Semantic-Aware Partitioning on RDF Graphs

Qiang Xu[1], Xin Wang[1,4,5]*, Junhu Wang[2], Yajun Yang[1,5], and Zhiyong Feng[3,5]

[1] School of Computer Science and Technology, Tianjin University, China
[2] School of Information & Communication Technology, Griffith University, Australia
[3] School of Computer Software, Tianjin University, China
[4] State Key Lab. for Novel Software Technology, Nanjing University, China
[5] Tianjin Key Laboratory of Cognitive Computing and Application, China
{xuqiang3,wangx,yjyang,zyfeng}@tju.edu.cn, j.wang@griffith.edu.au

Abstract. With the development of the Semantic Web, an increasingly large number of organizations represent their data in RDF format. A single machine cannot efficiently process complex queries on RDF graphs. It becomes necessary to use a distributed cluster to store and process large-scale RDF datasets that are required to be partitioned. In this paper, we propose a *semantic-aware partitioning* method for RDF graphs. Inspired by the PageRank algorithm, classes in the RDF *schema graphs* are ranked. A novel partitioning algorithm is proposed, which leverages the semantic information of RDF and reduces crossing edges between different fragments. The extensive experiments on both synthetic and real-world datasets show that our semantic-aware RDF graph partitioning outperforms the state-of-the-art methods by a large margin.

Keywords: graph partitioning; semantic-aware; RDF graph; schema

1 Introduction

The *Resource Description Framework* (RDF) is a W3C recommendation for describing and organizing resources on the Semantic Web. The RDF data model is a type of graph model, which consists of a set of triples (s, p, o), where s is the *subject*, p the *predicate*, and o the *object*. Each triple (s, p, o) represents a statement that s has the relationship p with o, thus being a directed edge in the RDF graph. With the Linked Data initiative, large amounts of RDF graphs have been publicly released, which often contain millions or even billions of triples.

How to partition a big RDF graph while reducing its crossing edges between different machines and improving the performance of RDF query processing has been recognized as a challenging issue. There exists several methods to partition RDF graphs in terms of vertices [2,3], edges [4], and paths [5]. However, these methods are all based on the traditional graph partitioning, which merely consider the structural information of graphs. In this paper, we propose a *semantic-aware* RDF partitioning algorithm that leverages the type information specified by the `rdf:type` predicate to achieve high-quality partitioning of RDF graphs.

* Corresponding author.

Our main contributions include: (1) we propose an efficient algorithm, inspired by PageRank, for ranking classes in RDF schema graphs; (2) a novel RDF partitioning method is proposed, which can use the semantic information embedded RDF graphs to reduce crossing edges between different fragments; (3) the extensive experiments on the synthetic and real-world RDF graphs have been conducted to verify the effectiveness and efficiency of our approach. The experimental results show that our approach outperforms the state-of-the-art methods by a large margin.

2 Related Work

The existing partitioning methods on general graphs can be classified as follows:

Vertex-based approach. Thus far, METIS [3] is widely considered to be the most popular graph partitioning algorithm. In [2], Huang et al propose an RDF partitioning approach based on METIS, which duplicates the vertices that are within n-hop distance from the boundary vertices and assigns them into the same partition. Although this approach can make more queries being executed locally, it may result in a large amount of data redundancy.

Edge-based approach. Margo et al [4] propose a scalable distributed graph partitioner (Sheep) which can produce edge partitions by reducing the graph to an elimination tree without sacrificing the partition quality. However, the cost of edge cut in this approach is unbounded and may be prohibitively high.

Path-based approach. Wu et al [5] present a method to partition RDF graphs with paths as the basic partition unit, which partitions paths sharing merged vertex into the same fragment. By using the end-to-end path, this method makes use of the structural information of RDF graphs to minimize the number of crossing edges. However, the sizes of paths may differ significantly, so using the number of paths as the balance metrics may lead to data skewness.

