Ontology-Based Semantic Search for Large-Scale RDF Data

Xiaolong Tang[1,2], Xin Wang[1,2,⋆], Zhiyong Feng[1,2], and Longxiang Jiang[1,2]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
[2] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
{txl1290,lxjiang2012}@gmail.com, {wangx,zyfeng}@tju.edu.cn

Abstract. In recent years, the Web of Data has emerged with the release of growing amount of Linked Data. Since traditional Information Retrieval (IR) technologies are no longer suit for the retrieval on Linked Data, it becomes difficult for ordinary users to retrieve the data efficiently and accurately. This paper presents a method of doing keyword search on Web of Data. We propose two distributed inverted index schemes, one of which is built from Linked Data and the other from the ontology. And as a necessary part of the ontology index, an ontology encoding scheme is also proposed. Based on the index schemes, we design an improved ranking algorithm named OntRank by introducing a semantic factor into the BM25F ranking model. The experimental evaluation illustrates the efficiency of constructing indexes and the precision of retrieval results.

Keywords: Linked Data, keyword search, distributed inverted index.

1 Introduction

The Resource Description Framework (RDF)[1] is a standard data model for describing the Semantic Web whose goal is making the information machine-readable. Linked Data[3] refers to graph-structured data that encoded in RDF and accessible via Hypertext Transfer Protocol (HTTP). In recent years, with the development of Linked Data, the researchers focus on constructing semantic search engines to access more accurate knowledge by taking advantage of the graph structure of Linked Data. However, the realization of the semantic search engine implies two major challenges: the system must scale to large amount of data and the search on Linked Data must be efficient and accurate.

A number of semantic search engines have been developed in past few years. But our survey on these semantic search engines reveals that although they improve the traditional IR technologies, a lot of vulnerabilities lie in these systems. For example, some of them[2,4,5] are not suitable for naive users who are not necessarily proficient in the semantic data and the structured query language. The others[9,13,14,8,10] have considered the importance of user-friendly, but the accuracy of their query results is at a low-level.

⋆ Corresponding author.

J. Wang et al. (Eds.): WAIM 2013, LNCS 7923, pp. 570–582, 2013.

In order to address aforementioned issues, we present a distributed semantic keyword search approach, which can provide an efficient and accurate search on Linked Data. Our work can be regarded as an extension to the traditional IR probabilistic retrieval model. More specifically, the main contributions of this paper include:

- A distributed inverted index scheme is proposed for Linked Data. And based on an ontology encoding scheme that we present in this paper, we design a specialized inverted index scheme for the ontology.
- We devise a new ranking algorithm, called OntRank, which is designed to improve the traditional IR ranking algorithm by introducing a semantic factor into the ranking function.
- We perform comprehensive experiments to demonstrate the performances of our index schemes and the accuracy of the OntRank algorithm.

The rest of the paper is organized as follows: Section 2 reviews related work. Section 3 describes the distributed inverted index schemes which are built from the RDF data and the ontology. The OntRank algorithm, which contains the semantic factor, will be proposed in Section 4. Section 5 reports the experimental results. In Section 6, we conclude and outlook the future work.

2 Related Work

Since Semantic Web develops rapidly in recent years, users look forward to making full use of the large amount of RDF data. For this purpose, lots of semantic search engines have been developed. Most of them are constructed based on the RDF query language, such as[2,4]. The characteristic of these search engines is that they provide a very complex structured query language, such as SPARQL[6] and SeRQL[11], to support RDF data accessing. However, they do not support keyword search on RDF data, so in order to access RDF data with these search engines, end users have to be quite familiar with the structure of RDF data and the syntax of structured query language.

To satisfy the requirements of ordinary end users, some keyword-based semantic search engines have been built. Currently, there are two main approaches on doing keyword search on RDF data: one of which uses the traditional IR technologies, the other one converts the keyword query into the structured query language. In the first approach, Semplore[8], K-Search[9] and Swoogle[10] combine the keyword retrieval with graph-structured RDF data, then compute a ranked list for the semantic documents. SWSE[13] is also a semantic keyword search engine. The main difference between SWSE and the aforementioned search engines is that, it provides an entity-centric search on RDF data. But all of them only adopt the traditional IR technologies to build the inverted index from RDF data, and they do not consider the semantic that exists in the keyword queries. SemSearch[14] is an example of the second approach, it transforms a keyword search into a structured SeRQL[11] query based on finding the matches of the two parts of the keyword search. However, this method may lose a lot of necessary matches when the length of the keywords is more than two.

