IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ARTIFICIAL INTELLIGENCE (AI)-EMPOWERED INTELLIGENT TRANSPORTATION SYSTEMS

Received February 26, 2020, accepted March 6, 2020, date of publication March 23, 2020, date of current version April 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982652

Artificial Intelligence Knowledge Graph for Dynamic Networks: An Incremental Partition Algorithm

YONGLIN LENG, HONGMIN WANG, AND FUYU LU
College of Information Science and Technology, Bohai University, Jinzhou, China

Corresponding author: Hongmin Wang (wanghongmin0704@163.com)

This work was supported in part by the Science and Technology Project of the Liaoning Provincial Education Department under Grant LQ2017004, in part by the Social Science Planning Foundation of Liaoning Province under Grant L14AGL002 and Grant L19BGL016, in part by the National Key Research and Development Program of China under Grant 2019YFD0901605, and in part by the Science and Technology Foundation of Liaoning Province under Grant 2019-ZD-0503 and Grant 20170540005.

ABSTRACT The quick and intelligent requests and answers in artificial intelligence (AI) are inseparable from intelligent data. Knowledge graph makes data more intelligent by establishing association among data, which provides convenience for intelligent search, reasoning and analysis of data. Resource Description Framework (RDF) is an effective data representation model of knowledge graph. This paper takes RDF as the research object and proposes an incremental partition method of intelligent data (IPID) to realize the distributed storage of large-scale AI data. First, IPID gives a mixed object function integrating edge cut and load balancing. Second, IPID devises the initial and incremental partitioning algorithms of RDF. The initial partition divides the original RDF graph into kernel vertices, boundary vertices and free vertices. The boundary and freedom nodes select the kernel vertex with the maximum gain of object function to form a sub-partition. And the incremental partition is in charge of the selection of sub-partition of new and deleted data by the object function. Meanwhile, the incremental partition algorithm would also execute a dynamic adjustment strategy at a certain time interval according to the balance and tightness of sub-partition to satisfy the partitioning object. Finally, IPID is tested on the knowledge graph datasets. The experimental results show that the object function guarantees the quality of knowledge graph partition in edge cut and load balancing, and effectively realizes the incremental partition.

INDEX TERMS Artificial intelligence, knowledge graph, RDF, incremental partition, dynamic adjustment.

I. INTRODUCTION

At present, the society has entered the era of artificial intelligence (AI). The development of AI has given birth to a large number of intelligent applications. Such as intelligent transportation, researchers around the world have been working on new automotive applications to create a comfortable and safer driving environment [1], [2]. However, these works are inseparable from the intelligent retrieval and reasoning. The intelligent retrieval and reasoning are based on intelligent Data. Knowledge graph effectively reflects the intelligence of data by establishing the fragmented data association [3].

RDF (resource description framework) is an effective data representation model of knowledge graph, which is recommended by W3C to describe the data resource and their relationships [4]. RDF uses (subject, predicate, object) to describe a characteristic of resource. Meanwhile, the subject and object are considered as two vertices, and the predicate is regarded as the directed edge from the subject to object. So a large number of triples form a directed graph, namely knowledge graph [5], [6].

With the rapid development of various intelligent fields, the scale of knowledge graph increases exponentially [7]. In the face of large-scale knowledge graph, the single machine can not meet the demand of data storage [8]. Distributed storage is an effective solution to large-scale data storage. Nevertheless, the premise of distributed storage is how to partition data effectively.

At present, most of the researches are focused on the static knowledge graph partitioning. This is because some domain knowledge graphs update slowly. But there are also some

The associate editor coordinating the review of this manuscript and approving it for publication was Amr Tolba.

areas, such as intelligent transportation system. Because of the continuous update of roads, vehicles and other information, the knowledge graph is constantly changing, which puts forward a new challenge to data partition [9]. That is how to reasonably adjust the change data and meet the partitioning objects of edge cut and load balancing.

Based on above all, this paper proposes an incremental partitioning method of intelligent data (IPID) to realize the distributed storage of large-scale knowledge graph. The contributions are summarized as follows:

(1) A mixed object function combining the minimum edge cut and the load balancing is designed in IPID.
(2) Based on this object function, an original knowledge graph partitioning algorithm is proposed in IPID. The algorithm divides the original RDF graph into kernel vertices, boundary vertices and freedom vertices. And then the boundary and freedom vertices select the kernel vertices with the maximum gain of object function to form a partitioning subgraph.
(3) For the change data, an incremental partitioning algorithm is devised. The algorithm realizes the partition selection of changed data by the object function. Meanwhile, a dynamic adjustment strategy is executed in IPID at a certain time interval to satisfy the partitioning object.
(4) The standard knowledge graph datasets are selected to verify the effectiveness of the algorithm. The experimental results show that the object function guarantees the partition quality of knowledge graph in edge cut and load balancing, and effectively realizes the incremental partition.

The other parts of the paper are summarized as follows: the second part introduces the related work; the third part describes the design of mixed object function, the fourth and fifth parts describe the initial and incremental graph partitioning algorithms under the mixed object function. Finally, the experimental results verify the IPID algorithm's ability and draw the conclusion.

II. RELATED WORK

RDF graph model developed in the semantic web domain can well reflect the data relationship in knowledge graph. With the increase of data scale, the single machine mode can not meet the demand of data storage. Distributed storage is an effective solution to large-scale data storage. Because the nature of graph makes a large number of associations among data, so the premise of distributed storage is how to improve the internal data association and reduce the relationship of computing nodes.

SPA [10] adopts the vertex blocks to partition RDF graph. VB-Partitioner [11] extends the vertex block by N hops to avoid the query communications among computing nodes. Aiming at the asymmetric relationship between vertices and the power characteristics of RDF graph, BRGP proposes a modularity-based label propagation algorithm for the rough partition of RDF graph. Wu *et al.* [12] propose a path partition algorithm, which decomposes RDF graph into many paths from source to sink, and then divides it by path. Huang et al directly apply METIS framework for RDF graph partition, and reduce the communication of computing nodes in SPARQL query by replica method [13]. Wang *et al.* use the label propagation algorithm to coarse RDF graph vertices, and combine METIS to partition the rough graph [14]. All of these algorithms are based on static RDF graph and do not give solutions to the dynamic graph.