RDF graphs are such kind of graphs that have inherent semantics. Thus far, the existing works of partitioning RDF graphs mainly include two types. The first type is to adapt the METIS method to some extent for RDF graphs. In fact, it is also a vertex-based approach with the feature of METIS, which cannot leverage the semantics embedded in RDF graphs. The second type is a naive hash partitioning approach, which generates a key according to each subject (or object) and assigns an RDF triple to a fragment simply based on its key. Thus, the number of crossing edges is large and the cost of RDF query processing may be expensive. In contrast, our partitioning method is able to take full advantages of the semantics inherently embedded in RDF graphs.

3 Preliminaries

An RDF dataset is a set of triples which can be represented as a directed labeled graph, i.e., RDF graph, which is defined as follows.

Definition 1. (RDF graph) *Let U and L be the disjoint infinite sets of URIs and literals, respectively. A tuple $(s, p, o) \in U \times U \times (U \cup L)$ is called an* RDF triple*, where s is the* subject*, p is the* predicate *(a.k.a.* property*), and o is the* object*. A finite set of RDF triples is called an* RDF graph.

Given an RDF graph T, we use $S(T)$, $P(T)$, and $O(T)$ to denote the set of subjects, predicates, and objects in T, respectively. For a certain subject $s_i \in S(T)$, we refer to the triples with the same subject s_i collectively as the *entity s_i*, denoted by $Ent(s_i) = \{t \in T \mid \exists p, o \text{ s.t. } t = (s_i, p, o)\}$.

We can use RDF Schema (RDFS) to define classes of entities and the relationships between these classes. For example, $(s, \texttt{rdf:type}, C)$ declares that the entity s is an instance of the class C. Given an RDF graph T, we assume that for each subject $s \in S(T)$ there exists at least a triple $(s, \texttt{rdf:type}, C) \in Ent(s)$, denoted by $s \in C$. We believe that this assumption is reasonable since every entity should belong to at least one type in the real world.

Given RDF graph T and a subject $s \in S(T)$, the *class set* of s is $\mathcal{C}(s) = \{C \mid s \in C \wedge \nexists C' \text{ s.t. } (C', \texttt{rdfs:subClassOf}, C) \in T\}$. The *direct class* of s is defined as $C(s) = \bigcap_{C_i \in \mathcal{C}(s)} C_i$. The *class system* of an RDF graph T is defined as $\mathcal{C}(T) = \{C(s) \mid s \in S(T)\}$.

Definition 2. (Schema graph) *Given an RDF graph T, the* schema graph *of T is an* undirected *labeled graph, denoted by $G_S(T) = (V_S, E_S, l_S)$, where (1) $V_S = \mathcal{C}(T)$ is the finite set of vertices; (2) $E_S \subseteq V_S \times V_S$ is the finite set of edges; (3) $l_S : E_S \to P(T)$ is a function that assigns a predicate as the label to an edge; (4) for an edge $e = (C_1, C_2) \in E_S$, a triple exists $(s, p, o) \in T$ such that $C_1 = C(s)$, $C_2 = C(o)$, and $l_S(e) = p$.*

Let T be an RDF graph and $C_i, C_j \in V_S$. We use $T(C_i, C_j)$ to denote the set of all triples whose subjects are of type C_i and objects are of type C_j. $T(C_i, C_j, p_k)$ is used to denote the subset of $T(C_i, C_j)$ where the predicates are p_k, and the set of predicates in $T(C_i, C_j)$ is denoted as $P(C_i, C_j)$.

Definition 3. (Predicate ratio) *Given an RDF graph T and its schema graph $G_S(T)$, the predicate ratio of a predicate p_k between two classes $C_i, C_j \in V_S$ is defined as $\mathsf{pr}(C_i, C_j, p_k) = \frac{|T(C_i, C_j, p_k)|}{|T(C_i, C_j)|}$.*

Given an RDF graph T and its schema graph $G_S(T)$, the cardinality of a predicate p_k between two classes $C_i, C_j \in V_S$ is denoted by $\mathsf{card}(C_i, C_j, p_k) = (m, n)$ which means that for each entity s in C_i there are on average n entities in C_j that are connected to s via triples, and vice versa, for each entity s in C_j there are on average m entities in C_i that are connected to s via triples.

