DCBDQuery – Query Language for RDF using Dynamic Concise Bounded Description

Xinpeng ZHANG† Masatoshi YOSHIKAWA‡

†‡ Graduate School of Informatics, Kyoto University Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501 Japan

E-mail: †xinpeng.zhang@db.soc.i.kyoto-u.ac.jp, ‡yoshikawa@i.kyoto-u.ac.jp

Abstract Along with the development of Semantic Web, available RDF data are increasing at a fast pace. How to store and query large amount of RDF data becomes an important issue of the Semantic Web. In this paper, we first discuss the usage of Concise Bounded Description (CBD). As an improvement, we define Dynamic Concise Bounded Description (DCBD) which is a general and dynamic version of CBD. We also propose a query language for RDF called DCBDQuery. DCBDQuery is used for constituting DCBD and finding meaningful reachable path or shortest path with respect to DCBD. Then we discuss an alternative approach for storing RDF data into Relational Database, called Updated Schema-aware Representation. We store RDF data into database using this representation, and then create graph in main memory from internal link statements. The DCBDQuery query engine is designed to access a hybrid data model of the database and the graph in main memory. Finally, the RDF data of DBLP++ is used to do experiment.

Keyword Semantic Web, RDF, query language, Reachability Query

DCBDQuery – 動的 CBD を利用した RDF 問合せ言語

張 信鵬† 吉川 正俊‡

†‡ 京都大学情報学研究科 〒606-8501 京都市左京区吉田本町

E-mail: †xinpeng.zhang@db.soc.i.kyoto-u.ac.jp, ‡yoshikawa@i.kyoto-u.ac.jp

あらまし Semantic Web の発展により，RDF データは急激に増加しており，大量の RDF データを効率よく格納・検索することが重要である．本論文では，CBD (Concise Bounded Description) の利用と制限について議論した上で，動的 CBD（DCBD）を定義する．DCBD はあらゆる CBD を包含し，一般的かつ動的に可変である。DCBD に関する検索を行うために，RDF 問合せ言語 DCBDQuery を提案する。DCBDQuery により、DCBD を構成すること及び DCBD を考慮した到達可能経路検索が実現できる。その後、RDF データを関係データベースに格納するための RDF スキーマを意識した Updated Schema-aware Representation を提案する。この手法を用い、データベースに格納されている内部リンクの文を取り出し、メインメモリ上でグラフを作成する。DCBDQuery の検索エンジンはデータベース上のデータとメインメモリ上のグラフからなる複合型のデータモデルに対し、検索を行う。

キーワード Semantic Web，RDF，問合せ言語，到達可能経路

1. Introduction

The Resource Description Framework (RDF) is a language for representing information about resources in the Semantic Web. Along with the increase of the size of RDF data, it is getting more and more critical to achieve an efficient way for optimal and consistent retrieval of knowledge about specific resources. Concise Bounded Description [1] offers a solution for this problem which is a general and broadly optimal unit of specific knowledge about that resource to be utilized by.

1.1. CBD and DCBD

Given a particular node in a particular RDF graph, a Concise Bounded Description (CBD) is a subgraph consisting of those statements which together constitute a focused body of knowledge about the resource denoted by that particular node. A CBD of a resource in terms of an RDF graph is an optimal unit of specific knowledge about that resource to be utilized by, interchanged between semantic web agents rather than the whole RDF graph. A CBD of a resource is also a meaningful query scope for query about that resource.

It is clear that a certain CBD is not surly to be an optimal form of description for every application or every user. CBD should be identified dynamically according to the changeable requirement of different applications or users. We propose a general solution: Dynamic Concise Bounded Description (DCBD). We give a definition of DCBD, which could cover almost all the definitions of different kinds of CBDs. We give the definition of DCBD with variables, including predicate weight, path weight limit and including direction. Users could constitute different kinds of DCBDs according to their needs easily by changing the values of these variables.

1.2. Querying RDF graphs

SPARQL [2] is a query language for RDF which is undergoing standardization by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

PSPARQL [3] is an extension of SPARQL query language with path expressions. It allows the use of regular expression patterns with variables in the predicate position of SPARQL graph patterns.

