

Optimization of RDF link traversal based query execution

Zhu Yanqin

Hua Ling

(School of Computer Science and Technology, Soochow University, Suzhou 215006, China)

Abstract: Aiming at the problem that only some types of SPARQL (simple protocol and resource description framework query language) queries can be answered by using the current resource description framework link traversal based query execution (RDF-LTE) approach, this paper discusses how the execution order of the triple pattern affects the query results and cost based on concrete SPARQL queries, and analyzes two properties of the web of linked data, missing backward links and missing contingency solution. Then three heuristic principles for logic query plan optimization, namely, the filtered basic graph pattern (FBGP) principle, the triple pattern chain principle and the seed URIs principle, are proposed. The three principles contribute to decrease the intermediate solutions and increase the types of queries that can be answered. The effectiveness and feasibility of the proposed approach is evaluated. The experimental results show that more query results can be returned with less cost, thus enabling users to develop the full potential of the web of linked data.

Key words: web of linked data; resource description framework link traversal based query execution (RDF-LTE); SPARQL query; query optimization

doi: 10.3969/j.issn.1003-7985.2013.01.006

In recent years, the web has evolved from a global information space of linked documents to one where both documents and data are linked. Underpinning this evolution is a set of best practices for publishing and connecting structured data on the web known as linked data^[1]. Nowadays, an increasing amount of data published on the web according to the linked data principles leads to the extension of the web. Because of its openness, query processing over the web of linked data meets new challenges.

There are several strategies for linked data query processing such as data warehousing, search engines, the query federation approach, etc.^[2-6]. The approaches described above need to know all relevant data in advance or restrict

themselves just to the selected sources, and do not tap the full potential of web. Hartig et al.^[7] proposed a new query execution paradigm for the web of data, namely RDF link traversal-based query execution (RDF-LTE). The main idea of RDF-LTE is to use the RDF links that exist between different data sources. It can discover potential relevant data sources by mapping the intermediate solutions during the query execution. In Ref. [7], Hartig et al. proposed an iterator-based implementation approach, which applies a synchronized pipeline of operators that allows it to temporarily reject certain input results.

However, because of the execution of fixed orders of the triple pattern in the iterator-based pipeline approach, RDF-LTE only can answer some types of SPARQL queries. Some execution orders can return results and others return no results. The execution order of the triple pattern affects the query cost and result completeness. To solve these problems, this paper focuses on query plan optimization, so that suitable execution orders for the triple patterns can be chosen.

This paper first gives a formal definition of RDF-LTE and analyzes the limitations of the approaches based on RDF-LTE. Then three principles for RDF-LTE query plan selection are proposed. Finally, we show our experimental results.

1 Formal Definition of RDF-LTE

Linked data sources use the RDF in various serialization syntaxes for encoding graph-structured data^[8]. Let U denote a set of URIs, B a set of blank nodes and L a set of literals. An RDF triple t is a 3-tuple: $t = (s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$, where s is called subject of t ; p is the predicate and o is the object. Sets of RDF triples are called RDF graphs. The notion of graph stems from the fact that RDF triples may be viewed as labeled edges connecting subjects to objects. The elements of $(U \cup B \cup L)$ are called RDF terms.

In the web of data, each entity has to be identified via the HTTP schema based URI. Let $U_{\text{resource}}: U_{\text{resource}} \subset U$ be the set of all these URIs. By looking up such a URI, we can retrieve RDF data about the entity which is identified by the URI. We formally identify the retrieve process as a function denoted as dereference. Dereference can be seen as a surjective function which returns an RDF object for each $u \in U_{\text{resource}}$; that is a set of RDF triples which can be retrieved by looking up such URI and describe the entity or resource identified by u . We denote

Received 2012-11-02.

Biography: Zhu Yanqin (1964—), female, doctor, professor, yqzhu@suda.edu.cn.

Foundation items: The National Natural Science Foundation of China (No. 61070170), the Natural Science Foundation of Higher Education Institutions of Jiangsu Province (No. 11KJB520017), Suzhou Application Foundation Research Project (No. SYG201238).