Definition 4. (Cardinality factor) *The cardinality factor of a predicate p_k between two classes $C_i, C_j \in V_S$ is defined as*

$$\mathsf{cf}(C_i, C_j, p_k) = \begin{cases} 1 & \textit{if } n \geq m \wedge \mathsf{card}(C_i, C_j, p_k) = (1, n) \\ -1 & \textit{if } n < m \wedge \mathsf{card}(C_i, C_j, p_k) = (m, 1) \\ \frac{n-m}{\max(m,n)} & \textit{otherwise} \end{cases} \quad (1)$$

4 Ranking RDF Classes

The algorithm classRank, inspired by PageRank, is shown in Algorithm 1. Intuitively, we consider the k classes with top-k highest *rank scores* (denoted as rs) as the *significant* classes. The differences between classRank and PageRank are: (1) instead of distributing rank score evenly in PageRank, in classRank, the rank score of each class is divided and distributed to its adjacent classes in proportion to $\mathsf{pr} \cdot |\mathsf{cf}|$; (2) a fragment of rank score is distributed to an adjacent class only if the corresponding $\mathsf{cf} < 0$.

Algorithm 1: classRank // Rank classes in an RDF schema graph

```
   Input  : A schema graph $G_S$ and the number of iterations $k$.
   Output: The schema graph $G_S$ with $rs$ on each vertex.
1  foreach class $C_i \in V_S$ do $C_i.rs \leftarrow 1/|V_S|$;          // Initialise $rs$
2  while $k > 0$ do
3      foreach class $C_i \in V_S$ do $C_i.\delta \leftarrow 0$;  // $\delta$ will record the change of $rs$
4      foreach class $C_i \in V_S$ do
5          $P_{in} \leftarrow \{(C_j, p_k) \mid \mathsf{cf}(C_i, C_j, p_k) < 0\}$;
6          foreach $(C_j, p_k) \in P_{in}$ do
7              $C_j.\delta \leftarrow C_j.\delta + C_i.rs \cdot \mathsf{pr}(C_i, C_j, p_k) \cdot |\mathsf{cf}(C_i, C_j, p_k)|$;
8              $C_i.\delta \leftarrow C_i.\delta - C_i.rs \cdot \mathsf{pr}(C_i, C_j, p_k) \cdot |\mathsf{cf}(C_i, C_j, p_k)|$;
9      foreach class $C_i \in V_S$ do $C_i.rs \leftarrow C_i.rs + C_i.\delta$;
10     $k \leftarrow k - 1$;
11 sort($C.rs$);                         // In descending order
12 return $G_S$
```

Algorithm 1 assigns each class in the RDF schema graph an initial rank score rs (line 1). The rank score rs of C_i is distributed to each of C_i's neighbouring classes C_j, and the amount of the rank score frament $rs \cdot \mathsf{pr}(C_i, C_j, p_k) \cdot |\mathsf{cf}(C_i, C_j, p_k)|$ is added to $C_j.rs$. This process (lines 2-10) will be iteratively executed k times. During each iteration, for each class C_i, the variable δ is used to record the changed value of rs. If the cf value of two adjacent classes C_i and C_j is less than zero (line 5), then C_i will distribute a fragment of rs to C_j (lines 6-8). At the end of each iteration, we update the rs of each class by adding its δ to rs (line 9).

It can be observed that the time complexity of classRank is bounded by $O(|V_S|^2|AE_m|)$, where $|V_S|$ is the size of the RDF schema graph and $|AE_m|$ is the maximum number of edges between adjacent vertices in the schema graph. Note that the size of the RDF schema graph G_S is much less than its corresponding RDF graph.

5 Semantic-Aware Partitioning

We can obtain the k classes with top-k highest rs values from the result of Algorithm 1. Based on these top-k ranked classes, we propose a novel RDF partitioning algorithm semPartition, which is able to leverage the inherent semantics embedded in RDF graphs as heuristic information to make a better RDF partitioning. Note that *entity* s is the basic partitioning unit in our algorithm.

