RDF Subgraph Matching by Means of Star Decomposition

Mingyan Wang[1,2], Qingrong Huang[3], Nan Wu[1,3], Ying Pan[1,3*]

[1] Guangxi Key Lab of Human-machine Interaction and Intelligent Decision, Nanning Normal University, China
[2] School of Physics and Electronics, Nanning Normal University, China
[3] School of Computer and Information Engineering, Nanning Normal University, China
wmywangming@163.com, rongronghuang@163.com, wunan_19980802@163.com, panying@nnnu.edu.cn

Abstract

With the continuous development of the network, the scale of RDF data is becoming larger and larger. In the face of large-scale RDF data processing, the traditional database query method has been unable to meet the needs. Due to the limited characteristics of subgraph matching, most existing algorithms often have the phenomenon that many subgraphs are repeatedly traversed during the query process, resulting in a large number of intermediate result sets and low query efficiency. The core problem to be solved is how to efficiently match subgraphs. In order to improve the query efficiency of RDF subgraphs in massive RDF data graphs and solve the problem of repeated calculation of some graphs in the query process of RDF subgraphs, an RDF subgraph query algorithm based on star decomposition is proposed in this paper. The algorithm uses graph structure to decompose RDF subgraphs into stars and uses a custom node cost model to calculate the query order of the star subgraphs. By decomposing, the amount of communication among subgraphs is reduced, and the communication cost for query processing is lowered. Moreover, utilizing the query order for RDF subgraph matching can effectively reduce the generation of intermediate result sets and accelerate the efficiency of subgraph matching. On this basis, the performances of the proposed algorithm and several other widely used algorithms are compared and analyzed on two different datasets. Experiments show that the proposed algorithm has better advantages in database recreation, memory size, and execution efficiency.

Keywords: RDF query, Subgraph matching, Subgraph query, Search problems, Star decomposition

1 Introduction

Resource Description Framework (RDF) is a standard data model for describing web resources. It was mainly designed to build the Semantic Web. In the past few decades, there have been related technologies applied to RDF graph queries in different fields, such as life sciences, bioinformatics, and social intelligence. It is suitable for the re-identification of social network users, socially intelligent transportation, and the anonymity of social networks in people’s daily lives. In such an environment, how to query RDF subgraphs more effectively has become a research focus.

Today, when RDF graph data is getting larger and larger, the query methods adopted by traditional relational databases are far from being able to meet the current processing needs. With the continuous development of the Semantic Web, the data in RDF format is increasing, and the number of RDF graphs is also increasing. At present, massive RDF data brings new problems to RDF graph data management. It is very important to accurately find the required data in the massive RDF graphs. Typically, a large number of RDF graphs are processed by graph mining [1], and the required data can be obtained. Graph mining is a branch of data mining, which represents a large amount of complex data in the form of graphs and infer useful knowledge by mining. Subgraph matching plays a very important role in the whole process of graph mining [2]. According to whether the graph has directionality or not, the graph is divided into the directed graph and the undirected graph. Subgraph matching refers to inputting a query graph and a queried data graph and outputting a set of subgraphs that satisfy the query graph matching conditions (that is, the matching results are consistent with the query graph topological structure and vertex labels). Using the idea of graph matching can effectively reduce the time of the RDF query. At present, the efficiency of large-scale RDF graph matching using existing algorithms is low. Although researchers have proposed many new methods to improve the current situation, there are still many deficiencies. Due to the limitations and characteristics of subgraph matching, most of the current algorithms are based on node iteration, using the relationship between nodes for analysis. Existing query algorithms often introduce a large number of connection operations, and repeatedly traverse some subgraphs during the query process, resulting in a large number of intermediate result sets, resulting in low query efficiency and low query performance. In addition, there are also great problems with database pre-creation, and memory storage, the query time is extended [3]. Because of the above problems, the solution for the current RDF graph query problem is not perfect, many researchers are seeking new algorithms to query the RDF graph. Therefore, how to reduce the generation of intermediate result sets, reduce repeated calculations and improve execution efficiency has become an important issue for current RDF graph queries.

In response to the above problems, this paper proposes an RDF Node Substitution Value-Star Query (RNSV-SQ) algorithm based on node cost value. The main contributions of this paper can be summarized as follows:

*Corresponding Author: Ying Pan; E-mail: panying@nnnu.edu.cn
DOI: 10.53106/160792642022122307015

1) We propose a star decomposition method, which decomposes RDF graphs into star subgraphs by using graph structure, and calculates the query order of these star subgraphs by using the user-defined node generation value model.