Our work is developed based on Jingwei system[15] and it is quite different from the aforementioned search engines. First, our inverted indexes are built both from RDF data and the ontology. Second, our method can recognize the semantics in the queries. In our system, the keyword query contains two parts, the main keywords and the auxiliary keywords, such as *red apple@banana*. We get the semantics by checking the relations between the two kinds of keywords. Third, our approach can get more accurate results for the keyword queries according to the semantics.

3 Distributed Index Scheme

In this section, we will introduce two distributed inverted index schemes in our system, one of which is built from Linked Data, the other one is from the ontology. And we refer to the inverted index constructed on the assertional part of Linked Data as *A-index*, and the inverted index on the terminological part as *T-index*.

3.1 Distributed A-Index

Unlike traditional unstructured data, RDF data has no clear boundaries on documents. However, documents are the essential elements for inverted indexes, so we first give the definition of *RDF documents*.

Definition 1. *Assume that there are two disjoint infinite sets U and L, where U is the set of RDF URIs and L is the set of RDF literals. An RDF triple is a tuple of the form*

$$(s,p,o) \in U \times U \times U \cup L$$

In this tuple, s is called the subject, p the predicate, and o the object.

Definition 2. *An RDF graph is a finite set of RDF triples. Let G be an RDF graph. 1) A subgraph of G is a subset of G. 2) subj(G) denotes the set of all subjects in G.*

Definition 3. *Given an RDF graph G and a subject $s \in subj(G)$, an RDF document for s is a subgraph D_s of G where $subj(D_s)=\{s\}$.*

In Definition 3, we define the *RDF documents* as the collection of triples with the same *subject*, and the documents ID is *s*. From the traditional IR viewpoint, the documents have several fields, such as title, abstract and text, and each of them has different weights. Similarly, *RDF documents* also have the concept of fields, which is divided based on the meaning of *p*. Since BM25F[16], the variant of BM25, has considered the influence of fields in the ranking algorithm, we decide to devise our *A-index* on top of BM25F. To satisfy the BM25F ranking algorithm, we have divided RDF data into 4 fields, i.e. *label, comment, type and others*, and the information of the fields will be stored in *A-index*. Figure 1 shows

Algorithm 1. Construction of *A-index*

Input: RDF Data
Output: *A-index*

```
1: map((s,p,o)):
2: f = {label, comment, type, others}
3: if o ∈ L then
4:    if filed(p) == f then
5:       for each term ∈ o do
6:          EMIT((term,s),(f,1))
7:       end for
8:    end if
9: end if
10: reduce((term,s),iterator values):
11: for value in values do
12:    field = value.getLeft()
13:    frequency = field.getSum()
14:    hashtable.put(field,frequency)
15: end for
16: set A-index { RK ← term, CN ← s, CV ← hashtable }
```

the structure of *A-index*, which is a 3-layer key-value structure, and the name of them are the row key (RK), the column name (CN), the column value (CV). In the design of *A-index*, we consider both efficiency and scalability, thus the MapReduce framework is adopted to build the index on top of Cassandra that is a distributed key-value NoSQL database. The constructing process of *A-index* is shown in Algorithm 1.

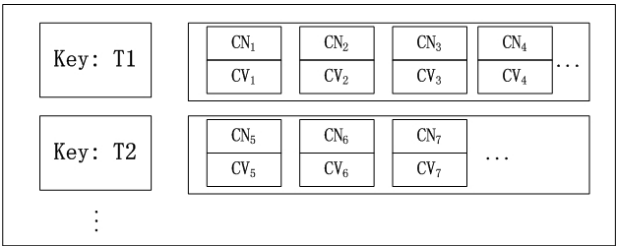

Fig. 1. The structure of A-index

In the map phase, we only process the triple (s,p,o) whose o is in L. In lines 3-9, we get the field ID and the document ID, for each term, and then emit them to the reduce function. And in the reduce phase, we first receive the term and document ID from map. Then in lines 11-15, we use a hash table to collect the detailed term frequency information, the key of the hash table is the field ID and the value is the corresponding term frequency. Line 16 inserts the term, the document ID and the hash table into the *A-index*.