We propose a query language for RDF called DCBDQuery. DCBDQuery could constitute DCBD from a certain RDF graph and find meaningful reachable path or shortest path of two nodes in a certain RDF graph. These two queries cannot be expressed by SPARQL or PSPARQL.

In the remainder of this paper, we give a definition of DCBD in Section 2, followed by the syntax and query forms of the DCBDQuery query language in Section 3. Before explaining the query process of DCBDQuery, we propose the Updated Schema-aware Representation in Section 4, by which we first store all the RDF data into Relational Database, and then load only the internal link statements into main memory for query, so as to achieve a best space-time trade-off. Then, query processing of DCBDQuery will be discussed in Section 5. In Section 6, we implement a system using the RDF data of DBLP++. Finally, we give the conclusions and future works.

2. Definition of CBD and DCBD

In this section, we view the definition of CBD first, and then define DCBD. Besides, we give examples of CBD and DCBD constituted from a sample RDF Data.

2.1. Definition of RDF Graph

First, let's look at the definition of RDF Graph from which CBD and DCBD are constituted. We suppose that there are no blank node and reification statement in the RDF Graph for the sake of simplicity.

Definition 1 (graph) If <S,P,O> is an RDF statement, S is called its subject, P its predicate, and O its object. subj(G) = {S| <S,P,O> ∈ G} is noted as the set of elements appearing as a subject in a statement of an RDF graph G. pred(G) and obj(G) are defined in the same way for predicates and objects. We call the nodes of G, denoted nodes(G) = subj(G) ∪ obj(G), the set of elements appearing either as subject or object in a statement of G. U denotes the set of urirefs. L denotes the set of literals. subj(G) ∈ U, pred(G) ∈ U, obj(G) ∈ U or obj(G) ∈ L. A statement of G is an element of statement(G) = subj(G) ∪ pred(G) ∪ obj(G). An RDF statement <S,P,O> is an internal link, if O is an instance of a class in the RDF schema of G; otherwise, it is an external link. path(G) denotes the set of path of G. p is a path of G, p(G) = {$<S_0,P_0,O_0>,\ldots,<S_k,P_k,O_k>$ | $<S_i,P_i,O_i>$ ∈ statement(G), $0 \le i \le k$, $k \in N$, $O_i = S_{i+1}$ or $O_i = O_{i+1}$ or $S_i = O_{i+1}$ or $S_i = S_{i+1}$}, p(G,V_s,V_e) = {$<S_0,P_0,O_0>,\ldots,<S_k,P_k,O_k>$ | $<S_i,P_i,O_i>$ ∈ statement(G), $0 \le i \le k$, $k \in N$, $V_s = S_0$ or $V_s = O_0$, $O_i = S_{i+1}$ or $O_i = O_{i+1}$ or $S_i = O_{i+1}$ or $S_i = S_{i+1}$, $V_e = S_k$ or $V_e = O_k$ }. Since <S,P,O> is directed, G is a directed graph, but path p is undirected.

2.2. A Definition of CBD

A kind of CBD is presented herein as a reasonably general and broadly optimal form of description. This CBD is constituted by including in the subgraph all statements in the source graph where the subject of the statement is the starting node.

Definition 2 (CBD) Let G be a source RDF graph. Given a node s ∈ nodes(G), taken to comprise a concise bounded description of G denoted by node s, CBD(s) is defined as follows:

- if <S,P,O> ∈ statement(G), O ∈ U or O ∈ L and S = s; include <S,P,O> in CBD(s).

