Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 7148071, 15 pages
https://doi.org/10.1155/2023/7148071

Research Article

RDF Subgraph Query Based on Common Subgraph in Distributed Environment

Qingrong Huang, Xiaocong Lai, Qianxiang Su, and Ying Pan

School of Computer and Information Engineering, Nanning Normal University, Nanning 530001, China

Correspondence should be addressed to Ying Pan; panying@nnnu.edu.cn

Received 22 January 2022; Revised 9 November 2022; Accepted 16 November 2022; Published 13 January 2023

Academic Editor: Iftikhar Ahmad

With the gradual development of the network, RDF graphs have become more and more complex as the scale of data increases; how to perform more effective query for massive RDF graphs is a hot topic of continuous research. The traditional methods of graph query and graph traversal produce great redundancy of intermediate results, and processing subgraph collection queries in stand-alone mode cannot perform efficient matching when the amount of data is extremely large. Moreover, when processing subgraph collection queries, it is necessary to iterate the query graph multiple times in the query of the common subgraph, and the execution efficiency is not high. In response to the above problems, a distributed query strategy of RDF subgraph set based on composite relation tree is proposed. Firstly, a corresponding composite relationship is established for RDF subgraph set, then the composite relation graph is clipped, and the redundant nodes and edges of the composite relation graph are deleted to obtain the composite relation tree. Finally, using the composite relation tree, a MapReduce-based RDF subgraph set query method is proposed, which can use parallel in the computing environment, the distributed query batch processing is performed on the RDF subgraph set, and the query result of the RDF subgraph set is obtained by traversing the composite relation tree. The experimental results show that the algorithm proposed in this paper can improve the query efficiency of RDF subgraph set.

1. Introduction

With the development of science and technology, more and more data, such as those in the fields of chemistry and bioinformatics, are described and processed in the form of resource description framework (RDF), which is a graph-based data model with strong expressiveness and simplicity characteristics.

As a resource description framework for knowledge representation and reasoning in the Semantic Web, RDF has attracted the attention of many researchers. Researchers usually use SPARQL Protocol and RDF Query Language (SPARQL) to retrieve RDF data. The problem of RDF subgraph query refers to how to find part of the data graph that matches the preset RDF subgraph in all RDF data graphs. At present, most of the traditional methods are to process a single RDF graph query sequentially.

With the release of a large amount of RDF data, the query problem usually needs to batch process the RDF subgraph set. For example, it is common in the Semantic Web to encounter many problems with multiple SPARQL queries that need to be processed simultaneously, in which case the collection of RDF subgraphs needs to be processed simultaneously. Because there are many identical partial graphs in RDF subgraph set, if each RDF graph is queried in turn, repeated traversal and repeated computation will occur, which makes the query inefficient and poor query performance.

In addition, using SPARQL to query large-scale RDF data often exceeds the processing capacity of a single computer. The core of large-scale RDF data query is to solve how to carry out subgraph matching efficiently, and there are two different directions to deal with subgraph matching: one direction is to develop a special RDF query processing engine using the characteristics of existing RDF data, and

the other direction is to develop an effective subgraph matching algorithm. Therefore, the subgraph matching method of distributed RDF graphs has attracted more and more attention. In distributed RDF query processing, the scale and complexity of RDF graphs are increasing, including a large number of graph data nodes and their node attributes, as well as the complex correlation between graph data nodes. The amount of data is huge, and different information sources make the form of data inconsistent. Moreover, the combination of different forms of data will produce many repeated and irrelevant intermediate result sets, and there will also be many isomorphic relationships of subgraphs, which reduces the query efficiency when querying RDF subgraphs.

In order to find an adaptive solution, which cannot only solve the problem of repeated calculation in RDF subgraph set query, but also effectively query large-scale RDF data, this paper makes an in-depth study on RDF subgraph query method and proposes a distributed query strategy of RDF subgraph set based on composite relation tree. The basic idea of this query strategy is as follows. Firstly, a corresponding composite relation graph is established for RDF subgraph set, and the redundant nodes and edges of the composite relation graph are deleted to obtain the composite relation tree. Then using the composite relation tree, a MapReduce-based RDF subgraph set query method is proposed, the distributed query batch processing is performed on the RDF subgraph set, and the query result of the RDF subgraph set is obtained by traversing the composite relation tree.

The remainder of this paper is organized as follows. Section 2 introduces the works relevant to our research. Section 3 gives the formal definition of RDF data model and RDF subgraph query. Section 4 introduces the construction of composite relation tree based on common subgraphs. Section 5 proposes a query algorithm based on MapReduce for RDF subgraph set. Section 6 evaluates our method. Finally, Section 7 concludes the paper and outlines our future work.

2. Related Work

2.1. Query of RDF Subgraph Set. The research focus of RDF subgraph set query is to design an efficient parallel execution strategy of RDF subgraph set, and the optimization execution strategy includes two aspects: one is the optimization of RDF subgraph query strategy and the other is the optimization of RDF subgraph query execution order [1]. In recent years, the related research is as follows.

Yu et al. [2] proposed a multipattern graph optimization algorithm, which used a compact tree-structured index to represent the structural correlation among multiple patterns and presented a dynamic update algorithm for fast insertion and deletion of indexes to avoid rebuilding indexes from scratch. At the same time, the execution sequence of dependency tree was used to query the RDF subgraph set, thus ultimately reducing the traversals of data graphs in the process of query. Ren and Wang [3] proposed a method to detect useful common subgraphs. This method defined the concept of trivertex label sequence, which was used to represent the common subgraphs of two RDF schema graphs, and then designed a novel grouping factor to divide the query graph into several groups. In each group, an RDF schema diagram containing mapping was established to represent the subgraph isomorphism between RDF subgraphs. The authors also used the heuristic algorithm to sort the execution sequence of RDF pattern graphs and effectively used the intermediate results in reducing the intermediate cache. Khan et al. [4] proposed a multidocument abstract summarization method based on semantic graph and improved sorting algorithm. Semantic graphs were constructed from source documents, and graph nodes represented predicate argument structures (PASs), the semantic structures of sentences, which were automatically identified by semantic role markers. Graph edges represented similarity weights, which were calculated by semantic similarity of PASs. This method used an improved graph node sorting algorithm and maximal marginal relevance to sort the important graph node PASs to reduce redundant PASs. Wang et al. [5] adopted the query rewriting and grouping methods to reduce the number of queries and repeated calculations and established an RDF storage index to quickly estimate the selection rate.

In addition to some of the studies mentioned above, RDF subgraph queries are also a class of graph search that can be applied to many scenarios through different forms of optimization. For example, solving dynamic vehicle path problems [6] and also rumor detection in different social network analyses [7]. Moreover, when the method can be applied to network searches, it can help to find the nodes that cause Doppler shifts [8].

To sum up, the existing RDF subgraph set query methods improve the query efficiency, but there are still the following challenging technical problems: (1) how to construct the combination relationship of RDF subgraph set for reducing the traversal times in the query process? (2) How to store the intermediate results to reduce resource utilization and improve the utilization of intermediate results in the query process? (3) How to design a suitable query algorithm to match all the result sets of RDF subgraphs with less traversal times?

2.2. Distributed RDF Query. In addition, researchers at home and abroad have done a lot of research on the key technologies of implementing RDF data query and processing system applications on distributed platforms [9].

Wang et al. [10] proposed a query processor based on StarMR (an efficient and scalable distributed algorithm based on star decomposition) and used MapReduce to perform subgraph matching query on large RDF graph data, then adopted two optimization strategies (postponing Cartesian products and filtering RDF property) to improve the basic StarMR algorithm. Peng et al. [11] proposed a graph-based distributed SPARQL query processing technology by using two partial evaluations and assembly methods. Two partial evaluation methods: one was to evaluate the cross-matching and overlap of each RDF fragment in parallel, and the other was to assemble these local segments through two assembly strategies to calculate the cross-matching. Two assembly methods (centralized and distributed assembly): one was to send all local matches to a single assembly site,

