B-SPARQL: A typed language for Querying the Big Knowledge

Ruqian Lu
Key Laboratory of MADIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
Chinese Academy of Sciences, Beijing 100190, China
rqlu@math.ac.cn

Chuanqing Wang
Key Laboratory of MADIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China
wangchuanqing15@mails.ucas.ac.cn

Xikun Huang
Key Laboratory of MADIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China
huangxikun14@mails.ucas.ac.cn

Songmao Zhang
Key Laboratory of MADIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
Chinese Academy of Sciences, Beijing 100190, China
smzhang@math.ac.cn

***Abstract*—We introduce B-SPARQL as an extension of the classical SPARQL to deal with the problem of querying big knowledge. B-SPARQL extends SPARQL mainly in two directions. The first one is to query big knowledge in a global way such as 'how much of the knowledge items of the knowledge base are confidential?' The second one is to query big knowledge in its translated digital form instead of in its original symbolic form. The former is necessary when evaluating and maintaining an immense knowledge base. The latter is useful when it is very inefficient or even impossible to deal with an immense knowledge base. In accordance, B-SPARQL not only deals with basic knowledge elements such as subjects, predicates and objects of RDF triples. It also deals with triples and knowledge bases and knowledge areas (parts of knowledge bases), including SPARQL endpoints, immediately by considering them as basic knowledge elements as well.**

Keywords*—*SPARQL, Big Knowledge, Semantic Web, RDF, Knowledge Graph, Complex network

I. Introduction

A. *Impact of Big Data (BD) and Big Knowledge (BK) on Knowledge Query Techniques*

The rise of big data (BD) study at the begin of 21th century is no doubt one of the most important streams in nowadays information processing. Various theories about BD such as 3V, 4V, 5V or even nV ($n > 4$) properties have been proposed. A different proposal of HACE (heterogeneous, autonomous, complex, evolving) theorem was made by Wu [1].

For most people, the first thing they want to do and the ultimate goal they want to reach with BD is to mine big knowledge (BK) from it. People started to propose and put forwards the terminology 'BK' in the second decade of this century. The slogan 'turn big data to big knowledge' has been a common appeal of many experts, including K. Murphy [2], A. Russel [3], J. Aron [4], J. Liebowitz [5] and Herschkowitz [6]. See also [7], [8] and [9] for similar works from China.

Ruqian Lu is the corresponding author. This work has been supported by National Key Research and Development Program of China under grant 2016YFB1000902. And NSFC Project 61232015, 61472412, 61621003, 61872352. And Tsinghua-Tencent- AMSS –Joint Project: WWW Knowledge Structure and its Application.

However, almost all these works just mentioned the significance of BK but without providing a specific definition for it. To the best of our knowledge the first systematical study of BK was made in [10], where 5 characteristics (massive concepts, massive connectedness, massive clean data resources, massive application cases, massive confidence) for BK, 5+1 characteristics (+ massive capabilities) for BK system (BKS), 6+2 characteristics (+ massive cumulativeness and massive concerns) for advanced BKS, and 8+2 characteristics (+ massive consistency and massive completeness) for future BKS are proposed. Furthermore, the characteristics of BK engineering (BKE) is also defined.

Still one step forwards along this line is the big wisdom (BW) concept proposed by M. Wu and X. Wu [11], representing a HAO/BIBLE framework, which integrates human intelligence (HI), artificial intelligence (AI), and organizational/business intelligence (O/BL) with BD analytics in large environments, for industrial intelligence in organizational activities. The spirit of HAO/BIBLE is a triple jump, where the first jump is BD analytics, second jump BK discovery and third jump complex problem solving.

But the most impressive BKE effort is the development of large knowledge graphs (KG) which is the strongest effort made by the BK community. According to a statistics we believe the KGs listed in Table I (data extracted from Table 5 of [10]) are powerful candidates of BK, where the threshold is 10M (100 K) for number of entities (concepts) and 1B (1M) for number of entity facts (concept facts).

Given that nowadays massive size knowledge graphs are already very popular, the question arises: is the traditional theory and technique of knowledge query still powerful enough to face the BD and BK challenges? In fact, under the background of BD and BK there are many new requirements that SPARQL like query techniques are not powerful enough to meet. For example,

978-1-7281-8156-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICBK50248.2020.00034

macroscopic query about KG's large scale global properties, structured query about KG's structured information at different levels, typed query about KG's data type information, semantic query about KG's natural language semantics, continuous query about KG's low dimensional vector representation, etc. Roughly speaking, ideas about introducing new query functions come from four major sources: challenges of BD and BK processing, new techniques developed for BD and BK processing, requirements of many application needs, and published proposals for developing new variants of SPARQL (see section II).

TABLE I. THE BIG KNOWLEDGE GRAPH

KG	Entity(10M) Concept(100K)	Entity Fact (1B) Concept Fact(1M)
Freebase[12]	44M(E)	3.1B(EF)
Google Knowledge Graph[13]	570M (E)	70B(EF)
Knowledge Vault[14]	45M (E)	1.6B(EF)
OpenKN [15]	20M(E)	2.2B(EF)
Probase [16]	5.4M(C)	20M(CF)
YAGO2 [17]	350K(C)	120M(CF)
OpenCyc3 [18]	177K(C)	1.5M(CF)
OpenCyc4 [18]	239K(C)	2.1M(CF)

B. B-SPARQL in General

SPARQL is a powerful query language operating on knowledge graphs. However, it still lacks some important facilities which are very useful in BD and BK processing. Adding these facilities to SPARQL to make it more powerful is the motivation of designing B-SPARQL.