EXAMPLE 1: Constitute CBD

Given the RDF graph in Figure 1 as input:

```
<?xml version="1.0"?
<rdf:RDF
    xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc   ="http://purl.org/dc/elements/1.1/"
```

```
      xmlns:dct   ="http://purl.org/dc/terms/"
      xmlns:foaf ="http://xmlns.com/foaf/0.1/">
  <rdf:Description rdf:about="http://dblp.l3s.de/d2r/AbiteboulHV95">
      <rdf:type>http://xmlns.com/foaf/0.1/document</rdf:type>
      <dc:title>Foundations of Databases.</dc:title>
      <dc:creator>http://dblp.l3s.de/d2r/Richard_Hull</dc:creator>
      <dct:references>http://dblp.l3s.de/d2r/GareyJ79</dct:references>
   </rdf:Description>
   <rdf:Description rdf:about="http://dblp.l3s.de/d2r/GareyJ79">
      <rdf:type>http://xmlns.com/foaf/0.1/document</rdf:type>
      <dc:title>Expressive Power of Query Languages.</dc:title>
      <dc:creator>http://dblp.l3s.de/d2r/Victor_Vianu</dc:creator>
      <dct:references>http://dblp.l3s.de/d2r/StimG83</dct:references>
   </rdf:Description>
   <rdf:Description rdf:about="http://dblp.l3s.de/d2r/ StimG83">
      <rdf:type>http://xmlns.com/foaf/0.1/document</rdf:type>
      <dc:title>Expressive Power of Query Languages.</dc:title>
      <dc:creator>http://dblp.l3s.de/d2r/Zane</dc:creator>
      <dct:references>http://dblp.l3s.de/d2r/TomH89</dct:references>
   </rdf:Description>
   <rdf:Description rdf:about="http://dblp.l3s.de/d2r/Richard_Hull">
      <rdf:type>http://xmlns.com/foaf/0.1/agent</rdf:type>
      <foaf:name>Richard Hull</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

Figure 1: Source RDF Graph

The CBD of *http://dblp.l3s.de/d2r/AbiteboulHV95* corresponds to the subgraph in Figure 2:

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf   ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc    ="http://purl.org/dc/elements/1.1/"
      xmlns:dct   ="http://purl.org/dc/terms/"
      xmlns:foaf ="http://xmlns.com/foaf/0.1/">
  <rdf:Description rdf:about="http://dblp.l3s.de/d2r/AbiteboulHV95">
      <rdf:type>http://xmlns.com/foaf/0.1/document</rdf:type>
      <dc:title>Foundations of Databases.</dc:title>
      <dc:creator>http://dblp.l3s.de/d2r/Richard_Hull</dc:creator>
      <dct:references>http://dblp.l3s.de/d2r/GareyJ79</dct:references>
   </rdf:Description>
</rdf:RDF>
```

Figure 2: Example of CBD

2.3. Definition of DCBD

We define DCBD in a dynamic and general way to satisfy different kinds of requirements. First, We give the definition of predicate weight and path weight used in the definition of DCBD.

Definition 3 (predicate weight) We assign weight w to each predicate in the source graph G.

- If $P_0 \in Pred(G)$,
  then, $w(P_0) \in R$, $0 \leq w(P_0) \leq 1$.

Definition 4 (path weight) Given a path p of G, path weight of p(G) is defined as:

- $weight(p(G)) = weight(<P_0>) \times \ldots \times weight(<P_k>)$.

Generally, given a particular node (the starting node) in a particular RDF graph (the source graph), predicate weight, path weight limit and including direction as input, a subgraph of that particular graph, taken to comprise a DCBD of the resource denoted by the starting node, is constituted by including statements where the subject or the object is the starting node first, and then recursively including statements where the subject or the object is the end node of the path of the current DCBD until the path weight getting smaller than the path weight limit.

Definition 5 (DCBD) Let G be a source RDF graph. Given a node $s \in nodes(G)$, predicate weight $weight(P_m)$ for $P_m \in pred(G)$ $(m = 0, 1, \ldots, k)$, path weight limit $pwl \in R$, $0 \leq pwl \leq 1$, including direction := {FORWARD or BACKWARD or BOTH}, taken to comprise a dynamic concise bounded description of G denoted by node s which is a subgraph of G, $DCBD(s, weight(P_m), pwl, direction)$, abbreviated to D in following, is defined as follows:

(i) If direction = FORWARD or BOTH,
  $<S,P,O> \in statement(G)$ and S = s; or
  direction = BACKWARD or BOTH,
  $<S,P,O> \in statement(G)$ and O = s,
  include <S,P,O> in D.

(ii) Recursively, if the following conditions (1) and (2) (or (1) and (3)) are satisfied, then include <S,P,O> in D.