and the other was to assemble a local match in multiple locations in parallel, which made the solution independent of partition and minimized the number of vertices and edges involved in the intermediate results. Wang et al. [12] designed a query processor SDEC based on MapReduce framework, which made use of the embedded semantics of RDF graphs and decomposed RDF graphs into star subgraphs, and then gave the matching sequence, so as to reduce the intermediate results in the matching process. Moreover, the delay Cartesian product operation was used to improve the performance of the basic algorithm. Wang et al. [13] proposed a distributed Pregel-based method DP2RPQ (Pregel-based parallel provenance-aware regular path queries) to evaluate source-aware RPQ (regular path query) on large RDF graphs. The method used Glushkoy automatic machine to track the matching process in parallel, then three optimization strategies were designed based on the cost model: vertex calculation optimization, message communication reduction, and count path mitigation, which significantly reduced the intermediate results of the basic DP2RPQ algorithm and overcame the count path problem to a certain extent. Rathore et al. [14] proposed a parallel processing system for large graphs based on Spark Streaming and Spark with GraphX tools, which utilized cyber-physical systems and sensor technology for continuous monitoring and mining of data in big cities. The system adopted a graph-oriented method and implemented big data processing in single node, dual-node Hadoop, and Spark server environments. Xu and Zhang [15] proposed a Spark-based RDF query architecture, which was studied on the basis of semantic connection set (SCS). The proposed architecture adopted the repartitioning mechanism of class data based on the vertical partition, which could reduce memory overhead and consume index data. Um et al. [16] designed RDF storage scheme, batch loading algorithm, and query processing technology for fast searching billions of triples. The storage scheme provided all triples patterns and tables for each predicate for fast searching. The batch loading algorithm converted the triples into encoded data, and used the MapReduce framework to load billions of triples. Although the experimental results have shown that it can effectively process billions of times the data, it was found that the encoding step required a very high disk I/O and time consumption. Guo et al. [17] proposed a memory distributed framework Leon+ to solve the RPQ problem in the knowledge graph environment. Leon+ utilized the connection pruning through a novel RDF digest technology and path partitioning strategy, thus reducing search space and installation communication costs. The authors also developed a subtle cost model for query planning to achieve the efficiency of a complex RPQ. Xu et al. [18] proposed a new distributed subgraph matching method SP-Tree, which used the Pregel model to effectively solve the problem of subgraph matching of large RDF graph data. In addition, two optimization techniques were proposed to improve the efficiency of the algorithm, one using RDF shapes to filter the locally computed and transmitted messages, and the other delaying the Cartesian product operations during the matching process to reduce intermediate results.

To sum up, although the subgraph query of RDF graph data has made a lot of achievements, it is still necessary to further study the query optimization method in distributed environment to search large-scale RDF graph data efficiently, in particular, when dealing with distributed graph data, the current methods have too much intermediate data cache and limited utilization of intermediate data [19]. Therefore, more graph structure and knowledge semantic information should be taken as the partition standard to conduct a more detailed study on graph partition algorithm [20], and the frontier research schemes of computing in distributed environment should be introduced into the field of RDF data processing to better realize the fast query of subgraph data [21, 22]. In order to solve the problem above, this paper decomposes the subgraph and formulates the query order of the decomposed graph and effectively uses the graph structure information and tree-level relationship to reduce the intermediate results, thus improving the query efficiency.

3. RDF Data Model and RDF Subgraph Query

RDF is a kind of data framework that can describe resource information accurately, and it is designed to promote data integration and distribution on Semantic Web. RDF is essentially a graph-based data model, which is specially researched by World Wide Web Consortium (W3C) to express resource information and exchange resource data. RDF data model and its query model are introduced as follows.

Definition 1 (RDF triples). RDF triples are statements, the format of each RDF triple is (s, p, and o), where s is a subject, o is an object, p is a predicate that describes the connection between resource s and resource o, and (s, p, and o) represent that resource s has an attribute p and its value is o. One RDF triple can be abstracted as a directed graph, where the subject and object are nodes, and the predicate is their edge.

Definition 2 (SPARQL query). The common type of SPARQL query statement can be simply described as $Q = SELECT\ T_Y\ \text{WHERE}\ C_X$, where C_X is the description of the result, and T_Y is a set of triple patterns. The subject s, predicate p, and object o of each triple pattern (s, p, and o) can be represented as variables or constants. In a given RDF data graph G, the triples of subgraphs are searched, and each subgraph is mapped to the nodes and edges of G through the schema variables in C_X, so as to get the desired result T_Y.

Definition 3 (RDF data graph). RDF data graph G is composed of multiple RDF triples, $G = (V_G, E_G, L_G, l_G)$, where V_G represents the node set of G, E_G represents the edge set of G, L_G represents the label mapping set of V_G, and l_G represents the label mapping function of $V_G \longrightarrow L_G$, which corresponds any node in V_G to the label in L_G.

Definition 4 (RDF subgraph). Every RDF graph that needs to be queried, and matched can be regarded as an RDF subgraph d, $d \in G$, and $d = (V_d, E_d, L_d, l_d)$, where V_d represents

the node set of d, E_d represents the edge set of d, L_d represents the label mapping set of V_d, and l_d represents the label mapping function of $V_d \longrightarrow L_d$, which corresponds any node in V_d to the label in L_d.

Definition 5 (Subgraph matching). Given a data graph $g = (V_g, E_g, L_g, l_g)$ and an RDF subgraph $d_i = (V_{di}, E_{di}, L_{di}, l_{di})$, there exists a bijective function f: $V_{di} \longrightarrow V_g$ from d_i to g, which meets the following conditions:

(1) $\forall v \in V_{di}$ and $f(v) \in V_g$, such that $l_g(f(\mathrm{v})) \in l_{di}$

(2) $\forall (k, v) \in E_{di}, f(f(v)$ and $f(k)) \in E_g$, such that $l_g(f(v), f(k)) \in l_{di}$

That is, d_i matches g, which is denoted as $d_i \subseteq g$.

RDF subgraph matching is a basic query type in RDF graph data management, which is an NP-complete problem.

Definition 6 (RDF subgraph set). RDF subgraph set, denoted as $D = \{d_1, d_2, \cdots, d_i\}$ is the set of multiple RDF subgraphs that need to be queried in the data graph G.

Definition 7 (Query of RDF subgraph set). Given a data graph G and an RDF subgraph set $D = \{d_1, d_2, \cdots, d_i\}$, for any RDF subgraph $d_i \in D$, the query of RDF subgraph set is to search for all data subgraphs in G that match to d_i according to the set execution order.

Generally, RDF subgraph query only needs to get the result set of one RDF subgraph each time, while the query of RDF subgraph set needs to get the result set of RDF subgraph set at the same time, so the query of RDF subgraph set is an extension of RDF subgraph query. The detailed notation and meaning are shown in Table 1.

For example, an RDF subgraph set $D = \{d_1, d_2, d_3, d_4, d_5\}$ as shown in Figure 1, where A-G represent the nodes of the subgraphs, and E_1-E_8 represent the edges of the subgraphs. It is important to note that these RDF graphs have many duplicates, such as multiple identical nodes or edges between d_1 and d_5, and according to Definition 5, $d_1 \subseteq d_3$.

Table 1: Notations and meanings.

Notations	Meanings
$Q = SELECT\ T_Y$ WHERE C_X	A common type of SPARQL query statement
$G = (V_G, E_G, L_G, l_G)$	An RDF data graph
$D = \{d_1, d_2, \cdots, d_i\}$	A query graph, which is a subgraph of G
$d = (V_d, E_d, L_d, l_d)$	An RDF subgraph set
Exit (d_i)	An existence probability
Grade (e_{ij})	A grade of the edges in a composite relational graph
U_{di}	An evaluation value of the node d_i in the composite relation graph
$M_{(v)}$	An adjacency table
$\Omega(d_j)$	An intermediate result set
$R = \{S_1, S_2, \cdots, S_t\}$	A star decomposition queue
$\Omega(S_t)$	A matching result
D_1	A composite relation tree

4. Construction of Composite Relation Tree Based on Common Subgraphs

4.1. Definition and Function of Composite Relation Tree. There are usually common subgraphs in the RDF subgraph set, which leads to a large number of repeated calculations in the subgraph query process. Therefore, how to solve the problem of repeated calculations in the process of RDF subgraph set query is the research focus. In order to deal with this problem, we use the common subgraphs to construct the composite relation tree, which describes the composite relationship of the RDF subgraph set, and we finally traverse the composite relation tree to reduce the repeated calculation of queries. Related concepts are introduced as follows.

Definition 8 (Composite relation). Composite relation represents the matching relation between RDF subgraphs or the inclusion relation extracted from the common subgraphs among RDF subgraphs.

Definition 9 (Composite relation graph). Composite relation graph describes the composite relation of the RDF subgraph set, where the nodes represent the RDF subgraph in the RDF subgraph set, and the directed edges represent the composite relationship among the subgraphs.

Definition 10 (Composite relation tree). Composite relation tree is a binary tree obtained by cutting the redundant nodes and edges of composite relation graph.

In order to facilitate the understanding of Definitions 8–10, we give the following examples. In Figure 1, the RDF subgraph d_1 is isomorphic to the subgraph d_3, that is, $d_1 \subseteq d_3$, then the subgraph d_1 has a compound relationship with the subgraph d_3. Figure 2 is a composite relation graph, in which subgraphs d_1 and d_3 are represented by nodes, and the edge connecting subgraphs d_1 and d_3 represent the composite relation between subgraphs d_1 and d_3. The compound relation tree is shown in Figure 3, which is a binary tree obtained by cutting redundant nodes and edges of the compound relation graph.