Zhu *et al.* propose an incremental clustering algorithm based on density which can realize the incremental partition of the graph, but the algorithm mainly tends to find the problem of overlapping communities in social networks, and does not consider the edge cut and load balancing [15]. Fennel is a flow-based graph partition framework, which combines some heuristic algorithms to achieve graph partition under the constraint of object function. Spinner proposed by Martella [16] is an incremental graph partitioning algorithm which extends the label propagation algorithm into a distributed graph partitioning algorithm based on the Pregel to ensure the scalability of the algorithm. Spinner can automatically adjust the result of graph partition according to the changes of the graph and computing environment to avoiding the repartition [17]. In addition, Spinner allows the global vertices to participate in the optimal calculation by restarting the iteration when dealing with incremental nodes. Although a very good result is obtained, but this way is very time-consuming. SPAR [18] divides the relevant information of social user into a storage node to ensure the partition of local semantic integrity while minimizing the copy. In addition, SPAR designs an adaptive segmentation strategy for the change of nodes, edges and storage nodes. Lv *et al.* design a dynamic partition method which gathers the high locality vertices and get a processing order by a priority-based scheduling algorithm [19]. LogGP [20] records, analyzes and reuses the historical statistical information to generate a hyper-graph. And then LogGP uses a novel hyper-graph streaming partitioning approach to refine the partitioning result. During the partitioning process, the LogGP evaluates the running time of each node according to the runtime statistics, and dynamically adjusts the subsequent execution.

III. MIXED OBJECT FUNCTION OF EDGE CUT AND LOAD BANLANCING

Given an RDF graph $G = (V, E)$, in which V and E are the set of vertices and edges, respectively. The $|V| = n$ and $|E| = m$ represent the number of vertices and edges. A k-way partition is to divide the RDF graph into k disjoint subgraphs $P = \{G_1, G_2, \ldots, G_k\}$, where $G_i = \{V_i, E_i\}$, $i = 1,2,\ldots, k$. For any $i \neq j$, $V_i \cap V_j = \varphi$ and $\sum_{i=1}^{k} V_i = V$.

Edge cut and loading balance are two important indexes to measure the quality of graph partitioning. Edge cut refer to the connecting edges across different subgraphs. The fewer edge cuts are, the fewer connections among subgraphs are, which also means less communication among storage nodes and high cohesion of data. The edge cuts between the subgraphs

G_i and G_j can be computed as follows:

$$inter\left(G_i, G_j\right) = \sum\nolimits_{a\in V_i, b\in V_j} e\left(a, b\right), \quad i \neq j \tag{1}$$

If the vertices a and b from two different subgraphs have an edge, then $e(a, b) = 1$, otherwise $e(a, b) = 0$. Corresponding to the edge cut, the edges inside any subgraph are defined as the inner cut which is described as follows:

$$intra(G_i) = \sum\nolimits_{a,b\in V_i} e(a, b) \tag{2}$$

Obviously, the higher the inner cut is, the closer the relationship is. That is to say the subgraph has a stronger independence.

The load balancing means that the number of vertices in each subgraph tends to be equal. Under the same hardware environment, the parallel execution efficiency is much higher if the vertex size of each subgraph tends to be equal. When the number of vertices in any subgraph G_i satisfies Eq. (3), the set of subgraphs P tends to be balanced. Among them, e is a floating factor between 0 and 1, which allows the graph size to float in a certain range.

$$\lceil n(1-e)/k \rceil \leq |V_i| \leq \lceil n(1+e)/k \rceil \tag{3}$$

From the above analysis, graph partitioning not only minimizes the edge cut, but also ensures the load balancing. So this paper builds a mixed object function $f(P)$ based on the edge cut and load balancing which satisfies the following requirements:

$$f(P) = cut(P) + balance(P) \tag{4}$$

where $cut(P)$ represents the edge cuts, for a k-way graph partitioning it satisfies Eq. (5):

$$cut(P) = \sum\nolimits_{i=1}^{k} \sum\nolimits_{j>i}^{k} inter(G_i, G_j) \tag{5}$$

$balance(P)$ represents the load balancing, it satisfies Eq. (6):

$$balance(P) = \sum\nolimits_{i=1}^{k} \sigma(G_i) \tag{6}$$

where

$$\sigma\left(G_i\right) = \left(|V_i| - \left\lceil \frac{n}{k} \right\rceil\right)^2 \tag{7}$$

Obviously, When the scale of the the subgraph is close to $\lceil \frac{n}{k} \rceil$, the value of $\sigma\left(G_i\right)$ is the smallest.

When the function $f(P)$ takes the minimum value, the graph partitioning result is the best. However, graph partitioning is a NP-complete problem, so it is difficult to obtain accurate partitioning results. So our goal is to find an approximate optimal solution. To achieve this goal, $f(P)$ is treated as follows:

Given a function $g(P)$ as Eq. (8):

$$g(P) = \sum\nolimits_{i=1}^{k} intra(G_i) - balance(P) \tag{8}$$

where $\sum_{i=1}^{k} intra(G_i)$ satisfies:

$$\sum\nolimits_{i=1}^{k} intra(G_i) = m - cut(P) \tag{9}$$

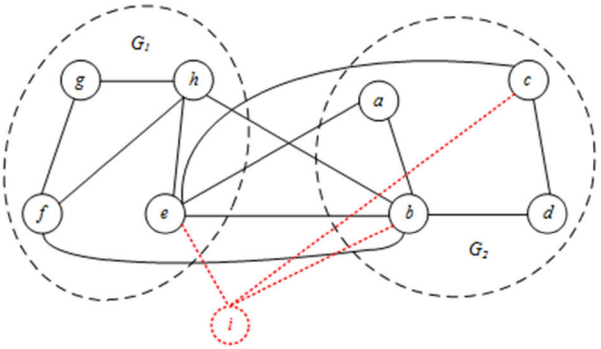

FIGURE 1. Calculation of the maximum gain value.

then from Eqs. (4, 8, 9):

$$g(P) = m - f(P) \tag{10}$$

If we find the minimum value of $f(P)$, it is equivalent to finding the maximum value of $g(P)$. Therefore, the partition of v can be determined by calculating the maximum gain of $\Delta g(P, i, v)$.