The first facility we introduce in B-SPARQL is a set of complex data types and batch processing operations on large granule knowledge. SPARQL queries operate on basic data types including IRI and literals. SPARQL also introduces group patterns to represent composed data structures. In B-SPARQL we go one step further to define, operate and query massive size data with complex data types. Currently B-SPARQL's data types include but are not limited to the following:

- *?Xany* means querying an entity *Xany*;
- *?Xpany* means querying a predicate *Xpany*;
- *?UPR(Xany1, XAny2)* means querying an unordered pair *Xany* of entities;
- *?OPR(Xany1,Xany2)* means querying an ordered pair *Xany* of entities;
- *?T(Xany)* means querying a triplet *Xany*.
- *?Tr(Xany)* means querying a tree *Xany*.
- *?A(Xany)* means querying an algorithm name *Xany*.
- *?S(anytype)* means querying a set of data of any type. This set may be a knowledge graph, part of a knowledge graph, a SPARQL endpoint or just several triplets.
- *?L(anytype)* means query a list of data separated by blancs.
- *?ID(Xany)* means queries an entity which is an identifier.
- *?SEQ(anytype)* means querying a sequence of data separated by commas.
- *?STR(anytype)* means querying a string of symbols not including separators such as comma, blanc, point, colon, etc.
- *?B(anytype)* means querying a bag (multi-set) of data of any type.
- *?E(anytype)* means querying a vector embedded from some data of any type.
- *?N(anytype)* means querying a network of data of any type.
- *?P(anytype)* means querying a path in some network.
- *?C(anytype)* means querying a chain in some network.

In the above notations, the identifier *Xany* means any variable name which is an identifier starting with the letter *X*. This rule discriminates variables from non-variables. As a consequence, except the basic queries (query an entity or a predicate), a B-SPARQL query is always attached with a type symbol such as *T, S, N* etc. Types can be nested. For example *?S(S(T(X)))* queries a power triple set variable *X*. In the above list limitation of type constructions is specified for some types. Nevertheless we cannot exhaust all possible data types at this place because the remaining type construction possibilities are still unlimited. Of course not any type construction is meaningful. For details of *?N(Xany)*, *?P(Xany)* and *?C(Xany)* see sections IV and V.

The second facility we introduce in B-SPARQL is knowledge embedding in digital vectors as an independent data type, which is characterized by the notation *?E*(anytype) mentioned above and can be combined with other data types to form higher level data structures. This is a technique which has been already introduced by some authors for increasing the power of SPARQL [19-22], but not yet integrated into SPARQL as an independent language (at least to the knowledge of us). In this paper we introduce embedding technique and vector translation technique systematically in B-SPARQL and make them enjoying an equal place as other data types have. Symbolic processing of BK is very inefficient if not totally intractable. Recently the knowledge embedding technique has made a big advancement. In B-SPARQL we extend the querying domain from symbolic and numerical space to digital vector spaces where many queries unsolvable in the former space can be resolved in the later space. In this way digital vectors become one of the new data types. *?E(T(X))* queries a digital vector variable of an embedded triplet over the embedding space.

The third facility is a specialization of the first facility with regard to generalization of the triple concept which is twofold. On the one hand, we generalize the terminal (head and tail of a triplet) concept by allowing it to be any structured data type which generalizes the expression concept of SPARQL. An entity is only the simplest form of an expression.

For example, the triplet

?S(T(Y)) = union (?S(T(Y1)),?S(T(Y2)))

means the triplet set variable *?S(T(Y))* is equal with the union of two set variables *?S(T(Y1))* and *?S(T(Y2))*.

In particular, a terminal of a triple can itself be a triplet or sequence of triplets. Following is a legal triplet in B-SPARQL, which includes two sub-triplets.

{{snow is white.} is true.} is well-known.

On the other hand, we also generalize the predicate of a triplet to be a procedure call or an algorithm to allow the triple mode: Source data producer product data. For example, the triplet *(? X square 4.)* means calculating the square root of 4.

The fourth facility we introduce in B-SPARQL is the query procedure (or simply procedure) concept. This is very useful in complex query writing and recursive query construction.

The definition of a procedure has a procedure head and a procedure body. The procedure head starts with the delimiter 'Select ?' and the procedure name, followed by a parenthesized sequence of parameters separated by a semicolon, where the output parameters (i.e. call by result parameters) are before the semicolon, while the input parameters (i.e. call by value parameters) are after the semicolon. All parameters are variables written in their typed forms. In a procedure call the output parameters remain in their variable form, while the input parameters may be variables or general expressions as mentioned in last paragraph. A query procedure can be called by other procedures or by itself recursively. Both kinds of parameters may be empty. However the semicolon in the procedure head must be always present.

For examples of procedure definition and call see the next sections.

The fifth facility we introduce in B-SPARQL is the simplification of writing as much as possible. For the moment it includes the following:

Triplet simplification: Triplets with repeated predicates can be saved. Following is a legal triplet.

all (?X1, ?X2) = all (?X3,?X4).

It equals to four triplets: *?X1 = ?X3. ?X1 = ?X4. ?X2 = ?X3. ?X2 = ?*X4.