  (1) $<S,P,O> \in statement(G)$, and $t \in nodes(D)$, and $\exists p(D,s,t)$ s.t. $weight(p(D,s,t)) \times weight(<S,P,O>) \geq pwl$.
  (2) direction = FORWARD or BOTH, and S = t.
  (3) direction = BACKWARD or BOTH, and O = t.

EXAMPLE 2: Constitute DCBD

Given the RDF graph in Figure 1 as input, and given the following:

weight(rdf:type) = 1.0,
weight(dc:title) = 1.0,
weight(foaf:name) = 1.0,
weight(dc:creator) = 0.9,
weight(dct:references) = 0.75,
path weight limit = 0.6,
include direction = BOTH

The DCBD of *http://dblp.l3s.de/d2r/AbiteboulHV95* corresponds to the subgraph in Figure 3:

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf   ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc    ="http://purl.org/dc/elements/1.1/"
      xmlns:dct   ="http://purl.org/dc/terms/"
      xmlns:foaf ="http://xmlns.com/foaf/0.1/">
  <rdf:Description rdf:about="http://dblp.l3s.de/d2r/AbiteboulHV95">
      <rdf:type>http://xmlns.com/foaf/0.1/document</rdf:type>
      <dc:title>Foundations of Databases.</dc:title>
      <dc:creator>http://dblp.l3s.de/d2r/Richard_Hull</dc:creator>
      <dct:references>http://dblp.l3s.de/d2r/GareyJ79</dct:references>
   </rdf:Description>
   <rdf:Description rdf:about="http://dblp.l3s.de/d2r/GareyJ79">
      <rdf:type>http://xmlns.com/foaf/0.1/document</rdf:type>
      <dc:title>Expressive Power of Query Languages.</dc:title>
      <dc:creator>http://dblp.l3s.de/d2r/Victor_Vianu</dc:creator>
   </rdf:Description>
   <rdf:Description rdf:about="http://dblp.l3s.de/d2r/Richard_Hull">
      <rdf:type>http://xmlns.com/foaf/0.1/agent</rdf:type>
      <foaf:name>Richard Hull</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

Figure 3: Example of DCBD

3. The DCBDQuery Query Language

In this section, we discuss the syntax and query forms of DCBDQuery.

3.1. DCBDQuery syntax

DCBDQuery has two query forms.

CONSTITUTE
Returns the DCBD of the given starting node constituted from the source RDF graph.

FIND
Returns the paths or shortest paths connecting any two of the given nodes together.

3.1.1. CONSTITUTE

The CONSTITUTE query returns an RDF subgraph of the resource RDF graph. The result subgraph constituted from statements included according to the definition of DCBD.

Grammar rule:
ConstituteQuery := CONSTITUTE StartingNodeClause CostituteDCBDClause
ConstituteDCBDClause *:=* GraphClause IncludeClause PredicateWeightClause LimitWeightClause

Starting node is given in the StartingNodeClause by URI. Source graph is given in the GraphClasue in literal. Including direction is set in IncludeClaused. Predicate weight is assigned to each predicate in Predicate-WeightClause. Path weight limit is given in Limit-WeightClause. A query example is shown as below:

Query 1
CONSTITUTE
FOR http://dblp.l3s.de/d2r/resource/ conf/KouhJY05
FROM GRAPH dblp
INCLUDE both
BY PREDICATE WEIGHT
http://purl.org/dc/terms/partOf = 0.6,
http://purl.org/dc/elements/1.1/creator = 0.9,
http://www.w3.org/2002/07/owl#sameAs = 1.0,
http://swrc.ontoware.org/ontology#journal = 0.6,
http://swrc.ontoware.org/ontology#series = 0.6,
http://swrc.ontoware.org/ontology#editor = 0.7,
http://purl.org/dc/terms/references = 0.75
LIMIT 0.5

3.1.2. FIND

The FIND query returns a list of path. The query executes recursively for any two of the given nodes, and returns the reachable path or shortest path of them in the source graph or in the DCBDs of them.

