INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research

International Journal of Metadata, Semantics and Ontologies

ISSN online: 1744-263X - ISSN print: 1744-2621
https://www.inderscience.com/ijmso

Pre-processing of RDF data for METIS partitioning

Siham Benhamed, Safia Nait-Bahloul

DOI: 10.1504/IJMSO.2023.10059687

Article History:

Received:	06 December 2022
Last revised:	08 June 2023
Accepted:	25 July 2023
Published online:	05 December 2023

Copyright © 2023 Inderscience Enterprises Ltd.

Pre-processing of RDF data for METIS partitioning

Siham Benhamed*

LITIO Laboratory,
University Oran 1, Ahmed Ben Bella,
Oran, Algeria
and
Department of Computing and Mathematics,
University Abdelhamid Ibn Badis – Mostaganem,
Mostaganem, Algeria
Email: siham.benhamed@univ-mosta.dz
*Corresponding author

Safia Nait-Bahloul

LITIO Laboratory,
Department of Computer Science,
University Oran 1, Ahmed Ben Bella,
Oran, Algeria
Email: nait-bahloul.safia@univ-oran1.dz

Abstract: The partitioning of RDF data on a large scale allows generating a set of RDF data subgraphs. METIS is a graph partitioning technique that minimises the cost of partitioning. METIS applies, among other things, to RDF graphs. However, the semantics introduced in the description of RDF data is not taken into account in the partitioning process in METIS. For this, we propose in this paper a step of pre-processing RDF data before partitioning these data. The objective of this step is to improve the quality of semantic partitioning of RDF graphs. The evaluation of the RDF pre-processing step for METIS was performed on real and synthetic data.

Keywords: RDF data; RDF graph; pre-processing; graph partitioning; METIS.

Reference to this paper should be made as follows: Benhamed, S. and Nait-Bahloul, S. (2023) 'Pre-processing of RDF data for METIS partitioning', *Int. J. Metadata Semantics and Ontologies*, Vol. 16, No. 2, pp.152–171.

Biographical notes: Siham Benhamed is a PhD student working as a Teacher at University of Abdlhamid Ibn Badis – Mostaganem, Algeria. The PhD program, she followed her core courses in Computer Science at The Faculty of Exact and Applied Sciences of Oran 1 Ahmed Ben Bella University, Algeria. She received her Diploma of Engineer degree in Computer Sciences and a Magister diploma from Oran 1 Ahmed Ben Bella University. From these studies, she developed an interest in the performance optimisation in the web semantic domain. Currently, she is working on a Research Project within the LITIO Laboratory at Oran 1 Ahmed Ben Bella University, Algeria, that explore the benefit of the graph partitioning of the RDF data to improve the quality of the partitioning of graphs RDF at the semantic level.

Safia Nait-Bahloul obtained her Doctorate degree in Computer Science from the University of Oran. Since 2011, she has been a Member of the LITIO Laboratory at the University of Oran, which was accredited in 2009. She manages a Research Team in the LITIO Laboratory on Data Engineering and Web Technology. Since 2008, she has also been responsible for an academic Master's degree in Information Systems and Web Technology. Her research interests include advanced aspects of databases, web technology and unsupervised classification. Her work has been published in several journals and conference proceedings. She has supervised several Doctoral and Master's candidates and Undergraduate Projects in the field of information research, clustering, MDA and security (access control).

Copyright © 2023 Inderscience Enterprises Ltd.

1 Introduction

The interest in RDF data partitioning is due to the growth of data on the web through the Resource Description Framework (RDF) (W3C, 2014a). Data partitioning allows the processing of large RDF data volumes at the scale of big data (Ragab et al., 2021) which is more difficult due to the size, heterogeneity and additional complexity provided by RDF reasoning. To overcome the challenge of the massive RDF data size, the data are represented by graphs. This data representation form allows keeping the dependency between data to formally deduce new knowledge from the existing ones. For this, the knowledge bases of the semantic web are generally represented and expressed by RDF triples, constituting large graphs that formally describe sets of web resources. Thus, this representation provides an intuitive interpretation of the resources and the relationships between the different resources. However, processing and analysing large RDF data graphs and quickly accessing relevant data from these graphs is more difficult and complex both in terms of execution time and in terms of the memory size. For this reason, graph partitioning techniques for partitioning RDF data have been introduced to improve their large-scale processing. These techniques allow large data sets to be distributed on different sites in order to deploy distributed and parallel architectures (Kernighan and Lin, 1970). These techniques are used when the distance between all the data in the set to be partitioned is unknown (Buluc and Hao, 2016).

Graph partitioning by the multilevel method (Barnard and Simon, 1993) has been used for RDF data through METIS (Karypis and Kumar, 1998b) which produces balanced partitions whose size is similar and the communication cost between them is minimised. However, in the semantic web context, METIS doesn't support the preservation of the graph semantics during the web data graph partitioning. Therefore, the ontology or vocabularies intended to structure RDF resources in a partition miss semantics and do not correspond to the basic definition of an ontology (Simperl et al., 2011). To do this, we propose in this paper an algorithm which allows to extend the METIS partitioning to preserve data semantics during RDF graph partitioning. We add an upstream RDF data processing phase in order to improve the partitioning quality, in particular by minimising the semantic communication rate between the different partitions. We propose an illustrative example of our approach on the partitioning result.

This paper is organised as follows: Section 2 presents the state of the question, Section 3 presents the RDF graph partitioning by METIS which we illustrate by an example of its progress and its limits. We present in Sections 4 and 5 our framework for preprocessing RDF for METIS partitioning with a detailed scenario. Section 6 presents the evaluation results of our approach and we end this paper with a conclusion in Section 7.

2 Related works

The use of METIS through its multilevel algorithm in RDF data processing systems has revealed that it is effective for RDF data graph partitioning. These systems require the execution of an additional upstream RDF data processing phase to improve their large-scale processing. The purpose of this phase is to convert the input RDF data into a graph with an adequate format with METIS (e.g. Soma and Prasanna, 2008; Huang et al., 2011; Slavov et al., 2012; Wang and Chiu, 2012; Lee and Liu, 2013a, Lee and Liu, 2013b; Lee et al., 2013; Zhang et al., 2013; Gurajada et al., 2014; Rakhmawati et al., 2014; Bok et al., 2019; Ramesh et al., 2021; Priyadarshi and Kochut, 2022). These systems execute additional operations to support the partitioning of the RDF graph by vertex while preserving the partitioning quality. These additional operations applied on the RDF data constitute the pre-processing phase.

The operations found include the deletion of data represented by (i) tuples involving the schema elements of Soma and Prasanna's approach (2008), (ii) triples whose predicate is 'rdf: type' or having the type sense in H-RDF-3X (Huang et al., 2011), (iii) duplicate vertices and edges in each partition in VB-Partitioner (Lee and Liu, 2013b), (iv) literal vertices in TriAD (Gurajada et al., 2014) and (v) duplicate triples in SPA (Lee et al., 2013). The ignoring operation of the edges orientation in the graph during partitioning was applied in the method of Slavov et al. (2012) and the search for connected components of the RDF graph was introduced in the approach of Wang and Chiu (2012). The data regrouping such as (i) the subjects set with similar properties in an entity class in EAGRE (Zhang et al., 2013), (ii) the graph vertices in VB-Partitioner (Lee and Liu, 2013b), (iii) RDF triples in SHAPE (Lee and Liu, 2013a), (iv) the use frequency over queries in Bok et al. approach (2019) and (v) the type of triples in Granite (Ramesh et al., 2021). The preprocessing in the system of Rakhmawati et al. (2014) is based only on objects identified by URI. Thus, we find the indexing of RDF triples in PartKG2Vec (Priyadarshi and Kochut, 2022) and the generalisation of the entity graph from the adjacency list elements of the RDF graph in Galicia et al. approach (Galicia et al., 2019).

Most results obtained of these systems concerning pre-processing are interesting, but do not take into consideration of semantic aspects of RDF graphs. Note, that partitioning SHAPE method (Lee and Liu, 2013a) partially takes into account the semantic aspects, but not the rich structural aspects of RDF. URI-based data clustering results in non-semantic partitioning if many vertices share prefixes (Lee and Liu, 2013a).

3 METIS partitioning of RDF data

METIS (Karypis and Kumar, 1998b) is a tool that allows graph partitioning and consists of a set of algorithms based on the multilevel paradigm (Barnard and Simon, 1993). The multilevel algorithm is based on the vertices number reducing principle through the adjacent vertices pairs grouping. Thus, the graph reduction to be partitioned keeps the graph topological properties to provide access to a global view. The multilevel algorithm is formed from algorithms set based on multilevel dichotomous recursion (multi-k-way) (Karypis and Kumar, 1997). It is composed of three very distinct phases (Karypis and Kumar, 1998a) which are contraction, partitioning and refining.

METIS has been applied on several data structures, including RDF. To do this, the RDF data must be transformed into a graph. This operation is fundamental in the RDF data processing.

An RDF graph is the grouping of an RDF document statements set on a given vocabulary expressed by triples. It describes a model of binary predicates in first-order logic, of the form *Property (Subject, Object)*. The RDF graph is directed and labelled, in which the vertices represent the subjects and objects of a triple, and the edges describe the assertions labelled by the predicates. In this paper, we take some semantic web definitions that exist in the literature (Definition 3.1 and Definition 3.2) in order to apply them in our RDF data pre-processing operation.

Definition 3.1 (RDF triple): *Let U be the URI references set, B the blank nodes (anonymous resource) set and L the literals (strings) set, an RDF triple is the triple (s, p, o) with* $s \in (U \cup B)$ *is the subject,* $p \in U$ *is the predicate and* $o \in (U \cup B \cup L)$ *is the object. The RDF triple is represented by a directed graph with a labelled edge such that* $S \xrightarrow{P} O$.

Definition 3.2 (RDF graph): *An RDF graph is a non-empty set of RDF triples. It is a multigraph with multiple directed edges labelled by the RDF triples predicates, this graph contains the RDF triples set. Let an RDF graph denoted* $G = (V, E, \Sigma_E, L_E)$ *where* $V \subset (U \cup B \cup L)$ *is the vertices set in the graph that identifies the subject or object of an RDF triple. The set* $E \subset V \times V$ *is an oriented edges multiset. The pair* $(u, v) \in E$ *denotes an edge oriented from u to v.* $\Sigma_E \in U$ *is the edge labels set (the predicates) and* L_E *is the correspondence set of a label* $\rho \in \Sigma_E$ *and its edge (u, v), with* $L_E \subset E \times \Sigma_E$. *An RDF graph* G_0 *is a subgraph of an RDF graph G if* $G_0 \subseteq G$.

The *RDF graph* partitioning consists in assigning a set of the graph vertices describing a resource via RDF triples to different partitions constituting sub-graphs in such a way that the partition size is not greater than a given size. The graph partitioning allows to reduce the graph browsing time, especially for large size graphs. The objective of the *RDF graph* partitioning problem is to compute partitions by minimising the edges number 'predicate' or the sum of their weights that connect partitions, and to minimise the overlaps set of different partitions. For that, METIS expresses the partitioning result of a vertex set *V* into *k* partitions by a vector $P[v]$ of length *n* representing the generated partitions number. $P[v]$ is created by METIS such that for any vertex $v \in V$, $P[v]$ contains an integer *i* between *1* and *k*, indicating the vertex assignment *v* to the partition P_i. $P[v]$ represents the METIS partitioning result; i.e., it presents the assignment assertions set of each graph vertex *G* to a particular partition. This assignment determines a mapping function between a vertex and its partition.

The partitioning quality in METIS depends on the graph characteristics, it is measured by the partitions size (balance) and the cut edges number which guarantee the total cost reduction of communications between the partitions (*edge-cut*) (Karypis and Kumar, 1998b). The balance determines the partitions stability and is calculated by dividing the maximum number of nodes in a partition by the average number of nodes per partition (total number of nodes in the graph over the number of partitions). The communication cost represents the number of nodes $V_i \in V$ whose edges bound the partitions (crossing edges between partitions). The edge-cut represents the number of edges whose incident nodes belong to different partitions.

The partitioning quality in METIS requires both minimising the cut function and balancing the computational load between the processors by producing the disjoint part balances.

3.1 RDF data preparation

We propose in the following example a METIS execution application on RDF data. Let's consider an RDF document in Figure 1(a), consisting of 16 triples, including 12 subjects, 11 objects and 12 predicates to be partitioned by using METIS into two partitions. This Figure is formed by triples set of the form $< S >< P >< O >$ constituting the RDF document. The METIS partitioning of the RDF document requires its representation into a graph (see Figure 1(b)) in an intermediate step before the graph transformation into a METIS graph. The METIS graph requires a number-based input structure of numerical type. The graph resources (subject or object) are mapped from Figure 1(b) into integer, see Figure 1(c).