2) We propose a subgraph matching algorithm based on above star decomposition. According to the query order of the star subgraphs, fewer intermediate results can be generated, and repeated calculations can be reduced, so as to improve the performance of the RDF subgraph query. It also improves the database pre-creation, storage memory size, and execution efficiency.

3) We compare and analyze the performances of the proposed algorithm and several other widely used algorithms on two different datasets. Experiments show that the algorithms proposed in this paper can effectively improve the query efficiency of RDF subgraphs.

The rest of this article is arranged as follows: The next section describes the current status of RDF subgraph query algorithm at home and abroad. Section 3 describes the RDF subgraph, subgraph matching, and other related concepts, and introduces our RDF subgraph query algorithm based on star decomposition in detail. Section 4 proves the superiority of our algorithm by conducting experiments on two datasets. In the last section 5, the content of the article is sorted out and summarized, and the future research direction is further prospected.

2 Related Work

To solve the RDF subgraph query problem is to solve the problem of subgraph matching. Its main difficulty is how to obtain results from polynomial time. When the traditional subgraph query algorithms are processing, a large number of intermediate result sets will often appear, resulting in a prolonged matching time. In recent years, many researchers have been researching and discussing the problem of subgraph matching, and have proposed many advanced methods, which have played a further development role in solving the problem of subgraph matching. The related research is mainly as follows:

In 1976, a practical search scheme for subgraph matching was given for the first time, which was based on the idea of backtracking. It also used some feasible solutions to iteratively generate complete solutions, and timely terminate some infeasible solutions in the query process, such as VF2 [4], GADDI [5], SPath [6], etc. VF2 used grammatical and semantic similarity to match nodes, and used specific vectors as data structures to reduce the time for matching large graphs. GADDI calculated the neighboring discriminating substructure (NDS) distance for each pair of adjacent nodes in the graph, and generated a series of unequal attributes based on this to eliminate the candidate results. SPath proposed a one-time path mechanism, which decomposes the query graph into the shortest path set in units of neighbor nodes, but it is not a one-time node mechanism. VF3 [7] proposed a feasibility result set, divided nodes into predecessors and successors, classified label nodes, and proposed feasibility rules using node structure information and edge label frequency. These algorithms filter the candidate set through different connection sequences, pruning schemes or some auxiliary information. The disadvantage is that the time complexity is high, so they are not suitable for the processing with large amounts of environmental data.

There are a large number of intermediate result sets and repeated calculations in the subgraph matching process, resulting in too long execution time. Ren et al. [8] proposed a general improvement method of existing matching methods based on the relationship between data nodes. Through observation and comparison, it was concluded that most of the subgraph matching algorithms have a large number of duplications of calculation, which can be avoided by the relationship between data vertices. Also proposed a novel method to reduce repetitive calculations. Wang et al. [9] provided many additional experiments based on the original subgraph matching search to detect the different effects of graph compression itself and candidate filtering on query performance. Experiments showed that the processing time was significantly reduced by using a separate compression step, and then the processing time can be further reduced by using a filtering step. Kyu et al. [10] proposed a chain-star index scheme and a query method of SPARQL query. In addition to supporting chain query and star query, this method also optimizes query time by reducing the memory for connection operations and storing data, thereby minimizing query execution time. On this basis, Kyu et al. [11] considered a search method based on RDF data graph structure, which can effectively reduce connection operations. It supported chain and star SPARQL queries, and was optimized for query performance. By considering the difference in the edge structure around all vertices, the chain-like and star-like subgraphs were extracted from the RDF data graph, and search methods that reduce the number of connections were favored to speed up query response time.

With the increasing complexity of the current query graph structure, the efficiency of graph-based query about SPARQL query processing has gradually been unable to keep up with the complexity of the graph structure. In analyzing the basic structure of the RDF graph, Park et al. [12] first conducted a comprehensive study of the existing cardinality estimation technology of subgraph matching query. Introduced a new framework on which all currently known technologies can be implemented, and an opinion on its performance can be given. A new realization has also been achieved with the representative cardinality estimation technology of graph databases and relational databases. Bi et al. [13] proposed a new framework of the delayed Cartesian product based on the structure of a query to minimize the redundant Cartesian products. A new path-based auxiliary data structure was also proposed, which performs subgraph matching through the generated matching order, thereby significantly reducing the exponential size of the existing path-based auxiliary data structure.