It is considerable that not all the reachable paths of two particular nodes in the whole RDF graph are meaningful. Finding out all the reachable paths or shortest paths in the whole graph is also an expensive work. Thus we define the query to just find reachable path or shortest path in the whole graph or in the DCBDs of the given nodes, with a weight limit of the reachable path or shortest path, so as to reduce the query scope to a meaningful area. If there is no reachable path or shortest path, of which the weight is heavier than the given reachable path weight limit, the given nodes are identified as unreachable.

The heavier the weight of the reachable path is, the stronger the relationship of these two nodes is. So paths with the heaviest weight are returned in shortest path query.

Grammar rule:
FindQuery := FIND (SHORTEST)? PATH FOR NodeClause *SourceClause ReachablepathLimitClause*
SourceClause :=
IN (GRAPH Graph PredicateWeightClause|
DCBD‘(’ ConstituteDCBDClause ‘)’)

Nodes are given in the NodeClause with a constraint of no more than 5 nodes. Finding reachable path or shortest path is identified by adding the keyword SHORTEST or not. Query object could be the whole RDF graph with

predicate weight identified in PredicateWeightClause, or the DCBDs of the given nodes constituted by the query conditions given in ConstituteDCBDClause. The reachable path weight limit is given in *ReachablepathLimitClause*. A shortest path query example is shown as below:

Query 2

```
FIND SHORTEST PATH FOR
http://dblp.l3s.de/d2r/resource/conf/IEEEscc/Ananth06,
http://dblp.l3s.de/d2r/resource/ conf/ACISicis/KouhJY05
IN DCBD (
FROM GRAPH dblp
INCLUDE both
BY PREDICATE WEIGHT
http://purl.org/dc/terms/partOf = 0.6,
http://purl.org/dc/elements/1.1/creator = 0.9,
http://www.w3.org/2002/07/owl#sameAs = 1.0,
http://swrc.ontoware.org/ontology#journal = 0.6,
http://swrc.ontoware.org/ontology#series = 0.6,
http://swrc.ontoware.org/ontology#editor = 0.7,
http://purl.org/dc/terms/references = 0.75
LIMIT 0.5)
LIMIT 0.25
```

4. Updated Schema-aware Representation

Before discuss the query processing, let's look at the Updated Schema-aware representation first. In Section 4.1 we introduce the Updated Schema-aware Representation and its attributes. Then we discuss the usage of the Updated Schema-aware Representation.

4.1. Introduction

With Updated Schema-aware Representation, basically per class with its external link properties and per internal link property in the RDF Schema is represented as a table in the Relational Database (see Figure 4). This Represen--tation approach can be generally used in any RDF data with some rules as follows:

1. Store multiple value external link property in a separate property table.
2. Every record of Class Table is encoded using an integer value to enable faster lookups.

The properties of this representation are:

1. Reduce self-join over triple stores. Since all data are stored in the same table in triple stores, many self-join are required for a path query. In contrast, with this representation more efficient fast merge join between different tables is required.
2. Query for a certain class can be processed very fast. Most of the time only one row of the class table is required to be accessed.
3. Only those tables accessed by a query need to be read. Besides, because there are only two integer columns in the property table of internal link, I/O costs can be reduced.
4. NULL value may appear in the property column of Class Table, thus some space will be wasted by these NULLs. However, because predicate is not stored for each statement, many spaces can be saved.

Class Table1

Id	rdf_about	property 1	…	property n
Integer	resource URI	property value		property value

…

Class Tablem

Id	rdf_about	property 1	…	property n
Integer	resource URI	property value		property value

Property Table1

Subject	Object
Id of Class Table	property value (Id or Literal)

…

Property Tablen

Subject	Object
Id of Class Table	property value (Id or Literal)

Figure 4: Updated Schema-aware Representation

4.2. Usage

Managing and querying huge RDF file in disk is not efficient. Loading all the RDF data into main memory is also an expensive work. In order to make a reliable and high speed query engine, we design the query engine to access a hybrid data model of relational database and graph in main memory.

First, we store RDF data into relational database using Updated Schema-aware Representation. Then, graph is created in main memory using the data in Property Tables of internal link. There are some reasons why we do not load data of external link.