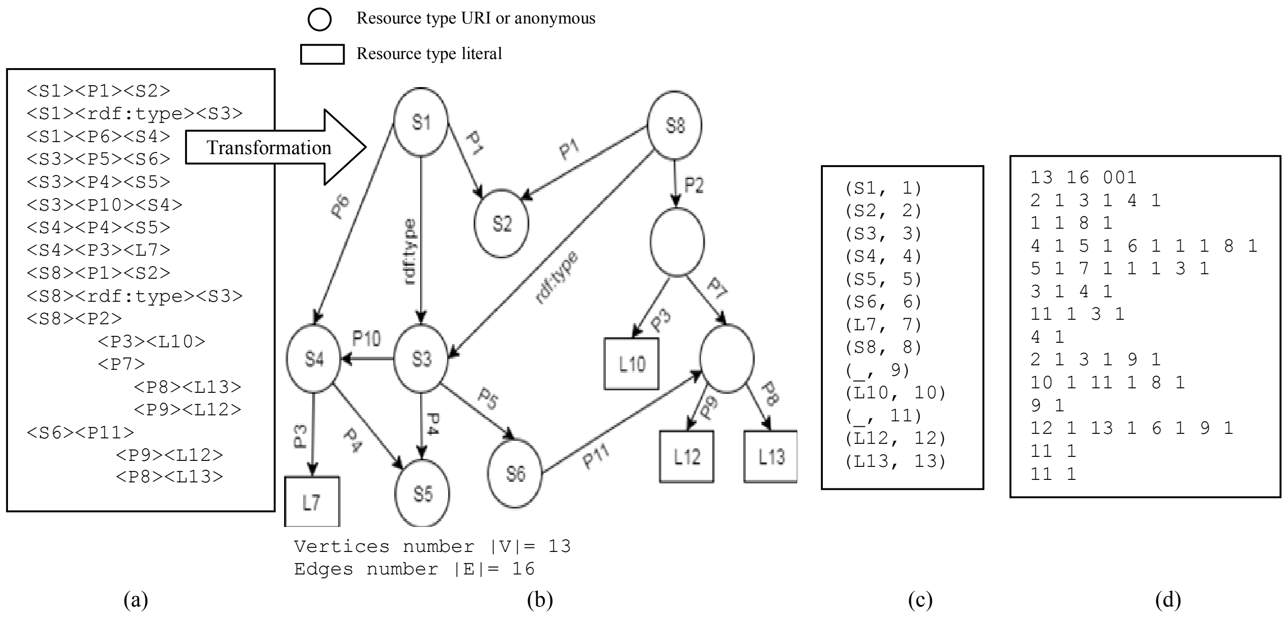
The METIS graph (see Figure 1(d)) is a text file that satisfies the graph representation form in METIS. The latter consists of *n*+1 lines, n being the graph order (see Figure 1(b)), the first line of which, called the header line, contains information on the graph size and the graph type, while the remaining n lines contain information on each graph vertex. They represent the adjacent vertices where the edges are weighted at 1. Figure 1(d) consists of 14 lines where the first line determines that the graph is composed of 13 nodes and 16 edges and that the graph is unweighted. For example, the second line content is 2 1 3 1 4 1, means that nodes 2, 3 and 4 are the adjacent nodes of node 1 and the edges weight is 1.

3.2 Application and interpretation of METIS

When METIS partitions Figure 1(d) into two partitions, it returns a file in which it assigns each node to a partition, the value 0 represents the node assignment to the first partition and the value 1 represents the node assignment to the second partition. The partitioning result is shown in Figure 2(a).

METIS achieves a balanced partitioning (see Figure 2(b)). Based on the metrics defined by Karypis and Kumar (1998b), the partitioning constraints provided by METIS in our example Figure 1(b) are: Edge-cut = 3, Communication volume = 5, Balance =1.077

Figure 1 Example of the RDF document by METIS (a) RDF document, (b) RDF graph, (c) Mapping and (d) METIS graph



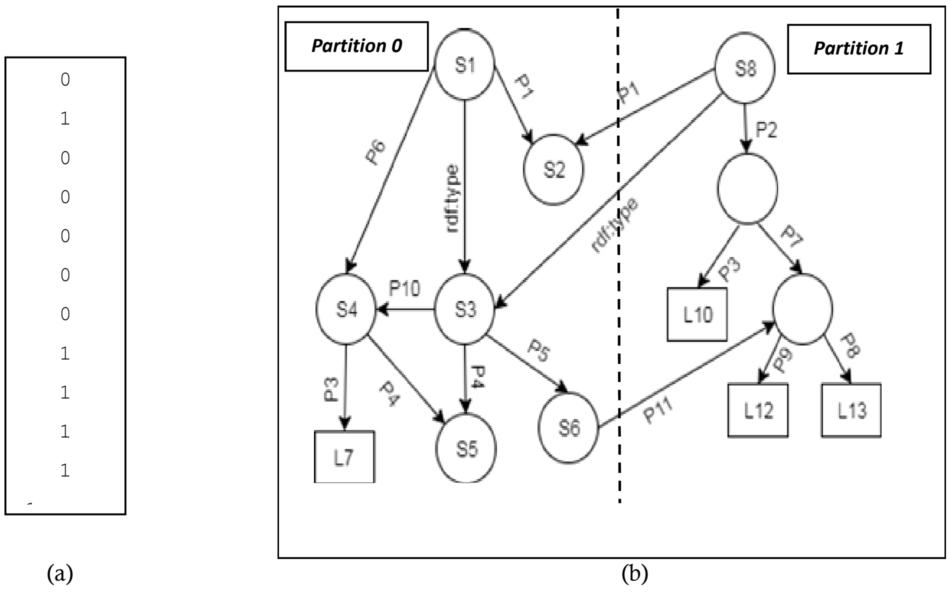
```
<S1><P1><S2>
<S1><rdf:type><S3>
<S1><P6><S4>
<S3><P5><S6>
<S3><P4><S5>
<S3><P10><S4>
<S4><P4><S5>
<S4><P3><L7>
<S8><P1><S2>
<S8><rdf:type><S3>
<S8><P2>
      <P3><L10>
      <P7>
         <P8><L13>
         <P9><L12>
<S6><P11>
        <P9><L12>
        <P8><L13>
```

```
Vertices number |V|= 13
Edges number |E|= 16
```

```
(S1, 1)
(S2, 2)
(S3, 3)
(S4, 4)
(S5, 5)
(S6, 6)
(L7, 7)
(S8, 8)
(_, 9)
(L10, 10)
(_, 11)
(L12, 12)
(L13, 13)
```

```
13 16 001
2 1 3 1 4 1
1 1 8 1
4 1 5 1 6 1 1 1 8 1
5 1 7 1 1 1 3 1
3 1 4 1
11 1 3 1
4 1
2 1 3 1 9 1
10 1 11 1 8 1
9 1
12 1 13 1 6 1 9 1
11 1
11 1
```

Figure 2 Partitioning of the example into 2 partitions by METIS (a) Partitioning result and (b) Graph partitioning



```
0
1
0
0
0
0
0
1
1
1
1
```

We note that in Figure 2(b), the cutting the edge labelled by rdf:type which is carrying semantics will cause a break in the second partition semantics since node S8 is an instance of class S3. This partitioning doesn't keep the initial semantics of the Figure 1(b) graph. In fact, METIS doesn't support the nature and type of intra-partition edges, and therefore it doesn't allow for keeping the link that defines the data semantics and the ontology concept. In addition, effects on inter-partition edges can change the meaning of the partitioned ontology. For example, partition 1 doesn't define the meaning of the initial defined ontology (see Figure 1). In order to improve this partitioning, we propose to keep the data semantics during the partitioning.

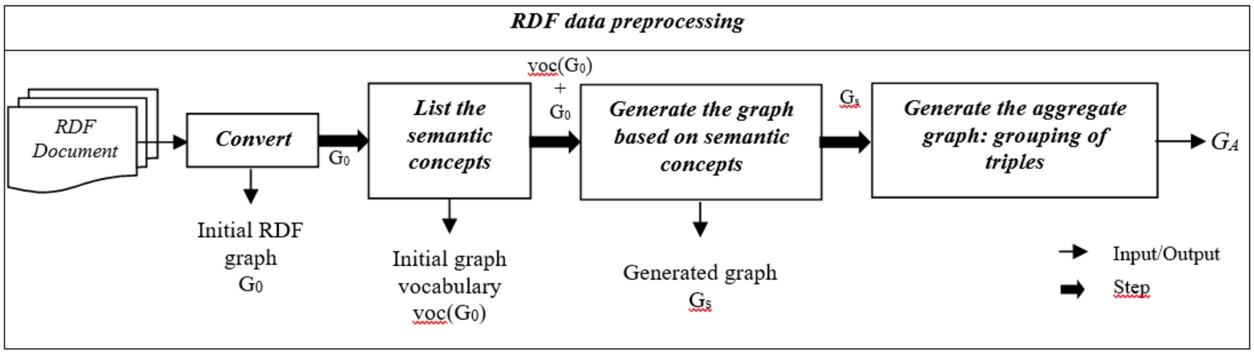
4 Framework for RDF pre-processing for METIS

RDF pre-processing for METIS aims at the semantic preservation expressed by the RDFS vocabulary (W3C, 2014b). To this end, we propose a framework (see Figure 3) for pre-processing RDF data to avoid the problem seen in the example in Figure 1 and Figure 2. This framework is defined by several ordered phases in which we inject semantic concepts set needed for pre-processing such as *RDF graph vocabulary*, *atom* and *term RDF*. RDF data semantic pre-processing consists of four phases including transforming the RDF document into the initial graph, maintaining the graph semantics, generating the graph and finally reducing it. The pre-processing RDF data algorithm is presented below.

Algorithm: RDF_Pre-processing

```
1. Input: doc_RDF
2. Output: graph_RDF $G_A$
3. Begin
4.     /*Convert RDF data to a graph
       $G_0 \leftarrow$ Conversion (doc_RDF)
5.     /* List the semantic concepts to generate the vocabulary of the initial graph $voc(G_0)$
       $voc(G0) \leftarrow$ Semantic_Concepts $(G_0)$
6.     /*Generate the $G_s$ graph based on the semantic concepts
       $G_s \leftarrow$ Generate_graph $(G_0, voc(G_0))$
7.     /*Generate the aggregate graph by grouping the triples
       $G_A \leftarrow$ Generate_aggregate_graph $(G_s)$
8. Return $G_A$
9. End
```

Figure 3 RDF data pre-processing framework


RDF data preprocessing
RDF Document
Convert
G0
Initial RDF graph G0
List the semantic concepts
Initial graph vocabulary voc(G0)
voc(G0) + G0
Generate the graph based on semantic concepts
Generated graph Gs
Gs
Generate the aggregate graph: grouping of triples
GA
Input/Output
Step

4.1 Conversion phase of RDF data into a graph

The first phase of the framework consists in converting the RDF document into the *initial RDF graph*. For this purpose, we propose the *Conversion* algorithm presented below. This algorithm considers an RDF document as input (line1) and creates as output an *RDF graph* (Definition 3.2) G_0 which consists of four sets V_0, E_0, Σ_{E0} and L_{E0} (line 4) representing respectively the nodes set, the edges set, the edge labels set and the set of correspondences between each edge and its appropriate label. After creating empty G_0 (line 4), we fill in the sets constituting the graph in question. For each triple (Definition 3.1) in the RDF document (lines 5 to 7), we assign the subject and object of the triple in the set V_0, the subject and object pair of the triple in the set E_0, the predicate of the triple in the set Σ_{E0} and the subject and edge pair with the predicate of the triple in the set L_{E0}. In other words, each RDF document triple (*s*, *p*, *o*) is converted into an edge $e \in E$ where $L_E(e) = p$ and $e = (s, o)$.

The graph G_0 is stored in a list according to a new structure proposed in our Definition 4.1, called the RDF graph resource list.

