, [18]) and XML (*e.g.*, [19]). Two major temporal dimensions are considered in temporal databases, which are *valid time* and *transaction time*. Valid time is the time when data is valid in the modeled world; the transaction time is the time when data is actually stored in the data model. More recently, the temporal RDF model is proposed. In [20], a temporal triple is defined as an RDF triple with a temporal label. Then a temporal triple has the form (a, b, c):[t_1, t_2], where (a, b, c) is a triple and [t_1, t_2] is a time interval. Then (a, b, c):[t_1, t_2] represents the time period when the triple is valid in the real world.

To represent historical information in RDF, temporal properties are considered in [25], where a temporal triple has a form of (s, p:[*start*, *end*], o). Here p:[*start*, *end*] is a temporal property and the effective time of property p is [*start*, *end*]. A straightforward extension to the case of indeterminate triples is proposed in [23]. The temporal RDF database consists of a set of temporally annotated RDF triples with a form of (*subject*, *property*:*annotation*, *object*) [24].

Although some efforts have devoted to RDF index and temporal RDF data modeling has received more attention, the research works of temporal RDF data processing are still scarce. Several issues such as *reasoning* [21] and *query* [22] are investigated in the context of temporal RDF. However, to the best of our knowledge, only the work in [24] investigates the issue of indexing temporal RDF data. The present paper is different from [24] in two major aspects. First, this paper can deal with the transaction time while the papers of [20, 24] can only deal with the valid time. Second, the tGRIN index structure proposed in [24] is built upon the relational databases which physically store temporal RDF data rather than directly upon temporal RDF triples.

3. TEMPORAL RDF INDEX

Let U be a set of URI references, L a set of literals, and P a set of properties. Then, an RDF triple is a triple (s, p, o) in $U \times P \times (U \cup L)$. The elements s, p, and o of RDF triples are called the *subject*, *property* (also called *predicate*) and *object*, respectively.

3.1 Structure of Temporal RDF Index

The basic idea of temporal RDF is to annotate triples or/and triple elements (*i.e.*, *subject*(*s*), *property*(*p*) and *object*(*o*)) with temporal labels. The type of temporal label can be transaction time, it is applied to record the time that a transaction (*e.g.*, an operation or change) occurs. If we operate on RDF data, the transaction time can faithfully record the history of data changes in the operation. Therefore, in this paper, we concentrate on transaction time as a main temporal dimension.

Definition 1: A temporal triple is an RDF triple with a time interval. Formally, a temporal RDF triple is denoted (*s*, *p*, *o*):[*ts*, *te*], in which *s*, *p*, and *o* represent the subject, predicate and object, respectively, and [*ts*, *te*] represents a transaction time starting at *ts* and ending at *te*. A temporal RDF data model is a set of temporal RDF triples with the form of $\{(s, p, o):[ts, te] \mid ts \leq te\}$. Here *ts* and *te* are the two time points.

t1	(sue, type, CEO):[4, 10]	t7	(likes, type, socialRel):[0, 20]
t2	(crispin, type, VP):[0, 20]	t8	(knows, type, socialRel):[0, 20]
t3	(sue, manage, joe):[9, 13]	t9	(yonei, knows, yong):[3, 9]
t4	(jane, firend, lucy):[3, 9]	t10	(yong, report, tamae):[3, 18]
t5	(crispin, knows, larry):[6, 15]	t11	(report, type, social):[9, 13]
t6	(larry, likes, sarah):[6, 15]	t12	(joe, report, jane):[3, 18]

Fig. 1. An example of temporal RDF data model.

Fig. 1 presents a simple example of temporal RDF data model which contains some temporal RDF triples.

To index temporal RDF data, we introduce two concepts named *interval list* and *triple group*. The interval list of a temporal RDF data model contains all unduplicated transaction time of the temporal triples. We get all the triples with the same transaction time to construct triple groups. Moreover, each transaction time in the interval list corresponds to a triple group and the interval list corresponds to a set of triple group.