4.2. Construction of Composite Relation Graph. In order to mine the possible composite relation in RDF subgraph set, this paper uses the maximum common subgraph to find the composite relation between any two RDF subgraphs in the subgraph set, and in order to better express the

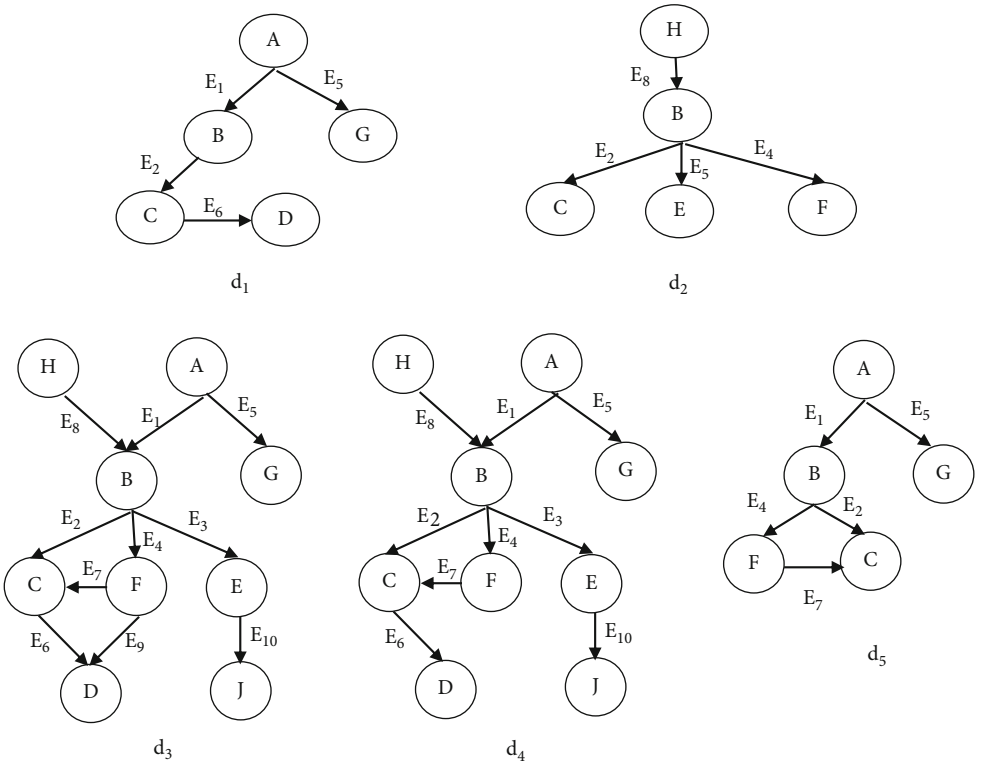

Figure 1: RDF subgraph set $D = \{d_1, d_2, d_3, d_4, d_5\}$.

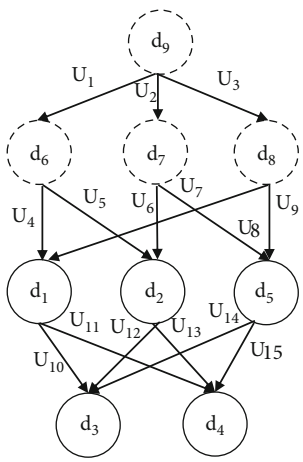

Figure 2: Composite relation graph.

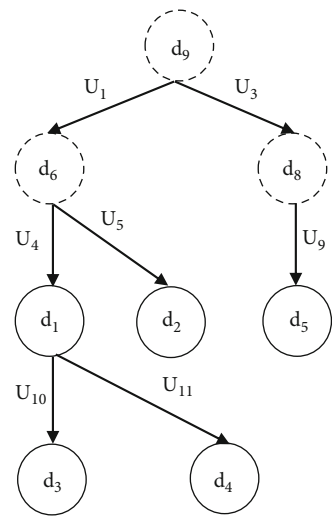

Figure 3: Composite relation tree.

compound relationship between RDF subgraphs, the relationship between RDF subgraphs is expressed in the form of a directed graph. In this way, not only the composite relation between RDF subgraphs in the set can be more intuitively expressed, but also the relationship between RDF subgraphs in the set can be more hierarchical. The basic idea of constructing composite relation graph is as follows: firstly, the maximum common subgraph algorithm is used to find the subgraph matching of RDF subgraph set. If there is subgraph matching in RDF subgraph set, then the composite relation between corresponding subgraphs is constructed. If there is no subgraph matching in the RDF subgraph set, then the composite relation between the RDF subgraphs and the common subgraphs is constructed.

The process of constructing composite relation graph of RDF subgraph set is as follows:

(1) The maximum common subgraph between subgraphs in RDF subgraph set is solved

(2) For two RDF subgraphs d_i and d_j, if their relationship is a subgraph matching relationship (such as

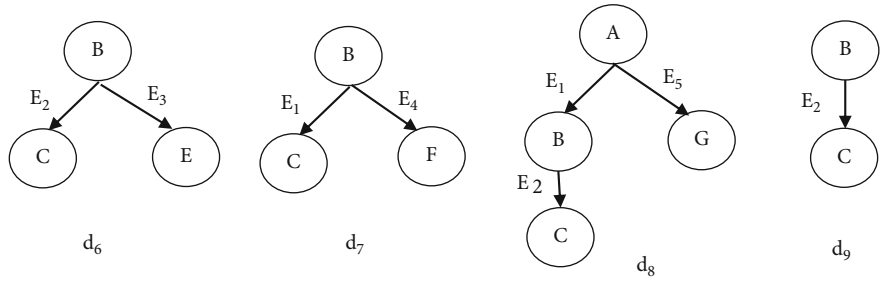

Figure 4: Maximum common subgraphs corresponding to RDF subgraph set in Figure 1.

$d_i \subseteq d_j$), then an edge from d_i to d_j is established between them. If d_i and d_j have the maximum common subgraph q, two edges pointing from q to d_i and d_j are established, and q is represented by the dotted-line node in the composite relation graph. Then, q is added to the set of RDF subgraphs

(3) For the maximum common subgraphs added into the subgraph set, continue to solve the maximum common subgraph between them, and repeat step (2)

(4) Repeat the operation of step (3) for the maximum common subgraph obtained in each round until the number of the maximum common subgraphs added to the RDF subgraph set in one round is one, then the construction of the composite relation graph is completed

For example, the composite relation graph for the RDF subgraph set shown in Figure 1 is constructed. Firstly, the maximum common subgraphs in the subgraph set are obtained, as shown in Figure 4. Then the composite relation graph shown in Figure 2 is constructed by steps (2)–(4), where the solid-line node (such as node d_3) represents the subgraph existing in the RDF subgraph set, the dotted-line node (such as node d_9) represents the maximum common subgraph added to the subgraph set, and the edge represents the matching relationship between the two subgraph nodes, such as $d_1 \subseteq d_3$.

4.3. Pruning of Composite Relation Graph. In order to simplify the composite relation graph, we use the clipping strategy to cut the redundant common subgraphs in the composite relation graph to get the composite relation tree. The main process is divided into two steps: one is to prune the nodes of the composite relation graph, and the other is to prune the edges of the composite relation graph. The relevant definitions of the clipping strategy are introduced as follows:

Definition 11 (Difference of two subgraphs). Given two RDF subgraphs d_a and d_b and $d_a \subseteq d_b$, then the difference of d_b minus d_a, denoted as R'_{ba}, is the nodes and edges of the remaining part of d_b after removing d_a The set of these remaining nodes is denoted as V'_{ba}, and the set of remaining edges is denoted as E'_{ba}.

Definition 12 (Grade of edges). The grade of edges in the composite relation graph is denoted as grade(e_{ij}), which is calculated as follows:

$$\text{grade}\left(e_{ij}\right) = \partial \times V\left(R'_{ji}\right) + \lambda \times E\left(R'_{ji}\right), \tag{1}$$

where e_{ij} is a directed edge from d_i to d_j, $V(R'_{ji})$ and $E(R'_{ji})$ represent the number of nodes and edges, respectively. ∂ represents the frequency of the remaining node set V'_{ji} in the data graph, ∂ = (number of times V'_{ji} appears in the data graph)/(total number of nodes in the data graph), and $\partial \in [0, 1]$. λ represents the frequency of the remaining edge set E'_{ji} in the data graph, λ = (number of times E'_{ji} appears in the data graph)/(total number of edges in the data graph), and $\in [0, 1]\lambda$.