1. External link is necessary knowledge about the subject node of that external link. So the predicate weight of external link is assigned to be 1.
2. The object node of external link is not instance of the class in the RDF schema of the RDF data, so that can not be query object.
3. Retrieve data of external link from database constructed with index is fast.

The detail of the query processing is discussed in Section 5.

5. Query Processing of DCBDQuery

In this section, we look at how we retrieve query answer from database and graph. In Section 5.1, we discuss how to create graph. Then the query processing is discussed for each query form separately.

5.1. Create graph

Because path query is processed without considering direction, undirected graph will be created. One statement is represented as one undirected edge with two nodes in the graph. On the other hand, direction information is necessary for constituting DCBD, because the including direction is given. The direction information should also be given in query answer. To solve this issue, we label the edge with the information of direction. And, a short name rather than URI of that property is used to represent the property of that edge.

Regarding node, because Id attribute values of each class are assigned separately, we label node with its Id with a prefix to represent the class type of that node.

5.2. Constitute DCBD

This query mainly constitutes DCBD from the graph in main memory. This is done by an algorithm which is an extension of the depth-first search algorithm. The algorithm constitutes DCBD from the given starting node, and explores as far as possible along each path until the path weight getting smaller than the given path weight limit. Besides, the data interchanging between database and main memory is needed.

The whole query process is as below:

1. Parse query statement based upon the syntax of DCBDQuery to check query and get query variables;
2. Search Id of the given node from class tables in the database;
3. Encode the label of the given node in the graph according to the labeling rule;
4. Constitute DCBD of the graph starting from the node with the label encoded in 3;
5. Retrieve all the external property of all the class nodes in the DCBD by Id decoded from label.

5.3. Find Reachable path

The main task of this query is finding reachable path or shortest path from the graph in main memory. An extension of the breath-first search is used to do this for any two of the given nodes recursively.

The algorithm of finding shortest path in DCBDs of the given nodes, explores all the neighboring nodes ($Node_1$, …, $Node_m$) of the two given nodes separately. Then for each of those nearest nodes of each of the two given nodes, explores their unexplored neighbor nodes in turn, until any one of the following conditions is satisfied, then stop exploring neighbor nodes of that node ($Node_i$).

(1) The path weight of the path starting from the given node to $Node_i$ is getting lighter than
   (i). the DCBD path weight limit, or
   (ii). the reachable path weight limit, or
   (iii). the heaviest path weight of the reachable path which have been found until now.

(2) $Node_i$ can connect the two given nodes together.

If (2) is satisfied, then the path connecting the two given nodes and $Node_i$ is a reachable path. The reachable path with the heaviest path weight is the shortest path.

Eliminate condition (i) above to find shortest path in whole graph. All reachable paths can be queried by without considering condition (iii).

The whole query process is shown as below:

1. Parse query statement based upon the syntax of DCBDQuery to check query and get query variables.
2. Search Ids of the given nodes from class tables in the database;
3. Encode the label of the given nodes in the graph according to the labeling rule;
4. Recursively find reachable path or shortest path in the graph for any two of the nodes with the label encoded in 3.
5. Retrieve URI of all the class nodes in the answer paths by id decoded from label.

6. Implementation and Experiments

We have implemented in Java a DCBDQuery query engine. The RDF data of DBLP is used for evaluating our system. The system runs on a server with four 3.80 GHz Xeon processor, 8 Gbytes of memory and 2MB L2 cache.

6.1. The RDF Data of DBLP

The RDF Data of DBLP++ is built by FacetedDBLP [7] which is an enhancement of DBLP plus additional key--words and abstracts as available on public web pages. The RDF data is updated once per week, which consists of approximately 28 million RDF triples. Figure 5 shows the internal link part of the RDF Schema of DBLP++.