Example 1: Let us look at the temporal RDF data model in Fig. 1. Its interval list is {[4, 10], [3, 18], [9, 13], [6, 15], [3, 9], [0, 20]}. For the interval [0, 20], we have a triple group {(likes, type, socialRel), (knows, type, socialRel), (crispin, type, VP)}.

K-D tree index is a space-partitioning data structure for organizing the points in a *k*-dimensional space, which is mainly used to retrieve multi-dimensional data with a very high efficiency [26]. Since the transaction time is two-dimensional data, we can construct a K-D tree (where *k* can represent 2) to index the temporal information of temporal RDF model. K-D tree is built by recursively decomposing the point set into two equal subsets according to the median data of the dimension whose variance of the data is larger. K-D tree index about the interval list of the temporal RDF model in Example 1, which is shown in Fig. 2, contains all elements of the interval list as the values of its nodes.

Note that the K-D tree can only index temporal RDF data at a level of time intervals. To speed up retrieval of triples, we do need to further index the triples in each triple

group. In the index about the triple group, we construct the index about triple group by extending the composition bitmap proposed in [9]. Considering various possible query conditions, we propose three composition indexes in this paper. The keys of composition bitmaps are *subject-predicate* (*s-p* for short), *predicate-object* (*p-o* for short), and *object-subject* (*o-s* for short) complex values, respectively. We store the above composite index in an array b_{mn}, where m and n are the numbers of rows and columns in the array, respectively. Generally speaking, we use b_{00} to store the corresponding transaction time of the triple group. In the *s-p* key bitmap, for example, $b_{[i][0]}$ $(i > 0)$ is applied to store all subjects, $b_{[0][j]}$ $(j > 0)$ to store all predicates, and $b_{[i][j]}$ $(i > 0$ and $j > 0)$ to store the corresponding objects. If a bitmap index contains such triples that have the same subject and predicate but different objects, that is, $b_{[i][j]}$ corresponds to a number of different objects, they are treated as the part of $b_{[i][j]}$.

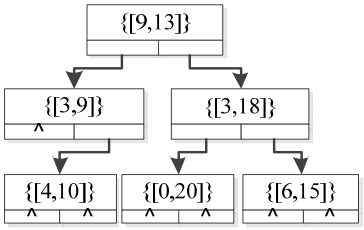

Fig. 2. *K-D* tree index.

[0,20]	type
crispin	{VP}
likes	{socialRel}
knows	{socialRel}

Fig. 3. *s-p* composite bitmap index.

[0,20]	socialRel
type	{crispin,likes,knows}

Fig. 4. *p-o* composite bitmap index.

[0,20]	crispin	likes	knows
VP	{type}	null	null
socialRel	null	{type}	{type}

Fig. 5. *o-s* composite bitmap index.

Let us look at the triple group that corresponds to the interval [0, 20] in Example 1. Three composition bitmap indexes for the triple group are showed in Figs. 3-5. The key values of these three composition bitmap indexes are *s-p*, *p-o*, and *o-s*, respectively.

Combinedly using the temporal K-D tree index and the composition bitmap indexes, we can construct the index structure of temporal RDF data. The temporal RDF index structure is a tree model. We define the temporal RDF index tree as follows.

Definition 2: The model of temporal RDF index tree has a form of (N, C, V).

(1) N is a set of nodes of the index tree. Three types of nodes can be identified: *non-leaf node* (denoted n_n), *leaf node* (denoted n_l) and *empty node* (denoted n_e). Then we have $N = \{n_n\} \cup \{n_l\} \cup \{n_e\}$.

(2) C: $N \rightarrow N$ is a set of parent-child relationships between nodes. For two nodes n_i and n_j, $C(n_i) = n_j$ means that n_i is the patent of n_j, in which $n_i \in \{n_n\}$ and $n_j \in \{n_n\} \cup \{n_l\} \cup \{n_e\}$. For node $n_i \in \{n_n\}$, it has three *leaf nodes* and two *non-leaf*/*empty nodes* as its children.