Definition 13 (Existence probability). Existence probability is used to represent the probability that the triples (s, p, and o) in the data graph matching the subgraph node d_i. It is calculated as follows:

$$\text{exit}(d_i) = \frac{d_{\text{num}}(s, p, o)}{G_{\text{num}}(s, p, o)}, \tag{2}$$

where d_{num} represents the number of triples containing d_i in the data graph, G_{num} represents the number of all triples in the data graph, and the value range of existence probability is [0, 1].

Definition 14 (Evaluation value of node). The evaluation value of the node d_i in the composite relation graph is denoted as U_{di}, which is calculated as follows:

$$U_{di} = \text{exit}(d_i) \times \sum_{j=1}^{k} \text{grade}\left(e_{ij}\right), \tag{3}$$

where k is the number of in-degree edges of d_i.

The main steps of clipping composite relation graph are as follows:

(1) Determine the subgraph nodes that can be pruned. Because the dotted-line nodes in the composite

relation graph are the public subgraphs, so the dotted-line nodes and their edges need to be clipped for simplifying the composite relation graph. The dotted-line node can be clipped if it meets the following condition: both its out-degrees and the in-degrees of its child nodes are greater than or equal to 2. For example, in the composite relation graph shown in Figure 2, the subgraph node d_6 has two out-degrees U_4 and U_5, and the child nodes connected to the out-degree edges are d_1 and d_2. Moreover, the subgraph node d_1 has two out-degrees U_4 and U_8, and d_2 has two in-degrees U_5 and U_6. Therefore, d_6 meets the condition and is determined as a prunable node. Similarly, d_7 and d_8 are also the prunable nodes

(2) Prune the subgraph nodes. The subgraph node with zero out-degree is found from the bottom of the composite relation graph, and the graph is traversed upward. If only one subgraph node in the same layer is a prunable subgraph node, this node and its connected edges are pruned. If there are multiple prunable subgraph nodes in the same layer, we need to compare the evaluation values of these subgraph nodes, and the node with the highest value is first pruned until there is no prunable subgraph node in this layer. Repeat the above pruning process until there are no prunable subgraph nodes in the composite relation graph, and then stop pruning the subgraph nodes

For example, it can be observed from Figure 2 that d_6, d_7, and d_8 are the subgraph nodes that can be clipped, and they are in the same layer. Therefore, compare the evaluation values of these subgraph nodes and crop the subgraph nodes with larger values first. Assuming that ∂ is 0.3, λ is 0.1, and the existence probability of the subgraph nodes is the same. According to Equation (3), U_{d6}, U_{d7} and U_{d8} can be calculated as: $U_{d6} = U_4 + U_5 = 1.2$, $U_{d7} = U_6 + U_7 = 1.7$, and $U_{d8} = U_8 + U_9 = 1.3$, so d_7 and its connected edges are clipped first, and again, we determine whether there are multiple prunable subgraph nodes in the same layer, and do the prune process until there are no prunable subgraph nodes. Finally, we get the simplest composite relation graph as shown in Figure 5.

(3) Prune the edges. For the subgraph nodes with multiple in-degrees, we need to compare the grades of their in-degree edges, which are calculated according to Equation (1) and keep the edge with the smallest value. If multiple edges have the same value, the existence probability of the upper layer nodes connected with the in-degree edges are compared, and the in-degree edge with low existence probability is retained. The existence probability is calculated according to Equation (2).

(4) Repeat steps (2) and (3) until each subgraph node has only one in-degree edge, then a binary tree is obtained, which is the composite relation tree

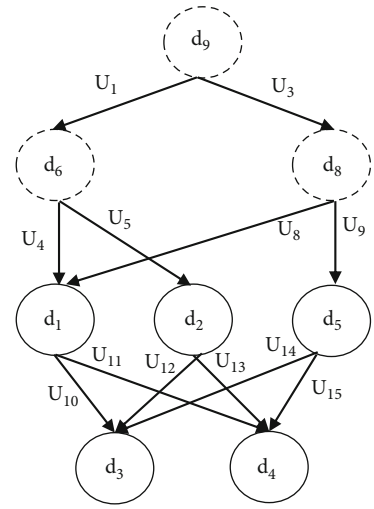

Figure 5: Simplest composite graph.

For example, in the simplest composite graph shown in Figure 5, d_1 has two in-degree edges U_4 and U_8, assuming that ∂ is 0.3 and λ is 0.1, according to Equation (1), the grade values of U_4 and U_8 are 0.4 and 0.8, respectively, so U_4 with the smallest value is retained, and U_8 is cut; d_3 has 3 in-degree edges U_{10}, U_{12}, and U_{14}, and their values are 1.4, 1.8, and 1.7, respectively. So, U_{10} with the smallest value is retained, while U_{12} and U_{14} are cut. Similarly, the edges of d_4 are clipped. Finally, the composite relation tree shown in Figure 3 can be obtained.

5. Query Algorithm Based on MapReduce for RDF Subgraph Set

The RDF subgraph set query problem is mainly to solve the problem of how to reduce the double calculation between subgraphs. Therefore, this section proposes an algorithm, composite relation tree query (CRTQ), for querying RDF subgraph set using the composite relation tree in the MapReduce environment.

MapReduce is a programming mode specially designed for the parallel operation of various large-scale datasets. "Map" and "reduce" are the two main basic ideas. Using these ideas, it is very convenient for programmers to use the distributed system to operate their own applications independently.

The main idea of CRTQ algorithm is as follows. Firstly, the data of the RDF subgraph collection is distributed and stored in multiple tables, and these tables can be used to perform MapReduce calculations on the adjacency table of each subject node at the same time. Then, each RDF subgraph in the corresponding composite relation tree is traversed and queried in the order of hierarchy. When traversing the query subgraph, the subgraph is decomposed into a star shape, a star-shaped decomposition queue is obtained, and SPARQL query is performed under the operation of MapReduce. In addition to the root subgraph node, other subgraph nodes are queried in the intermediate result set of their parent

CRTQ (D_1, $M(v)$)
Input: Composite relation tree D_1, adjacency table $M(v)$
Output: Query result set $\Omega(T) = \{\Omega(d_1), \Omega(d_2), \ldots, \Omega(d_n)\}$
FOR EACH $d_j \in D_1$
    $\Omega_1 =$ **CALL***SubgraphMatching* (d_j);
    IF $\exists dj.Leaf(v) \in \Omega_1$ **THEN** $\Omega(d_j) = \Omega_1$;
    j++;
    $\Omega(T) = \Omega(d_j) \cup \Omega(d_{j+1})$; //Get the matching result set of D_1
END FOR
RETURN $\Omega(T)$;
FUNCTION *SubgraphMatching* (d_j)
    IF d_j=*d.root* **THEN** $R \leftarrow$ **CALL***StarDecomposition* (d_j);
    WHILE $R \neq \varnothing$ **DO**
    $t \leftarrow 1$, $S_t \leftarrow R.dequeue$;
    CALL*map*($\varnothing$, $M(v)$);
    IF $t > 1$ **THEN**
    CALL*map*($\varnothing$, β); //$\beta \in \Omega(P_{t-1})$
    $\Omega(d_j) =$ **CALL***reduce*(β_{key}, $\Omega(P_{t-1})$, $\Omega(S_t)$));
    END IF(
    t++;
    END WHILE
    ELSE
    $R \leftarrow$ **CALL***StarDecomposition* (d_j);
    WHILE $R \neq \varnothing$ **DO**
    $t \leftarrow 1$, $S_t \leftarrow R.dequeue$;
    CALL*map*($\varnothing$, $\Omega(d_j$.*Parent*)); //query d_j in the result set of the parent graph
    IF $t > 1$ **THEN**
    CALL*map*($\varnothing$, β); //$\beta \in \Omega(P_{t-1})$
    $\Omega(d_j) =$ **CALL***reduce*(β_{key}, ($\Omega(P_{t-1})$, $\Omega(S_t)$));
    END IF
    t++;
    END WHILE
    END IF
RETURN $\Omega(d_j)$;
  FUNCTION *map*($\varnothing$, $M(v)$ *or* β)
    IF input $M(v)$ **THEN**
        $\Omega(S_t) \leftarrow$ **CALL***StarMatching* (S_t, $M(v)$);
        IF $t = 1$ **THEN**
            $\beta \leftarrow \Omega(S_1)$;
            RETURN ($\varnothing$, β);
        ELSE
            FOR EACH $\beta \in \Omega(S_t)$
                $\beta_{key} = \{(v', \beta(v')) | v' \in V(P_{t-1}) \cap V(S_t)\}$; //get the matching result of $\{v'\}$ as the key value
                RETURN (β_{key}, β);
            END FOR
        END IF
    ELSE
        $\beta_{key} = \{(v', \beta(v')) | v' \in V(P_{t-1}) \cap V(S_t)\}$;
        RETURN (β_{key}, β); //the input value is $\Omega(P_{t-1})$, the output is (β_{key}, β)
    END IF
  FUNCTION *reduce*(β_{key}, ($\Omega(P_{t-1})$, $\Omega(S_t)$))
      FOR EACH $(\beta, \beta') \longleftarrow \{\Omega(P_{t-1}) \times \Omega(S_t)\}$
        RETURN $\Omega(P_t) \longleftarrow \beta \cup \beta'$; //get the current matching result set $\Omega(P_t)$
      END FOR

Algorithm 1

Table 2: The main characteristics of datasets.