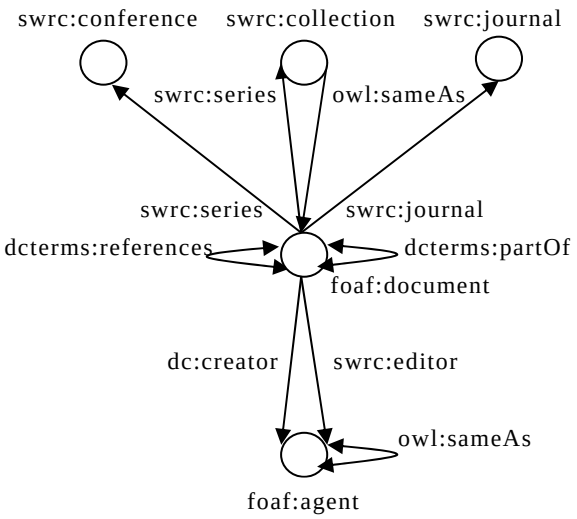

Figure 5: A part of the RDFS of DBLP++

6.2. System

We designed the relational schema for RDF data of DBLP++ using Updated Schema-aware Representation. We loaded all the data downloaded from the site of FacetedDBLP into database which took almost 30 hours. The DCBDQuery query engine queried successfully all tests. The time of creating graph in main memory is about 1180 seconds. The graph contains 1,546,238 nodes and 3,922,201 edges.

Time complexity of the algorithm of constituting DCBD and finding reachable path or shortest path are both proportional to the number of nodes plus the number of edges in the graph they traverse (O(|N| + |E|)).

Table 1 lists the query time of some CONSTITUTE query tests with the number of nodes and number of edges of the result DCBD.

number of nodes	number of edges	query time (millisecond)
2	1	147
21	20	163
126	246	186
399	518	284
929	928	286
4265	4936	353
16157	35174	1627
63756	93755	46960
176332	246143	50932

Table 1: Query time of CONSTITUTE

With a relatively larger reachable path weight limit, exploring process of FIND query finishes faster. Besides, in order to obtain meaningful reachable path or shortest path, relatively large DCBD path weight limit or reachable path weight limit should be given. From these and the experiment results, we can say that the DCBDQuery query engine could return meaningful result of FIND query efficiently.

7. Conclusions and Future Works

Concise Bounded Description (CBD) is an optimal unit about a certain resource in a whole RDF graph for information interchanging or querying about that resource. Although there are already several kinds of CBD existed, any kind of them is application dependent. Towards this problem, we proposed the definition of Dynamic Concise Bounded Description (DCBD), which is general and customizable. According to the definition of DCBD, we proposed a query language named DCBDQuery for query DCBD and reachable path or shortest path.

Further, we proposed Updated Schema-aware Represen--tation for storing RDF data into relational database which showed better performance than triple stores. Finally, we created a DCBDQuery query engine, which access a hybrid data model constructed of relational database and graph in main memory. With this hybrid data model, query can be executed efficiently.

As future work, we plan to find an efficient approach for query CBD directly from Relational Database. Adding SPARQL query into the DCBDQuery is also considerable. To help user to get meaningful query answer, reasonable value of the variables in query will be considered too.

References

[1] Patrick Stickler, “CBD - Concise Bounded Descrip--tion”, W3C Member Submission 3 June 2005. http://www.w3.org/Submission/CBD/.

[2] Eric Prud'hommeaux, Andy Seaborne, SPARQL Query Language for RDF, W3C Recommendation 15 January 2008, W3C Recommendation 15 January 2008.

[3] Faisal Alkhateeb, Jean-François Baget and Jérôme Euzenat, “RDF with regular expressions”, Research Rreport 6191, INRIA (2007).

[4] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen: “Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema”, Proceedings of the First Internation Semantic Web Conference, I. Horrocks and J. Hendler, pp.54-68, Springer Verlag, July 2002.

[5] Yannis Theoharis, Vassilis Christophides, Grigoris Karvounarakis: “Benchmarking Database Representation of RDF/S Stores”, ISWC2005, LNCS 3729, p. 685 ff. Springer-Verlag Berlin Heidelberg, 2005.

[6] Daniel J.Abadi, Adam Marcus, Samuel R.Maeed, and Kate Hollenbach, “Scalable Semantic Web Data Management Using Vertical Partitioning”, VLDB ’07, September 23-28, 2007, Vienna, Austria.

[7] Jörg Diederich, Wolf-Tilo Balke and Uwe Thaden. “About FacetedDBLP”, 13 November 2007. http:// dblp. l3s. de/dblp++.php.