(3) V: $N \rightarrow$value is an assignment function. For a non-leaf node $n_n \in N$, $V(n_n)$ is a list of time intervals (*i.e.*, $V(n_n) = \{[ts, te]\}$). For a leaf node n_l, $V(n_l)$ is a composition bitmap $\{b_{[i][j]}\}$.

We give an example to illustrate the temporal RDF index tree. In Fig. 2, we show the K-D tree about the interval list of the temporal data model in Fig1 and then we need add the local indexes to the index tree. The nodes of the index tree usually have 5 child nodes, two are child nodes of the original K-D tree and the other three are the composition bitmap indexes. We have the temporal RDF tree index of the temporal RDF data in Fig. 1, which is shown in Fig. 6.

To sum up, we first construct the temporal K-D tree index of the interval list as the global index and then construct the composition bitmap index of the triple group as the local index. When querying temporal RDF triples, the global index is used to locate the triple groups and then the local composite bitmap index is used to locate the triples that satisfy the given temporal query. Compared with the composite bitmaps in [9] for regular RDF data without temporal information, the size of the local composite bitmap index that is used for a given temporal querying is very small. So, the proposed index is more efficient for massive temporal RDF data querying than the index proposed in [9].

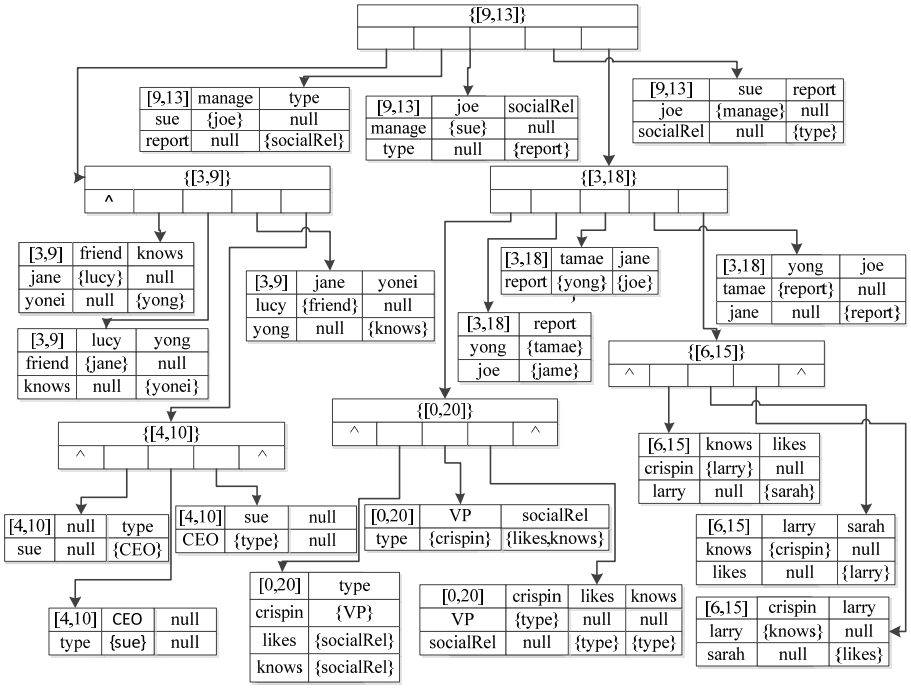

Fig. 6. Temporal RDF index.

3.2 Algorithm of Indexing Temporal RDF

Our index approach is to find suitable triple groups according to the time intervals and then validate the appropriate triples in the triple group. We first propose Algorithm 1 to build the global index which is actually used to build the temporal K-D tree index and then use Algorithm 2 to create the local indexes which are composition bitmap indexes.

Algorithm 1 shows the overall processing of building the global index of temporal

RDF model. We traverse the dataset to get the interval list *I* for the non-leaf node value of the global index (line 1). Then we start to build K-D tree, if *I* is null, the algorithm returns null (line 3). This means that a branch of the index tree is built. Otherwise, we process *I* to obtain the split dimension *D* and split value *Mid* (lines 4-6). We process each element in *I* to obtain the variance of each dimension and compare all the variance to get the larger variance and the related dimension (lines 13-21). We get the median *Mid* of dimension *D* by processing interval list *I* on the dimension *D* (line 6). The interval in which *Mid* is located is obtained by *D* and *I*, which is the value of the current node *Node* (line 7). Next, we need to build the child nodes of *Node*. Our index also contains local indexes, which is mainly realized by calling Algorithm 2 (line 10). The construction of the other non-leaf nodes is mainly to recursively call the part of creating nodes itself (lines 8-12). The other non-leaf nodes of the index tree are created with the same dataset and different interval list. We can construct the index tree until the nodes are all created.