Citation: Zhu Yanqin, Hua Ling. Optimization of RDF link traversal based query execution [J]. Journal of Southeast University (English Edition), 2013, 29(1): 27 – 32. [doi: 10.3969/j.issn.1003-7985.2013.01.006]

$$\forall u \in U_{\text{Resource}}, \exists (s, p, o) \in \text{dereference}(u): s = u \vee p = u \vee o = u$$

According to the linked data principles^[1], dereference is not an injective. It is possible that the same RDF object may be retrieved by dereferencing different URIs. Besides, for each $u \notin U_{\text{Resource}}$, dereference u will return a NULL RDF object: $\text{dereference}(u) = \emptyset$.

SPARQL is the standard language for querying RDF data. Important parts of SPARQL queries are basic graph patterns (BGPs) and sub-graph matching. In this paper we are concerned with answering BGP queries.

A BGP is a set of triple patterns (s, p, o) where every s, p and o is either a variable or a constant, that is an infinite subset of set $(U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$ where V is an infinite set of queries variables, distinct from U, B and L . The elements of a BGP are called triple pattern. For each triple pattern tp we use $\text{uris}(tp)$ and $\text{vars}(tp)$ to denote the set of all URIs and query variables contained in the tp . A matching triple in RDF object D' for a triple pattern $(?s, ?p, ?o)$ is any RDF triple $(s, p, o) \in G$ with

$$(?s \notin V \rightarrow ?s = s) \wedge (?p \notin V \rightarrow ?p = p) \wedge (?o \notin V \rightarrow ?o = o)$$

2 Analysis of RDF-LTE: Query Cost and Result

In Ref. [7], Hartig et al. introduced the idea of the RDF-LTE using an iterator-based implementation. Here we discuss the query cost and result completeness of RDF-LTE.

2.1 Iterator-based pipeline implementation

In the semantic web client library, RDF-LTE is implemented using a synchronized pipeline of operators. The pipeline is implemented as a chain of iterators I_1, I_2, \dots, I_n where each iterator I_i is responsible for a triple pattern tp_i from BGP $b = \{tp_1, tp_2, \dots, tp_n\}$. Each iterator returns so-

lution mappings that are solutions for a BGP consisting of the triple pattern of the corresponding iterator and all the triple patterns of the proceeding iterators; i. e., I_i returns solution mapping for BGP $\{tp_1, tp_2, \dots, tp_i\}$. To determine these solutions, each iterator executes the following steps recursively:

1) Each iterator I_i obtains a solution mapping from μ' of its predecessor and applies the solution mapping to its triple pattern tp_i , resulting in a triple pattern $tp'_i = \mu'[tp_i]$.

2) If the query-local dataset does not contain the RDF object that can be retrieved from dereferencing all the URIs in tp'_i , it is added to the query-local dataset.

3) The iterator tries to generate (intermediate) solutions by finding matching triples for tp'_i in the query-local dataset.

An iterator is a group of three functions, namely Open, GetNext and Close. Open initializes the data structures needed to perform the operation. GetNext returns the next result of the operation. Close ends its iteration and releases allocated resources. The above three steps are corresponding to the GetNext function of the iterators.

2.2 Execution order and result completeness

Although the algorithm entails less execution time, it does not guarantee to return all the solutions for a SPARQL query. In this section we discuss the reasons for this limitation based on a concrete SPARQL query.

Example 1 SPARQL query Q1

```
?x rdf: type <http://.../A> ..... tp1
?x num1: predicate1 ?y ..... tp2
?y foaf: knows <http://www.abc.com/people/hua-ling> ..... tp3
?y num2: predicate2 ?z ..... tp4
```

The above SPARQL query Q2 is a user-submitted BGP query and Fig. 1 shows the RDF objects retrieved during query processing.