3.2 Distributed T-Index

Although the *A-index* can accomplish the mission of doing keyword search on RDF data, the semantics in RDF data is not exploited. Thus, we build the *T-index* as a supplement of the *A-index* by indexing the ontology. And in order to use the ontology conveniently, an ontology encoding scheme is also proposed. This section will introduce the expression of the ontology and the construction of the *T-index* in detail.

3.2.1 Ontology Encoding

As we know, the ontology contains abundant semantics, which can be applied to express relations of entities. However, recognizing relations of entities is quite difficult if we directly store and use the ontology. In addition, the ontology is usually updated to enrich its semantics, so the management of the ontology will have a high cost. ORDPATH[7] is a hierarchical coding scheme whose benefits are two-fold. The first one is that the ORDPATH scheme allows inserting new nodes at arbitrary positions in the tree structure without relabeling any old nodes, so the update costs of ORDPATH will be at a low-level. The other one is that the code of ORDPATH can be compressed into a binary form which is easy for comparing ORDPATH values. In this way, we can recognize whether two nodes are child-parent nodes or sibling nodes by only comparing their binary code length. Obviously, ORDPATH is appropriate to ontology encoding, and the application of ORDPATH can represent the subsumption relationship appropriately. Thus, we adopt ORDPATH as the ontology encoding scheme.

Example 1. In Fig. 2, G is a part of the tree structure of entity classes which encoded with ORDPATHs. We can see that there are five layers from the root node *owl:Thing* to the leaf node *Pine* and the corresponding ORDPATHs code is under them. It is easy to identify the subsumption relationship of entity classes based on the comparison of ORDPATH. The code of *Plant* is 1.25.3, and we can infer that its ancient nodes is encoded with 1, 1.25, its sibling node is 1.25.5

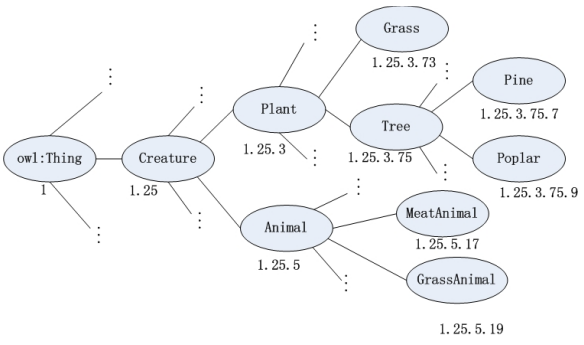

Fig. 2. The inheritance relationship of entity classes encoded by ORDPATHs

and one of its children nodes is 1.25.3.75, the corresponding entity classes are *owl:Thing, Creature, Animal* and *Tree*. According to the insertion strategy of ORDPATH, we can insert new nodes 1.25.1 to the left of *Plant* and 1.25.7 to the right of *Animal*. In addition, we can also insert the nodes at arbitrary positions, such as the position between *Plant* and *Animal*, with an even integer. 1.25.4.1 is an example of sibling nodes between *Plant* and *Animal*, and we can see that the even integers do not count for ancestry: 1.25.4.1 is a child of 1.25. This property shows that ORDPATHs is scalable and the old nodes need not to be relabeled when we update ORDPATHs, so it has a high update performance.

3.2.2 T-index Architecture

Definition 4. *Given a set of all terms T, an RDF document D and an entity class C. t is an element of T, if the entity class of subj(D) is C and t is in D, then we call that t relates to the entity class C, which is expressed as* $t \sim C$*, and we use class(t) to express the set of entity classes that relate to t. For* T_i *that is a subset of T, if* $t \sim C$ *and* $t \in T_i$*, we call* $T_i \sim C$*. We define CLASS(*T_i*) as the set of all the entity classes that relate to* t_i *in* T_i*, and CLASS(*T_i*)* $= \bigcup_{t_i \in T_i} class(t_i)$.