Algorithm 1: Global index

```
Input:   temporal RDF dataset S
Output: temporal RDF index
1.  I ← getTimeInterval(S)                    // traverse the dataset to get all time intervals
2.  CreateNode (I, S)
3.      If I is null Return null
4.      Split (I)
5.          D ← judgeDimension (I)
6.          Mid ← getMidValue (I, D)          // get the median in dimension D
7.      Node.Data ← getMidInterval (D, I, Mid)
                        // get the list of intervals whose value is Mid on dimension D
8.      I_left ← getleftInterval (D, I)
                        // get the list of intervals whose value is less than Mid on dimension D
9.      Node.left ← CreateNode (I_left, S)
10.     call local index (Node.Data, S)
11.     I_right ← getrightInterval (D, I)
                        // get the list of intervals whose value is greater than Mid on dimension D
12.     Node.right ← CreateNode (I_right, S)
13.judgeDimension(I)                          // get the split dimension
14.     Set maxDimension=0, maxVariance=0;
15.     For (all dimension in I)
16.         Set var=getVariance(I, dimension)
17.         If(var>maxVariance)
18.             maxVariance=var
19.             maxDimension=dimension
20.     dimension=maxDimension;
21.     return dimension;
```

In Algorithm 1, we call Algorithm 2 to build a local index. Algorithm 2 is actually used to build the composition bitmap indexes and create the leaf nodes of temporal RDF index structure. The key values of composition bitmaps are subject-predicate pairs.

Algorithm 2 is used to construct the local index of temporal RDF index. It is possible that there are multiple intervals related to the median and all possible intervals need to be considered (line 1). Generally, an interval corresponds to three composite bitmap indexes. The first element b_{00} of the bitmap *b* is the time interval *i*, which is related to the

value of its parent node in Algorithm 1. We need to obtain the triple group G that is related to the interval i from the temporal RDF dataset S (line 2). We process the dataset S to obtain the subjects of triples and put all the subjects into the subject set {*Subject*} without duplicates (line 3). We use the similar method to obtain the predicate set {*predicate*} (line 4). Then, we can store the elements of the subject set {*Subject*} and the predicate set {*predicate*} in the first column b_{p0} of bitmap and the first row b_{0q} of bitmap sequentially (lines 5-6). Finally, we need to store the object in the bitmap (lines 7-9). We process each triple t in G and store the object of a triple t in the right position.

The time complexity of the algorithm1 is $O(k \times n \times \log n)$ and the time complexity of Algorithm 2 is $O(d \times n)$, where k is the number of time intervals, n is the number of temporal triples, d is the size of the interval list.

Algorithm 2: Local index

Input: time interval I, temporal RDF dataset S
Output: bitmap index b
1. For(all i in the interval list I)
2. $b_{00} \leftarrow i$
3. $G \leftarrow$ getTripleGroup (S, i) // get all triples in interval i in dataset S
4. {Subject} $\leftarrow$ getSubject (S) // get all the subjects of the dataset S
5. {predicate} $\leftarrow$ getPredicate (S) // get all the predicates of the dataset S
6. $b_{p0} \leftarrow$ {Subject} // $0 < p < n$ (n is the number of rows in the bitmap)
7. $b_{0q} \leftarrow$ {Predicate} // $0 < q < m$ (m is the number of columns in the bitmap)
8. For all $t \in G$
9. if (t.subject = b_{p0}&&t.predicate = b_{0q})
// the subject of the triple t, t.predicate: the predicate of triple t
10. $b_{pq} = t$.object

4. EXPERIMENTAL EVALUATION

To evaluate the performances of our approach for temporal RDF data index, we conduct some experiments with datasets and report experimental results. We first present several datasets we apply in our experiments and then our experimental results.