Algorithm 2: semPartition `// Partition an RDF graph`

```
Input  : An RDF graph $T$, the schema graph $G_S$ with top-$k$ classes returned
         by Algorithm 1, the number of fragments $n$, and the threshold.
Output: A fragmentation $\mathcal{F} = \{F_1, \ldots, F_n\}$ of $T$.
1  for $i \leftarrow 1$ to $k$ do                                  // Partitioning phase
2      foreach $s \in C$ do                              // Class $C$ with the highest $rs$
3          if $s$ has not been processed then
4              $d \leftarrow \mathsf{hash}(s) \bmod n$; $F_d \leftarrow F_d \cup Ent(s)$;
5      foreach $C_i \in \{C$'s neighbouring classes$\}$ do
6          foreach $p \in \{$edges between $C$ and $C_i\}$ do
7              if $\mathsf{cf}(C, C_i, p) > threshold \wedge T(C, C_i, p)$ has not been processed then
8                  foreach $o \in \{o \mid \exists\ (s, p, o) \in T \wedge s \in C \wedge o \in C_i\}$ do
9                      if $\mathsf{frag}(o) = null$ then
10                         $d \leftarrow frag(s)$; $F_d \leftarrow F_d \cup Ent(o)$;
11 foreach $T(C_i, C_j, p)$ that has not been processed do   // Postprocessing phase
12     switch $(s, p, o) \in T(C_i, C_j, p)$ do
13         case $\mathsf{frag}(s) \neq null \wedge \mathsf{frag}(o) = null$
14             $d \leftarrow \mathsf{frag}(s)$; $F_d \leftarrow F_d \cup Ent(o)$;
15         case $\mathsf{frag}(s) = null \wedge \mathsf{frag}(o) \neq null$
16             $d \leftarrow \mathsf{frag}(o)$; $F_d \leftarrow F_d \cup Ent(s)$;
17         case $\mathsf{frag}(s) = null \wedge \mathsf{frag}(o) = null$
18             if $\mathsf{cf}(C_i, C_j, p) > threshold$ then
19                 $d \leftarrow \mathsf{hash}(s) \bmod n$; $F_d \leftarrow F_d \cup Ent(s)$;
20             else
21                 $d \leftarrow \mathsf{hash}(o) \bmod n$; $F_d \leftarrow F_d \cup Ent(o)$;
22 return $\mathcal{F} = \{F_1, \ldots, F_n\}$;
```

Algorithm 2 consists of two phases. During the partitioning phase (lines 1-10), the algorithm starts with the class C with the highest rs. For each subject s in the class C, $Ent(s)$ is assigned to a certain fragment F_d randomly by a hash function hash(s) on subjects and a modulo of the number of fragments n (lines 2-4). Then, for each neighbouring class C_i of C, the factor $\mathsf{cf}(C, C_i, p)$ and the *threshold* are used to decide whether an object o in class C_i need to be distributed into the same fragment as the subject s (i.e., $\mathsf{frag}(s)$) (lines 5-10). If $\mathsf{cf}(C, C_i, p) > \mathit{threshold}$ and $T(C, C_i, p)$ has not been processed yet, then objects in class C_i will be partitioned (lines 8-10), otherwise the partitioning of C_i will be ignored and the algorithm will (go to line 2 to) start a new iteration to process the class with the next highest rs in the top-k ranked classes.