Since each s has at least one entity class and the relation between the entity classes can describe the relation between the corresponding subjects, we consider that a similar correlation exists between the terms of RDF documents and entity classes. Thus, we decide to construct the *T-index* according to the relation between terms and entity classes, which is defined in Definition 4. The index structure and the method for creating the *T-index* are like what we discussed in section 3.1. But a little difference between them is that the construction of the *T-index* has two mappers and two reducers. Algorithm 2 is the pseudo-code of building the *T-index*. The inputs are two files in DBpedia, where the o in the *instance_type* is the type of the s and the o in the *short_abstract* is the abstract of the s.

In the map phase of the first MapReduce, lines 2-7 identify which file the triple (s,p,o) belongs to, and then emit the s, o and an identifier to the reducer. Then in the corresponding reduce phase, lines 10-13 get the most accurate ORDPATH of the entity classes of the s, when o is the type of s. After that, we emit the ORDPATHs and the abstracts of the same s as an intermediate file, which is the input of the second MapReduce job. And in the second MapReduce job, we first emit each term in the abstracts and the corresponding ORDPATHs in lines 20-22. Then lines 24-26 make a statistic on $class(t)$ for each term t, which contains the ORDPATHs and the corresponding weights. Finally, we insert the terms, the ORDPATHs and the weights into the *T-index*.

Algorithm 2. Construction of *T-index*

Input: *instance_type, short_abstract*
Output: *T-index*

```
first_map((s,p,o)):
if (s,p,o)∈ instance_type then
    EMIT(s,(o,type))
end if
if (s,p,o)∈ short_abstract then
    EMIT(s,(o,term))
end if
first_reduce(s,iterator values):
for value in values do
    if value.getRight() == type then
        subj_type = value.getLeft()
        ordpaths = subj_type.getOrdpaths()
    end if
    if value.getRight() == term then
        text == value.getLeft()
    end if
end for
EMIT(ordpaths,text)
second_map(ordpaths,text):
for each term ∈ text do
    EMIT(term,(ordpaths,1))
end for
second_reduce(term,iterator values):
for value in values do
    ordpaths = value.getLeft()
    calculate weight w of ordpaths
    set T-index { RK ← term, CN ← ordpaths, CV ← w }
end for
```

4 Semantic Ranking Algorithm

Based on the indexes built in section 3, we propose an improved ranking algorithm, called OntRank, which take advantage of the relation between the data to improve the effectiveness of document ranking.

4.1 RO Calculation

Definition 5. *Given a set of entity classes C, the entity classes C_0, C_1 and $C_2 \in C$, and C_0 is the minimum common parent class of C_1 and C_2. If the main keywords $T_1 \sim C_1$ and the auxiliary keywords $T_2 \sim C_2$, the distance between C_1 and C_2 through C_0 is called RO(Rank Order), which reveal the relevance of T_1 and T_2, and expressed as RO(C_1, C_2). RO(T_1, T_2) is a set of RO(C_i, C_j) that $C_i \in CLASS(T_1)$ and $C_j \in CLASS(T_2)$.*

Algorithm 3. Calculating $RO(T_1, T_2)$

Input: the main keywords T_1, the auxiliary keywords T_2
Output: rank order $RO(T_1, T_2)$

```
1: setofOrdpathA = T1.getOrdpaths();
2: setofOrdpathB = T2.getTopKOrdpaths();
3: for each ordpathsA in setofOrdpathA do
4:    for each ordpathsB in setofOrdpathB do
5:       ordpathsC = getMiniCommonOrdpaths(ordpathsA, ordpathsB);
6:       RO(A,B) = |ordpaths A - ordpaths C| + |ordpahts B - ordpahts C|;
7:       if RO(A,*) not in RO(T1, T2) then
8:          add RO(A,B) to RO(T1, T2)
9:       end if
10:      if RO(A,*) in RO(T1, T2), but RO(A,B) < RO(A,*) then
11:         modify RO(A,*) to RO(A,B)
12:      end if
13:   end for
14: end for
15: return RO(T1, T2)
```

In Definition 5, the main keywords and the auxiliary keywords are the two parts of our keyword query introduced in section 2, and both of them are sets of terms, so they have the characteristics of Definition 4. Since the semantics in the queries is essential to our ranking algorithm, we define RO as the semantic factor which is calculated by the correlation of entity classes related to the keywords. The corresponding pseudo-code of calculating $RO(T_1, T_2)$ is shown in Algorithm 3.