Algorithm: Conversion

```
1. Input: doc_RDF
2. Output: RDF_graph $G_0$
3. Begin
4. $G_0 \leftarrow$ Empty_RDF_graph $(V_0, E_0, \Sigma_{E0}, L_{E0})$
5. For each $(s, p, o) \in$ doc_RDF do
6.         $G_0 \leftarrow G_0 U(\{s, o\}, \{(s, o)\}, \{p\}, \{((s, o), p)\})$
7. End For each
8. Return $G_0$
9. End
```

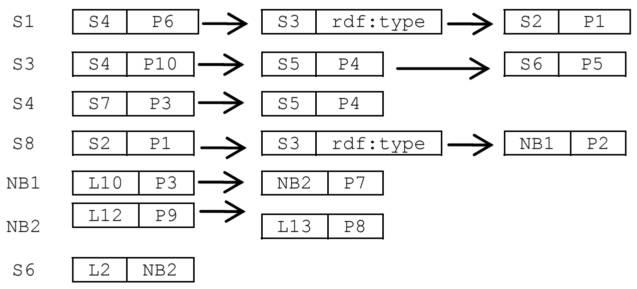
Definition 4.1 (RDF graph resource list): *the resource list L of the RDF graph G is a chained list of size $|V|$ that contains all adjacent vertices of each vertex $v_i \in V$ of G. It associates with each vertex v_i a list L_i containing a set of pairs of the adjacent vertex u with the edge label ρ such that*:

$L = \{L_i \ / \ i \in [1, \ |V|]\}$ and

$$L_i = \{(u,\rho)/u \in V, \rho \in \Sigma_E, (v_i,u) \in E, ((v_i,u),\rho) \in L_E\}$$

This list is determined according to the resource type to which we add the property value of a triple. The structure of the *RDF graph resource list* classifies the RDF document resources by triples (nodes of type subject index the adjacent nodes set which are of type object with the edges labels set which are of type predicate). We present in the Figure 4 below the *RDF graph resource list* appropriate of the example proposed in Figure 1(b).

Figure 4 RDF graph resource list of the example in Figure 1(b)



The space complexity of the *Conversion* algorithm is of order $O(|V| + |E|)$. It depends on the structure of the graph data description, which allows indicating the links between the vertices. In addition, the *RDF graph resource list* structure provides the incidence vertices list with linear complexity.

4.2 Listing phase of the semantic concepts

In the second phase of the *RDF data pre-processing* framework, we propose to inventory the graph semantic concepts in order to determine the graph elements that contribute to the resources semantic definition in order to determine the semantic graph. The *Semantic_Concepts* algorithm generates semantic concepts based on a set of notions that we propose to use. The semantic notions are: *semantic description* (Definition 4.2), *RDF vocabulary* (Definition 4.3), *RDF atom* (Definition 4.4), *RDF atom term* (Definition 4.5), *RDF atom vocabulary* (Definition 4.6), *RDF graph vocabulary* (Definition 4.7) and *semantic RDF graph* (Definition 4.8).

Definition 4.2 (Semantic description): *Let $(S \times P \times O)$ an RDF triples and P' is the predicate set of the specification of the RDF model syntax defined in the RDF schema such that $P' \subset P$ and P' = {Property, Class, subPropertyOf, subClassOf, domain, range, type}. A semantic description p is a predicate witch define the subject s as an instance of the object o, such that $p \in P'$.*

Definition 4.3 (RDF vocabulary): *Let $(S \times P \times O)$ is RDF triples set, the RDF vocabulary $(S' \times P' \times O')$ is a subset of (S,P,O) such that $S' \subset S$, $P' \subset P, O' \subset O$ and $\forall p \in P'$, p is a semantic description that semantically define an object $o \in O$ by the subject $s \in S$.*

Definition 4.4 (RDF atom): *Let an RDF document consisting of triples set $(S \times P \times O)$ such that S is the subject set, P is the predicate set and O is the object set, an RDF atom $A = (S' \times P' \times O')$ is an RDF vocabulary such that such that $S' \subset S$, $P' \subset P, O' \subset O$ and $\forall (s,p,o) \in (S' \times P' \times O')$, p is a semantic description.*

Definition 4.5 (RDF atom term): *Given an RDF atom A, $t(s)$, $t(p)$ and $t(o)$ each represents, respectively the subject atom term, the predicate atom term and the object atom term set such that:*

$$t(s) = \{s \ / \ s \in S', A = (S' \times P' \times O')\}$$

$$t(p) = \{p \ / \ p \in P', A = (S' \times P' \times O')\}$$

$$t(o) = \{o \ / \ o \in O', \ A = (S' \times P' \times O')\}$$

Definition 4.6 (RDF atom vocabulary): *Given $A = (S' \times P' \times O')$ the RDF atoms set and t(s) is the subject atom term set, the atom vocabulary $voc_i(A)$ is defined for each $s_i \in t(s)$ such that $voc_i(A) = \{(\{s_i\} \times P' \times O')/p \in P, o \in O$ and $(s_i,p,o) \in A\}$. The RDF atom vocabulary $voc_i(A)$ in an RDF graph represents the smallest sub-graph of semantic description to each subject atom term of $t(s)$.*

Definition 4.7 (RDF graph vocabulary): *Let $voc_i(A)$ is the RDF atom vocabulary set of each subject atom term of t(s), the vocabulary of an RDF graph G is $voc(G)$ such that $voc(G) = \{Uvoc_i(A)/\forall s_i \in t(s)\}$.*

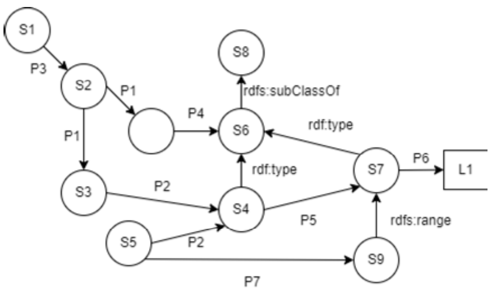
Definition 4.8 (Semantic RDF graph): *Given $A = (S' \times P' \times O')$ an RDF atom, a semantic RDF graph G' is an RDF graph generated by the RDF graph vocabulary, such that $G' = (V', E', \Sigma'_E, L'_E)$ where $V' \subset S'$, $E' \subset \{(S' \times O')\}$, $\Sigma'_E \subset P'$ and $L'_E \subset \{((S' \times O') \times P')\}$.*

We propose, in the Figure 5 below, an example of semantic concepts of an RDF document (see Figure 5(a)). The graphical representation of the document is shown in Figure 5(b). The corresponding basic semantic concepts are presented in Figure 5(c) and the appropriate graphical representations are shown in Figures 5 (d), 5(e), 5(f), 5(g) and 5(h).

Figure 5 Example of semantic concepts of an RDF document (a) RDF document, (b) Initial Graph G_0, (c) Corresponding basic semantic concepts, (d) voc(A1), (e) voc(A2), (f) voc(A3), (g) voc(A4) and (h) Semantic RDF graph *G*

```
<S1><P3><S2>
<S2><P1><S3>
<S2><P1>
      <P4><S6>
<S3><P1><S2>
<S3><P2><S4>
<S4><P5><S7>
<S4><rdf:type><S6>
<S5><P2><S4>
<S5><P7><S9>
<S6><rdfs:subClassOf><S8>
<S7><P6><L1>
<S7><rdf:type><S6>
<S9>rdf:range><S7>
```

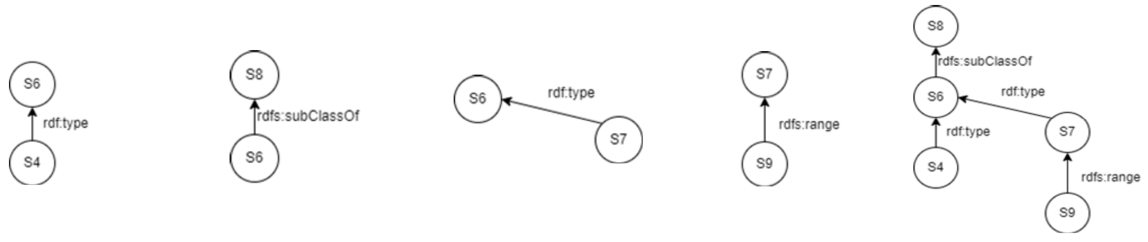
(a)



(b)

A = {(S4, rdf:type, S6), (S6, rdfs:subClassOf, S8), (S7, rdf:type, S6), (S9, rdf:range, S7)}
t(s)= {S4, S6, S7, S9}
t(p)= {rdf:type, rdfs:subClassOf, rdf:range }
t(o)= {S6, S8, S6, S7}
voc$_1$ (A) = {(S4, rdf:type, S6)}
voc$_2$(A) = {(S6, rdfs:subClassOf, S8)}
voc$_3$(A) = {(S7, rdf:type, S6)}
voc$_4$(A) = {(S9, rdf:range, S7)}
voc(G_0) = {(S4, rdf:type, S6), (S6, rdfs:subClassOf, S8), (S7, rdf:type, S6), (S9, rdf:range, S7)}

(c)



(d) (e) (f) (g) (h)

The *Semantic_Concepts* algorithm receives the *RDF graph* G_0 as input (line 1) and generates the *RDF graph vocabulary* $voc(G)$ as output (line 2). To do this, after initialising the sets of *RDF atoms* and *RDF atom terms* to empty (line 4). We search in all the graph for semantic labels (having an RDFS description), and for each of them we add to the set initialised before the determining elements (1) the nodes and edge labels set to the *RDF atoms* set (line 8), (2) the subject type nodes set to the *subject atom term* set (line 9), (3) the edge labels set to the *predicate atom terms* set (line 10) and (4) the object type nodes set to *object atom terms* set (line 11). After processing the whole graph, we generate $voc(A_i)$, the *RDF atom vocabularies* from the *RDF atoms* (lines 14 to 17). In lines 18 to 20, we generate the vocabulary of the initial graph $voc(G_0)$.

Algorithm: Semantic_Concepts

```
1.  Input: RDF_graph $G_0$
2.  Output: RDF_graph_vocabulary $voc(G_0)$
3.  Begin
4.  $A$, $t(s)$, $t(p)$, $t(o)$ ← {}
5.  For each $((v_i, v_j), \rho) \in L_{E0}$ do
6.      If est_RDFS(ρ) then
7.          For each $((v_i, v_j), \rho) \in L_{E0}$ do
8.              $A \leftarrow A \cup \{(v_i, \rho, v_j)\}$
9.              $t(s) \leftarrow t(s) \cup \{v_i\}$
10.             $t(p) \leftarrow t(p) \cup \{\rho\}$
11.             $t(o) \leftarrow t(o) \cup \{v_j\}$
12.         End For each
13.     End If
14. For each $v_i \in t(s)$ do /* Vary $v_i$ and consider all adjacent $v_j$
15.     $voc(A_i) \leftarrow \{\ \}$
16.     $voc(A_i) \leftarrow voc(A_i) \cup \{(v_i, \rho, v_j)\}$
17. End For each
18. $voc(G_0)$ ← Empty_ RDF_graph_vocabulary ()
19. $voc(G_0)$ ←union $(voc(A_i))$
20. Return $voc(G_0)$
21. End
```

4.3 Graph generation phase based on semantic concepts

The purpose of the graph generation phase in the framework is to reduce the initial graph size while preserving the semantic aspects. It consists of three consecutive operations including (1) removing *blank nodes* in G_0, (2) removing literal nodes in G_0 and (3) removing *atoms* based on *voc*(G_0). We give illustrative examples in each operation with their respective algorithm. Section 5 is for a scenario grouping all the framework phases (see Figure 3). For the first operation, we refer to the definition of the *blank node* (Berner-Lee, 1998) (Definition 4.8).

Definition 4.9 (Blank Node): *A Blank node also called an anonymous node or resource is a vertex v in the RDF graph* $v \in V$ *which is not identified by a URI such that* $v \in B$.

4.3.1 Removing blank nodes

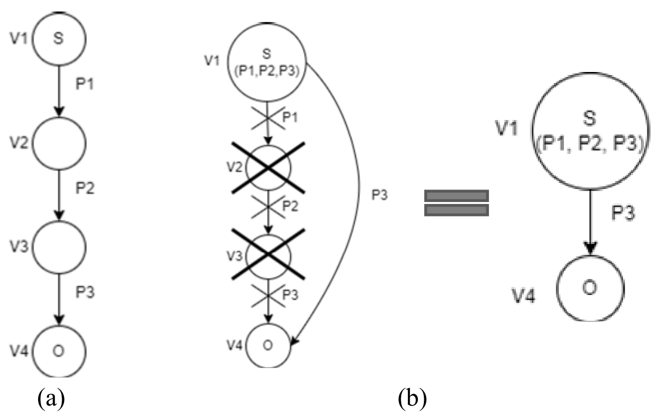
The algorithm of the first operation is called *generate_graph_without_blanks*. The latter receives as input the *RDF graph* G_0 (line 1) and returns it as output without *blank nodes* (line 2). To do this, we traverse all nodes v_i in the L_{E0} set (line 4) and treat for each edge the case of terminal nodes v_i that are blank (lines 5 to 15). We count the *blank nodes* path from the terminal node v_j (lines 7 to 9). Then, we group the edges labels between these *blank nodes* into the source node v_i at line 11 and remove the *blank nodes* with their labels in line 12. At the end, we create an edge between the source node v_i and the adjacent non-blank node of the terminal node (last white node in the chain) in line 14.