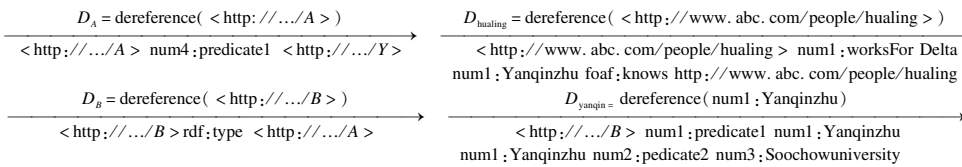


Fig. 1 RDF objects retrieved during query processing

To execute the BGP query in Fig. 1, RDF-LTE begins by dereferencing the URI $\langle \text{http://.../A} \rangle$ and retrieving its RDF objects. Formally, I_1 dereferences the seed URI and ensures that the query-local dataset contains the RDF object $D_A = \text{dereference}(\langle \text{http://.../A} \rangle)$. I_2 requests the intermediate solution from its predecessor I_1 . But to create the intermediate solution, I_1 tries to find the matching triple for tp_1 ; unfortunately, the query-local dataset D_A does not contain any matching triples (see Fig. 1). Hence, I_1 cannot provide any intermediate solution and thus the final query result is empty. Even if we initialize

the empty query-local dataset with all the reachable RDF objects by dereferencing all the URIs in the query, i. e. $D_A = \text{dereference}(\langle \text{http://.../A} \rangle)$ and $D_{\text{hualing}} = \text{dereference}(\langle \text{http://www.abc.com/people/hualing} \rangle)$, we can not find a matching triple for triple pattern tp_1 . And the result of the query is also empty.

However, if we execute tp_3 first and tp_1 finally, this will result in one solution for the BGP query:

$$\mu = \{ (?x, \text{http://.../B}), (?y, \text{num1: Yanqinzhu}), (?z, \text{num3: Soochowuniversity}) \}$$

From the query process of Q1, we can find that for different execution orders of triple patterns in the BGP, the RDF-LTE may return different results. We will analyze the reasons in the following.

No backward links: In web of linked data, an RDF triple in the form of (uri_s, uri_p, uri_o) contained in the RDF object dereference(uri_s) (or dereference(uri_o)) may not be contained in the RDF object dereference(uri_o) (or dereference(uri_s)). We refer to the character of the web as no backward links. It is possible that one execution order for a query may discover a matching triple that another order misses. No backward links is one of the reasons for different results in Q1.

Contingency solution: For two triple patterns $tp_i, tp_j (i < j)$, each operator I_i retrieves the intermediate solution from their predecessor iterators, fetches the RDF objects for triple pattern matching and generates the intermediate solutions which will further be added to the query-local dataset. However, the RDF objects retrieved for the execution of the triple pattern tp_i may contain a matching triple t^* for triple pattern tp_j which will be executed later by any of the iterators. It is not guaranteed that the RDF objects with the triple t^* will also be discovered and retrieved during the execution of the triple pattern tp_j . Such a type of solution generated based on t^* is a contingency solution. Current RDF-LTE takes the existing order in the user query and t^* will only be discovered after tp_j has been executed, so these solutions can never be generated.

As can be seen from the analysis above, the result completeness of RDF-LTE relies on the execution order of the triple patterns in the BGP queries. Some execution orders can return results and others return no results. This implies that certain orders are more suitable than others. Even if all the orders cannot be guaranteed to return all solutions, one selection of specific orders will provide more solutions than others or take less query execution time. In the next section we will present several query plan selecting principles for query plan optimization.

3 Query Plan Optimization over the Web of Data

Logical query optimization is to use equalities of query expressions to transform a logical query plan into an equivalent query plan that is likely to be executed faster or with less costs. In this paper, we regard an ordered BGP as a logical query plan. So the creation of such an optimized plan is to choose a suitable execution order for the triple patterns in a given BGP. Since there are multiple orders for a given BGP, it is possible to create different plans.

3.1 Metrics of query plans

Different query plans for the same BGP have different characteristics, resulting in different execution performances. Usually traditional query optimizations use the

query cost to measure different query plans. However, in contrast to traditional query execution, RDF-LTE may result in different sets of solutions for different query plans for the same BGP query as we discussed in the previous section. So for RDF-LTE, we assess query plans not only based on their cost but also on their benefit.