Datasets	Number of nodes	Number of edges	Edge/node ratio
DBpedia2015A-1 M	90512	270745	2.991
DBpedia2015A-25 M	2005251	4010863	2.000
DBpedia2015A-50 M	3860248	7450162	1.930
WatDiv-1 M	132478	384015	2.899
WatDiv-50 M	4342095	8074015	1.859
WatDiv-100 M	9145251	24774138	2.709
LUBM-10 M	404897	797108	1.969
LUBM-50 M	4196332	8518821	2.030
LUBM-100 M	8246487	17652144	2.141

subgraph in order, and the intermediate result set without leaf subgraph node is filtered out in each query. In this way, the generation of irrelevant results and intermediate result sets during the query of the next layer of subgraphs can be reduced, which can greatly reduce the operation of repeated calculations. At the same time, the result set of all the subgraphs in the tree can be obtained by traversing the composite relation tree once.

The specific steps of CRTQ algorithm are as follows:

(1) Generate the corresponding composite relation tree D_1 of RDF subgraph set, use the adjacency tables $M_{(v)}$ to store the data graph G, and use the adjacency tables $T(v)$ to store the tree D_1. Where $D_1 = \{d_1, d_2, \cdots, d_n\}$, $n \in [1, j]$, and j is the number of RDF subgraphs in the composite relation tree

(2) Query the root subgraph *d.root* of $D1$ and match *d.root* with G, filter the matching result set without the leaf subgraph node $dj.Leaf(v)$, and the result set $\Omega(d_j)$ obtained is the intermediate result set

(3) Traverse the next subgraph d_j in the composite relation tree D_1 according to the order of hierarchy and perform star decomposition on the subgraph d_j, then obtain the star decomposition queue $R = \{S_1, S_2, \cdots, S_t\}$, $t \in [1, s]$, and s is the number of star decomposition of the subgraph d_j

(4) Each subgraph d_j performs query matching in the intermediate result set of $\Omega(d_j.Parent)$ which obtained from the query of its parent graph *d.Parent*. In each round of MapReduce, the Map function sequentially matches a star S_t in R on the adjacency table of each node to obtain the matching result $\Omega(S_t)$

(5) The matching result set of the node intersection of the local query subgraph P_{t-1} and the star-shaped S_t is used as the key β_{key} connecting Map and Reduce. In the Reduce function, the matching result $\Omega(P_{t-1})$ of the partial query subgraph P_{t-1} obtained in the last round is combined with the matching result $\Omega(S_t)$ to obtain $\Omega(P_t)$

(6) Repeat steps (4) and (5) to get the matching result set Ω_1 of all stars in queue R, traverse the intermediate result set Ω_1, and filter the result set without leaf subgraph node $d_j.Leaf(v)$ to get $\Omega(d_j)$

(7) Continue to query the next layer of subgraphs, repeat steps (3)-(6), until all the subgraphs d_n of the composite relation D_1 have been queried

The main pseudocodes of CRTQ algorithm proposed in this paper are as follows:

The time complexity of the CRTQ algorithm is $O(V^s \bullet \max N^{s \bullet \max M})$. Where s is the number of stars, V is the number of nodes in D_1, $\max N$ is the maximum out-degree of the node in G, and *maxM* is the maximum out-degree of the subgraph node in D_1. In the matching of s star patterns, the maximum time consumption occurs when the node in G has the maximum out-degree $\max N$ and the subgraph node in D_1 has the maximum out-degree $\max M$. Although the subgraph set query method proposed in this paper will cause a certain amount of time consumption in the process of constructing a composite relation tree. However, the query method proposed in this paper can obtain the query results of multiple RDF subgraphs by traversing the composite relation tree, thus greatly reducing the query time.

6. Evaluation

In this section, the experiments have been performed to compare our CRTQ algorithm with other query algorithms MPT [2], MQO [3], and CS-MQO [5] and to evaluate the scalability of CRTQ when the data size changes and the cluster points change.

6.1. Experimental Setup. The experiment was carried out on a distributed cluster with six identical computing nodes, each computing node is Intel (R) Core (TM) i7-4720HQ@3.60GHz 8 core processor with 16GB memory and 1 T hard disk. 1000 Mbps Ethernet was used for communication between nodes. The development environment

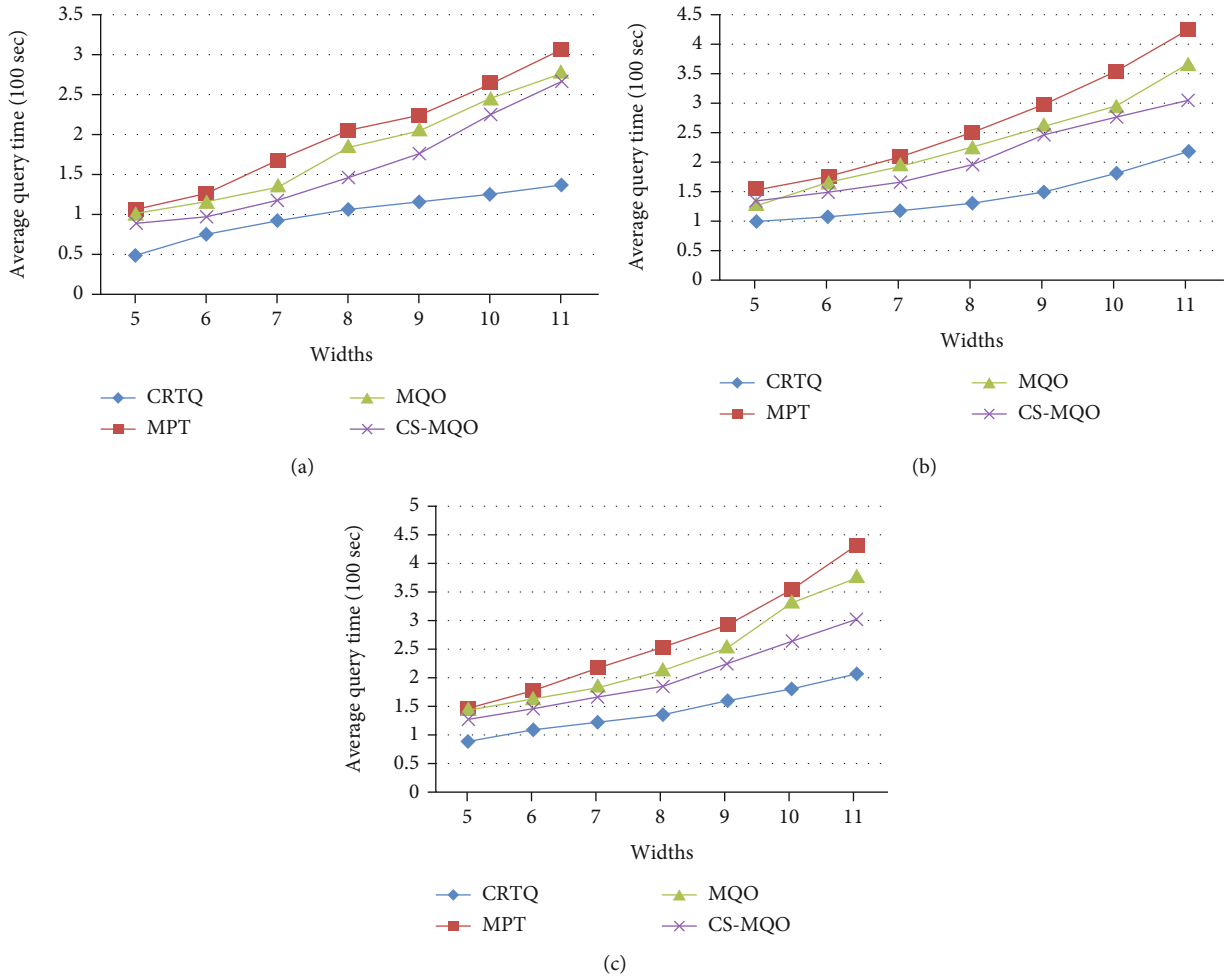

Figure 6: Query efficiency under different widths of composite relation trees. (a) DBpedia2015A-50 M. (b) WatDiv-100 M. (c) LUBM-100 M.

of the algorithm was Eclipse 2018 and Hadoop 2.7.4, and the language used was Java, JDK1.8.