In Algorithm 3, we obtain the corresponding ORDPATHs of $CLASS(T_1)$ in line 1. To avoid too many classes related to the auxiliary keywords that may influence the query accuracy, line 2 gets the top-k correlative ORDPATHs based on the weights of ORDPATHs. Lines 3-14 calculate RO. Line 5 gets the minimum parent class of the chosen classes, such as A and B. Then in line 6, we calculate the minimum path length between A and B. In case 1, the hash table $RO(T_1, T_2)$ do not have the element of $RO(A,*)$, where * can be any ORDPATHs of *setofOrdpathB*, then we add $RO(A,B)$ into the hash table as $RO(A,*)$. In lines 10-12, although $RO(T_1, T_2)$ have the element of $RO(A,*)$, the minimum path length between A and B is less than $RO(A,*)$, we update $RO(A,*)$ with $RO(A,B)$.

Obviously, the time complexity of Algorithm 3 is $O(|M| \times |N|)$, that $|M|$ is the number of $CLASS(T_1)$ and $|N|$ is the number of top-k $CLASS(T_2)$. The main space cost is the storage of entity classes which related to keywords. Hence, the overall space complexity is $O(|M| + |N|)$.

Example 2. G(Fig.2) in section 3.2.1 is an example of entity classes which encoded with ORDPATHs. We suppose that $CLASS(T_1)$ is $\{Poplar, Grass\}$ and $CLASS(T_2)$ is $\{Pine\}$. First, we get the shortest path between *Poplar* and *Pine*, which is *Poplar-Tree-Pine*, so $RO(Poplar,Pine) = 2$, then we add it to $RO(T_1,T_2)$. Similarly, we can also conclude $RO(Grass,Pine) = 3$ when $T_1 \sim Grass$ and $T_2 \sim Pine$. In this condition, $RO(T_1,T_2) = \{(1.25.3.75.9, 2), (1.25.3.73, 3)\}$.

4.2 OntRank Algorithm

Algorithm OntRank uses RO as the semantic factor to improve the effectiveness of the BM25F ranking function. In this section, a detailed description will be presented, which is about calculating the score of documents.

First of all, the BM25F ranking function should set several boost factors before calculating the score of documents. In our system, the boost factors are: $label = 30.0$, $comment = 5.0$, $type = 10.0$, $others = 1.0$. Moreover, BM25F also requires additional parameters, whose value is $K_1 = 4.9$, $b_{label} = 0.6$, $b_{comment} = 0.5$, $b_{type} = 0.5$ and $b_{others} = 0.4$. And in this paper, we use $score^{BM25F}(Q, D)$ to express the score of the query which based on the BM25F ranking algorithm. So the score of the OntRank ranking algorithm is calculated by

$$score^{OntRank}(Q, D) = score^{BM25F}(Q, D) \times (1 + \tfrac{1}{RO+1})$$

where RO is the concept defined in section 4.1.

In our method, RO is a list of non-negative integers, and with the value of RO decreasing, the correlation between the main keywords and the auxiliary keywords becomes higher. Here, we use $RO + 1$ to avoid the condition that $RO = 0$. Moreover, it can be seen that $\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4} \cdots\}$ are the values of different $\frac{1}{RO+1}$, the differences between neighboring values are $\{\frac{1}{2}, \frac{1}{6}, \frac{1}{12} \cdots\}$. It is obvious that the value of RO improves more significantly when the relationship between the two kinds of keywords is getting closer. Meanwhile, the class of a returned document must be one of the classes related to the main keywords. It means that RO is also the relationship between returned documents and the auxiliary keywords. Thus, the score of documents can be distinguished clearly according to the different relevance levels of the documents and the auxiliary keywords.

5 Evaluation

In this section, we describe our evaluation method on the proposed indexes and algorithms, and show the experimental results. Since the solutions mentioned in the INEX 2009 is designed for semi-structured data and up to now we have not found a open source semantic search engine for RDF data, we only compare our method with the BM25F model.

5.1 Setting

All experiments are carried out by using a 4-nodes cluster as the underlying storage system. Each node has Intel Q8400 CPU and 8 GB memory. The operating system is Ubuntu 10.04. The database is Cassandra 0.8.1, and the Cassandra heap was set to 4GB. We use Apache Tomcat 7.0 as the Web server.