Procedure head simplification: Since the delimiter '? Select' is always heading a procedure definition or call, and since all variable names have a leading '*X*' letter, it is not necessary to put a question mark before the parameters in procedure head.

Since B-SPARQL can process structured knowledge at different levels, there is an option of using structured or elementary knowledge representation in a query. The former is simpler while the latter is more detailed. The structured representation of a query is also called a macro.

II. Evolvement of SPARQL and Related Works

SPARQL 1.0, the first version of SPARQL, was published in 2008[23]. Its current version SPARQL 1.1[24], published in 2011, is a powerful language recommended by the W3C, where many new features are introduced, such as grouping, aggregation, federated query, etc. More details of SPARQL 1.1 can be found in SPARQL by Example, a tutorial created by W3C SPARQL Working Group [25]. There are many implementations and support tools for SPARQL, such as the well-known Apache Jena [26].

After SPARQL has been widely accepted as a standard tool for knowledge query on RDF data bases, experts have been still busy in proposing various extensions, generalizations and specializations of SPARQL to get more advanced and easy to use knowledge querying tools on different kinds of KGs.

The first kind of new features introduced in SPARQL extended its knowledge structuring and operation abilities. At this place it worth mentioning the enrichment of SPARQL path operations leading to introduction of new SPARQL variations characterized by rich path operations. Examples include Path SPARQL, or simply PSPARQL[27], which introduces regular expression operators in SPARQL to process path patterns, SPARQLeR[28], extensions of PSPARQL which treat paths as first-class objects and GLEEN [29]with a more rich path expression syntax. Other added ingredients include data processing operations like Insert, update, and delete, more expression patterns for select operations, new aggregates and subqueries, etc.[30].

The second kind of advancing SPARQL is the extension of knowledge representation type it covers. The most meaningful extensions are towards extending knowledge representation in the Lee's pyramid framework [31]. The query scope of SPARQL has been extended from just RDF to RDFS [26], then to OWL[32], further to DL[33], and finally even to full text querying[34].

On the other hand, numerical processing of knowledge is introduced by incorporating embedding techniques in SPARQL, e.g. L. Zhang et.al developed a tool TrQuery by introducing embedding in SPARQL processing to allow approximate solutions when exact solutions are not available [19]. H. Kang et. al integrated embedding technique into SPARQL to optimize the performance of nearest neighbor (NN) search [20]. M. Kulmanov et. al introduced Vec2SPARQL to allow jointly querying vector functions with machine learning models within a single SPARQL query [21], while S. Kumari, R et. al just embedded the predicates of triples to make a more efficient search in information retrieval [22].

The third kind of renewing SPARQL is the introduction of a new language paradigm. Corby and Zucker developed an extension of SPARQL, called FunSPARQL [35], which extends SPARQL to a functional language on top of SPARQL filter expression language by adding a set of constraints expression facilities which enable programmers to define extension functions to empower the filter function of SPARQL. Considering the high cost of SPARQL application for large clusters data computing, Zadeh has developed distributed

SPARQL computing facilities [36] to parallelize the query processes.

The final extension of SPARQL is to extend its application domain. There are extensions of SPARQL to temporal knowledge domains [37], to geo-spatial knowledge domains[38], and to temporal- spatial knowledge domains[39]. Andrejev and Risch developed an extension of SPARQL in scientific domains [40]. The new facilities of the proposed language SciSPARQL with its prototype implementation include expressions, numeric multi-dimensional arrays, user-defined functions, aggregate functions and function views. For applications such as Internet of Things there are always continuous streams of knowledge which need to be processed timely. C-SPARQL [41] and DESERT [42] are extensions of SPARQL for processing such streams.

To the best of our knowledge, after investigating the literature, we did not find research works towards BD and BK as we do in this paper.

III. QUERYING BIG KNOWLEDGE PROPERTIES

A. Big Knowledge Quantity

Example 3.1.1: Find *K* most frequent predicates from *M* largest knowledge areas:

```
Select ?KMmost(S(OPR(XPre, Ynum)); Yk, Ym, S(S(T(Ygl))))
Where{Call select ?Mmost(S(S(T(Ypart))); Ym, S(S(T(Ygl)))).
    Call select ?Kmost(S(OPR(XPre, Ynum)); Yk, S(T(Yloc)))
     where {?S(T(Yloc)) in ?S(S(T(Ypart)))}.}
   Group by ?XPre.
   Order by desc(?Ynum).
   Limit ?Yk.
Select ?Kmost(S(OPR(XPre, Ynum)); Yk, S(T(Yloc)))
Where {{?Xa ?XPre ?Xb.} in ?S(T(Yloc)).}
    Group by ?XPre.
    Order by desc (count(?XPre)).
    Limit ?Yk.
    ?Ynum := count(?XPre).
Select ?Mmost(S(S(T(Ypart))); Ym, S(S(T(Yglob))))
 Where { ?S(S(T(Ypart))) subsetOf ?S(S(T(Yglob))).}
    Order by desc(count(anyof(?S(S(T(Ypart)))))).
    Limit ?Ym.
```

The call statement is:

Call select ?KMmost(S(OPR(XPre, Ynum)); 10, 3, {DBpedia, YAGO, Wikidata}).