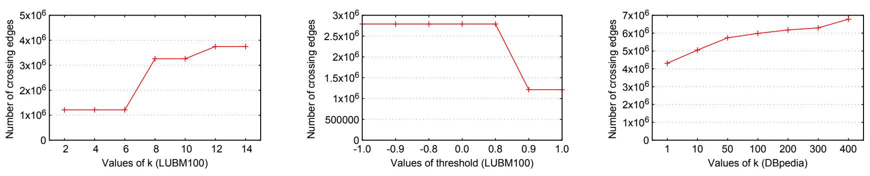
In the postprocessing phase (lines 11-21), the remaining RDF triples are partitioned in three cases: (1) if the subject s has been assigned to a fragment but not the object o, then $Ent(o)$ is assigned to the same fragment as s (lines 13-14); (2) if the object o has been assigned to a fragment but not the subject s, then $Ent(s)$ is assigned to the same fragment as o (lines 15-16); (3) if neither the subject s nor object o has been assigned to a fragment, then the assignment of s or o is determined by the comparison of $\mathsf{cf}(C_i, C_j, p)$ and $\mathit{threshold}$.

We have proved that Algorithm 2 can get a fragmentation $\mathcal{F} = \{F_1, \ldots, F_n\}$ of an RDF graph T, and every $(s, p, o) \in T$ satisfies $(s, p, o) \in F_i \wedge F_i \in \mathcal{F}$.

It can be observed that the time complexity of semPartition is bounded by $O(|deg_m||AE_m||C_m| + |E_S|)$, where $|deg_m|$ is the largest degree in G_S, $|AE_m|$ is the maximum number of edges between adjacent vertices in G_S, $|C_m|$ is the largest number of subjects in a class, and $|E_S|$ is the number of edges in G_S.

6 Experiments

We have conducted the experiments on both synthetic and real-world RDF graphs to verify the effectiveness and efficiency of our method. The prototype program, which is implemented in Python, is deployed on a desktop computer that has a Intel i5-6500 CPU with 4 cores of 3.2GHz, 8GB memory, 500GB disk, and 64-bit Ubuntu 14.04 as the OS. We generated 3 synthetic datasets using the LUBM benchmark, i.e., LUBM3, LUBM50, and LUBM100, which have 337K, 6.9M, and 13.9M RDF triples, respectively. We also run our algorithms on the real-world RDF graph DBpedia (version 2015-10) with 23M RDF triples. We compared our method with hash partitioning and METIS, and we did not consider the method in [5] because its sophisticated partitioning heuristics suffer from high preprocessing cost and high replication that is verified by [1].

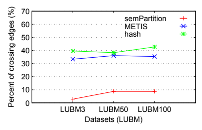


(a) Changing k (b) Changing $\mathit{threshold}$ (c) Changing k

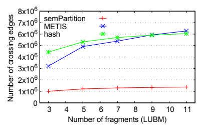
Fig. 1. The result of changing parameters

Experiment 1: Performance with different parameters. Let n denotes the number of fragment. We carried out experiments on LUBM100 and DBpedia, in which k varies from 2 to 14 and 1 to 400, respectively. As shown in Fig. 1(a) and 1(c), where $n = 5$, different values of k have significant impact on the partitioning results. The results are much better with $k = 2, 4, 6$ on LUBM100 and $k = 1$ on DBpedia. After k is over a certain value, the larger k is, the more subjects whose classes in top-k are partitioned randomly rather than according to the cf factor between classes, which results in the increasing of crossing edges. As shown in Fig. 1(b), semPartition was executed over LUBM100 with $k = 6$ and $n = 5$. When $threshold \geq 0.9$, the results are the best. The reason is that the $threshold$ determines the strength of the cf factor between classes.

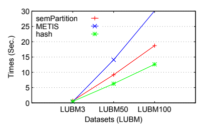
Experiment 2: Comparison between algorithms. We compared semPartition with hash partitioning and METIS. The comparison experiments were conducted on LUBM and DBpedia whose results are shown in Fig. 2.