Algorithm: Generate_graph_without_blanks

```
1.  Input: RDF_graph $G_0$
2.  Output: RDF_graph $G_0$
3.  Begin
4.  For each $((v_i,\ v_j),\rho_j) \in L_{E0}$ do
5.    If $(v_j \in B)$ do
6.      $k$=0
7.      Repeat
8.        $k = k + 1$
9.      Until $v_{j+k} \notin B$
10.     For each $k$
11.       $v_i = v_i + (\rho_j + \rho_{j+k})$/* + is the concatenate operator
12.       $G_0 \leftarrow G_0 / (\{v_{j+k}\}, \{(v_{i+k}, v_{j+k})\}, \{\rho_{j+k}\}, \{((v_{i+k}, v_{j+k})\rho_{j+k})\})$
13.     End For each
14.     $G_0 \leftarrow G_0 U(\{\}, \{(v_i, v_{j+k})\}, \{\rho_{j+k}\}, \{((v_i, v_{j+k}), \rho_{j+k})\})$
15.   End If
16. End For each
17. Return $G_0$
18. End
```

We propose below an example of a graph formed by 4 nodes, of which 2 nodes constitute a chain of *blank nodes* (see Figure 6(a)). We show in Figure 6(b), the result of removing the *blank nodes*. The latter consists of the two non-blank nodes S and O, with S maintaining the *blank nodes* labels P1, P2 and P3. Nodes S and O are connected by the last *blank node* label P3.

Figure 6 Example of removing blank nodes (a) Example of a graph and (b) Result of the removing



Passing the information from the *blank nodes* to an adjacent non-blank node keeps the information of the triple (s, p, o) since we don't want to break the semantic link induced by the *blank node*. In addition, removing the *blank nodes* simplifies the graph partitioning while keeping the links invoked by the *blank nodes* in the triples.

4.3.2 Removing literal nodes

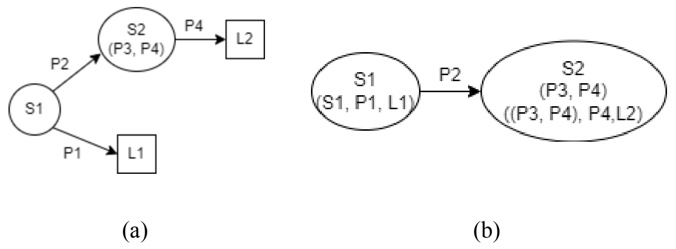
The *removing literal nodes* operation consists of removing the edges that connect the nodes of the string literals set. This operation is presented in the *Graph_without_literal* algorithm. The latter receives the RDF graph G$_0$ returned by the *Generate_graph_without_blanks* algorithm (line 1) and returns this graph without literal object nodes (line 2). To do this, we go through all the graph edges (line 4) and when the terminal node v_j is a literal object (line 5), we add this node and the label of the associated edge to the source node v_i (line 6), then we remove this node and the associated edge (line 7).

Algorithm: Graph_without_literal

```
1.  Input: RDF_graph $G_0$ /* graph without blank node
2.  Output: RDF_graph $G_0$
3.  Begin
4.  For each $((v_i,\ v_j),\rho_j) \in L_{Es}$ do
5.    If $(v_j \in L_{E0})$ do
6.      $v_i = v_i + ((v_i,\ v_j),\rho)$
7.      $G_0 \leftarrow G_0 / (\{v_j\}, \{(v_i,\ v_j)\}, \{\rho\}, \{((v_i,\ v_j),\rho)\})$
8.    End If
9.      End For each
10. Return $G_0$
11. End
```

We propose in Figure 7 an example of the literal nodes removal. The proposed graph (see Figure 7 (a)) consists of 4 nodes S1, S2 (P3, P4), L1, L2 of which nodes L1 and L2 are literal objects and node S2 (P3, P4) is a node resulting from a removing of a *blank node* that was between edges P3 and P4. The result of the second operation of removing is shown in Figure 7(b). The resulting graph consists of two nodes and an edge, where the source node contains the label and literal object of the first removed node and the terminal node contains the label and literal object of the second removed node.

Figure 7 Example of removing the literal nodes (a) Example of a graph and (b) Result of the removing



The nodes removing representing literal objects allows them to be preserved, since the links to and from the literals must not be, in any case, cut in the partitioning. Moreover, this operation leads to reductions in METIS execution time and respectively in memory space according to the results obtained in TriAD (Gurajada et al., 2014) which adopts this operation.

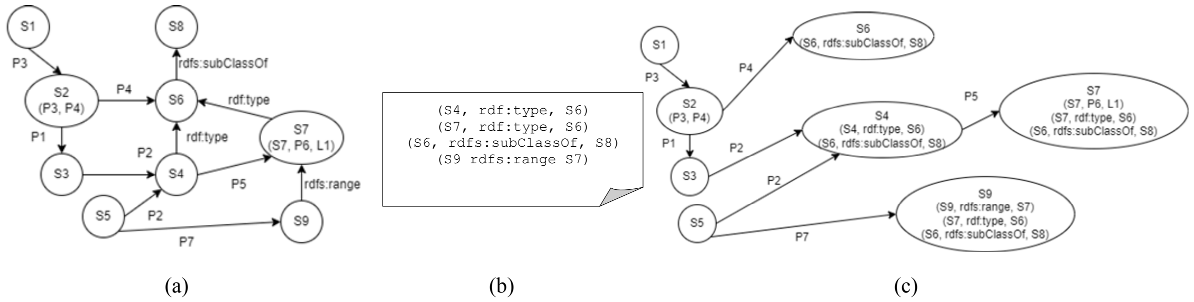
4.3.3 Removing atoms

The last operation of this phase is the removal of the resources semantic descriptions, it is presented in *Graph_without_atom* algorithm. The latter produces the graph G_s (line 3) from the *RDF graph* G_0, result of *Graph_without_literal* algorithm (line 1) and the *RDF graph vocabulary* $voc(G_0)$ which is generated by *Semantic_concepts* algorithm. To do this, we first make a copy of the graph G_0 in G_s (line 5) on which we proceed to remove the *atoms*. For this purpose, we traverse all the edges labels of the graph (line 6) and enumerate the path of the nodes starting from the source node v_i which is composed of edges labelled set by a *predicate atom term* (semantic description) in lines 8 to 11. This path is called semantic path. Next, we add the path atom information to the source node v_i (line 12) and remove the nodes from the path if they are not source nodes or terminal nodes of an edge labelled with a *predicate atom term* (lines 13 to 17).

Algorithm: Graph_without_atom

```
1.  Input : RDF_graph $G_0$ /* graph without blank node and without literal
2.  RDF_graph_vocabulary $voc(G_0)$
3.  Output: RDF_graph $G_s$
4.  Begin
5.  $G_s \leftarrow G_0$
6.  For each $((v_i, v_j), \rho_j) \in L_{Es}$
7.    $K = 0$
8.    Repeat /* Browse the semantic path
9.        $v_i \leftarrow v_i + (v_{i+k}, \rho_{j+k}, v_{j+k})$
10.       $k \leftarrow k + 1$ / *$k$ is the path length
11.   Until $(\rho_{j+k} \neq t(p))$ et $(v_{i+k},\ v_{j+k}) \notin E_S$
12.   $G_s \leftarrow G_s U\left(\{(v_i,\ v_{j+k+1})\}, \{\rho_{j+k+1}\}, \{((v_i,\ v_{j+k+1}), \rho_{j+k+1})\}\right)$
13.   For $c = 1 \ldots k$ do
14.     If $\left(\nexists\left(((v, v_{j+c}), \rho\right) \in L_{Es} \text{ et } \rho \neq t(p))\right)$
15.         $G_s \leftarrow G_s / \left(\{v_{j+c}\}, \{(v_{j+c-1}, v_{j+c})\}, \{\rho_{j+c}\}, \{((v_{j+c-1}, v_{j+c}), \rho_{j+c})\}\right)$
16.     End If
17.   End For
18. End For each
19. Return $G_s$
20. End
```

Figure 8 Example of graph generation G_s, (a) Graph G_0, (b) Vocabulary $voc(G_0)$, (c) generation result G_s


S1
P3
S2 (P3, P4)
P4
S8
rdfs:subClassOf
S6
rdf:type
S7 (S7, P6, L1)
P1
S3
P2
rdf:type
S4
P5
rdfs:range
S5
P2
S9
P7
(S4, rdf:type, S6)
(S7, rdf:type, S6)
(S6, rdfs:subClassOf, S8)
(S9 rdfs:range S7)
S6 (S6, rdfs:subClassOf, S8)
S1
P4
P3
S2 (P3, P4)
P5
S7 (S7, P6, L1) (S7, rdf:type, S6) (S6, rdfs:subClassOf, S8)
P1
S4 (S4, rdf:type, S6) (S6, rdfs:subClassOf, S8)
P2
S3
P2
S9 (S9, rdfs:range, S7) (S7, rdf:type, S6) (S6, rdfs:subClassOf, S8)
S5
P7
(a)
(b)
(c)

We propose in the Figure 8 below the generation of the graph G_s of the example proposed previously in Figure 5(b). The graph G_0 in Figure 8(a) represents the result of the *Graph_without_literal* algorithm and $voc(G_0)$ is the *RDF graph vocabulary* (see Figure 8(b)) generated by the *Semantic_Concepts* algorithm. The graph G_0 consists of nine nodes S1, S2 (P3, P4), S3, S4, S5, S6, S7 (S7, P6, L1), S8 and S9 of which node S2 (P3, P4) is the result of the *blank node* removing between the edges P3 and P4 and node S7 (S7, P6, L1) is the result of the literal node L1 removing. The graph also consists of eleven edges. This graph consists of four semantic paths: (1) the path of nodes S4, S6 and S8, (2) the path of nodes S7, S6 and S8, (3) the path of nodes S6 and S8 and (4) the path of nodes S9, S7, S6 and S8.

The algorithm *Graph_without_atom*, after making a copy of G_0 in G_s, it traverses the semantic paths in which, it transfers the nodes of the *RDF atoms* of each path to the appropriate source node. For example, in Figure 8(c) the node S6 is extended by the *atom* (S6, rdfs:subClassOf, S8), it is then removed since it is not connected to any other (non-semantic) edges.The edges labels that belong to the *RDF atom terms* set of type predicate $t(p)$ define an RDF class or an RDF property. Removing these edges allows us to keep the triples that determine RDF properties and classes. The partitioning of the graph G_s allows a semantic partitioning due to the fact that the cutting of the edges carrying semantics is discarded by the last operation which consists in transferring the properties of the semantic edges in the source nodes (see Figure 8). The o-TG (*v*) aggregation is performed when for the edges set, the terminal node *v* is the same (lines 17 to 26).

4.4 Generation phase of the aggregate graph

This phase of the framework groups the triples sets of the *RDF graph* in order to minimise their number and consequently reduce the graph size to be partitioned by METIS. It consists in building the *triples groups* set from the *atoms* of the *RDF graph* G_s generated by the *Graph_without_atom* algorithm. These sets are generated following the grouping principle (Definition 4.9) proposed in SHAPE (Lee and Liu, 2013a) in order to generate the G_A aggregate graph.

Definition 4.10 (Triple Group): *Given an RDF graph* $G = (V, E, \Sigma_E, L_E)$, *The grouping vertex u denoted s-TG (u) of vertex* $v \in V$ *is the triples set in which their subject is u with s-TG* $(u) = \{(v, \rho) \mid u \in V, \rho \in \Sigma_E, u = v\}$. *Similarly, the sets o-TG and p-TG of v are defined as o-TG* $(v) = \{(u, \rho) \mid v \in V, \rho \in \Sigma_E, u = v\}$ *and p-TG*$(\rho) = \{(u, v) \mid (u, v) \in E\}$, *respectively.*

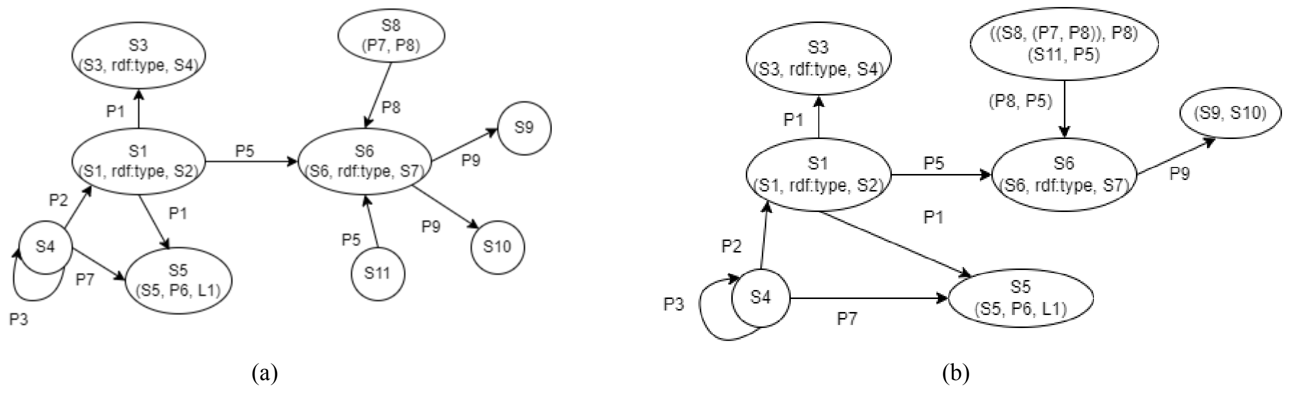
The generation of the aggregate graph consists of grouping nodes that do not contain semantic descriptions whose edges share the same node (source or terminal) containing a semantic description. The generation of the G_A aggregate graph is performed by the *Generate_aggregate_graph* algorithm. The latter receives the graph G_s (line 1) from the previous phase and generates the aggregate graph G_A as output (line 2). To do this, we first copy the content of the graph G_s into G_A (line 4), then we go through all the graph nodes (line 5) and we apply the aggregation when the node contains an *atom* (lines 6 to 23). The aggregation *s-TG* (*v*) is performed when for edges set, the source node *v* is the same (lines 7 to 16). In this case, we group the terminal nodes that do not contain *atoms* (lines 7 to 8) and we group all the labels in this set when they are different (lines 10 to 12). Then we remove all these edges and their terminal nodes (line 13). At this point, we generate the edge of the *s-TG* (*v*) aggregation (line 16). In this case, we group source nodes that do not contain *atoms* (lines 17 to 18) and group all labels in this set when they are different (lines 20 to 22). Then, we remove all these edges and their source nodes (line 23), thus, we generate the edge of the *o-TG* (*v*) aggregation (line 26).