B. Big Knowledge Connectedness

The massive connectedness of BK does not only depend on the quantity on its connections (edges between knowledge elements), but also on the harmonic distribution of connections among its elements (concepts, entities, articles, etc.). There are two measures for evaluating the harmony of connectedness of BK elements [10]. The first measure calculates the harmony of its global connectedness which is the number of connected element pairs divided by the number of all element pairs. The second measure is the average value of local connectedness (VOLC) of all entities, where entity e's VOLC is the number n of e's neighbor pairs connected by a triplet divided by the total number m of e's all neighbor pairs. According to a calculation of Chen and Huang [10], there is a big difference of connection harmony with regard to Wiki like knowledge distribution [43] and KG like distribution [44]. The following two examples showcase the query algorithms of these two harmony measures.

Example 3.2.1: Calculate the global connectedness

```
Select ? UPR(X) (Count (?UPR(X)) as ?Xtotal1)
  Where { ?UPR(X) = (?X1, ?X2). ?X1 != ?X2.}
     Filter (?UPR(X) in ?C(Y)).
       Count (?UPR(X)) as ?Xtotal2.
       Xrate1 := ?Xtotal2 /?Xtotal1
```

Example 3.2.2: Calculate the average local connectedness

```
Select ?X3 (Avg (?Xrate1) as ?Xavgrate)
Where {Select ? UPR(X) (Count (?UPR(X)) as ?Xtotal1)
  Where { c-distance (?X1, ?X3) = 1. c-distance (?X2, ?X3) = 1.
?X1 != ?X2. ?UPR(X) = (?X1, ?X2).}
filter (c-distance (?X1, ?X2) = 1)
       Count (?UPR(X)) as ?Xtotal2.
      ?Xrate1 := ?Xtotal2 /?Xtotal1}
```

C. Big Knowledge Completeness

There are different definitions about knowledge completeness. In this paper we rely on the definition of Murphy [2] who has claimed that knowledge graphs may be incomplete with respect to predicates in triplets. For example, he has discovered that the 'profession' predicate of about 68% of persons mentioned in Freebase is missing.

Example 3.3.1: How much is the knowledge graph complete?

According to Murphy [2], the concept 'complete' is defined as follows: A knowledge graph is called complete with regard to some predicate *p'* if whenever there is a triplet *T = (e1, p', e2)*, then for any other subject *e3*, where *e1* and *e3* are in the same class, there should be a triplet *T2 = (e3, p', e4)*.

Further we define degree of completeness as the rate: number of predicates satisfying the completeness principle/number of all predicates. A B-SPARQL query calculating this rate is as follows:

```
Select ?count(distinct(?XP1)) as ?Xcomplete
```

?count(distinct(?XP2)) as ?Xtotal ?Xcomprate Where{ ?X2 ?XP1 ?Y2.

Where {?X1 ?XP1 ?Y1. ?X2 ?XP2 ?Y3. sameClass (?X1, ?X2) = true }}

Filter(STR(?X1) != STR(?X2))

?Xcomprate = ?Xcomplete/?Xtotal.

IV. Querying Big Knowledge Structures

In B-SPARQL, the BK structures are mainly concerned with nets, paths, chains, sets, bags and their nested constructions. Their construction (nesting) rules have been specified in section I.B. We explain some of the semantics of these structured notations below.

?N (T(X)) queries a triple network *X* with entities as nodes and triplets as edges. Each edge is directed from head to tail of a triple. Two triples are connected if the first one's subject or object is equal to the second one's subject or object.

?P(T(X)) queries (finite) triple paths. Each path is a connected sequence of (from subject to object) uni-directional triplets. The length of a path is the number of triplets it contains. The p-distance between two entities is the length of the shortest path connecting them, if it exists.

?C (T(X)) queries (finite) triple chains. Each chain is a connected sequence of triplets. The length of a chain is the number of triplets it contains. The c-distance between two entities is the length of the shortest chain connecting them, if it exists.

?N (T(X)), *?P (T(X))*, *?C (T(X))* can be written as *?N(X)*, *?P(X)*, *?C(X)* for short.

A. Mining Minimal Paths and Chains

A triple net is maximal if there is no triple which is connected to it but does not belong to it.

Example 4.1.1: Given a maximal triplet net, find all minimal paths *?P(X)* between two terminals *e1* and *e2*. Following is the macro form of the query.

Select ?P(X)
Where {?P(X) head e1. ?P(X) tail e2. Length (?P(X)) = minimal.}

where head (tail) means first (last) terminal of ?P(X). In a compiling process, it will be translated to the following query procedure:

Select ? minepath (P (X);Y1,Y2)
Where {?Y1 = ?Y2.|| ?Y1 ?X1' ?Y2. || ?Y1 ?X1' ?X2. ?X2 ?X3' ?Y2.
|| ?Y1 ?X1' ?X2. ?X3 ?X4' ?Y2.
call Select ? minepath (P (Y);X2,X3).}

A call of this procedure with real parameters *e1* and *e2* look like:

Call Select ? minepath (P (X); e1, e2).

Example 4.1.2 : Given a triplet net, to find are all minimal chains *?C(X)* between two entities *e1* and *e2*. This is much more complicated and tedious than finding the paths, since a chain is only a pseudo-path such that each edge (triple) in it can take any direction (forwards or backwards). Following is a macro for this task.