Three standard RDF datasets DBPedia2015A [23], WatDiv [24], and LUBM [25] were used in the experiment. DBPedia2015A is a dataset about sports and sports events. WatDiv is a dataset about e-commerce, which includes information about users, retailers, and products. LUBM is the data benchmark of Lehigh University, which contains the relevant teaching information data of the university. These datasets allow users to generate the datasets of different scales according to their needs. We use the method described in [26] to extract and expand the original datasets and generate, respectively. 1 M, 50 M, and 100 M RDF triples of WatDiv; 1 M, 25 M, and 50 M RDF triples of DBpedia2015A; 10 M, 50 M, and 100 M RDF triples of LUBM. The main characteristics of these datasets are shown in Table 2.

6.2. Comparison of Query Efficiency. RDF query subgraph sets were generated by randomly selecting RDF subgraphs in the datasets, and the composite relation trees of these subgraph sets were generated according to the method in Section 4. These composite relation trees have a width of 5-11 and a depth of 5-11, respectively.

To compare the query efficiency of CRTQ algorithm with MPT, MQO, and CS-MQO algorithms, we use six identical computing nodes (cluster nodes) and performed experiments on DBPedia2015A-50 M dataset and Watdiv-100 M dataset, respectively, under the condition of the same depth and the different widths of the composite relation trees. For the different width, each query is run 5 times, and the average query time is taken.

It can be seen from the experimental results shown in Figure 6, with the increasing width of the composite relation tree, the amount of data in the RDF graph is also increasing, and the average query time of the four algorithms is constantly increasing. Moreover, all the query time of CRTQ algorithm is less than 250 s, and it can be seen that CRTQ algorithm has the obvious advantages when the width of the composite relation tree increases. In particular, when the width of composite relation tree is 11, the average query time of the CRTQ algorithm on three datasets is about 51%, 46%, and 40% higher than that of MPT, MQO, and CS-MQO algorithms, respectively.

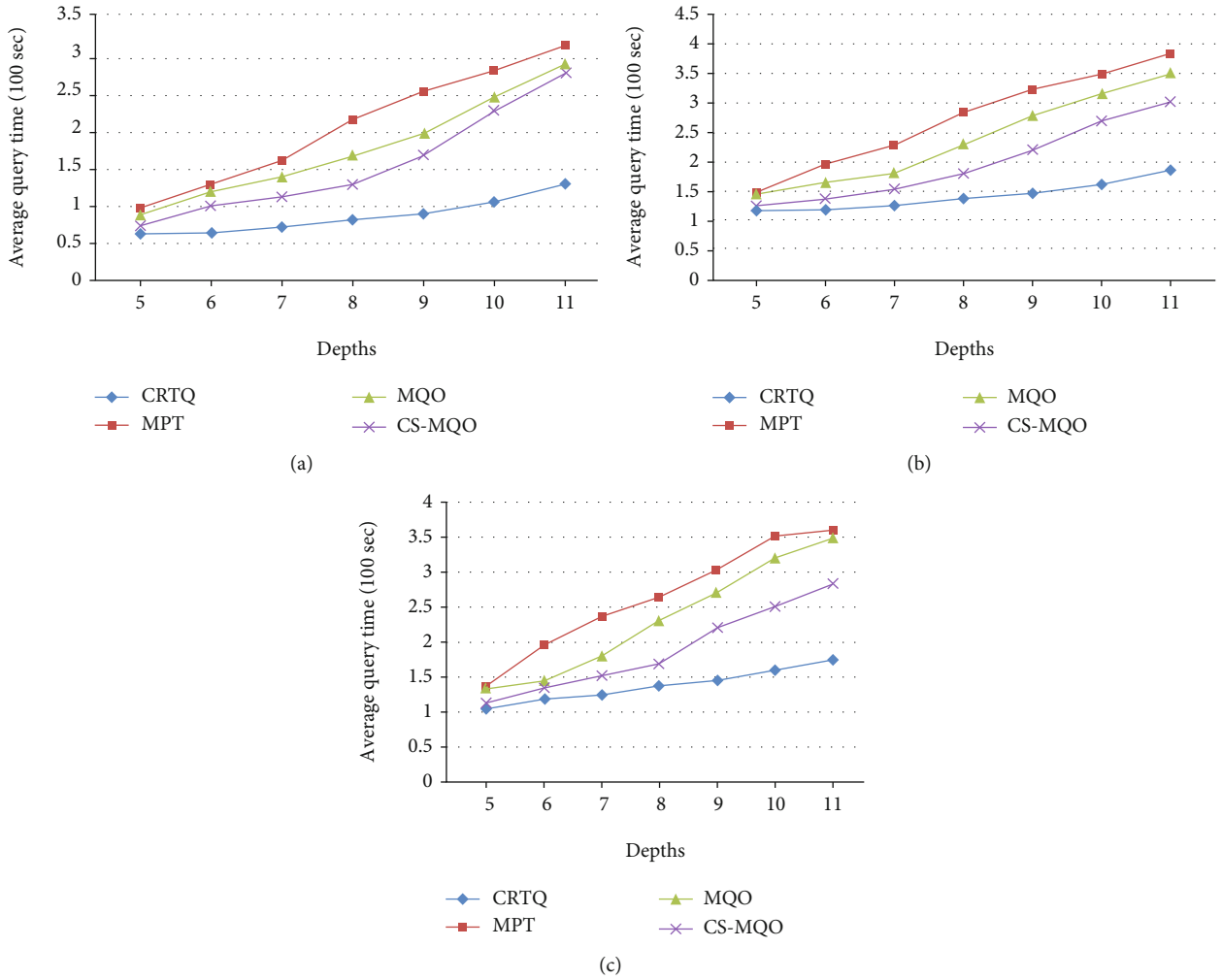

Figure 7: Query efficiency under different depths of composite relation trees. (a) DBpedia2015A-50 M. (b) WatDiv-100 M. (c) LUBM-100 M.

Similar to the above experiment, we compare the query efficiency of these algorithms under the condition of the same width and the different depths of composite relation tree. The experimental results are shown in Figure 7.

From Figure 7, we can see that with the increasing depth of the composite relation trees, the amount of data in the RDF graph is also increasing, and the average query time of the four algorithms is constantly increasing. Moreover, all the average query time of CRTQ algorithm is less than 200 s, and it can be seen that CRTQ algorithm has the obvious advantages when the depth of the composite relation tree increases. In particular, when the depth of composite relation tree is 11, the average query time of CRTQ algorithm on three datasets is about 54%, 50%, and 43% higher than that of MPT, MQO, and CS-MQO algorithms, respectively.

In addition to the above query efficiency comparison, we also perform a graph preprocessing performance comparison. The experimental results are shown in Figure 8.

From Figure 8(a), we can see that the database precreation time for CRTQ and MQO is slightly higher than that of the MPT algorithm for the three datasets during database creation because CRTQ needs to generate composite association trees and MQO needs to determine multipattern matching sequences through mapping; while CS-MQO has the longest precreation time because it needs to utilize vertical storage and semantic clipping.

In addition, we can understand from Figure 8(b) that CRTQ is the best in terms of cache space size, while the average cache space size required by CS-MQO and MPT is about 1.3-1.5 times that of CRTQ. This is because the CS-MQO algorithm converts all queries into undirected connected graphs, and the cost of finding common subgraphs among multiple graphs is relatively high. While MPT is divided into two steps, basic pattern matching and extended pattern matching, and each time the set of matching results of basic patterns is used as input for extended pattern matching, so there is still a part of the data graph that needs to be traversed several times, and its intermediate results need to be saved each time, which greatly increases the space complexity.