Algorithm: Generate_aggregate_graph

```
1.  Input: RDF_graph $G_s$
2.  Output: RDF_graph $G_A$
3.  Begin
4.   $G_A \leftarrow G_s$
5.  Foreach $v \in V_s$ do
6.    If $v \in t(s)$ do /* $v$ contains an atom
7.        For each $((v, v_j), \rho_j) \in L_{Es}$ do /* s-TG (v) grouping
8.          If $v_j \notin t(s)$
9.            $v' \leftarrow v' + (v_j, \rho_j)$
10.           If $\rho' \neq \rho_j$ do /* p-TG ($\rho$) grouping
11.             $\rho' \leftarrow \rho' + \rho_j$
12.           End If
13.           $G_A \leftarrow G_A / (\{v_j\}, \{(v, v_j)\}, \{\rho_j\}, \{((v, v_j), \rho_j)\})$
14.         End If
15.       End For each
16.       G_A← $G_A \leftarrow G_A U (\{\{v'\}(v_i, v'), \rho', ((v_i, v'), \rho')\})$
17.       For each $((v_i, v), \rho_j) \in L_{Es}$ do /* o-TG(v) grouping
18.         If $v_i \notin t(s)$ do
19.           $v'' \leftarrow v'' + (v_i, \rho_i)$
20.           If $\rho'' \neq \rho_i$ do /* p-TG (ρ) grouping
21.             $\rho'' \leftarrow \rho'' + \rho_i$
22.           End If
23.           $G_A \leftarrow G_A / (\{v_i\}, \{(v_i, v)\}, \{\rho_i\}, \{((v_i, v), \rho_i)\})$
24.         End If
25.       End For each
26.       $G_A \leftarrow G_A U (\{v''\}(v'', v), \rho'', \{((v'', v), \rho'')\})$
27.   End If
28.   End For each
29. Return $G_A$
30. End
```

We propose in Figure 9 an example of aggregate graph generation. Let a *RDF graph* G_s (see Figure 9 (a)) consist of nine nodes S1 (S1, rdf:type, S2), S3 (S3, rdf:type, S2), S4, S5 (S6, P6, L1), S6 (S6 rdf:type, S7), S8 (P7, P8), S9, S10 and S11. Nodes S1 (S1, rdf:type, S2), S3 (S3, rdf:type, S2) and S6 (S6 rdf:type, S7) are the removal result of *atoms* (S1, rdf:type, S2), (S3, rdf:type, S2) and (S6 rdf:type, S7) is generated by the *Semantic_Concepts* algorithm. Node S5 (S6, P6, L1) is the removal result of the L1 literal node generated by the *Graph_without_literal* algorithm, and node S8 (P7, P8) is the removal result of a *blank node* between the edges labelled by P7 and P8 generated by the *Generate_graph_without_blanks* algorithm. The aggregate graph generation in this example (see Figure 9 (b)) groups the adjacent nodes of node S6 (S6 rdf:type, S7) that do not contain a semantic description. In this case, the s-TG (v) and p-TG (*v*) aggregations are applied for nodes S9, S10, since they share the same source node and their edges, and are labelled with the same value. The o-TG (*v*) aggregation is applied to nodes S8 (P7, P8) and S11 because they have the same terminal node.

Figure 9 Example of graph aggregate generation G_A (a) Graph G_s and (b) Aggregation result



5 METIS partitioning scenario with RDF pre-processing

5.1 RDF pre-processing

In this section, we propose the complete application defined in the *RDF data pre-processing* framework as an example, to do so, we will take the graph of Figure 1(a) and we apply the *RDF_preprocessing* algorithm. The aim of this process is to show the impact of the proposed RDF pre-processing on an *RDF graph* compared to partitioning the *RDF graph* without pre-processing.

The result of phase 1 is identical to that of Figure 1(b) since the aim of this phase is to convert the document into a graph, but it should be noted that the complexity of this algorithm has been improved by means of representing the graph by our list structure.

In the *listing phase of the semantic concepts*, the graph vocabulary must be created to maintain the semantics of the RDF document in the *RDF graph*. The *graph vocabulary* $voc(G_0)$ represents the contents of the graph resource list, which allows the preservation of the graph semantics while the graph is stored in the linked list. The corresponding basic semantic concepts are as follows:

A = (S1, rdf:type, S2), (S2, rdf:type, S3)

t(*s*)= {S1, S8}

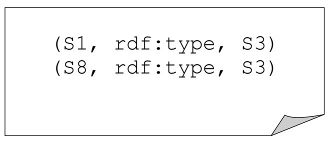
t(*p*)= {rdf:type }

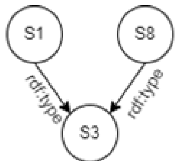
t(*o*)= {S3}

voc(*A*) = {(S1, rdf:type, S3), (S8, rdf:type, S3)}

The *graph vocabulary* $voc(G_0)$ is shown in Figure 10(a) and the *semantic RDF graph* is shown in Figure 10(b).

Figure 10 The semantic concepts of Figure 1(a) (a) The graph vocabulary $voc(G_0)$ and (b) The semantic RDF graph *G*

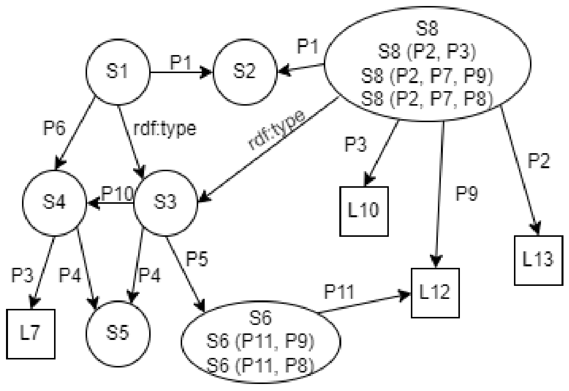

```
(S1, rdf:type, S3)
(S8, rdf:type, S3)
```



(a) (b)

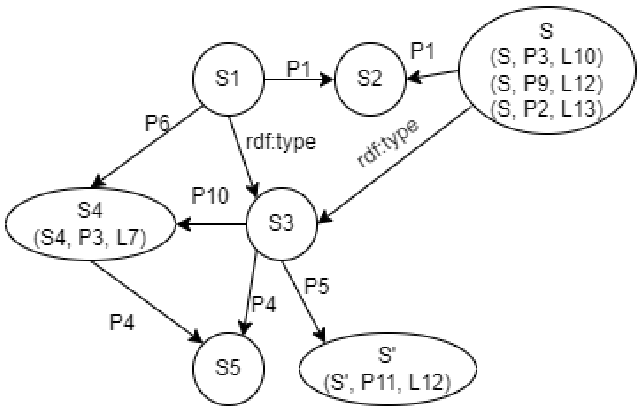
The *Graph generation phase based on semantic concepts* first removes the two *blank nodes* from the initial graph G_0 (see Figure 11). Then, it removes the literal nodes L7, L10, L12 and L13 (see Figure 12).

Figure 11 RDF graph without blank nodes from Figure 1(a)



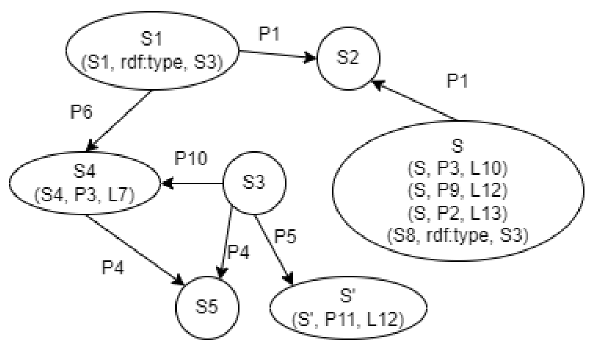
We replace S8 S8 (P2, P3) S8 (P2, P7, P9) S8 (P2, P7, P8) with S and S6 S6 (P11, P9) S6 (P11, P8) with S' in Figure 12.

Figure 12 G_0 graph without blank nodes and without literals from Figure 1(a)



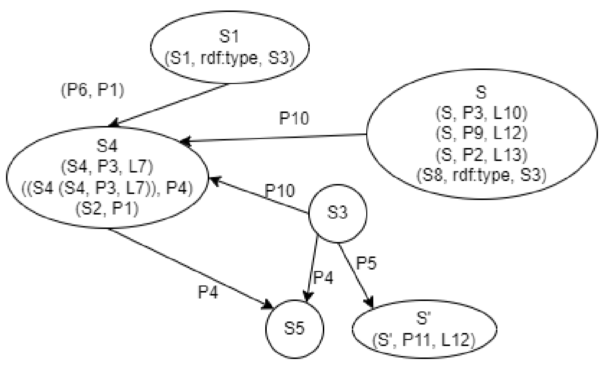
The operation of *removing atoms* generates the graph G_s, it transfers the *atoms* (S1, rdf:type, S3) and (S8, rdf:type, S3) into nodes S1 and S, respectively. The result of this operation is shown in Figure 13.

Figure 13 RDF graph G_s from Figure 1(a)



The *generation phase of the aggregate graph* groups the two terminal nodes S1 and S4 (S4, P3, L7) and the edges P1 and P6 since their source node contains the *RDF atom* S1, rdf:type, S3. The result of this phase is shown in Figure 14.

Figure 14 graph G_A from Figure 1(a)



5.2 METIS partitioning

Once the pre-processing of the RDF data is complete, it is necessary to go through the G_A mapping and create the METIS graph to partition the data.

The mapping decodes all elements of the graph G_A (nodes and edges) as shown in Figure 15(a) and the transformation result is shown in Figure 15(b). The representation of the *RDF graph resource list* which mentions the label of the edges is presented in Figure 15(b). For example, the first line determines that node 1 and 2 are adjacent and that their edge is labelled with predicate 7.

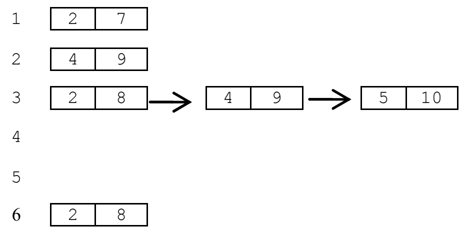
The pre-processing applied by our framework to the graph in Figure 1(b) produced a list of size 12 (see Figure 15(a)) which is smaller than the list produced without pre-processing from the initial graph (see Figure 4) which is of size 23 knowing that both lists represent the same graph. The pre-processing thus makes it possible to reduce the size of the METIS graph (see Figure 15(c)). In effect, the METIS graph produced by the *RDF data pre-processing* framework consists of six nodes and six edges, while the METIS graph (see Figure 1(d)) produced without pre-processing consists of thirteen nodes and sixteen edges.