Select ?C(X)
Where {?C(X) terminal e1. ?C(X) terminal e2.}

Following is the same query in detailed form which makes use of query procedure definition and a function 'undirect' producing a reversed form of a triplet as its alternative. More exactly, undirect *(?X1, ?X2', ?X3) = {?X1, ?X2', ?X3 || ?X3, ?X2', ?X1}*.

Select ? minechain (C(X);Y1,Y2)
Where {T1|| T2 T3 || T4 T5 call Select ? minechain (C(X);X2,X3) }

Above we use simplified notation to avoid very tedious writing, where

T1 = {?Y1 ?X1' ?Y2.} T2 = {?Y1 ?X1' ?X2.} T3 = {?X2 ?X4' ?Y2.}
T4 = {?Y1 ?X1' ?X2.} T5 = {?X3 ?X5' ?Y2.}

Example 4.1.3: Find all maximal triplet nets *N(T(X))*, where ?C(Y) and ?C(Y1) mean querying a triplet chain *Y* or *Y1* respectively.

Select ?N(T(X))

Where { all (?X1, ?X2) inside ?N(T(X)). ?X1 != ?X2.

all (?X1,?X2) inside ?C(Y). ?C(Y) inside ?N(T(X)).

Filter not exists { all (?X1, ?X3) inside ?C(Y1).

Where {?X3 outside ?N(T(X)).} }}

B. Mining Frequent Subnets

The following example finds most frequent subnets from a set of nets.

Example 4.2.1: Given a set *S(N(U))* of nets. The query *?N(T(X))* finds all frequent subnets contained in *S(N(U))* and counts their numbers. We define *X* is a frequent subnet only when that number exceeds 3/10 of size (*U*).

Select ?N(T(X)) (count(?N(Y)) as ?Xnum)

Where{ ?N(Y) in S(N(U)).

?N(T(X)) partof ?N(Y). }

Group by (?N(T(X))) }

*Filter (?Xnum > 3*size(U)/10)}*

Order by desc(?Xnum)

V. Querying Complex Network Properties

There are two well-known complex network models: the small world model [45] and the scale free model [46]. In this section we only discuss the small world model. In this model it is claimed that the distance of most pairs of nodes in the network is very small, usually no more than six (?!).

Considering a knowledge graph KG as a network, we want to check to what degree KG fulfils the small world property. This is done by calculating the average c-distance between any pair of randomly selected entities.

Example 5.1. We calculate the chain distance (instead of path distance) between two entities *e1* and *e2* since the edges may be forwards or backwards. Besides, it is not always meaningful to ask for distance between any *e1* and *e2*. For example *?distance (3,4)* is meaningless (not equal to 1!). Those entities which are likely to appear as subjects of triplets have the type ID (i.e. identifiers). We request that only distances between two ID type entities could be queried. The query below calculates the distance ?X of the entity pair *ID(Y1)*, *ID(Y2)* with initial value *?X1*.

Select ? c-distance (X ;ID(Y1),ID(Y2), X1).

Where { ?ID(Y1) = ?ID(Y2). ?X := ?X1.

|| all (?ID(Y1),?ID(Y2)) in ?T(Y). ?X := ?X1 +1.

|| all (?ID(Y1),?ID(Y3)) in ?T(Y). all (?ID(Y2),?ID(Y4)) in ?T(Y).

Call Select ? c-distance (X ;ID(Y3),ID(Y4), X1+2).

Select ? Ave-distance (Y4; Y5, Y6, Y7)

Where { ?Y5 > 0.

Call Select ? distance (X; Random (ID(Y), Y1) , Random (ID(Y), Y2), 0, Y),

Call Select ? Ave-distance (Y4; Y5-1, Y6, X+Y7, Y)}

?Y4 := (?X + ?Y7)/?Y6.

Thus, if we want to calculate the average value of shortest distances among 100 randomly selected entity pairs, the call should be:

Call Select ? Ave-distance (Y4; 100, 100, 0).

In the 'Ave-distance' procedure the result parameter *Y4* is the average value of entity pair distances, while *X* is the calculated distance of a single pair of entities. The input parameters *Y5* and *Y6* are both the number of tested entity pairs, where *Y5* is used for loop time counting while *Y6* is used for final average computing, *Y7* is the accumulated distance values during computation.

A calculation of average entity pair distance is done in experiment 7.3 for three KGs. The result is shown in Table IV.

VI. B-SPARQL's Semantic Query of Big Knowledge

A good approach of dealing with knowledge semantics is to embed the original knowledge representation to a digital vector space and to do computation in this space instead of doing reasoning in the original symbolic space. The knowledge vectors in the new space then can be dealt with as the semantics of the original knowledge representation. It's resorting on embedding techniques that calculating approximate or heuristic results with risk evaluation becomes possible. In a previous paper [47], we have applied the embedding technique to data items at a higher level than entities, such as triple level, snapshot level, topic level, etc. In this paper we generalize it further to more complex data structures including paths, nets, trees, etc.

An embedding based technique for knowledge evaluation is a model that makes use of an arithmetical (or more complex) relation between subject s, predicate p and object o of a triple [48]. In these models, a triple is considered as a translation in the vector space. As result of the rule 'translating s with p to get o', the translated s should be close enough to o if the algorithm is good enough and if the knowledge represented by the triplet is precise enough. This view provides insight into the semantics of knowledge graphs. Some examples of these models are TransE [49], TransR [50] and TransH [51].