In Fig.1, let $n/k = 4$, the current $g(P)$ values of two partitioning subgraphs G_1 and G_2 are both 7. When the new node i is added to G_1 and G_2, respectively, the values of $g(P)$ are 7 and 8, and the gain values are $\Delta g\left(P, G_1, i\right) = 1$和$\Delta g\left(P, G_2, i\right) = 0$. So we select the G_2. It can also be seen intuitively from Fig.1 that the edge cut produced by adding I *to* G_2 is smaller than that of G_1, and the change of load balancing is consistent.

IV. INITIAL GRAPH PARTITIONING

In this section, an improved k-way Greedy Graph Growing Partitioning (IKGGGP) [21] based on mixed object function is proposed to partition the original RDF graph. IKGGGP first divides the graph vertices into kernel vertices belonging to only one sub-partition, boundary vertices existing in multiple sub-partitions and free vertices not belonging to any partitions. Fig.2 describes the three types of vertices, in which all the connections of red kernel vertices located in partition A, while the connections of purple boundary vertices belong to multiple partitions, and the blue free vertices do not belong any partitions. So for the boundary and free vertices, IKGGGP needs to choose a suitable partition by calculating the maximum gain of $g(P)$.

A. SELECTION OF KERNEL VERTEX

SPARQL query is a subgraph matching problem in essence. So if the closely related vertices locate in the same storage node, the communication will be greatly reduced. When a vertex is very similar to its neighbors, it is easier for it to become a kernel vertex.

Giver an RDF graph, $\Gamma(v)$ is the neighbor set of v. $\Gamma_\varepsilon(v)$ represents a set of vertices whose similarity with v is greater or equal to the threshold value ε. If the number of $|\Gamma_\varepsilon(v)|$ exceeds the threshold δ, then the vertex v is a kernel vertex.

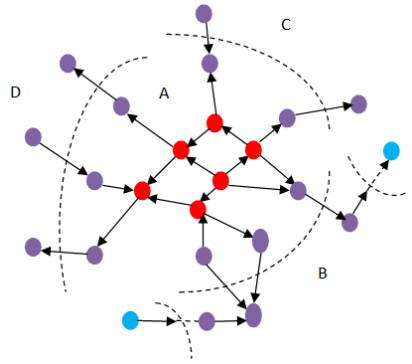

FIGURE 2. Division of initial graph vertices.

Searching for the kernel vertex needs to compute the similarity between vertex and its neighbors. The traditional methods for calculating the similarity are based on the structure information of vertices, such as random walk [22], common neighbor [23], and so on. Because of the semantic feature of knowledge graph, there are some semantic relations among nodes. If the vertices with close semantic association are divided into the same partition, it will be more conducive to the parallel query. Therefore, IKGGGP devises a vertex similarity measure method based on semantics.

We know that RDF graph uses Uniform Resource Identifier (URI) to identify vertices and edges. URI reflects the information of vertex class and superclass through directory structure. If two vertices have more common superclass, they have greater similarities. So we use $sup(v)$ to calculate the superclass of vertex v. The semantic similarity between u and v is measured as follows:

$$similarity\,(u, v) = \frac{|sup(u) \cap sup(v)|}{|sup(u) \cup sup(v)|} \tag{11}$$

When the kernel vertices are determined, we can divide the graph initially. Algorithm1 gives the pseudo code of the initial partition.

B. DISTRIBUTION OF BOUNDARY AND FREE VERTICES

After the kernel vertices have been determined, there are another two types of vertices, namely the boundary vertices and free vertices. These two kinds of vertices need to calculate the maximum gain value $g(P)$ to select the partition. It can be described as follows:

$$partition(v) = argmax\,\Delta g(P, v, i), \quad i = 1, 2, \ldots, k \tag{12}$$

The execution steps of vertex allocation in IKGGGP algorithm are as follows:

Step 1: Calculate the gain value Δg of all the vertices to be allocated to each partition;

Step 2: Select the vertex v that produces the maximum gain, assign it to partition i with maximum gain, and remove v from the vertex set to be allocated;

Step 3: If the vertex set to be allocated is not empty, update the gain value of the remaining vertices, return to step 1, otherwise, go to step 4;

Step 4: Output the results of partition.

Algorithm 1 Initial Partition

Input: the core set of vertex *core*, graph G
Output: initial partition $P = \{G_1, G_2, \ldots, G_k\}$
Method:

```
Initial queue(q); p < −1;
Do while p <= k
  v <–select(core);
  remove(core, v);
  inqueue(q, v);
  i< −1;
  partition(v) < −p;
  number(v) < −i;
  i < − i + 1; G_p = G_p ⋃ v;
  Do while not isempty(q) and i < n/k
    v <–outqueue(q);
    for each u in Γ_ε(v)
      if u ∈ core
          remove (core, u);
      if u ∉ G_p
          inqueue (q, u);
          partition (u) < −p;
          number (u) < −i;
      if i = n/k
          exit for
      i < −i + 1; G_p = G_p ⋃ u;
    end for
  end do
  p = p+1;
  clearqueue(q);
end do
```

In order to improve the efficiency, IKGGGP gives an improve strategy. Firstly, the algorithm uses a two-dimensional array to store the gain value Δg. Each row of the array corresponds to a vertex, and each column corresponds to a partition. When the vertex v is assigned to the partition i, the gain variation of other vertices w in each partition as follows:

$$\begin{cases} \Gamma(v) - \Gamma_i(v) + 2\left(\frac{n}{k} - 1 - |v_i|\right) w \in \Gamma(v)\ not\ in\ G_i \\ \Gamma(v) - \Gamma_i(v) + \frac{2n}{k} - 1 - 2|v_i|\,, others \end{cases} \tag{13}$$

Secondly, in order to easily find the vertex that produces the maximum gain, IKGGGP uses an adjacency list to record the corresponding information of maximum gain value of each vertex. Each array element is a value in the maximum gain range and sorted in descending order. At the same time, each element points to a two-way linked list to store the vertex information that produces the same maximum gain value.