5.2 Dataset and Queries

For the evaluation we use the DBpedia 3.7 dataset excluding pagelinks; each characteristic of the DBpedia dataset is listed in Table 1. Although we only use

the DBpedia, our method is not optimized for the dataset. In other words, our method is general enough to be used for other datasets.

Table 1. Dataset characteristics

name	value
distinct triples	43,853,549
distinct subjects	8,870,118
distinct predicates	1,299
distinct objects	18,286,543

For the reasons that a) we adopt the INEX evaluation framework to provide a mean for evaluating the semantic keyword search approach b) Wikipedia is the source of the INEX collection. We have to preprocess the dataset that is build from the intersection of DBpedia and the INEX-Wikipedia collection[12]. The result is listed in Table 2.

Table 2. Dataset preprocessing result

dataset name	entity number
DBpedia	364,5384
Wikipedia	266,6190
Intersection(DBpedia, Wikipedia)	244,9229

The query set in our evaluation framework is taken from the INEX 2009 contest. INEX2009 offers the judgments of 68 queries. Each query consists of three parts with the same topic: title, description and narrative. In our experiments we adjust the titles by adding the semantics of descriptions and narratives into them. In this way the query will be more expressive and fit better to the semantic search. The average length of the queries is around 4 words. However, not all the semantics of queries suit for the OntRank algorithm, we take 50 of them as the final query set.

We take the IR metrics as the evaluation criterion. These metrics are generally oriented in two main directions: the ability to retrieve relevant documents and the ability to rank them properly. In our experiment, we use part of them: Mean Average Precision (MAP), which provides a single-figure measure of quality across recall levels; Geometric Mean Average Precision (GMAP) that is a variant of MAP by using a geometric mean; Precision after K documents (P@K) which measures the precision after K documents have been retrieved; and R-Precision that measures the precision after R documents have been retrieved, where R is the total number of relevant documents for a query.

5.3 Results: Indexing Cost

We performed an experiment on measuring the indexing cost with the increase of the number of nodes. The results are shown in Figure 3. Note that the indexing cost of *T-index* is the total executing time of two MapReduce jobs. From

the results, we observe that the time of constructing index will decrease by increasing the number of nodes. However, because of the information exchange between nodes, the downward trend of executing time are not linear. In addition, the scalability of our system is also reflected in the results, the problem of the explosive growth of Linked Data can be resolved merely by increasing the computer numbers.

5.4 Results: Query Accuracy

The query accuracy is measured by using a tool named trec_eval[17]. This tool is implemented for producing the IR metrics. Figure 4 shows the evaluation results obtained by two different methods, one is with OntRank and the other is only use the BM25F ranking function.

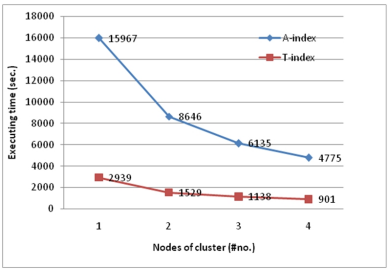

Fig. 3. System indexing cost

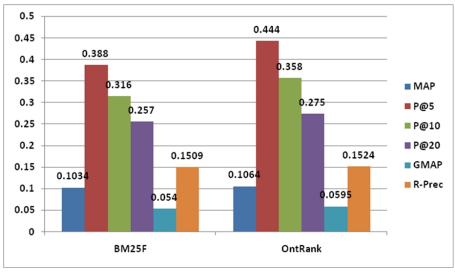

Fig. 4. IR metrics of BM25F and OntRank

The results draw from the experiments show that the OntRank ranking algorithm is better than BM25F in every single measure, especially P@K. This is not surprising since we add the semantics into the queries, so the search system can return more relevant documents after K documents have been retrieved. We can see that OntRank obtains the best performance, although some of the IR criteria do not improve significantly, such as MAP, GMAP and R-Prec. We have analyzed the reason for it, and find that there are many factors restricting the improvement of the results:

1. The relevant documents of some queries is about concepts, such as theories or algorithms, and their entity class is owl:Thing, so we can not get the corresponding auxiliary keywords, and the OntRank ranking algorithm remains ineffective.
2. Some entities do not have abstract, so some relevant documents may not be found, the OntRank ranking algorithm has no effect on them.
3. There are some mistakes in the classification of DBpedia, for example, the entity of Pavarotti belongs to dbpedia:Person in DBpedia, but its proper class is dbpedia:Musicalartist. And what is worse, this phenomenon is prevalent in the querying process. Hence, these mistakes will have a significant influence on the *RO* value which is the main impact factor of the IR metrics.