The most well-known example of applying embedding technique to KG manipulation is knowledge prediction and completion. It is often used to completing a triplet by predicting its missing terminal. However in this section we will only propose a solution for knowledge classification with embedding.

Example 6.1: Knowledge Classification

We assume similar predicates represent similar knowledge. Thus to classify triples roughly equals to classify their predicates. In the 'Classify' query below, sub-query 'Collect' gathers all predicates *S(XPr)* of the KG *S(T(X))* and put them in descending order. It further selects from *S(XPr)* a subset *S(XMpr)* of *?Xmany* most frequent predicates. The sub-query 'Embed' embeds the both predicate sets in two vector sets *S(E(XPre))* and *S(E(XMpre))* respectively. The sub-query 'Cluster' divides *S(E(XPre))* in *?Xmany* vector clusters, where each cluster contains a frequent (predicate) vector. Each vector *v* of *S(E(XPre))* belongs to that cluster where the distance between its frequent vector and v is minimal among all frequent vectors. At last the sub-query 'Reverse' transforms the vector clusters back to predicate clusters. Then it builds *?Xmany* triplet clusters, where each triplet belongs to that cluster to which its predicate belongs. To save space we only give a list of query procedure calls as summary.

Select ? Classify (S(S(T(Y)));S(T(X)), Xmany)

Where {Call Select ?Collect (S(XPr), S(XMpr); S(T(X)), Xmany)

Call Select ?Embed (S(E(XPre)), S(E(XMpre)); S(XPr), S(XMpr))

Call Select ? Cluster (S(S(E(Xres)));S(E(XPre)) , S(E(XMpre)),Xmany)

Call Select ? Reverse (S(S(T(Xres))); S(S(E(Xeres))));

VII. B-Spa1.0: Semantics, Implementation and Experiments

B-Spa1.0 is the first implementation of B-SPARQL. It is written in PYTHON [52] language and makes use of the open-source PYTHON library RDFLib [53]. B-Spa1.0 extends RDFLib by introducing new modules to implement new functions of B-SPARQL. These modules partly enrich the original functions and partly implement brand new functions, such as structured data types, query procedures, embedding functions, etc. Following are the results of four experiments which are typical in B-SPARQL service.

Due to the above reason, B-Spa1.0 makes use of a transformational semantics. The syntax of B-Sparql can be divided in three parts. The first part contains B-Sparql's query clauses in identical Sparql form. They will be submitted to RDFLib Python interpreter without any change. The second part is a set of B-Sparql's concise representations of Sparql's rather complex compound query clauses. They will be first transformed to Sparql form and then submitted to the interpreter. The third part contains new semantic primitives such as structured data types and embedded data types. New Python routings have been written to interpret them in run time. Figure 1 shows the mechanism of this transformational semantics, where B to S means transforming B-Sparql to Sparql, S(B) by P means Sparql (B-Sparql) interpreted by Python.

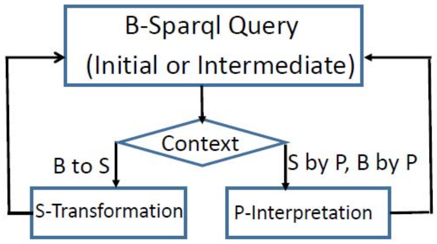

Fig. 1 Transformational Semantics of B-Sparql

Experiment VII.1 (Example 3.1.1)

In Table II we only list three most frequent predicates of each KG. The global result is (W1, W2, W3, Y1, Y2, D1, Y3, D2, D3). Roughly Wikidata > YAGO > DBpedia, except that D1 > Y3.

TABLE II. THREE MOST FREQUENT PREDICATES AND THEIR FREQUENCIES OF 3 KGS

DBpedia	YAGO	Wikidata
http://www.w3.org/1999/02/22-rdf-syntax-ns#type 113715836	http://www.w3.org/2000/01/rdf-schema#comment 139236795	http://schema.org/description 2205858413
http://www.w3.org/2002/07/owl#sameAs 33623686	http://www.w3.org/2000/01/rdf-schema#label 137296236	http://www.w3.org/1999/02/22-rdf-syntax-ns#type 1155990655
http://purl.org/dc/terms/subject 23990492	http://www.w3.org/1999/02/22-rdf-syntax-ns#type 55748756	http://wikiba.se/ontology#rank 1049614942

A. Experiment VII.2 (Example 3.3.1)

The data in Table III is the result of querying the KG NELL [54] for evaluating the degree of its predicate completeness. Table III shows its number of predicates and complete predicates and the completeness rate.

TABLE III. NELL'S COMPLETENESS OF PREDICATES

# predicates	# complete predicates	Completeness rate
828	355	355/828=0.429

B. Experiment VII.3 (Example 4.1.2 and Example 5.1)

To calculate SW property of KG, we randomly select 100 pairs of entities from each of the KGs DBpedia, Orpha [55] and Rhea[56] and calculate their average distances (AVGD) of entity pairs. Table IV shows the AVGD of each KG and their small world rate (6/AVG distance). Table IV shows that all three KGs have a good small world property (>=1). This may be a characteristic of big KGs.

TABLE IV. SMALL WORLD RATE OF THREE KGS

	DBpedia	Orpha	Rhea
Size	438336346	2296871	3291266
Average distance	3.32	2.24	1.95
S.W. rate	1.807	2.679	3.077

C. Experiment VII.4 (Example 6.1)

To do knowledge classification, we apply the query procedure ?Classify to DBpedia with Word2Vec and *?Xmany = 3*. Table V shows the result (how many 'semantically similar' predicates and triplets are in each cluster). Fig. 2 shows the vector clusters.