V. INCREMENTAL GRAPH PARTITIONING

In some application areas, knowledge graph is constantly changing, so the corresponding graph partition algorithm should also give reasonable solutions to these changes to ensure that the data partition always meets the requirements

the objects of distributed storage. In view of the dynamic change of RDF graph, this paper designs an incremental partition and adjustment strategy of RDF graph.

A. INSERT TRIPLES

There are three cases of inserting the new triples. First, the subject and object both exist in the original graph. Second, there is only one subject or object in the original graph. And the third, both of the subject and object are non-existent.

For a new triple, if the subject and object already exist in the partition, inserting a triple is to add a new relationship. If s, $o \in V_i$, adding a triple does not change the target function, so you can insert the triple directly. If $s \in V_i$, $o \in V_j$, and $i \neq j$, it is necessary to judge the following conditions and select the case with the maximum gain value to add.

Without moving any vertex and adding an edge directly between s and o, the gain value Δg is -1.

Under the premise of ensuring the load balancing of graph partition, the vertex s or o is migrated to the other partition, the gain value Δg changes as shown in Eq. (14).

$$\begin{cases} |\Gamma_j(s)| - |\Gamma_i(s)| + 2\left(|V_i| - |V_j| - 1\right), & s,\ o \in V_j \\ |\Gamma_i(o)| - \left|\Gamma_j(o)\right| + 2\left(\left|V_j\right| - |V_i| - 1\right), & s,\ o \in V_i \end{cases} \tag{14}$$

Assume the existing vertex s in the partition V_i, and the new added vertex is o, then vertex o has two options. One is the partition i, the other is the left partitions. The gain value is shown in Eq. (15). Similarly, the partition with the maximum gain value is selected on the premise of load balancing.

$$\begin{cases} \dfrac{2n}{k} - 1 - 2|v_i|, & s, o \in V_i \\ 2(\dfrac{n}{k} - 1 - |v_j|), & j \neq i \text{且} s \in V_i,\ \ o \in V_j \end{cases} \tag{15}$$

If the subject and object of a triple do not exist, then they have nothing associations with the vertices of graph. The strategy is to allocate the triple to the partition with the least vertices.

B. DELETE TRIPLES

Deleting a triple is to delete a directed edge from s to o. if $|\Gamma(s)| > 1$ and $|\Gamma(o)| > 1$, and s and o belong to different partitions, then the deletion of edge can reduce the overall edge cut without affecting the load balancing, so delete the edge directly.

If $|\Gamma(s)| = 1$ or $|\Gamma(o)| = 1$, then the vertex with one neighbor needs to be deleted together. The reduction of edge cut will improve the partition quality, but the deletion of vertices may lead to the load imbalance. Meanwhile, the deletion of triples only adjusts the allocation of nodes locally, the graph partition may appear locally optimal after a time interval. In order to avoid the above situation, after each T time period the graph will be dynamically adjusted.

Because of the deletion of triples, the partition of subgraphs will be unbalanced. If the subgraph G_i does not meet the requirements of load balancing, it needs to be adjusted.

According to Eq. (3), there are two cases that do not satisfy the load balancing.

First, the set of vertices *of* the subgraph satisfies $|V_i| < \lceil n(1-e)/k \rceil$. For this case, assuming the neighborhood vertex set of the subgraph G_i is $\Gamma(V_i)$, $\forall v \in \Gamma(V_i)$, $P(v) = j$. Moving vertex v from the partition j to the partition i, the gain value of the object function is as follows:

$$\Delta g = \Gamma_j(v) - \Gamma_i(v) + 2(\left|V_j\right| - |V_i| - 1) \tag{16}$$

Select the vertex v from $\Gamma(V_i)$ which can get the the maximum gain value and move it to the partition G_i until $|V_i|$ satisfies the Eq. (3).

For another case, the set of vertices of the subgraph G_i satisfies $|V_i| > \lceil n(1+e)/k \rceil$. We use $\eta(V_i)$ to represent the set of vertices associated with other subgraphs in the subgraph G_i. Then $\forall v \in \eta(V_i), \exists u \in \Gamma(v)$, $P(u) = j, j \neq i$, when v is moved from partition i to j, the gain value of the object function is:

$$\Delta g = \Gamma_i(v) - \Gamma_j(v) + 2(|V_i| - \left|V_j\right| - 1) \tag{17}$$

Again, select the vertex v from $\Gamma(V_i)$ which can get the maximum gain value and move it to the partition G_i until $|V_i|$ satisfies Eq. (3).

C. DYNAMIC ADJUSTMENT STRATEGY

When the partition meets load balancing, the inner vertices in subgraph are closely related and the relationships between subgraphs should be sparse. In this paper, the cohesion and coupling of subgraphs are used to measure the compactness of partition subgraphs. Eq. (18, 19) give the method to calculate the degree of cohesion and coupling.

$$cohesion(G_i) = \frac{intra(G_i)}{intra(G_i) + \sum_{j=1, j\neq i}^{k} inter(G_i, G_j)} \tag{18}$$

$$connection(G_i, G_j) = \frac{inter(G_i, G_j)}{intra(G_i) + \sum_{j=1, j\neq i}^{k} inter(G_i, G_j)} \tag{19}$$

For any partition subgraph, if the cohesion degree is greater than the coupling degree between it and other subgraphs, it means that the internal vertices of the subgraph are closely related. If the cohesion of a subgraph is lower than its coupling with other subgraphs, that means there are some vertices in the subgraph, which are more closely related to other subgraphs.