4.1 Experimental Datasets

We apply several RDF datasets which originate from the LUBM (Lehigh University Benchmark) [27]. The LUBM is developed to facilitate the evaluation of Semantic Web repositories in a standard and systematic way. We can obtain datasets by the generator of the LUBM. Note that the triples in the LUBM do not contain temporal information. For our experiments of temporal RDF index, we need to generate temporal RDF dataset by adding temporal information to the classical triples of the LUBM dataset randomly.

We evaluate the performance of our index approach by choosing several datasets with different size of interval lists and different numbers of triples. For this purpose, we use the LUBM datasets and create eight different datasets by randomly allocating time intervals to triples. To better evaluate the temporal RDF index, we use the LUBM datasets to compare the performance of index. The characteristics of the eight datasets are summarized in Table 1.

Table 1. Characteristics of LUBM datasets.

Dataset name	Size of interval list	Number of triples	Dataset name	Size of interval list	Number of triples
ds1	20	80	*ds2*	20	820
ds3	100	820	*ds4*	100	3320
ds5	*200*	*3320*	*ds6*	*200*	*6620*
ds7	*400*	*6620*	*ds8*	*400*	*9960*

4.2 Evaluation Results

Our experiments are implemented in JDK 1.8.0 with Eclipse and MySQL and performed on a system with Intel i5-2450m 2.5GHz processor, 2GB RAM, and Windows 7 operating system. We evaluate our index approach from two major aspects, which are index performances and querying performances.

(1) Index Performances

We evaluate the index performances of our index approach from several aspects, including *the building time of the index*, *the node number of the index*, and *the storage space of the index*. We observe how they vary over different datasets.

First, we evaluate the time of building indexes over different datasets in Table 2. It is shown in Fig. 7 that, for two datasets having the same number of triples but different sizes of interval lists (*e.g.*, ds1 and ds2), indexing the datasets with big size of interval list costs more than that with the small size of interval list; for two datasets having the same size of interval lists but different numbers of triples (*e.g.*, Dataset2 and datasets3), indexing the datasets with more triples costs more than that with fewer triples. So, the time of building index is related to the number of triples and the size of interval list.

Second, we evaluate the storage space of index over different datasets in Table 2. We respectively show in Figs. 8 and 9 how many nodes and spaces are used by the indexes. It can be seen from Fig. 8 that, for two datasets having the same number of triples

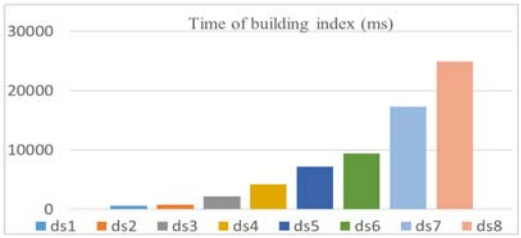

Fig. 7. Time of building index on different datasets.

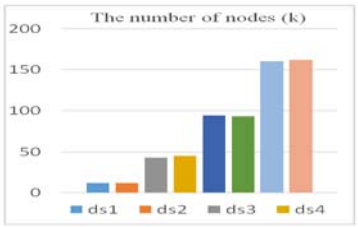

Fig. 8. Number of index nodes.

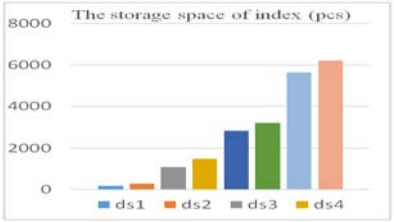

Fig. 9. Storage space of index.

but different sizes of interval lists (*e.g.*, ds2 and ds3), the index over the dataset with a big size of interval list contains more index nodes than the dataset with a small size of interval list; for two datasets having the same size of interval lists but different numbers of triples (*e.g.*, ds1 and ds2), the index over the dataset with more triples contains more index nodes than the dataset with fewer triples. The storage space of the index is closely related to the number of non-leaf nodes, which determines the number of composition bitmap indexes. So, like what we see in Fig. 8, we can see from Fig. 9 that the indexes over the datasets with more triples and big size of interval lists need more storage space than the datasets with fewer triples and small size of interval lists.