The METIS partitioning of the METIS graph (see Figure 15(b)) into two partitions generates the vector $P[v]$ (see Figure 16(a)). The latter assigns nodes S3, S5 and S' (S', P11, L12) to partition 0 and nodes S1 (S1, rdf:type, S3), S4 (S4, P3, L7) ((S4(S4, P3, L7)) P4) (S2,P1) and S (S, P3, L10) (S, P9, L12) (S, P2, L13) (S8, rdf :type, S3) to partition 1 (see Figure 16(b))

Figure 15 Creation of the METIS graph (a) Mapping, (b) Graph G_A and (c) Graph METIS

```
S1(S1, rdf:type, S3)→ 1
S4(S4, P3, L7) ((S4(S4, P3, L7)), P4)→ 2
S3 → 3
S5 → 4
S’ (S’, P11, L12) → 5
S (S, P3, L10) (S, P9, L12) (S, P2,
L13) (S8, rdf:type, S3)→ 6
P6, P1 → 7
P10 → 8
P4 → 9
P5 → 10
```

(a)



(b)

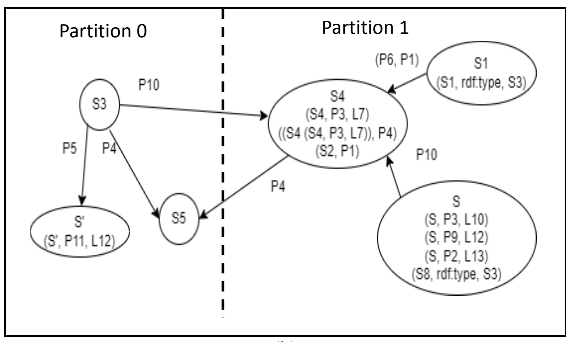
```
6 6 001
2 1
4 1 1 6 1 3 1
2 1 4 1 5 1
3 1 2 1
3 1
2 1
```

(c)

Figure 16 Partitioning the graph into two partitions (a) vector $P[v]$ and (b) graph G_A partitioning

```
0
0
1
1
0
```

(a)



(b)

The result obtained (see Figure 16(b)) is different from that obtained in Figure 2(b) (Sub-section 3.2) of the same graph with the same number of partitions. In the partitioning of Figure 16(b), there are two cut edges without the edges connecting the *blank nodes*, the literal values and the edges carrying semantic descriptions being cut. In effect, these edges do not exist in the METIS graph produced by the aggregate graph G_A, the final result of the *RDF data preprocessing* framework. These edges are removed and preserved in the nodes as indicated in the *Generate_graph_without_blanks*, *Graph_without_literal*, *Graph_without_atom* and *Generate_aggregate_graph* algorithms.

We further illustrate below the graph partitioning in Figure 15(b) into three and four partitions. The result of the partitioning into three partitions generates the vector $P[v]$ (see Figure 17(a)) assigning nodes S3 and S (S, P3, L10) (S, P9, L12) (S, P2, L13) (S8, rdf:type, S3) to partition 0, nodes S5 and S4 (S4, P3, L7) ((S4 (S4, P3, L7)) P4) (S2, P1) to partition 1 and nodes S' (S', P11, L12) and S1 (S1, rdf:type, S3) to partition 2. This partitioning (see Figure 17(b)) causes the edges P5, P4, P10 and (P6, P1) to be cut.

The result of the partitioning into four partitions generates the vector $P[v]$ (see Figure 18(a)) assigning node S (S, P3, L10) (S, P9, L12) (S, P2, L13) (S8, rdf:type, S3) to partition 0, node S1 (S1, rdf:type, S3) to partition 1, nodes S4 (S4, P3, L7) ((S4 (S4, P3, L7)) P4) (S2, P1) and nodes S' (S', P11, L12) to partition 2 and nodes S3 and S5 to partition 3. This partitioning (see Figure 18(b)) causes the edges P5, P4, P10 and (P6, P1) to be cut.

The graph partitioning in Figure 15(b) into three and four partitions is balanced but with a high-communication ratio but no break of a semantic description, a link to a white node or a link to literal.

Figure 17 Partitioning the graph into three partitions (a) vector $P[v]$ and (b) graph G_A partitioning

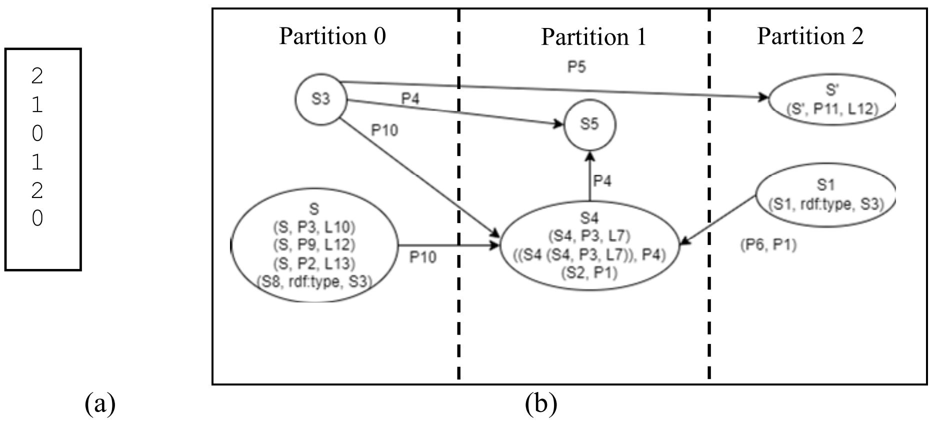
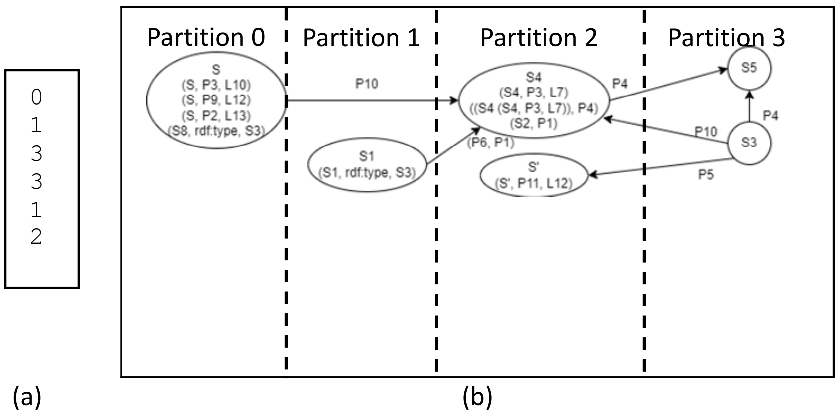


Figure 18 Partitioning the graph into four partitions (a) vector $P[v]$ and (b) graph G_A partitioning



6 Evaluation

For the evaluation, we performed our experiments on real and synthetic data sets of varying domains and sizes (four different sizes for each data set that increase proportionally). For the synthetic data, we used three representative RDF benchmarks WatDiv '*Waterloo SPARQL Diversity Test Suite*' (http://dsg.uwaterloo.ca/watdiv/), the popular LUBM benchmark *Lehigh University Benchmark* (http://swat.cse.lehigh.edu/projects/lubm/index.htm) in Guo et al. (2005) and BSBM *The Berlin SPARQL Benchmark* (http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/V1/spec/index.html#datagenerator).

The WatDiv data set has a set of resources generated according to an e-commerce database schema, including information about users, retailers and products. The number of RDF triples generated in WatDiv is determined by the appropriate 'scaling factor' parameter. In our evaluations, we have assigned the values 0.5, 1, 10 and 100 for the scaling factor to generate approximately 10,000 triples, 100,000 triples, 1,000,000 triples and 10,000,000 triples, respectively.

LUBM presents an ontology for an academic domain. To exploit the LUBM data set in our evaluation, we used the UBA 1.7 generator to generate an ontology covering 10, 50, 80 and 100 universities, respectively.

The BSBM data set is built around an e-commerce use case in which products set offered by different producers is judged by consumers set. We used the number of products as a scaling factor to generate the BSBM data set containing approximately 50,000 triples, 250,000 triples, 1,000,000 triples and 5,000,000 triples, respectively.

We also used the DBpedia real data set (https://wiki.dbpedia.org/Datasets). The latter is an academic and community-based project for the automatic exploration and extraction of data derived from Wikipedia. We used four DBpedia data sets containing open domain knowledge about sport and sport events. The number of triples in DBpedia ranges from 30,000 to 10,000,000 as shown in Table 1.

We opted to test the *RDF data preprocessing* framework performance using these data sets given the specific characteristics of each of these data sets. Data in WatDiv is heterogeneous where some is well structured and contains few optional attributes, while others are less well structured. It contains a more distinct set of predicates than LUBM, but with only around 15% of triples containing a semantic description. This set contains 12% of *blank nodes*, which allows us to test the impact of the *Generate_graph_without_blanks* (Sub-section 4.3.1) algorithm. LUBM is a data set rich in semantic concept, it contains about 31% of triples containing a semantic description (*Atom*) but with a reduced number of 'rdf:type' properties. The data set therefore contains around 32% of literal objects and 32% of triples constituting the *triple group*. This allows to evaluate the algorithms: *Graph_without_atom* (Sub-section 4.3.3), *Generate_aggregate_graph* (Sub-section 4.4) and *Graph_without_literal* (Sub-section 4.3.2). The latter is also evaluated in the BSBM data set since it contains 55% literal objects. The content of the DBpedia data is different from one set to another since they are real even if they are extracted from the same domain (sport). These data generate asymmetric real RDF graphs that are different (Moreira et al., 2021).

We present in the table below, the data sets characteristics such as size, number of triples, number of subjects, number of predicates, number of objects, percentage of the *blank nodes*, the percentage of literal, the percentage of triples constituting the *Atom* and the percentage of triples constituting *triples groups*.

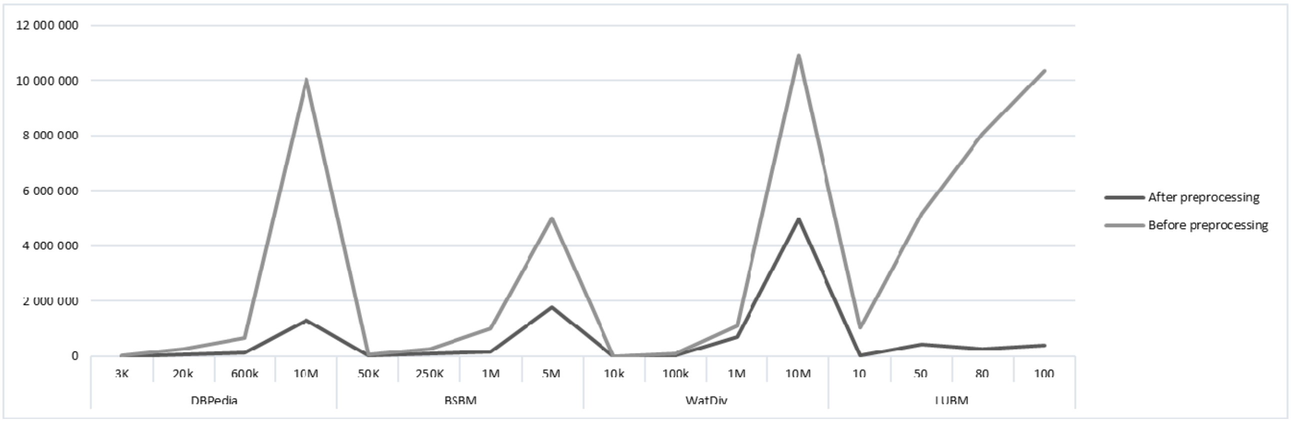
The evaluation of the *RDF_Preprocessing* algorithm was carried out on a machine equipped with an Intel core i5-5300M processor with 2.3GHz*4, 500 Gb of disks, 16 Gb of RAM under Ubuntu 14.04 LT and we used METIS version 5.1.0 (http://glaros.dtc.umn.edu/gkhome/).

We present in Figure 19, the triples reduction ratio of our framework execution on data sets.