Fig. 3 gives the result of swapping the vertices b and e in the two partition subgraphs G_1 and G_2. In Fig. 3, the dotted line represents the inner cuts and the solid line is the edge cuts. For the subgraph G_1, the degree of cohesion and coupling before exchange are 4/10 and 6/10, respectively. When the vertices are swapped, the two values are 5/10 and 5/10, respectively. Similarly, subgraphs G_2 also changed from 5/11 and 6/11 to 5/10 and 5/10. It can be seen that the cohesion of G_1 and G_2 is improved, while the coupling degree is decreased. And the most obvious performance is the edge cut decreased.

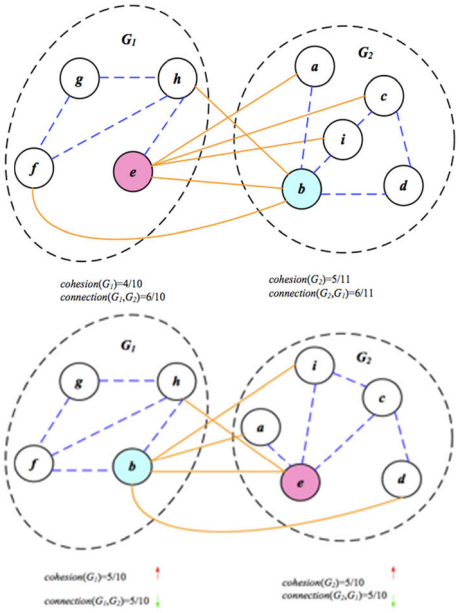

FIGURE 3. Adjustment of balance.

Given $\eta(V_i)$ represents the vertex set associated with other subgraphs in G_i. $\forall v \in \eta(V_i)$, $u \in \Gamma_j(v)$, $j = 1, 2 \ldots, k$ and $j \neq i$, if $\Gamma(u) < \Gamma(v)$ and the neighbor of u and v in the subgraphs G_i and G_j satisfies Eq. (20), then the vertices u and v are exchanged.

$$\Gamma_i(v) + \Gamma_j(u) < \Gamma_j(v) + \Gamma_i(u) \tag{20}$$

The steps to adjust the cohesion of the subgraph are as follows:

Step1: Calculate the degree of cohesion and coupling between the partition subgraphs according to the Eqs. (18, 19);

Step2: If there is a subgraph, which satisfies cohesion(G_i) < connection(G_i, G_j), $j = 1, 2, \ldots, k$, then perform step 3; otherwise, go to step 5;

Step3: If there exist $v \in \eta(V_i)$, $u \in \Gamma_j(v)$ and $\Gamma(u) < \Gamma(v)$, and satisfy Eq. (20) at the same time, then exchange the vertices u and v; otherwise, go to step 5;

Step4: Update the degree of cohesion and coupling between subgraphs, then go to step 2;

Step5: Output the adjusted partition results.

VI. EXPERIMENTS

In this paper, the performance of IPID algorithm is verified from two aspects: the initial and incremental graph partition. The test datasets include two knowledge graphs: LUBM and UniProt.

A. DATASETS AND SETTING

LUBM is a kind of standard synthetic dataset, which is used to describe an ontology of college departments, classes and professors and their relationships. UniProt is a free access real dataset of protein sequence and function information. Table 1 lists the basic information about these two datasets.

TABLE 1. Statistic of datasets used in experiments.

Dataset	#Vertex	#Edge
LUBM50	1,706,230	6,888,642
LUBM2000	66,059,204	276,345,040
UniProt1	20,372,462	100,000,0000
UniProt2	139,942,781	687,025,165

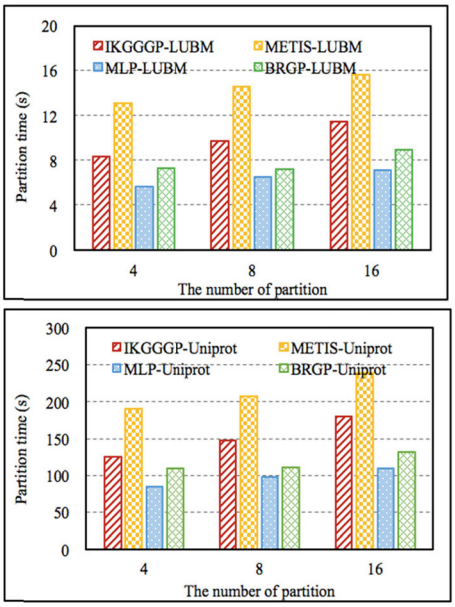

FIGURE 4. Comparison of partition time for initial partition.

The experiment environment in hardware is: Inter Xeon 2.00GHz × 24 processor, 20GB memory. The software environment is: 64-bit Linux operating system, C++ and compiled using G++, with-O2 option to optimize.

B. PARTITIONING PERFORMANCE OF IKGGGP

We select the LUBM50 and UniProt1 to verify the initial graph partition algorithm IKGGGP. The comparison algorithms include METIS, MLP + METIS and BRGP. The parameters are $\varepsilon = 0.3$, $\delta = 2m/n$.

Each partition algorithm is performed three times, and then take the average value of the result. For load balancing, we take the mean ratio of the largest subgraph and the partition subgraph. Both algorithms divide the dataset into three different size, namely, 4, 8 and 16. The comparison results include three aspects: partition time, edge cut and load balancing.

As shown in Fig. 4, the partition time of IKGGGP algorithm is better than METIS, and lower than MLP + METIS and BRGP. The execution efficiency of IKGGGP, MLP + METIS and BRGP is not directly related to the vertex degree distribution, so they are not affected by the high degree vertices.