To sum up, the proposed temporal RDF index is suitable for temporal RDF datasets that contain many triples with the same subjects, predicates, objects and time intervals.

(2) Querying Performances

The index is generally used to improve querying performances. We define two types of queries: path query and star query, path query consists of several connected triple patterns that form a path and a star query consists of more than two path queries and the paths share exactly a common center node.

Here we apply two types of queries over the datasets in Table 2 and have *Query1* and *Query2*. In addition, we introduce a time interval query which is used to get the interval of triples and have *Query3*. Fig. 10 shows the times of each type of query over the datasets of Table 2 by using and not using the RDF index structure.

It can be seen from Fig. 10 that, for a given type of query and a given dataset, the query with index costs far less than without index. This is because these queries can rapidly locate right triples and save much time in searching useless triples. In addition, it can be seen in Fig. 10 that, for a given dataset, Query3 with index costs less than without index. But compared with Query1 and Query2 with index, Query3 with index cost more. This is because Query3 with the index is a kind of interval-based query, which needs to search all the local index and then return the related interval. Query1 and Query2 with indexes search the global index firstly to avoid searching the useless local index.

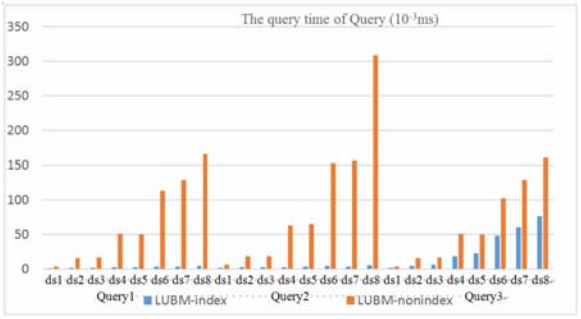

Fig. 10. The time of queries.

5. CONCLUSIONS

Time information widely exists in various real-world applications and RDF has been an important model for semantic data processing. In this paper, we present a novel

approach to index temporal RDF data, which can improve temporal RDF queries. The proposed two-level index contains a global index structured as a K-D tree and a local index structured as a composition bitmap index. The experimental results on datasets show that our index approach is scalable and efficient. In our future work, we will optimize our index approach to reduce the index space size and further improve query performance according to data scale and querying categories of temporal RDF.

REFERENCES

1. G. Klyne and J. J. Carroll, "Resource description framework (RDF): Concepts and abstract syntax," W3C Recommendation.
2. Bio2RDF, https://github.com/bio2rdf/bio2rdf-scripts/wiki.
3. DBpedia, http://wiki.dbpedia.org/about.
4. Lexvo, http://www.lexvo.org/linkeddata/tutorial.html.
5. LinkedMDB, http://linkedmdb.org/.
6. C. Bizer, T. Heath, and T. Berners-Lee, "Linked data – the story so far," *International Journal of Semantic Web and Information Systems*, Vol. 5, 2009, pp. 1-22.
7. Z. M. Ma, M. A. M. Capretz, and L. Yan, "Storing massive resource description framework (RDF) data: a survey," *Knowledge Engineering Review*, Vol. 31, 2016, pp. 391-413.
8. C. Weiss, P. Karras, and A. Bernstein, "Hexastore: Sextuple indexing for semantic web data management," *Proceedings of the VLDB Endowment*, Vol. 1, 2008, pp. 1008-1019.
9. K. Madduri and K. S. Wu, "Massive-scale RDF processing using compressed bitmap indexes," in *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, 2011, pp. 470-479.
10. M. Bae, J. Kihm, S. Kang, and S. Oh, "Indexing and querying algorithm based on structure indexing for managing massive-scale RDF data," *Journal of Intelligent and Fuzzy Systems*, Vol. 27, 2014, pp. 575-587.
11. K. Kim, B. Moon, and H. J. Kim, "RG-index: An RDF graph index for efficient SPARQL query processing," *Expert Systems with Applications*, Vol. 41, 2014, pp. 4596-4607.
12. F. Picalausa *et al.*, "A structural approach to indexing triples," in *Proceedings of the 9th Extended Semantic Web Conference*, 2012, pp. 406-421.
13. T. Tran, G. Ladwig, and S. Rudolph, "Managing structured and semistructured RDF data using structure indexes," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, 2013, pp. 2076-2089.
14. N. Q. Mehmood, R. Culmone, and L. Mostarda, "Modeling temporal aspects of sensor data for MongoDB NoSQL database," *Journal of Big Data*, Vol. 4, 2017, p. 8.
15. W. Z. Nie, H. Y. Cheng, and Y. T. Su, "Modeling temporal information of mitotic for mitotic event detection," *IEEE Transactions on Big Data*, Vol. 3, 2017, pp. 458-469.
16. A. Soylu *et al.*, "Querying industrial stream-temporal data: An ontology-based visual approach," *Journal of Ambient Intelligence and Smart Environments*, Vol. 9, 2017, pp. 77-95.
17. C. Claramunt *et al.*, "Special issue on spatial and temporal database management," *GeoInformatica*, Vol. 21, 2017, pp. 667-668.