6.3. Query Performance of Different Data Sizes. We compare the query efficiency of CRTQ, MPT, MQO, and CS-MQO algorithms under the following conditions: (1) the width/

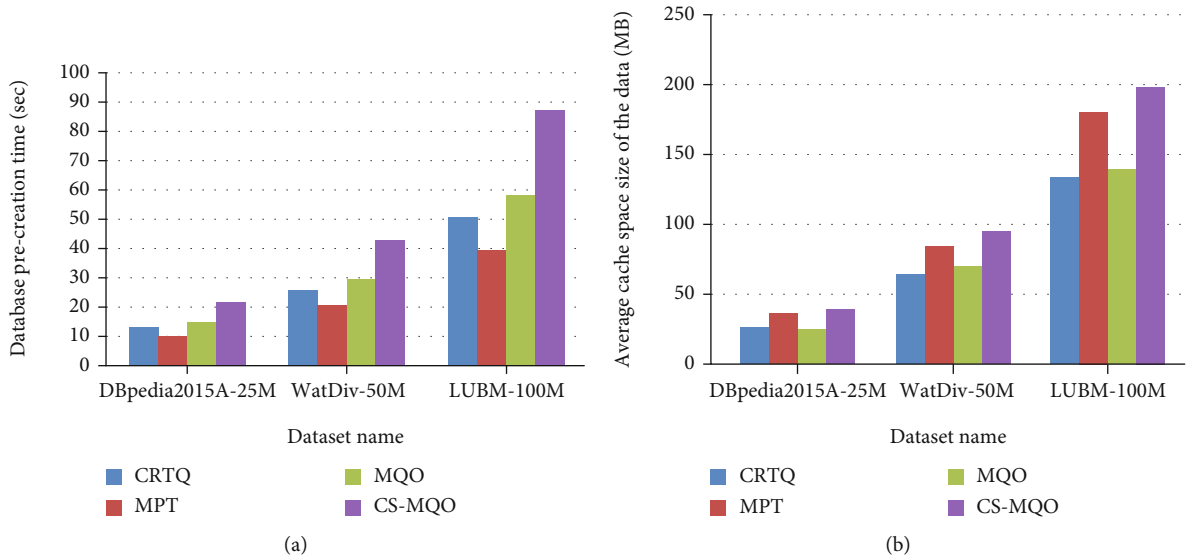

Figure 8: Preprocessing performance of graphs in datasets DBpedia2015A-50 M, WatDiv-100 M, and LUBM-100 M.

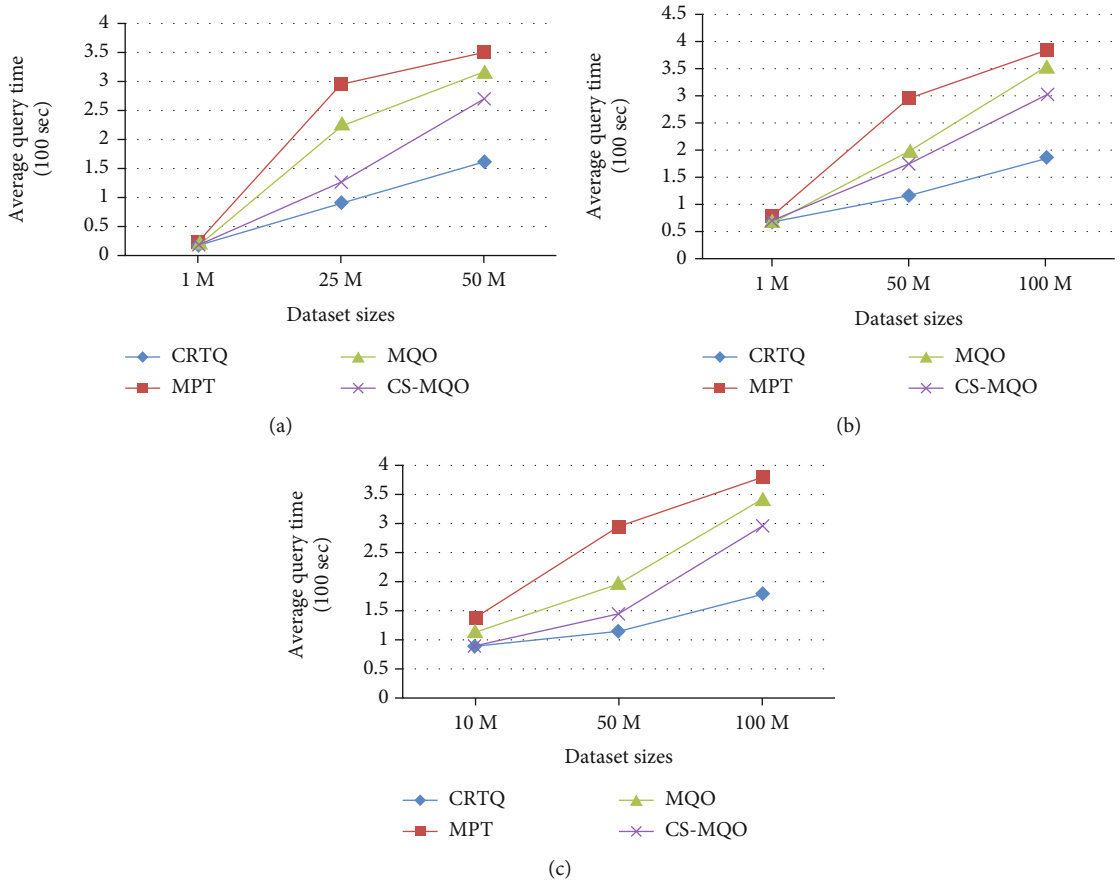

Figure 9: Query time of different dataset sizes. (a) DBpedia2015A. (b) WatDiv. (c) LUBM.

depth of composite relation tree are the same; (2) the depth of RDF query subgraph is the same and its range is 5-15; (3) the number of computing nodes is 6; (4) the DBpedia2015A dataset size increases from 1 M to 50 M, the size of WatDiv dataset increases from 1 M to 100 M, and the size of LUBM dataset increases from 10 M to 100 M.

From the experimental results shown in Figure 9, it can be seen that the query time of the CRTQ algorithm proposed

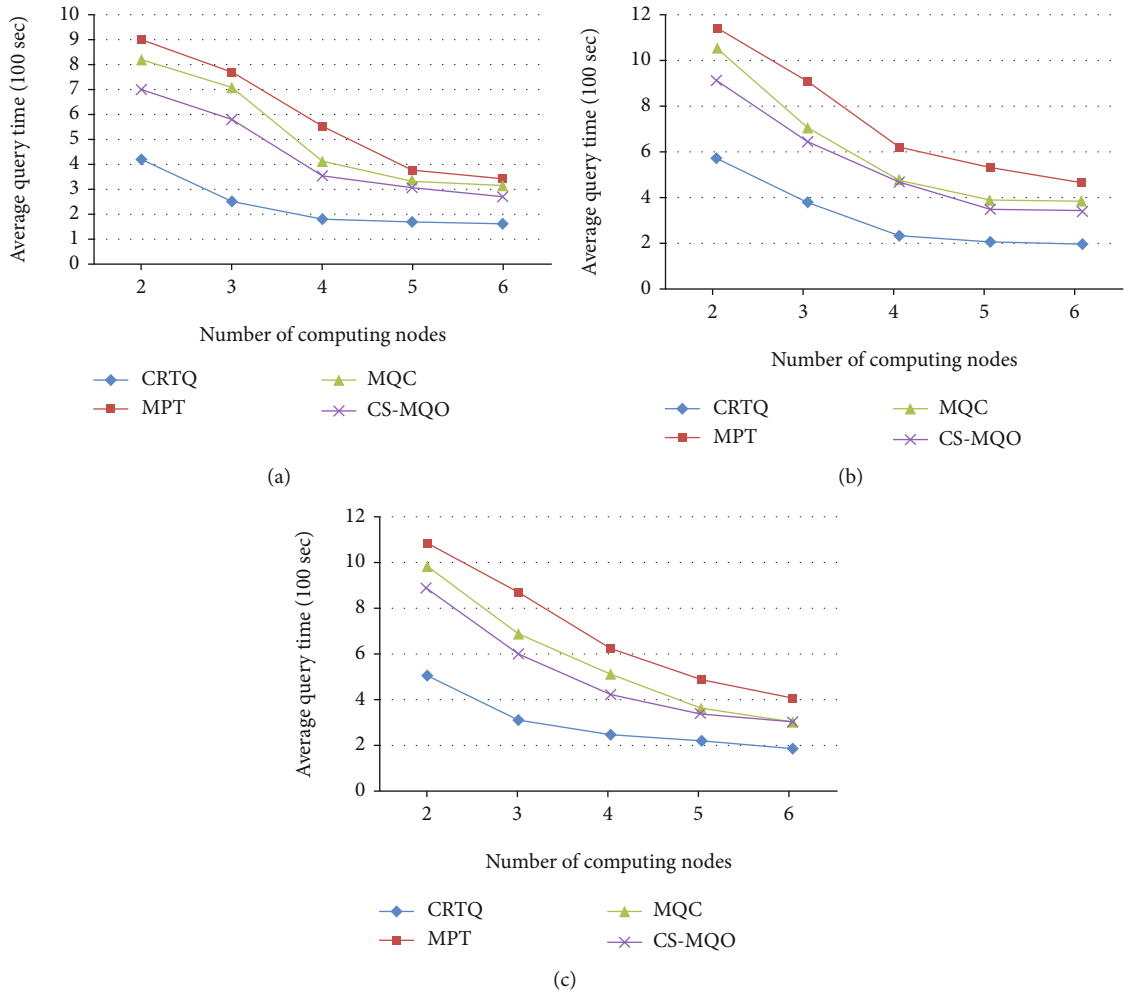

Figure 10: Query time for the different number of computing nodes. (a) DBpedia2015A-50 M. (b) WatDiv-100 M. (c) LUBM-100 M.

in this paper is less than the others algorithms. With the increase of data size, the query time of the four query algorithms is increasing, but the increase of query time of MPT, MQO, and CS-MQO is much larger than that of CRTQ. Moreover, the increase of query time of CRTQ is always in a relatively stable state with the increase of dataset.