Table 1 RDF data set characteristics

Data set		*Size (MB)*	*#Triple*	*#Subjet*	*#Predicate*	*#Object*	*% Blank node*	*% Literal*	*% Atom*	*% Triple group*
DBPedia	3 K	4.2	30,318	3955	23	16,425	0	42.84	22.08	13.54
	20 k	248.7	255,987	5679	159	4594	0	49.54	10.59	16.87
	600 k	470.6	658,307	59,019	4057	1254,295	0	36.85	18.54	26.70
	10 M	1392.6	10,058,978	597,264	67,241	981,967	0	38.05	9.54	39.48
BSBM V1.0	50 K	14.7	50,116	4900	40	11,888	0	55.32	10.32	9.77
	250 K	74.1	250,492	23,178	40	52,174	0	55.12	17.82	9.89
	1 M	298.3	1,000,226	92,044	40	201,091	0	55.38	20.09	10.16
	5 M	1465.9	5,000,453	458,141	40	969,744	0	55.37	19.37	10.29
WatDiv	10 k	28.5	9335	545	86	1781	12.26	13.57	4.65	3.61
	100 k	156.4	93,256	5450	86	17,795	12.29	13.29	4.91	3.92
	1 M	628.5	1,092,358	52,123	86	97,526	12.29	14.54	5.05	4.32
	10 M	1506.6	10,916,457	521,585	86	1,005,832	12.35	17.07	5.24	4.29
LUBM	10	107.6	1,045,739	165,439	18	124,612	0	32.36	32.49	32.49
	50	567.2	5,165,386	815,478	18	607,563	0	31.0	30.16	30.19
	80	753.5	8,067,027	1,792,587	18	1,299,760	0	32.81	32.06	32.02
	100	1109.5	10,347,240	1,632,638	18	1,215,002	0	32.05	32.12	32.1

Figure 19 Triples number before and after pre-processing



The obtained results show that the preprocessing established by our framework reduces the triples number in all data sets. We note that the reduction rate of triples number in the synthetic data sets (LUBM, WatDiv and BSBM) is monotonous whatever the number of triples and the size of these sets. On the other side, the DBpedia data reduction rate is different from one size to another. The triple reduction rate recorded in the LUBM, DBpedia, BSBM and WatDiv data sets is approximately 97%, 78%, 70% and 47%, respectively. The result obtained in LUBM is more important whatever the size of these data sets. This rate depends on the number of *blank nodes*, literal objects, triples *Atom* and triples constituting *triple group*. *Blank nodes* are deleted by the *Generate_graph_without_blanks* algorithm in only the WatDiv database. The number of literal objects deleted by *Graph_without_literal* algorithm is high in BSBM, DBPedia and LUBM data sets, but in WatDiv the number of literal objects removed is low. Triples *Atom* are deleted by *Graph_without_atom* algorithm in LUBM which contains a large number, and average in DBpedia and BSBM contrary to WatDiv which contains few RDF *Atom* (5%). The triples constituting *triple group* are deleted by *Generate_aggregate_graph* algorithm. The number of these deleted triples is high in LUBM (32%), medium in DBpedia (24%) and in BSBN and low in WatDiv (4%).

The reduction of triples is the result of preprocessing operations on the *RDF graph* relating to the nature of nodes (subject and object) and edges (predicate). To do so, we have counted the number of nodes and edges deleted in our framework. In Figure 20, we present the number of deleted nodes and edges to see the impact of the triple reduction on the number of deleted nodes and edges in the initial graph G_0.

The results show that our framework favours the reduction of the nodes of the initial graph G_0 (see Figure 20) in most data sets except LUBM data set. Indeed, nodes and edges are deleted throughout the preprocessing process according to the content of the data sets following the operations of removing *blank nodes*, literal nodes, *atoms* and *triples groups* on the initial graph G_0. The results show that the algorithms *Generate_graph_without_blanks*, *Graph_without_atom* and *Graph_without_literal* favour the reduction of the number of nodes and the *Generate_aggregate_graph* algorithm favours the reduction of edges independently of the size of the data sets, which justifies the result obtained in LUBM. Reminding that LUBM through our Framework generates many *triples groups*.

The number of nodes and edges removed affects the METIS graph size which represents the input file to METIS. We present in Figure 21 the METIS input size before and after the pre-processing as a function of the size of the data sets.

Figure 20 Number of nodes and edges removed in the pre-processing framework

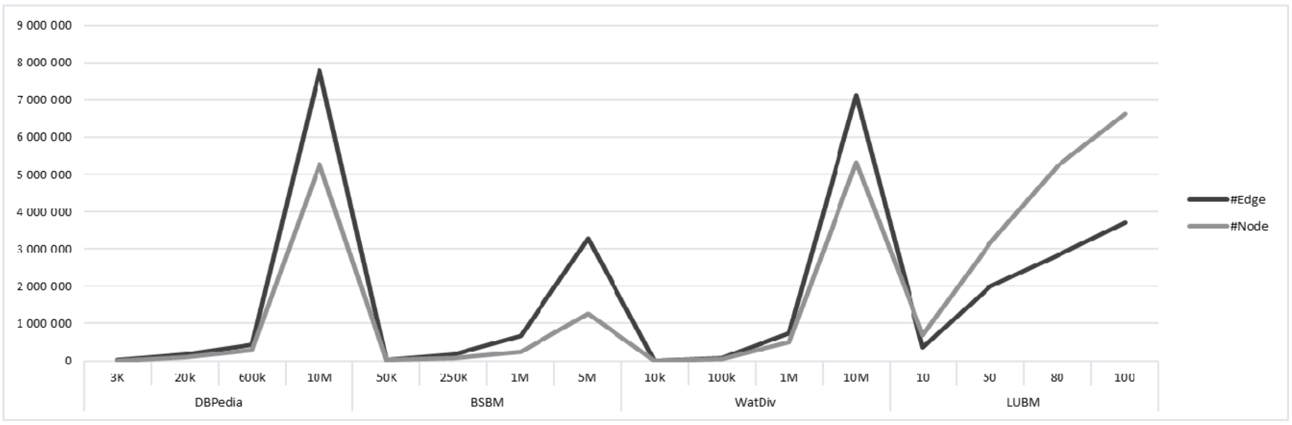
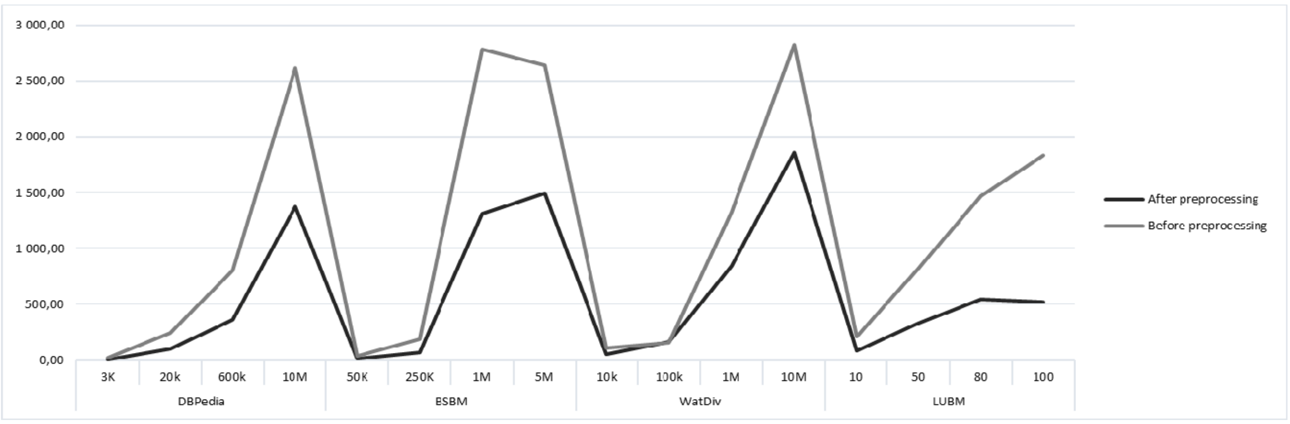


Figure 21 METIS graph (METIS Input) size before and after pre-processing (MB)



We notice in the results (see Figure 21) that the METIS graph size is reduced for all data sets in the *RDF data pre-processing* framework. The reduction rate of the METIS graph size recorded in the DBpedia, BSBM, WatDiv and LUBM data sets is respectively equal to 54%, 53%, 68% and 64%. We noticed that this reduction rate depends on the data size and the type of each data set. The reduction rate is important only when the size of the data sets is large and when the data sets contain a considerable number of literal objects as for BSBM or of *triples groups* and RDF *Atom* as for LUBM.

We thus present in the Figure below (see Figure 22) the length of the vector $P[v]$ (METIS output). The length value of this vector is determined by the order of the initial graph G_0 when the partitioning is performed without partitioning. On the other hand, when the partitioning is performed after the RDF data pre-processing, the length value of the vector $P[v]$ is determined by the order of the aggregation graph G_A.

We have noticed that the vector $P[v]$ length is slightly reduced in our framework. When the sum of literal objects, of *triples groups*, and *RDF Atom* removed from a data set by *Graph_without_literal*, *Generate_aggregate_graph* and *Graph_without_atom* algorithms exceeds 27%, the reduction in the length of the vector $P[v]$ becomes considerable. Consequently, these results show that *Generate_graph_without_blanks* algorithm doesn't sufficiently reduce the length of the vector $P[v]$ and that *Graph_without_literal*, *Generate_aggregate_graph* and *Graph_without_atom* algorithms contribute to the considerable reduction of $P[v]$ when they reduce the number of triples of the data sets.

On the other hand, in order to know the impact of our pre-processing on the partitioning quality, we launched a partitioning of the data sets by varying the partitions number to 5, 25, 50 and 100.

The analysis was done on the number of edges cuts of semantic description, the partitioning time and the memory size for each number of partitions (see Figures 23, 24 and 25).

The number of edges cuts of semantic description after our pre-processing is always equal to 0 whatever the data set size, the content of the data or the number of partitions. Therefore, we present in the Figure 23, the number of cuts of these edges before the pre-processing.

Figure 22 Length of the vector $P[v]$ (METIS output)

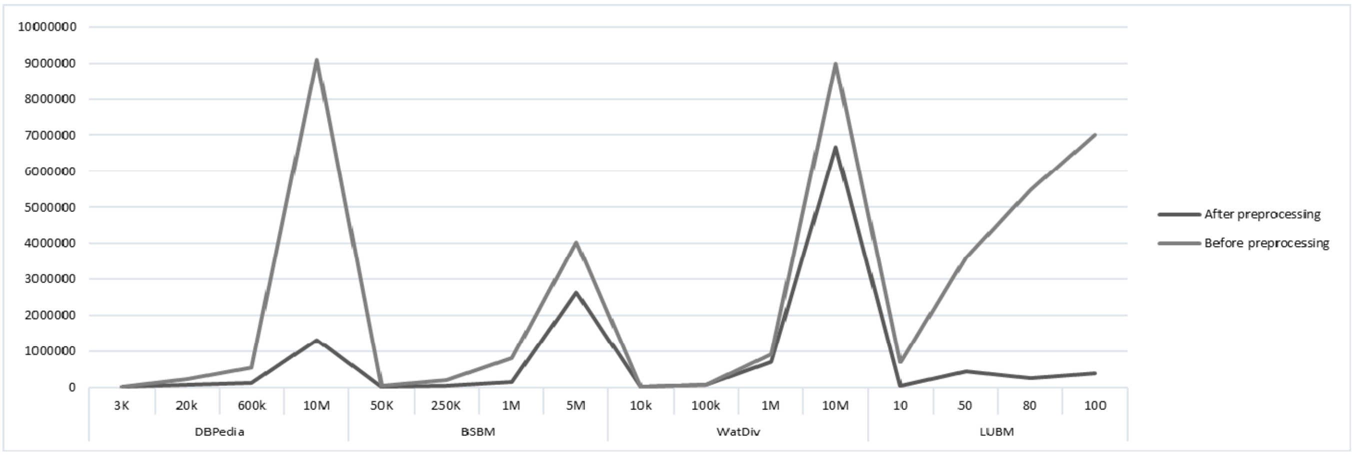
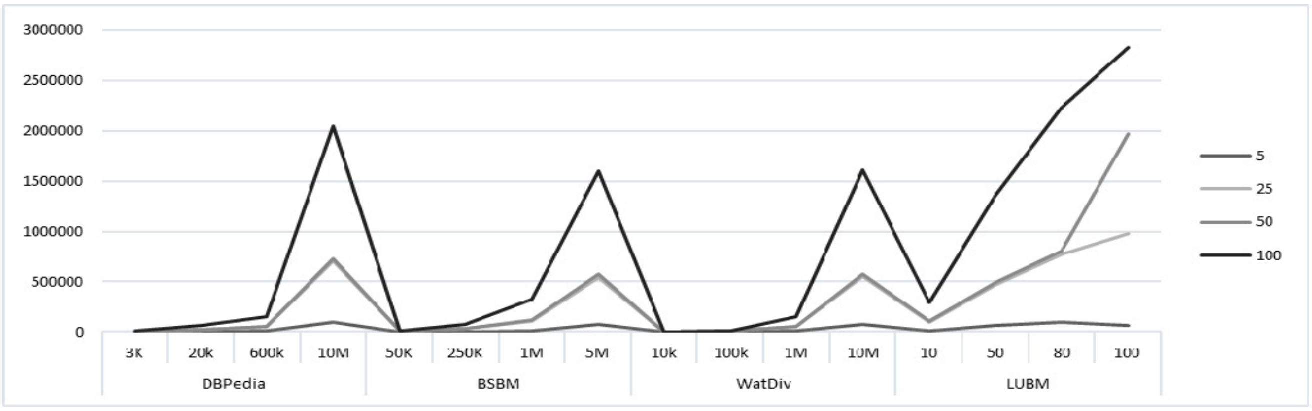


Figure 23 Number of edges cuts of semantic description



We note that when data sets are partitioned without pre-processing, the number of edges cuts of semantic description generally according to the number of partitions. The number of edges cuts is small when the number of partitions is equal to 5 for all data sets. In this case, the ranking of data sets in ascending order of the number of semantic descriptions edges is as follows: WatDiv, DBpedia, BSBM and LUBM. On the other hand, the number of edges cuts of semantic description is high when the number of partitions is equal to 100 for all data sets. The ranking of the data sets in descending order of the number of edges is as follows: WatDiv, DBpedia, BSBM and LUBM.