The vertex coarsening speed of METIS algorithm is affected by vertex degrees [24], [25]. There are some central vertices in the knowledge graph which have extensive connections with other resources. So the efficiency of METIS is

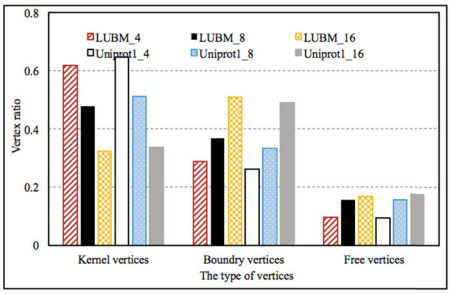

FIGURE 5. Statistics of vertex for initial partitioning.

lower than the other three algorithms. The result proves that the general graph partition framework is not suitable for the partition of knowledge graph.

For MLP+METIS and BRGP algorithms, the vertices are coarsened layer by layer by using LPA algorithm. The time complexity of LPA in each layer is $O(tm)$, where t is the number of iterations, and m is the number of edges in each layer. The IKGGGP algorithm needs to calculate the similarity between the vertex and its neighborhood to find out the core vertex, and then allocate the left nodes. The time complexity is $O(m + n)$. However, the IKGGGP algorithm is lower than MLP+METIS and BRGP in efficiency, because the IKGGGP algorithm needs to constantly update the maximum gain value of the vertices to be distributed.

In addition, the partition efficiency of IKGGGP algorithm is related to the number of distributed vertices. Fig. 5 lists the proportion of three types of vertices generated in the initial partition. It can be seen that the number of kernel vertices decreases with the increase of partition number, but the total number is still more than 30%. Therefore, at least 30% vertices do not participate in the redistribution. It is also a reason why the time efficiency of IKGGGP algorithm is higher than METIS.

MLP and BRGP both select LPA as their coarsening method. BRGP uses modularity to replace the select scheme of MLP based on common neighborhoods, which improves the quality of coarsened graph. Compared with LPA for indirect selection edge cut and load balancing optimization, the gain function introduced of IKGGGP algorithm directly selects the optimal condition of edge cut and load balancing for vertex distribution, so in Figs. (6, 7) the results of IKGGGP algorithm have more advantages in several algorithms.

C. INCREMENTAL GRAPH PARTITION

According to the dynamic change of RDF graph, the incremental partition and adjustment strategy is implemented on the result of initial partition. The experiment verifies the performance by continuously inserting triples.

The two comparison algorithms are Spinner and Fennel. Firstly, Lubm50 and Uniprot1 are executed on IKGGGP and Spinner as initial static data, and other data are generated by dynamic insertion. Fennel directly increases the data in the way of flow. The final data volume of the three algorithms

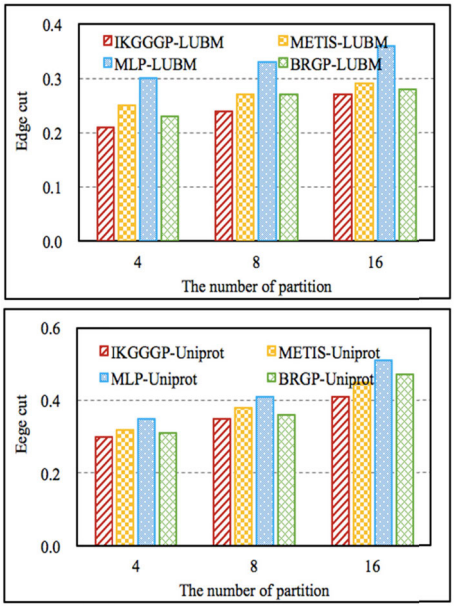

FIGURE 6. Comparison of edge cut for initial partition.

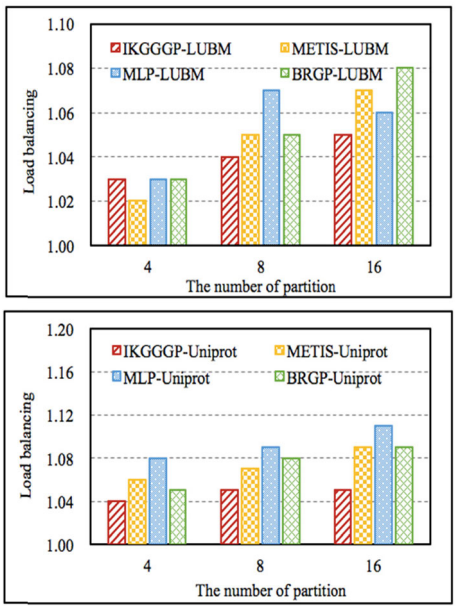

FIGURE 7. Comparison of load balancing for initial partition.

reach Lubm2000 and UniProt2 datasets. The dynamic adjustment interval of IPID is set to $T = 5S$.

Fig. 8 shows the three algorithms' experimental results in partition time, edge cut and load balancing. Because Fennel algorithm adopts flow partition and the time efficiency is linear, so it has the highest partition efficiency on both datasets. The partition efficiency of Spinner is lower than that of Fennel and IPID. The reason lies that Spinner restarts iteration to find the optimal partition each time, so the efficiency will gradually decrease with the increase of the number of vertices. Although IKGGGP algorithm used by IPID in original graph partition is not efficient, but in incremental graph, because IPID adopts linear allocation scheme and vertex adjustment

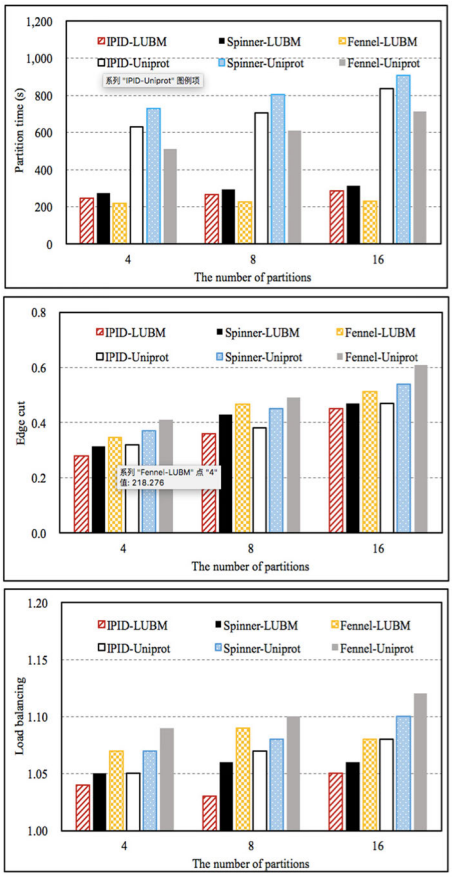

FIGURE 8. Performance verification of incremental partitioning algorithm.