Cost: The query cost of RDF-LTE can be measured in terms of query execution time, the number of URIs dereferenced or the overall RDF objects retrieved during query processing. In this paper, we use the query execution time to measure the query plan because it implicitly includes many other measures. For instance, the query execution time is dominated by delays resulting from the dereference of URIs. With the increase in the number of the RDF objects retrieved from the web, the time to execute the query may increase accordingly.

Benefit: RDF-LTE may return solution sets with different cardinalities as we have discussed above. So we should also consider the benefit of different query plans, that is, the number of solutions that an execution of a query plan returns.

3.2 Query plan optimization

Usually a function is used to estimate the cost and benefit of query plans, which need information about reachable RDF objects and the characters of data sources involved in the execution of the query. In the scenario of RDF-LTE we do not assume any such information, all we have is just a query and an empty query-local dataset. We do not know what RDF objects will be discovered during the query execution and what URIs will be dereferenced. Based on complete lack of information, we propose to select logic query plans based on the following three principles.

3.2.1 FBGP principle

In SPARQL queries, the basic graph patterns can be mixed with value constraints (FILTER) and other graph patterns. The execution of the basic graph patterns and value constraints is order independent; that means the structure of two basic graph patterns BGP_1 and BGP_2 separated by a constraint C can be transformed into one equivalent basic graph pattern followed by the constraint. We refer to a basic graph pattern followed by one or more constraints as a filtered basic graph pattern (FBGP).

FBGP principle includes the following steps:

- 1) If a BGP is FBGP, rules based on relational algebra are used to rewrite the original query before query execution, so that BGPs are merged and variables are replaced by constants from filter expressions. For instance, variables that occur in filters with an equal operator will be replaced by a given value; the filter expression that cannot be replaced will be moved to the end of the query.
- 2) The filter triple pattern should be placed close to the seed triple pattern as much as possible so that the number

of the intermediate solution is decreased.

An example is shown below. In an original SPARQL query submitted by the user, there are two separate BGPs, each with one triple pattern. After the above two steps, two separate BGPs are merged and variables are replaced in the rewritten query. In this example, ? name is replaced by “hualing”.

Example 2 Application of FBGP principle

Original SPARQL query:

```
SELECT ? mbox WHERE {
  { ? x foaf:name ? name. }
  FILTER ( ? name = "hualing" )
  && regex ( ? mbox, "hualing" )
  { ? x foaf:mbox ? mbox. }
}
```

Query after rewriting:

```
SELECT ? mbox WHERE {
  ? x foaf:name "hualing".
  ? x foaf:mbox ? mbox.
  FILTER ( ? mbox, "hualing" )
}
```

For an ordered BGP, after the transformation of step 1), the query variables in the filter triple pattern must be those that appeared in one preceding triple pattern. Formally, if a triple pattern tp_i in an ordered BGP $\bar{b} = (tp_1, tp_2, \dots, tp_n)$ is a filter triple pattern, it must hold $\forall v \in \text{vars}(tp_i) : (\exists j < i : v \in \text{vars}(tp_j))$. The proposal of step 2) is to reduce cost. Making the filter triple pattern as close as possible to the seed URI will decrease the number of the intermediate solutions.

3.2.2 Triple pattern chain principle

For an ordered BGP, the variables in each triple pattern should occur in at least one of the preceding triple patterns. Formally, for an ordered BGP $\bar{b} = (tp_1, tp_2, \dots, tp_n)$, if and only if it holds for each $i \in \{2, 3, \dots, n\}$, $\exists v \in \text{vars}(tp_i) : (\exists j < i : \exists v \in \text{vars}(tp_j))$, we regard it as a triple pattern chain. The triple pattern chain principle enables each iterator I_j to always reuse some of the bindings in the intermediate solutions obtained from their predecessors I_i .

Definition 1 Connected graph pattern (CBGP): a BGP $b = \{tp_1, tp_2, \dots, tp_n\}$ is a CBGP if it holds $\forall b_1, b_2 \in b : b_1 \cup b_2 = b \wedge (\exists tp_i \in b_1, tp_j \in b_2 : \text{vars}(tp_i) \cap \text{vars}(tp_j) \neq \emptyset)$.