Fortunately, all of them are due to the quality of the dataset. So we can believe that with the improvement of the quality of the dataset, the IR evaluation criterion will improve more significantly.

6 Conclusion and Future Work

In this paper, we study the problem of doing keyword search on large-scale RDF data, and propose an effective approach to get more accurate query results. To support keyword search, we use MapReduce to build the inverted indexes both from Linked Data and the ontology. And in order to build the *T-index*, we design an ontology encoding scheme. Based on the indexes, we present a new ranking algorithm by adding the semantic factor, which is calculated by the relation between the main keywords and the auxiliary keywords, into the BM25F ranking function. And through the experimental results, we confirm the efficiency and the accuracy of our approach. In the future, we will keep on optimizing the OntRank algorithm. Moreover, we will seek a more granular knowledge base, which has a more accurate classification on entities, to compensate for the deficiencies of the existing DBpedia ontology data.

Acknowledgements. This work was supported by the National Natural Science Foundation of China (Grant No. 61100049, 61070202) and the National High-tech R&D Program of China (863 Program) (Grant No. 2013AA013204).

References

1. Klyne, G., Carroll, J., McBride, B.: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation. W3C (2004)
2. Corby, O., Dieng-Kuntz, R., Faron-Zucker, C.: Querying the Semantic Web with Corese Search Engine. In: Proceedings of 15th ECAI/PAIS, Valencia, ES (2004)
3. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
4. Shadbolt, N., Gibbins, N., Glaser, H., et al.: Cs Aktive Space or how we learned to stop worrying and love the Semantic Web. IEEE Intelligent Systems, 41–47 (2004)
5. Zhang, L., Yu, Y., Zhou, J., Lin, C., Yang, Y.: An Enhanced Model for Searching in Semantic Portals. In: WWW (2005)
6. SPARQL 1.1, `http://www.w3.org/TR/2012/PR-sparql11-overview-20121108/`
7. O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G., Westbury, N.: ORDPATHs: Insert-Friendly XML Node Labels. In: Proceedings of ACM Conference on Management of Data (SIGMOD), pp. 903–908 (2004)
8. Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., Pan, Y.: Semplore: A Scalable IR Approach to Search the Web of Data. Web Semantics: Science, Services and Agents on the World Wide Web 7(3), 177–188 (2009)
9. Bhagdev, R., Chapman, S., Ciravegna, F., Lanfranchi, V., Petrelli, D.: Hybrid Search: Eectively Combining Keywords and Semantic Searches. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 554–568. Springer, Heidelberg (2008)

10. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: A Search and Metadata Engine for the Semantic Web. In: Proc. of the 13th ACM CIKM Conf. ACM Press, New York (2004)
11. SeRQL, http://www.w3.org/2001/sw/wiki/SeRQL
12. Perez-Aguera, J.R., Arroyo, J., Greenberg, J., et al.: INEX+DBpedia: A Corpus for Semantic Search Evaluation. In: WWW, pp. 1161–1162 (2010)
13. Hogan, A., Harth, A., Umbrich, J., et al.: Searching and Browsing Linked Data with SWSE: The Semantic Web Search Engine. Web Semantics: Science, Services and Agents on the World Wide Web, 365–401 (2011)
14. Lei, Y., Uren, V., Motta, E.: SemSearch: A Search Engine for the Semantic Web. In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS (LNAI), vol. 4248, pp. 238–245. Springer, Heidelberg (2006)
15. Wang, X., Jiang, L.: Jingwei+ A Distributed Large-scale RDF Data Server. In: Proceedings of Asia-Pacific Web Conference (2011)
16. Stephen, R., Hugo, Z.: The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval, 333–389 (2009)
17. Trec_eval, http://trec.nist.gov/treceval/