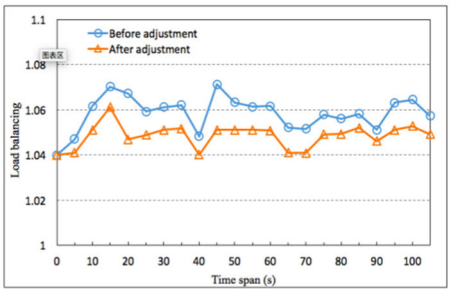

FIGURE 9. The influence of dynamic adjustment on the load balancing of graph.

is also based on local subgraph, so its efficiency is between Spinner and Fennel.

In the experimental results of edge cut and load balancing, IPID is better than the other two algorithms. Although Fennel algorithm is also based on edge cut and load balancing object function, but IPID adds dynamic adjustment function, so the whole partition does not fall into the local optimum. The load balancing of Spinner is based on edge. In this paper, the balance is mainly based on nodes, so the balance of Spinner is lower than that of IPID and Fennel.

In order to verify the validity of triple deletion and dynamic adjustment, we use a random way to delete some triples from the partition graph every interval T. Fig. 9 shows the change of load balancing before and after dynamic adjustment on

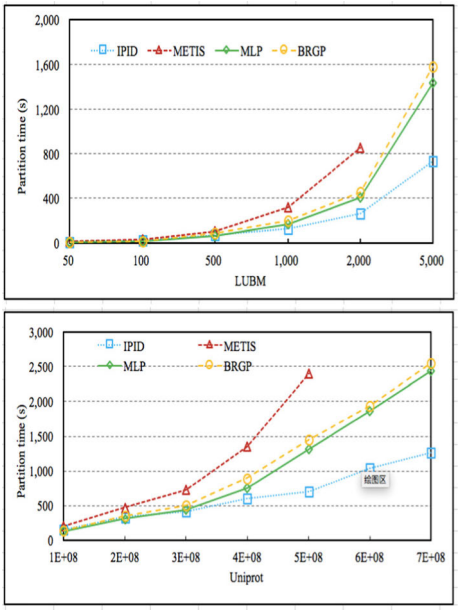

FIGURE 10. Incremental partitioning efficiency of IPID.

UniProt dataset. It can be seen that the reduction of graph vertices leads to the increase of the imbalance degree of the largest subgraph. When IPID adds dynamic adjustment function, the balance will be lower than that before adjustment, so that the balance of the whole graph partition will always be maintained in a small range of changes.

D. PERFORMANCE OF IPID IN INCREMENTAL GRAPH

Although Fig. 4 shows that the efficiency of IKGGGP is lower than that of MLP + METIS and BRGP. But its edge cut and load balancing are better than the other three algorithms.

However, with the increase of data, the partition efficiency has changed significantly. IPID uses IKGGGP as the initial partition result, and then adopt the incremental vertex assignment and auto adjustment strategy. MLP + METIS, METIS and BRGP adopt the way of repartition when the new data comes. Fig. 10 shows the time performance of two datasets which are divided into eight partitions using different partition algorithms. Among them, IPID selects Lubm50 and UniProt1 as the initial datasets.

The partition time of METIS, MLP+METIS and BRGP algorithm in different size are obtained by directly applying the algorithm to the whole datasets. It can be seen that with the increase of dataset size, the partition time of IPID algorithm increases slowly, while METIS, MLP + METIS and BRGP algorithms are obvious higher than IPID.

And the larger the dataset is, the better the IPID algorithm is. The reason is that IPID only considers the assignment of change vertices, so the partition speed is very fast. However, the other three algorithms need more iterations to get the final result when the data size changes. So IPID is effective in incremental data partition. In addition, with the increase of data scale, METIS algorithm can not realize data partition in Lubm5000 dataset and the UniProt dataset up to 500 million. Because the size of the data exceeds the memory.

VII. CONCLUSION

Knowledge graph includes a large number of domain knowledge, which can effectively reflect the relationship between knowledge. RDF data model is one of the effective ways to express knowledge graph. With the continuous expansion of domain knowledge, distributed storage is an inevitable trend for large-scale knowledge graph storage. In some fields, such as intelligent transportation, knowledge and information are constantly changing. In this paper, an incremental partition algorithm is designed for the dynamic domain knowledge. It provides an effective allocation scheme for data distributed storage. This scheme effectively solves the problems of edge cut and load balancing in distributed storage by defining the mixed object function. At the same time, IPID fully considers the semantic and structural information of knowledge graph, so that the partition results are more conductive to intelligent data retrieval. The experimental results also show that the effectiveness of the object function and the efficiency of the incremental partition and adjustment algorithm.

REFERENCES

[1] Z. Ning, Y. Feng, M. Collotta, X. Kong, X. Wang, L. Guo, X. Hu, and B. Hu, "Deep learning in edge of vehicles: Exploring trirelationship for data transmission," *IEEE Trans. Ind. Informat.*, vol. 15, no. 10, pp. 5737–5746, Oct. 2019.

[2] Z. Ning, J. Huang, X. Wang, J. J. P. C. Rodrigues, and L. Guo, "Mobile edge computing-enabled Internet of vehicles: Toward energy-efficient scheduling," *IEEE Netw.*, vol. 33, no. 5, pp. 198–205, Sep. 2019.