In addition, from the experimental results of WatDiv dataset shown in Figure 9, we get the following conclusion: even if the RDF data structure of the WatDiv dataset is more complex, when the WatDiv dataset size increases from 1 M to 100 M, CRTQ algorithm can cope with the expansion of data scale, and the change of query time is very stable. For example, the query time of CRTQ algorithm only increases from 65 s at 1 M to 185 s at 100 M, while the query time of MPT increases from 78 s to 385 s, the query time of MQO increases from 68 s to 353 s, and the query time of CS-MQO increases from 69 s to 302 s.

6.4. Scalability Evaluation of the Algorithm. In this section, the scalability of the four algorithms is evaluated by changing the number of computing nodes. The experimental results are shown in Figure 10. As can be seen from Figure 10, when the number of computing nodes increases from 2 to 6, the query time of the four algorithms on the datasets (DBpedia2015A-50 M, WatDiv-100 M, and LUBM-100 M) decreases and tends to stabilize. That is, the larger the number of computing nodes, the higher the query processing efficiency of the four algorithms. In addition, compared with other algorithms, CRTQ algorithm has a shorter query time and more effective scalability in distributed environment.

7. Conclusions and Future Work

In order to improve the query efficiency of RDF subgraphs in massive RDF data graphs and solve the problem of repeated calculation in the query process of RDF subgraph set, a distributed query method of RDF subgraph set based on composite relation tree is proposed in this paper. We first construct a corresponding composite relation graph for RDF subgraph set. To simplify the query process, we propose an evaluation model that weighs the nodes and edges of composite relation graph, and then removes the redundant nodes and edges. We also propose a MapReduce-based RDF subgraph set query method, which carries out the distributed query batch processing on the RDF subgraph set,

and we obtain the query results of the RDF subgraph set by traversing the composite relation tree. Experiments show that the work in this paper can improve the query efficiency of RDF subgraph set.

In the future research, we can make the following improvements: (1) further experiments on large real datasets or further query research on datasets with fixed frequency changes and updates; (2) the index of subgraph matching order can be added to the composite relation tree, and the number of subgraph traversal can be reduced to a certain extent through the effective index method; (3) in the intermediate result processing, in addition to star decomposition, other improved graph structure decomposition methods can be introduced, such as linear type and snowflake type.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

We declare that we have no conflict of interest.

Acknowledgments

This research was supported by the National Natural Science Foundation of China under Grant Nos. 62267005 and 61862010, and the Guangxi Collaborative Innovation Center of Multisource Information Integration and Intelligent Processing.

References

[1] M. Wylot, M. Hauswirth, P. Cudré-Mauroux, and S. Sakr, "RDF data storage and query processing schemes," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–36, 2019.

[2] J. Yu, X. Liu, Y. Liu, and Y. Hu, "Multiple pattern graph correlations for efficient graph pattern matching," in *Proceedings of the IEEE/ACS 14th International Conference on Computer Systems and Applications*, pp. 469–474, Hammamet, Tunisia, 2017.

[3] X. Ren and J. Wang, "Multi-query optimization for subgraph isomorphism search," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 121–132, 2016.

[4] A. Khan, N. Salim, H. Farman et al., "Abstractive text summarization based on improved semantic graph approach," *International Journal of Parallel Programming*, vol. 46, no. 5, pp. 992–1016, 2018.

[5] M. Wang, H. Fu, and F. Xu, "RDF multi-query optimization algorithm for query rewriting using common subgraphs," in *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, pp. 1–8, Sanya, China, 2019.

[6] A. Yavary and H. Sajedi, "Solving dynamic vehicle routing problem with pickup and delivery by CLARITY method," in *Proceedings of the 22nd International Conference on Intelligent Engineering Systems (INES)*, pp. 207–212, Las Palmas de Gran Canaria, Spain, June 2018.

[7] A. Yavary and H. Sajedi, "Rumor detection on Twitter using extracted patterns from conversational tree," in *Proceedings of the 4th International Conference on Web Research (ICWR)*, pp. 78–85, Tehran, Iran, 2018.

[8] A. R. Nafchi, M. Esmaeili, A. Ghasempour, E. Hamke, B. Santhanam, and R. Jordan, "Mitigating the time-varying doppler shift in high-mobility wireless communications using multi-angle centered discrete fractional Fourier transform," in *Proceedings of the 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 607–612, New York, NY, USA, December 2021.

[9] I. Abdelaziz, R. Harbi, Z. Khayyat, and P. Kalnis, "A survey and experimental comparison of distributed SPARQL engines for very large RDF data," *Proceedings of the VLDB Endowment*, vol. 10, no. 13, pp. 2049–2060, 2017.

[10] X. Wang, L. Chai, Q. Xu et al., "Efficient subgraph matching on large RDF graphs using MapReduce," *Data Science and Engineering*, vol. 4, no. 1, pp. 24–43, 2019.

[11] P. Peng, L. Zou, M. T. Özsu, L. Chen, and D. Zhao, "Processing SPARQL queries over distributed RDF graphs," *The VLDB Journal*, vol. 25, no. 2, pp. 243–268, 2016.

[12] X. Wang, Q. Xu, L. L. Chai, Y. J. Yang, and Y. P. Chai, "Efficient distributed query processing on large scale RDF graph data," *Journal of Software*, vol. 30, no. 3, pp. 498–514, 2019.

[13] X. Wang, S. Wang, Y. Xin, Y. Yang, J. Li, and X. Wang, "Distributed Pregel-based provenance-aware regular path query processing on RDF knowledge graphs," *World Wide Web*, vol. 23, no. 3, pp. 1465–1496, 2020.

[14] M. M. Rathore, S. Attique Shah, A. Awad, D. Shukla, S. Vimal, and A. Paul, "A cyber-physical system and graph-based approach for transportation management in smart cities," *Sustainability*, vol. 13, no. 14, p. 7606, 2021.

[15] J. Xu and C. Zhang, "Semantic connection set-based massive RDF data query processing in spark environment," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, 2019.

[16] J. H. Um, S. Lee, T. H. Kim, C. H. Jeong, S. K. Song, and H. Jung, "Distributed RDF store for efficient searching billions of triples based on Hadoop," *The Journal of Supercomputing*, vol. 72, no. 5, pp. 1825–1840, 2016.

[17] X. Guo, H. Gao, and Z. Zou, "Distributed processing of regular path queries in RDF graphs," *Knowledge and Information Systems*, vol. 63, no. 4, pp. 993–1027, 2021.

[18] Q. Xu, X. Wang, J. Li, Q. Zhang, and L. Chai, "Distributed subgraph matching on big knowledge graphs using Pregel," *IEEE Access*, vol. 7, pp. 116453–116464, 2019.

[19] C. Ranichandra and B. K. Tripathy, "Architecture for distributed query processing using the RDF data in cloud environment," *Evolutionary Intelligence*, vol. 14, no. 2, pp. 567–575, 2021.

[20] A. Paul, "Graph based M2M optimization in an loT environment," in *Proceedings of the 2013 Research in adaptive and convergent systems*, pp. 45-46, Montreal, Quebec, Canada, 2013.

[21] M. M. U. Rathore, M. J. J. Gul, A. Paul et al., "Multilevel graph-based decision making in big scholarly data: an approach to identify expert reviewer, finding quality impact factor, ranking journals and researchers," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 280–292, 2021.

[22] R. Schroeder, R. R. M. Penteado, and C. S. Hara, "A data distribution model for RDF," *Distributed and Parallel Databases*, vol. 39, no. 1, pp. 129–167, 2021.

[23] "DBpedia2015A [DB]," last accessed 2021/05/12, http://downloads.dbpedia.org/2015-04.

[24] "WatDiv [DB]," last accessed 2021/05/12, http://dsg.uwaterloo.ca/watdiv.

[25] "LUBM [DB]," http://swat.cse.lehigh.edu/projects/lubm.

[26] M. Spasić, M. Jovanovik, and A. Prat-Pérez, "An RDF dataset generator for the social network benchmark with real-world coherence," in *Proceedings of the Workshop on Benchmarking Linked Data 2016*, pp. 1–8, Kobe, Japan, 2016.