18. A. Tansel, J. Clifford, and S. Gadia, *Temporal Databases: Theory*, *Design and Implementation*, Benjamin/Cummings, San Francisco, 1993.
19. F. Rizzolo and A. A. Vaisman, "Temporal XML: modeling, indexing, and query processing," *The VLDB Journal*, Vol. 17, 2008, pp. 1179-1212.
20. C. Gutierrez, C. A. Hurtado, and A. A. Vaisman, "Introducing time into RDF," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, 2007, pp. 207-218.
21. C. A. Hurtado and A. A. Vaisman, "Reasoning with temporal constraints in RDF," in *Proceedings of the 4th International Workshop on Principles and Practice of Semantic Web Reasoning*, 2006, pp. 164-178.
22. J. Tappolet and A. Bernstein, "Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL," in *Proceedings of the 6th European Semantic Web Conference*, 2009, pp. 308-322.
23. O. Udrea, D. R. Recupero, and V. S. Subrahmanian, "Annotated RDF," *ACM Transactions on Computational Logic*, Vol. 11, 2010, pp. 10:1-10:41.
24. A. Pugliese, O. Udrea, and V. S. Subrahmanian, "Scaling RDF with time," in *Proceedings of the 17th International Conference on World Wide Web*, 2008, pp. 605-614.
25. B. McBride and M. Butler, "Representing and querying historical information in RDF with application to E-discovery," HP Laboratories Technical Report, HPL-2009-261, 2009.
26. Z. Y. Zheng and Y. Tan, "An indexed K-D tree for neighborhood generation in swarm robotics simulation," in *Proceedings of the 4th International Conference on Advances in Swarm Intelligence*, 2013, pp. 53-62.
27. http://swat.cse.lehigh.edu/projects/lubm.
28. R. Z. Ma, X. Y. Jia, J. W. Cheng, and R. A. Angryk, "SPARQL queries on RDF with fuzzy constraints and preferences," *Journal of Intelligent & Fuzzy Systems*, Vol. 30, 2016, pp. 183-195.
29. L. Yan, R. Z. Ma, D. Z. Li, and J. W. Cheng, "RDF approximate queries based on semantic similarity," *Computing*, Vol. 99, 2017, pp. 481-491.

Ping Zhao received her Bachelor degree in Computer Science and Technology from Jiangsu University, China. She is currently pursuing the M.S. degree in the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. Her research interests include RDF data and the Semantic Web.

Li Yan is a Full Professor at Nanjing University of Aeronautics and Astronautics, China. She received her Ph.D. degree from Northeastern University, China. Her research interests include databases, XML and the Semantic Web with a special focus on spatiotemporal information and uncertainty. She has published more than sixty papers on these topics. She is also the author of two monographs published by Springer.