We present in Figure 24, the partitioning time of data sets before and after pre-processing as a function of the number of partitions.

We note that the partitioning time recorded after pre-processing is always reduced independently of the specific characteristics of these data sets. This is justified by the size of the data sets which is reduced following the pre-processing. Thus, the partitioning time is reduced depending on the size of the data sets and the number of partitions. The shortest time after pre-processing of DBpedia, BSBM, LUBM and WatDiv data sets is recorded when the size of these data sets is small and the number of partitions is equal to 100. On the other hand, the longest partitioning time before pre-processing is recorded when the size of these data sets is large and the number of partitions is equal to 5.

We present in Figure 25 the memory size used in partitioning with and without the *RDF data pre-processing* framework as a function of the partitions number.

Figure 24 Partitioning time in seconds

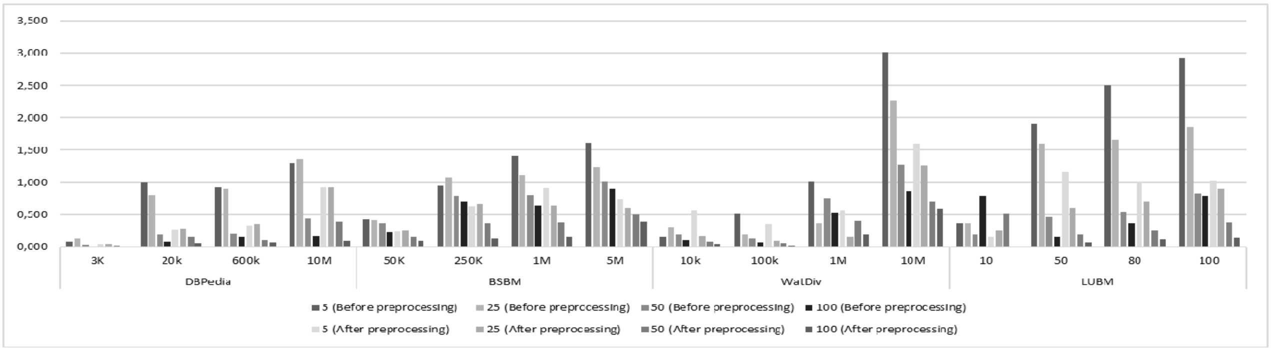
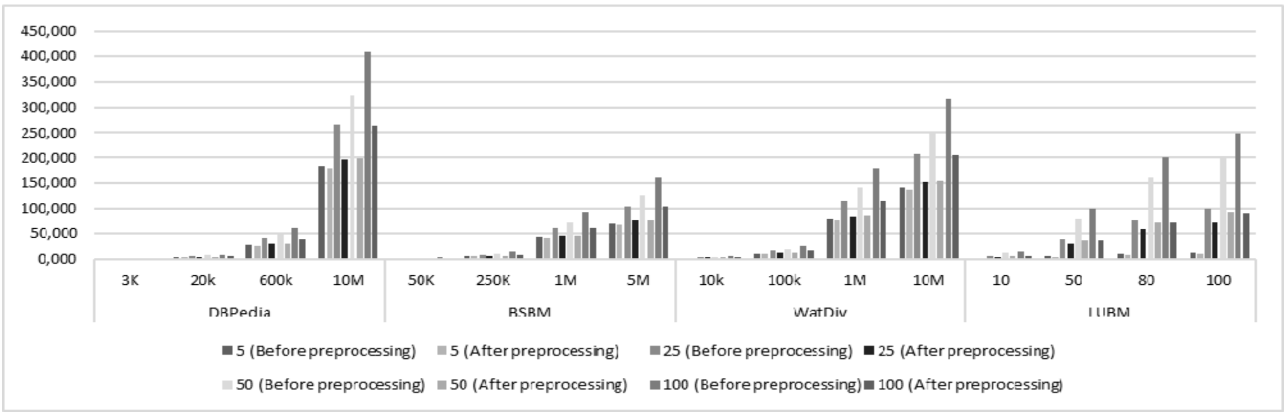


Figure 25 Maximum memory used 'MB' during partitioning

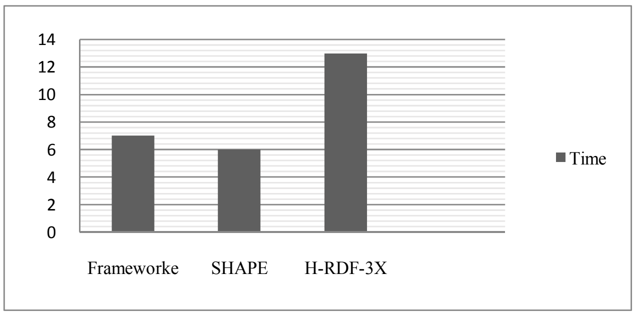


The results obtained show that pre-processing reduces the memory size used as a function of the number of partitions and the size of the data set, but independently of the specific characteristics of the data sets. The memory size is reduced a little bit when the number of partitions is small (5 partitions) in all data sets. The rate of memory reduction increases when the number of partitions is large enough (25, 50 and 100 partitions) and becomes more important when the data set size is large enough (see Figure 25).

Thus, we measured the pre-processing time of our framework on LUBM 100 and compared it to the pre-processing time occupied by the H-RDF-3X system and the SHAPE system.

The results obtained (see Figure 26) show that the SHAPE system requires 6 minutes of time and the H-RDF-3X system 13 minutes of time. Let us remember that these systems require a replication step after pre-processing and during partitioning. Our system requires 7 minutes of time with a step of conservation of the total semantic concepts and which will reduce the number of data to replicate. The partition generation step is not covered in this paper.

Figure 26 Pre-processing time of the LUBM 100 data set in minutes



7 Conclusions

The RDF data partitioning by METIS generates partitions formed a set of closest triples. As METIS doesn't take into account the data semantics, the generated partitions may lose semantics compared to the initial graph before partitioning. This affects the partitions quality and reduces the query processing performance. In addition, METIS doesn't scale well with very large graphs due to their intensive memory usage and high computational cost. In fact, the more connected graph is the more difficult to partition.

To do this, we have proposed in this paper an *RDF data pre-processing* framework based on data semantic preservation and graph size reduction. The preservation of the data semantics was achieved through several algorithms aiming at the preservation of anonymous resources and semantic concepts listed in the RDF document such as *RDF atoms*, *RDF atom terms*, *RDF atom vocabularies* and *RDF graph vocabulary*. The reduction of the graph was achieved by removing the literal data, preserving the semantics of the data and grouping the data. This reduces both the size of the RDF data to be partitioned and the degree of connectivity of the data by preserving their semantics.

We have performed various implementations of our framework based on real such as DBpedia and synthetic data such as BSBM, WatDiv and LUBM. We evaluated the impact of our approach on the ratio of reduced triples, the number of nodes and edges removed, the size of the METIS input file, the size of the METIS output file, the number of edges cuts of semantic description, the partitioning time and the memory size used. We also measured the data pre-processing time and compared with other systems (H-RDF-3X and SHAPE).

After the pre-processing performed in this paper, our perspective is the generation of partitions and the study of the performance of SPARQL queries based on large RDF data sets since the partitioning quality influences query processing performance (Kalogeros et al., 2020).

References

Barnard, S.T. and Simon, H.D. (1993) 'A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems', *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pp.711–718.

Benlic, A.U. and Hao, J.K. (2013) 'Breakout local search for the quadratic assignment problem', *Applied Mathematics and Computation*, Vol. 219, No. 9, pp.4800–4815.

Berner-Lee, T. (1998) *What the Semantic Web can Represent*, W3C 1998.

Bok, K., Kim, J. and Yoo, J. (2019) 'Dynamic partitioning supporting load balancing for distributed RDF graph stores', *Symmetry*, Vol. 11, No. 7, Article number 926.

Galicia, J., Mesmoudi, A., Bellatreche, L. and Ordonez, C. (2019) 'Reverse partitioning for SPARQL queries: principles and performance analysis', *Proceedings of the International Conference on Database and Expert Systems Applications* Springer, Cham, pp.174–183.

Guo, Y., Pan, Z. and Heflin, J. (2005) 'LUBM: a benchmark for OWL knowledge base systems', *Journal of Web Semantics*, Vol. 3, Nos. 2/3, pp.158–182.

Gurajada, S., Seufert, S., Miliaraki, I. and Theobald, M. (2014) 'TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.289–300.

Huang, J., Abadi, D.J. and Ren, K. (2011) 'Scalable SPARQL querying of large RDF graphs', *Proceedings of the VLDB Endowment*, Vol. 4, No. 11, pp.1123–1134.

Kalogeros, E., Gergatsoulis, M. and Damigos, M. (2020) 'Document-based RDF storage method for parallel evaluation of basic graph pattern queries', *International Journal of Metadata, Semantics and Ontologies*, Vol. 14, No. 1, pp.63–80.

Karypis, G. and Kumar, V. (1997) 'A coarse-grain parallel formulation of multilevel k-way graph-partitioning algorithm', *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, USA.

Karypis, G. and Kumar, V. (1998a) *Metis: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, University of Minnesota, version 4.0, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis.

Karypis, G. and Kumar, V. (1998b) 'Multilevel algorithms for multi-constraint graph partitioning', *Proceedings of the ACM/IEEE Conference on Supercomputing.*

Kernighan, B.W. and Lin, S. (1970) 'An efficient heuristic procedure for partitioning graphs', *The Bell System Technical Journal*, Vol. 49, No. 2, pp.291–307.

Lee, K. and Liu, L. (2013) 'Scaling queries over big RDF graphs with semantic hash partitioning', *Proceedings of the VLDB Endowment,* Vol. 14, No. 6, pp.1894–1905.

Lee, K. and Liu, L. (2013), 'Efficient data partitioning model for heterogeneous graphs in the cloud', *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp.1–12.

Lee, K., Liu, L., Tang, Y., Zhang, Q. and Zhou, Y. (2013) 'Efficient and customizable data partitioning framework for distributed big RDF data processing in the cloud', *Proceedings of the IEEE Sixth International Conference on Cloud Computing*, pp.327–334.

Moreira, J., Neto, E.C. and Barbosa, L. (2021) 'Analysis of structured data on Wikipedia', *International Journal of Metadata, Semantics and Ontologies*, Vol. 15, No. 1, pp.71–86.

Priyadarshi, A. and Kochut, K.J. (2022) 'PartKG2Vec: embedding of partitioned knowledge graphs', *Proceedings of the International Conference on Knowledge Science, Engineering and Management*, Springer, Cham, pp.359–370.

Ragab, M., Awaysheh, F.M. and Tommasini, R. (2021) 'Bench-ranking: a first step towards prescriptive performance analyses for big data frameworks', *Proceedings of the IEEE International Conference on Big Data (Big Data)*, IEEE Computer Society, Los Alamitos, CA, USA, pp.241–251.

Rakhmawati, N.A., Karnstedt, M., Hausenblas, M. and Decker, S. (2014) 'On metrics for measuring fragmentation of federation over SPARQL endpoints', *Proceedings of the 10th International Conference on Web Information Systems and Technologies WEBIST*, pp.119–126.

Ramesh, S., Baranawal, A. and Simmhan, Y. (2021) 'Granite: a distributed engine for scalable path queries over temporal property graphs', *Journal of Parallel and Distributed Computing (JPDC)*, Vol. 151, pp.94–111.

Simperl, E., Sarasua, C., Ungrangsi, R. and Bürger, T. (2011) 'Ontology metadata for ontology reuse', *International Journal of Metadata, Semantics and Ontologies*, Vol. 1, Nos. 1/1, pp.11–111.

Slavov, V., Rao, P., Barenkala, D. and Paturi, S. (2012) 'Towards RDF query processing on the intel SCC', *Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium*, pp.7–12.

Soma, R. and Prasanna, V.K (2008) *A Data Partitioning Approach for Parallelizing Rule Based Inferencing for Materialized Owl Knowledge Bases*, Technical Report, University of Southern California.

W3C (2014a) *RDF 1.1 concepts and abstract syntax.* Available online at: https://www.w3.org/TR/rdf11-concepts/

W3C (2014b) *RDF Schema 1.1.* Available online at: http://www.w3.org/TR/rdf-schema/

Wang, R. and Chiu, K. (2012) 'A graph partitioning approach to distributed RDF stores', *Proceedings of the IEEE 10th International Symposium on Parallel and Distributed Processing with Applications ISPA*, pp.411–418.

Zhang, X., Chen, L., Tong, Y. and Wang, M. (2013) 'EAGRE: towards scalable I/O efficient SPARQL query evaluation on the cloud', *Proceedings of the IEEE 29th International Conference on Data Engineering ICDE*, pp.565–576.