For a CBGP, we can always find a triple pattern order that satisfies the principle triple pattern chain. So we will use them for triple pattern matching and guarantee that it will never return empty results.

3.2.3 Seed URIs principle

This principle requires that the first triple in the ordered BGP should be a candidate seed triple pattern. We regard the triple pattern which contains at least one HTTP URI as the candidate seed triple pattern. For a BGP query, it may contain several candidate triple patterns. The candi-

date triple pattern we select as the first triple pattern is called the seed triple pattern.

For a new SPARQL query, the seed URIs principle ensures that when the query execution begins with an empty query-local dataset, the URIs in the seed triple pattern can be dereferenced to serve as a start to find matching triples. By finding the matching triples we can construct intermediate solutions and dereference them to augment the query-local dataset. Therefore, it is reasonable to select a candidate seed triple pattern as the first triple pattern in the query plan.

4 Experimental Evaluation

All the experiments are made on a Pentium 4 machine based on the semantic web client library. For query plan optimization, we use two synthetic BGP queries, neither of them can be answered using data from a single data provider alone. So they can be seen as being executed on the web of linked data. For each query, we generate all the query plans that satisfy the proposed principles. To evaluate the effectiveness of the proposed principles, we consider the following several metrics: query execution time, the number of results returned, and the location of the filter triple pattern. To avoid the effect of the cold start^[9] and cache^[10], we run each plan 6 times where the first time is not considered for measurement. Using a breadth first crawl of depth four starting at Dailymedorga; Mylan_Pharmaceuticals_Inc, we collect millions of triples. The data collected represents a heterogeneous web of linked data. The query is generated by randomly picking a subject from the input data and arbitrarily selecting distinct outgoing links. Then we substitute subject or object with variables. The following shows the example query we conduct.

Example 3 TestQuery1

```
SELECT ? cn? bd2 WHERE {
  Dailymedorga; Mylan_Pharmaceuticals_Inc dailymed;
  producesDrug? bd.
  ? bd dailymed:gene ricDrug ? gd.
  ? gd drugbank:possibleDiseaseTarget ? dt.
  ? dt diseasome:name "Epilepsy".
  ? bd dailymed:activeIngredient ? ai.
  ? bd2 dailymed:activeIngredient ? ai.
  ? c dailymed:producesDrug ? bd2.
  ? c rdfs:label ? cn.
}
```

The above TestQuery1 for query plan selection contains eight triple patterns, and only one of them qualifies for the seed URIs principle. The initial position of the filter triple pattern (? dt diseasome:name “Epilepsy”) is 4. For this query, about 35 query plans qualify for the three principles, and each of them has a filter triple pattern.

Fig. 2 illustrates the query results and retrieved RDF

objects for all the 35 query plans.

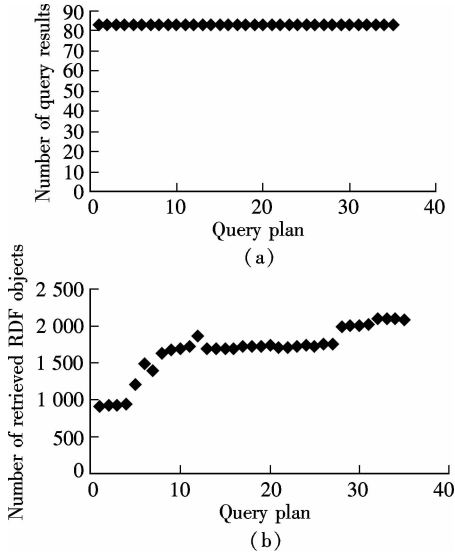


Fig. 2 Measurements for TestQuery1. (a) Query results for all query plans; (b) Retrieved RDF objects for all query plans

Each point in Fig. 2 represents a query plan. For all the plans, 83 solutions are returned, the same for each plan. However, the retrieved RDF objects differ significantly. To analyze the reasons that cause the difference, all the plans are divided into four groups according to their query execution time. For each group we compare their execution time, filter triple pattern locations and the number of retrieved RDF objects. Tab. 1 summarizes the comparison of these metrics.