TABLE V. CLASSIFY DBPEDIA IN THREE CLUSTERS

DBpedia	type	sameAs	subject
# predicates	686	1191	453
# triplets	130107097	59400920	36419711

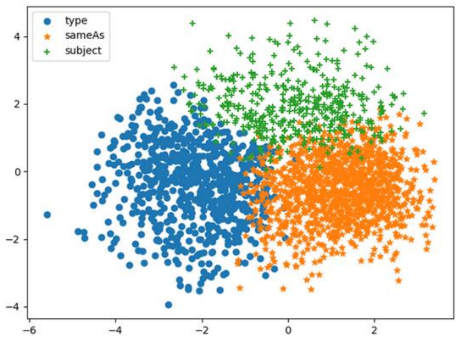

Fig. 2 Distribution of 3 clusters

I. FUTURE WORKS

This paper is only the first report of our work on BK query language. Lots of things are yet to be done. To name a few, a more rigorous syntax, a formal semantics, an optimized implementation, some practical applications, etc. are being looking forward to.

REFERENCES

[1] X. Wu, X. Zhu, G. Wu, W. Ding, Data mining with big data. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(1): pp.97-107

[2] K. Murphy, From big data to big knowledge, in Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage., 2013,

[3] A. Russell, Turning big data into big knowledge. http://socialscience.ucdavis.edu/iss-journal/features/turning-big-data-into-big-knowledge, 2018.

[4] J. Aaron, Turning big data into big knowledge, https://thestack.com/big-data/ 2015/12/07/turning-big-data-into-big-knowledge/, 2018.

[5] J. Liebowitz, "How to extract big knowledge from big data?" https://www.sas.com/en_us/insights/articles/big-data/big-knowledge-big-data.html, 2018.

[6] Hershkovitz, https://www.linkedin.com/pulse/analysttoolkit-from-big-data- knowledge-dr-shay-hershkovitz, 2016.

[7] X.Wu, H.Chen, G.Wu, J.Liu, Q.Zheng, X.He, A.Zhou, Z.Zhao, B.Wei, Y.Li, Q.Zhang, S.Zhang, R.Lu, N.Zheng, Knowledge engineering with big data. IEEE Intelligent Systems, pp. 46-55, 30(5), 2015.

[8] X.Wu, J. He. R.Lu, N.Zheng, From Big Data to Big Knowledge: HACE + BigKE, ACTA AUTOMATICA SINICA, pp. 965-982, 42(7), 2016.

[9] X. Wu, H. Chen, J. Liu, G. Wu, R. Lu, N. Zheng, Knowledge Engineering with Big Data (BigKE): A 54-Month, 45-Million, 15-Institution National Grand Project, IEEE Access, Vol 5, pp.12696-12701, 2017.

[10] R. Lu, X. Jin, S. Zhang, M. Qiu, X. Wu, A Study on Big Knowledge and Its Engineering Issues, IEEE Transactions on Knowledge and Data Engineering, Vol: 31(9), pp.1630-1644, 2019.

[11] M. Wu, X. Wu, On big wisdom, Knowledge and Information Systems 58: pp. 1–8, 2019.

[12] Explore Freebase data, http://www.freebase.com, 2013.

[13] A. Akesson, Google my business profiles start ranking in nonbranded searches, https://www.venndigital.co.uk/blog/ google-my-business-profiles-startranking, 2018.

[14] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy,T. Strohmann, S. Sun, and W. Zhang, Knowledge vault: A web-scale approach to probabilistic knowledge fusion, 20th ACM SIGKDD, pp. 601–610,2014.

[15] Y. Jia, Y. Wang, X. Cheng, X. Jin, and J. Guo, OpenKN: An open knowledge computational engine for network big data, IEEE/ACM Int. Conf. Adv. Soc. Netw. Anal. Mining, pp. 657–664, 2014.

[16] Probase, https://www.microsoft.com/enus/research/project/probase/

[17] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, G. Weikum, YAGO2: Exploring and querying world knowledge in time, space, context, and many languages, Int. Conf. Companion WorldWideWeb, pp. 229–232, 2011.

[18] "OpenCyc" [Online]. Available: http://www.baike.com/wiki/OpenCyc.

[19] L. Zhang, X. Zhang, Z. Feng, TrQuery: An embedding-based framework for recommending SPARQL queries, arXiv:1806.06205v1 [cs.DB] 16 Jun 2018.

[20] H. Kang, S. Hong, K. Lee, N. Park, S. Kwon, On integrating knowledge graph embedding into SPARQL query processing, (MSIP) [No. CRC-15-05-ETRI].

[21] M. Kulmanov, S. Kafkas, A. Karwath, A. Malic, G. V. Gkoutos,M. Dumontier, R. Hoehndorf, Vec2SPARQL: integrating SPARQL queries and knowledge graph embeddings, bioRxiv preprint first posted online Nov. 7, 2018.

[22] S. Kumari, R. Pandey, A. Singh, H. Pathak, SPARQL: Semantic information retrieval by embedding prepositions, International Journal of Network Security & Its Applications (IJNSA), Vol.6, No.1, 2014.