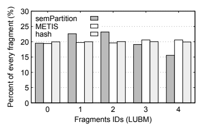
(a) Datasets



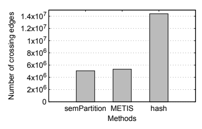
(b) Fragments



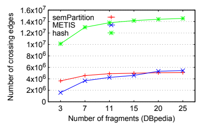
(c) Time



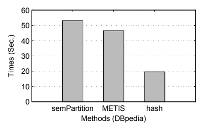
(d) Balance



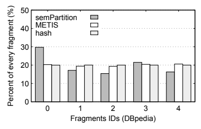
(e) DBpedia



(f) Fragments



(g) Time



(h) Balance

Fig. 2. Comparison between algorithms

1) Different size of datasets. We conducted experiments on three LUBM datasets with $n = 5$. When $k = 6$ and $threshold = 0.9$, our algorithm showed a much better result than hash partitioning and METIS, as shown in Fig. 2(a). We can observe that semPartition demonstrates a better fragmentation in which the numbers of crossing edges are 9.5k, 604.6k, and 1212.7k on LUBM3, 50, and 100, respectively. Our algorithm can generate the best partitioning on LUBM. For DBpedia, as shown in Fig. 2(e), when $k = 1$, $threshold = 0$, and $n = 20$, the number of the crossing edges of METIS and hash partitioning is more than that of semPartition. We believe that the reason why our method outperforms METIS includes: (1) semPartition fully leverages semantics in RDF graphs, while MEITS only considers the structure of graphs; (2) semPartition uses RDF triples as the units of partitioning, while METIS uses vertices as the units of partitioning.

2) Scalability by varying the number of fragments. The performance of semPartiton was verified on LUBM100 by varying n from 3 to 11 with $k = 2$, as shown in Fig. 2(b). We can see that our method is always the best one. For DBpedia, when changing n from 3 to 25, as shown in Fig. 2(f), the numbers of crossing edges in all methods have increased, which verifies our intuition. However, the growth rate of crossing edges in METIS is higher than that of our method, and the number of crossing edges of METIS has overtaken that of our method after the number of fragments becomes larger than 20.

3) Efficiency on different datasets. We verified the time efficiency of our method on LUBM and DBpedia. As shown in Fig. 2(c) and 2(g), hash partitioning has the least execution time on both types of datasets, while the execution times of our algorithm are competitive to that of METIS, since they are of the same order of magnitude though semPartition is just slower than METIS by a constant factor 1.14 on DBpedia. Not surprisingly, as hash partitioning does not need to consider any structural or semantic information, it is the fastest one.

4) Balance on different datasets. As shown in Fig. 2(d) and 2(h), when $n = 5$, we can observe that semPartition partitions RDF graphs approximately uniformly on LUBM100 and DBpedia, which is slightly inferior to METIS and hash partitioning. However, METIS overemphasizes the trade-off between balance and crossing edges, while our algorithm sacrifices a little bit balance to reduce the number of crossing edges. Therefore, the communication cost of parallel processing on RDF graphs partitioned by our algorithm can be reduced.

7 Conclusion

We have presented a semantic-aware method for partitioning RDF graphs based on ranked classes in RDF schema graphs. We argue that a partitioner should not only consider the structure of RDF graphs, but also take into account the inherent RDF semantics. Our experimental results on both synthetic and real-word data have verified the effectiveness and efficiency of our method, which outperforms the state-of-the-art RDF graph partitioning methods by a large margin.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (61572353), the Natural Science Foundation of Tianjin (17JCYBJC15400), the Open Fund Project of State Key Lab. for Novel Software Technology (Nanjing University) (KFKT2015B20), and the Australian Research Council (ARC) Discovery Project (DP130103051).

References

1. Harbi, R., Abdelaziz, I., Kalnis, P., Mamoulis, N.: Evaluating sparql queries on massive rdf datasets. Proceedings of the Vldb Endowment 8(12), 1848–1851 (2015)
2. Huang, J., Abadi, D.J., Ren, K.: Scalable sparql querying of large rdf graphs. Proceedings of the Vldb Endowment 4 (2011)
3. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1), 359–392 (2006)
4. Margo, D., Seltzer, M.: A scalable distributed graph partitioner. Proceedings of the Vldb Endowment 8(12), 1478–1489 (2015)
5. Wu, B., Zhou, Y., Yuan, P., Liu, L., Jin, H.: Scalable sparql querying using path partitioning. In: IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April. pp. 795–806 (2015)