Tab. 1 Comparison of metrics for all query plans of TestQuery1

Items	Group 1	Group 2	Group 3	Group4
Query execution time/s	< 20	[20,35]	[35,60]	>60
Number of query plans	4	5	18	8
Location of filter triple pattern	4.8	6.0	7.5	7.7
Number of retrieved RDF objects	922	1 577	1 715	2 049
Number of results returned	83	83	83	83

The location of the filter triple pattern confirms the effectiveness of FBGP principles. The fewer effective query plans in group 3 and group 4 contain the filter triple patterns in a 7-8 position. For more effective plans in group 1 and group 2 they are in a 4-6 position and closer to the seed triple pattern. Similarly, the numbers of retrieved RDF objects for group 1 and group 2 are much fewer than the ones for group 3 and group 4. The main reason is that the locations of group 1 and group 2 are closer to the seed triple pattern. Thus it brings much fewer intermediate solutions and fewer RDF objects retrieved. However, TestQuery1 cannot reflect the effects of no backward links and lack of contingency solutions. So we construct the following TestQuery2.

Example 4 TestQuery2

```

prefix doap: < http://usefulinc.com/ns/doap# >
prefix con: < http://www.w3.org/2000/10/swap/pim/contact# >
prefix rdfs: < http://www.w3.org/2000/01/rdf-schema# >
prefix foaf: < http://xmlns.com/foaf/0.1/ >
prefix rdf: < http://www.w3.org/1999/02/22-rdf-syntax-ns# >
prefix owl: < http://www.w3.org/2002/07/owl# >
SELECT ? s ? loc
WHERE {
  < http://www.w3.org/2000/10/swap/data#Cwm > doap:developer? f.
  ? f foaf:knows? s.
  ? s rdf:type? m.
  ? m rdfs:subClassOf? M.
  ? s owl:sameAs? loc.
  ? loc foaf:name "Tim Berners-Lee".
  < http://www.w3.org/2000/10/swap/data#Cwm > doap:developer? loc.
}

```

The above query contains seven triple patterns, two of which qualify as the seed triple pattern. The initial position of the filter triple pattern (? loc foaf:name "Tim Berners-Lee") is 6. There are 56 different query plans following the triple pattern chain principle. Fig. 3 depicts the number of query results and query execution time for different query plans. Each point in the figure represents a query plan.

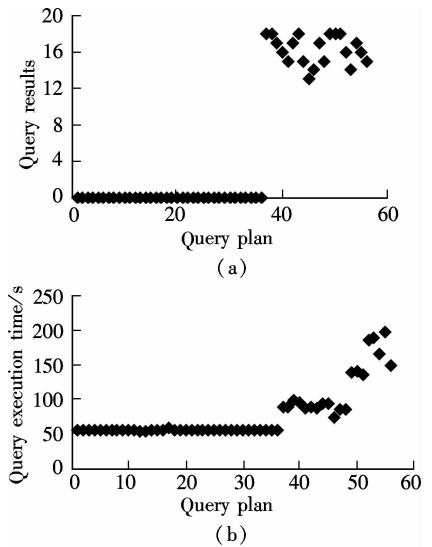


Fig. 3 Measurements for TestQuery2. (a) Results for all query plans; (b) Query execution time for all plans

To analyze the influence of the proposed principles, we divide all the query plans into two groups, and compare different groups with the following four metrics: average query execution time, number of query plans, the location of the filter triple pattern, and the number of results returned. As can be seen from Tab. 2, all the plans in one

of these groups do not provide any solutions. This problem can be attributed to no backward links. It confirms the above theoretical analysis. The location of the filter triple pattern reveals the impact of the FBGP principle. The fewer efficient plans in group 2 contain the filter triple pattern in position 3. 8 and 4. 6. For more efficient query plans, they are in position 3. 1 or 3. 3. Hence, it takes less execution time if they are closer to the seed triple pattern.