[23] SPARQL 1.0 (Jan 2008), hMp://www.w3.org/TR/rdf-- ‐ sparql-- ‐ query/

[24] SPARQL 1.1 (March 2013) hMp://www.w3.org/TR/sparql11-- ‐ overview

[25] Feigenbaum, E. Prud'hommeau, Sparql by Example, Cambridge Semantics, https://www.cambridgesemantics.com/blog/semantic-university/learn-sparql/sparql-by-example/

[26] hMp://jena.apache.org

[27] PSPARQL grammar, Inria, 2006, 2008, http://exmo.inrialpes.fr/software/psparql/psparqlgrammar.html

[28] K.J.Kochut, M.Janik, SPARQLeR: Extended Sparql for Semantic Association Discovery, ESWC 2007. LNCS 4519.

[29] GLEEN: Regular Paths for ARQ SparQL, Structural Informatics Group, University of Washington, http://sig.biostr.washington.edu/projects/ontviews/gleen/index.html.

[30] Sparql extensions, W3C-Wiki, https://www.w3.org/wiki/SPARQL/Extensions.

[31] E.Bernes-Lee, J.Hendler, O.Lassila, The Semantic Web, Scientific American, May 2001.

[32] OpenLink Virtuoso, W3C Sematic Web, https://www.w3.org/2001/sw/wiki /OpenLink_Virtuoso.

[33] E.Sirin, B.Parsia, Sparql-DL, Sparql query for OWL-DL, http://www.doc88.com/p-070396196293.html

[34] A. K. Yassine Settouti, et. al. SPARQL-Based Full-Text Search Add Filter Function (STAFF), WSCAR'2014.

[35] O. Corby, C. F. Zucker, FunSPARQL: Extension of SPARQL with a Functional Language, RR-8814, Inria Sophia Antipolis; I3S. Hal-01236947, 2015.

[36] R. Zadeh, Distributed Computing with Spark, Institute for Computational and mathematical Engineering at Stanford University.

[37] F.Grandi, T-SPARQL: A TSQL2-like Temporal Query Language for RDF[C]// Local Fourteenth East-european Conference on Advances in Databases & Information Systems. DBLP, 2010.

[38] M. Perry and J. Herring, OGC GeoSPARQL - A Geographic Query Language for RDF Data, Version 1.0, Open Geospatial Consortium, 2012.

[39] G.Garbis, K.MPereta, et. al., An implementation of a temporal and spatial extension of RDF and Sparql on top of MonetDB phase-1, TELEIOS FP7-257662 project.29th Feb. 2012.

[40] A. Andrejev, T. Risch, Scientific SPARQL: Semantic Web Queries over Scientific Data, Uppsala University, 2011

[41] D.F.Barbieri, D.Braga, S.Ceri, E.D.Valle, M.Grossniklaus, C-SPARQL: A CONTINUOUS QUERY LANGUAGE FOR RDF DATA STREAMS, International Journal of semantic computing, 4(1), 3-25, 2010.

[42] Farah Erim, I.Lytra, DESERT: A Continuous SPARQL Query Engine for On-Demand Query Answering, International Journal of semantic computing, 12(3), 373-397, 2018.

[43] X. Huang, On the harmonic connectivity of knowledge graphs, Acad. Math. Syst. Sci., Chinese Acad. Sci., Beijing, China, Tech. Rep. TR-2018-02-07, 2018.

[44] G. Chen, On the harmonic connectivity of Wikipedia articles, Institute of Computing Technology, Chinese Acad. Sci., Beijing, China, Tech. Rep. TR-2018-02-09, 2018.

[45] M. E. J. Newman, Models of the Small World, A Review, arXiv:cond-mat/0001118v2 [cond-mat.stat-mech] 9 May 2000.

[46] A.-L. Barabási, R. Albert, Emergence of Scaling in Random Networks, SCIENCE VOL 286 1999.

[47] R. Lu, C. Fei, C. Wang, S. Gao, H. Qiu, S. Zhang, C. Cao, HAPE: A Programmable Big Knowledge Graph Platform, Information Sciences, 509 (2020) 87–103.

[48] Q.Wang,Z.Mao,B.Wang,L.Guo, Knowledge Graph Embedding: A Survey of Approaches and Applications, pp.2724 - 2743, 29(12), 2017.

[49] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko. Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, pp. 2787–2795, 2013.

[50] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. 29th AAAI, pp.2181-2187, 2015.

[51] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. 28th AAAI, pp. 1112–1119, 2014.

[52] T.E. Oliphant, Python for scientific computing, Comput. Sci. Eng. 9 (3) (2007) 10–20.

[53] D. Krech, Rdflib: a python library for working with rdf, https://github.com/RDFLib/rdflib (2006).

[54] Mitchell T, Cohen W, Hruschka E, et al. Never-ending learning[J]. Communications of the ACM, 2018, 61(5): 103-115.

[55] Orphadata: Free access data from Orphanet. © INSERM 1999. Available on http://www.orphadata.org. Data version (XML data version).

[56] T. Lombardot, A. Morgat, K. B. Axelsen, L. Aimo, N. H. Nouspikel, A. Niknejad, A. Ignatchenko, I. Xenarios, E. Coudert, N. Redaschi, A. Bridge, Updates in Rhea: SPARQLing biochemical reaction data, Nucleic Acids Research, 47(D1), pp. 596–600, 2019.