[3] A. Tolba and E. Elashkar, "Soft computing approaches based bookmark selection and clustering techniques for social tagging systems," *Cluster Comput.*, vol. 22, no. S2, pp. 3183–3189, Mar. 2019.

[4] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao, "Semantic SPARQL similarity search over RDF knowledge graphs," *Proc. VLDB Endowment*, vol. 9, no. 11, pp. 840–851, Jul. 2016.

[5] N. Zaki, C. Tennakoon, and H. A. Ashwal, "Knowledge graph construction and search for biological databases," in *Proc. Int. Conf. Res. Innov. Inf. Syst. (ICRIIS)*, Jul. 2017, pp. 1–6.

[6] O. AlFarraj, A. Tolba, S. Alkhalaf, and A. AlZubi, "Neighbor predictive adaptive handoff algorithm for improving mobility management in VANETs," *Comput. Netw.*, vol. 151, pp. 224–231, Mar. 2019.

[7] Z. Ning, Y. Li, P. Dong, X. Wang, M. S. Obaidat, X. Hu, L. Guo, Y. Guo, J. Huang, and B. Hu, "When deep reinforcement learning meets 5G-enabled vehicular networks: A distributed offloading framework for traffic big data," *IEEE Trans. Ind. Informat.*, vol. 16, no. 2, pp. 1352–1361, Feb. 2020.

[8] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, p. 60, 2019.

[9] A. Tolba, "Content accessibility preference approach for improving service optimality in Internet of vehicles," *Comput. Netw.*, vol. 152, pp. 78–86, Apr. 2019.

[10] K. Lee, L. Liu, Y. Tang, Q. Zhang, and Y. Zhou, "Efficient and customizable data partitioning framework for distributed big RDF data processing in the cloud," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, Jun. 2013, pp. 327–334.

[11] K. Lee and L. Liu, "Efficient data partitioning model for heterogeneous graphs in the cloud," in *Proc. Int. Conf. High Perform. Comput., Netw.*, 2013, pp. 1–12.

[12] B. Wu, Y. Zhou, P. Yuan, L. Liu, and H. Jin, "Scalable SPARQL Querying using path partitioning," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 795–806.

[13] J. Huang, D. J. Abadi, and K. Ren, "Scalable sparql Querying of large rdf graphs," *Proc. VLDB Endowment*, vol. 4, no. 11, pp. 1123–1134, 2011.

[14] L. Wang, Y. Xiao, B. Shao, and H. Wang, "How to partition a billion-node graph," in *Proc. IEEE 30th Int. Conf. Data Eng.*, Mar. 2014, pp. 568–579.

[15] M. Zhu, F. Meng, and Y. Zhou, "Density-based link clustering algorithm for overlapping community detection," *J. Comput. Res. Develop.*, vol. 50, no. 12, pp. 2520–2530, Dec. 2013.

[16] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: Streaming graph partitioning for massive scale graphs," in *Proc. 7th ACM Int. Conf. Web Search Data Mining (WSDM)*, 2014, pp. 333–342.

[17] C. Martella, D. Logothetis, A. Loukas, and G. Siganos, "Spinner: Scalable graph partitioning in the cloud," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 1083–1094.

[18] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The little Engine(s) that could: Scaling online social networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 1162–1175, Aug. 2012.

[19] X. Lv, W. Xiao, Y. Zhang, X. Liao, H. Jin, and Q. Hua, "An effective framework for asynchronous incremental graph processing," *Frontiers Comput. Sci.*, vol. 13, no. 3, pp. 539–551, Jun. 2019.

[20] N. Xu, L. Chen, and B. Cui, "LogGP: A log-based dynamic graph partitioning method," *Proc. VLDB Endowment*, vol. 7, no. 14, pp. 1917–1928, Oct. 2014.

[21] M. Predari and A. Esnard, "A k-Way greedy graph partitioning with initial fixed vertices for parallel applications," in *Proc. 24th Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process. (PDP)*, Feb. 2016, pp. 280–287.

[22] X. Wang, Z. Ning, X. Hu, L. Wang, L. Guo, B. Hu, and X. Wu, "Future communications and energy management in the Internet of vehicles: Toward intelligent energy-harvesting," *IEEE Wireless Commun.*, vol. 26, no. 6, pp. 87–93, Dec. 2019.

[23] H. Khosravi-Farsani, M. Nematbakhsh, and G. Lausen, "SRank: Shortest paths as distance between nodes of a graph with application to RDF clustering," *J. Inf. Sci.*, vol. 39, no. 2, pp. 198–210, Apr. 2013.

[24] Z. Ning, P. Dong, and X. Wang, "Mobile edge computing enabled 5G health monitoring for Internet of medical Things: A decentralized game theoretic approach," *IEEE J. Sel. Areas Commun.*, to be published.

[25] Z. Ning, K. Zhang, and X. Wang, "Joint computing and caching in 5G-envisioned Internet of vehicles: A deep reinforcement learning-based traffic control system," *IEEE Trans. Intell. Transp. Syst.*, early access, Feb. 5, 2020, doi: 10.1109/TITS.2020.2970276.

YONGLIN LENG received the B.S. and M.S. degrees in computer science and technology from Bohai University, in 2003 and 2009, respectively, and the Ph.D. degree in software engineering from the Dalian University of Technology, in 2018. She is currently an Associate Professor with Bohai University. Her research interests mainly focus on the storage and index of RDF graph data, clustering, and filling for incomplete data.

HONGMIN WANG received the B.S. degree in computer science and technology from Liaoning University, in 2002, and the M.S. degree in education from Bohai University, in 2008. She is currently an Associate Professor with Bohai University. Her research interest mainly focuses on the storage of RDF graph data.

FUYU LU received the B.S. degree in computer science and technology from Bohai University, in 2003, and the M.S. degree in computer science and technology from the Dalian University of Technology, in 2009. He is currently an Instructor with Bohai University. His research interests mainly focus on the retrieval and reasoning of artificial intelligence data.