Tab.2 Comparison of metrics for all query plans of TestQuery2

Items	Group 1		Group 2	
Average query execution time/s	< 60	[60,80]	[80,100]	>100
Number of query plans	29	7	11	9
Location of filter triple pattern	3. 1	3. 3	3. 8	4. 6
Number of results returned	0	0	17. 6	17. 4

5 Conclusion

In this paper, we first give a formal definition of RDF-LTE and analyze the query cost and result completeness of RDF-LTE based on the concrete SPARQL queries. Then, we discuss its limitations and find that by using current approaches, many RDF objects which do not contribute to answering the final query are retrieved and empty results are returned for some types of queries. Three principles for logical query planning are proposed under the assumption of lack of related information. The experimental results demonstrate the effectiveness of our proposed approaches. It is shown that more query results can be returned with less cost.

In the future, we will aim at using the information discovered at run-time, tracking the provenance of the data source to rank the query result. Besides, future work will also include the application of RDF-LTE in certain areas.

References

[1] Berners-Lee T. Linked data[EB/OL]. (2006-06-18)

[2012-09-20]. <http://www.w3.org/DesignIssues/LinkedData.html>.
[2] Hartig O, Langegger A. A database perspective on consuming linked data on the web[J]. *Datenbank-Spektrum*, 2010, 10(2):57–66.
[3] Oren E, Delbru R, Catasta M, et al. Sindice.com: a document-oriented index for open linked data[J]. *International Journal of Metadata Semantics and Ontologies*, 2008, 3(1):27–52.
[4] Cheng G, Qu Y. Searching linked objects with falcons: approach, implementation and evaluation[J]. *International Journal on Semantic Web and Information Systems*, 2009, 5(3):49–70.
[5] Sheth A P, Larson J A. Federated database systems for managing distributed, heterogeneous, and autonomous databases[J]. *ACM Computing Surveys*, 1990, 22(3):183–236.
[6] Quilitz B, Leser U. Querying distributed RDF data sources with SPARQL[C]//*Proceedings of the 5th European Semantic Web Conference on the Semantic Web*. Canary Islands, Spain, 2008:524–538.
[7] Hartig O, Bizer C, Freytag J C. Executing SPARQL queries over the web of linked data[C]//*Proceedings of the 8th International Semantic Web Conference*. Washington DC, USA, 2009:293–309.
[8] Klyne G, Garroll J J. Resource description framework (RDF): concepts and abstract syntax[EB/OL]. (2004-02-10) [2012-09-20]. <http://www.w3.org/TR/rdf-concepts/>.
[9] Stuckenschmidt H, Vdovjak R, Houben G-J. Index structures and algorithms for querying distributed RDF repositories[C]//*Proceedings of the 13th International Conference on World Wide Web*. New York, USA, 2004: 631–639.
[10] Lampo T, Vidal M E, Danilow J, et al. To cache or not to cache: the effects of warming cache in complex SPARQL queries[C]// *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2011:716–733.

基于 RDF 链接遍历查询方案的优化

朱艳琴 花 岭

(苏州大学计算机科学与技术学院, 苏州 215006)

摘要:针对采用现有的 RDF 链接遍历查询执行方案只能回答部分类型 SPARQL 查询的问题,结合具体的 SPARQL 查询,讨论了元组模式执行顺序对查询结果及查询代价的影响,分析了互联数据 Web 的 2 个问题:缺乏反向链接性与不支持偶然发现的解.然后,提出了 3 个启发式的逻辑查询计划优化原则:FBGP 原则、元组模式链原则和种子 URIs 原则.这 3 个原则有助于减少中间解和增加可回答的查询类型数目.并通过实验证明了其有效性和可行性.实验结果表明,优化后的方案能够以较小的代价得到更多的查询结果,从而有助于用户更好地发挥互联数据 Web 的潜能.

关键词:互联数据 Web; RDF-LTE; SPARQL 查询;查询优化

中图分类号:TP391