The study found that frequent subgraph mining (FSM) in graph mining can quickly find all subgraphs of the graphs. Frequent Subgraph Patterns (FSPs), which work well for graph mining on small and medium-sized graphs, can speed up query time for subgraphs on complex graph structures. Rehman et al. [14] proposed a new optimization method to show the association between frequent and optimized subgraph patterns. This approach explored whether there was a potential correlation between FSPs and optimized subgraphs, and can further reduce FSPs. In order to improve the efficiency of the algorithm, a new conceptual framework A-RAnked Frequent pattern-growth Framework (A-RAFF) was proposed

[15], this framework adds a sorting function to the mining process, which can sort the found frequent subgraphs. In addition, A-RAFF embedded the ranking of FSGs found in the mining process [2], so that A-RAFF can reduce the work of generating a large number of useless frequent subgraphs under the premise of efficient computation. To deal with a large number of FSPs challenges [16], a new ranking method FSP-Rank was used by A-RAFF, which effectively reduces the repetition and huge frequent patterns, and effectively speeds up the processing time.

Researchers also improve the execution efficiency by graph division. Guan et al. [17] proposed a subgraph matching method based on the structure segmentation of query graph. By disassembling a complete query graph into many simple query subgraphs, the search space was reduced through the adjacent subgraph structure, and matching subgraphs were found in the data graph according to the search area. Finally, output the resulting graph by adding related subgraphs. This method can improve query time and efficiency when processing complex query graphs. Ning et al. [18] proposed a dominance-partitioned strategy under the premise of load-balancing on subgraph division, which partitions large RDF graphs without affecting the knowledge structure. The subgraph matching algorithm through a dominance-partitioned strategy to perform all matching subgraphs on the RDF graph of the cluster partition.

In addition to the above methods, Li et al. [19] solved the problem of answering queries about fuzzy RDF data graphs. By modeling the RDF data onto the fuzzy graph, the RDF queries and the query search of the subgraph of the fuzzy graph can be regarded as equivalent, and these subgraphs were highly likely to match the given query graph. Sakr et al. [20] designed many centralized RDF query processing systems. In these query systems, the storage and query processing of RDF datasets were managed on a single node. An overview of various technologies, systems, and a large number of RDF data models for centrally querying RDF datasets were efficient, and scalable RDF query processing solutions.

Although the above methods have made a lot of achievements in RDF subgraph processing, they still spend a lot of time in many aspects when processing data, resulting in low efficiency. For example, in the preprocessing of the relational structure of RDF data, there are still a large number of intermediate result sets, which will consume a lot of time, and the efficiency of database pre-creation, storage memory size, and execution speed is not high [3].

3 RDF Subgraph Matching Algorithm Based on Star Decomposition

In the process of RDF subgraph matching, the intermediate result set is one of the important factors that affect the query efficiency, so it is very important to study how to reduce the generation of the intermediate result set to improve the query efficiency. Because for queries, different orders will lead to different results. In this section, the cost value of the structure between nodes and the degree of analysis was used to obtain the optimal order for the query, thereby reducing the generation of intermediate result sets, and improving the efficiency of the subgraph query.

3.1 Related Concepts

Definition 1 (RDF data graph) RDF data graph $G = (V, E, L, \varphi)$ is a directed label graph, where V represents the set of nodes. E represents the set of directed edges connected to the nodes in V. $E: (v, v')$ represents a directed function, representing a directed edge from v to v', where $v, v' \in V$. L is the label set of edges and nodes. $\varphi: V \cup B \rightarrow (L \cup var)$ represents a label function that maps nodes or edges to corresponding labels, B represents blank node.

Definition 2 (RDF subgraph) When querying the RDF data graph, it can be found that each RDF graph used for the query can be regarded as a subgraph G_g, $G_g \in G$. In the RDF subgraph $G_g = (V_g, E_g, L_g, \varphi_g)$, V_g is the node-set of G_g, E_g is the edge set of G_g, L_g is the label mapping set of V_g, and φ_g represents the label mapping function of $V_g \rightarrow L_g$, any node s and the label in L_g can correspond to each other.

The processing of RDF graphs can be abstractly regarded as directed graph processing, and each query RDF graph can be regarded as an RDF subgraph.

Definition 3 (Subgraph matching) Given two graphs $G = (V, E, L, \varphi)$ and $G_g = (V_g, E_g, L_g, \varphi_g)$, the subgraph matching mapping f: $V_g \rightarrow V$ represents the mapping from the nodes of G_g to the nodes of G. For any node $u \in V_g$, there is $\varphi_g(u) = \varphi(f(u))$. For each edge $(u_i, u_j) \in E_g$ ((u_i, u_j) represents a directed edge from u_i to u_j), there is an edge $(f(u_i), f(u_j)) \in E$.

Definition 4 (RDF star-shaped subgraph) Given a subgraph $G^* = (V^*, E^*, L^*, \varphi^*)$ of an RDF graph G, if and only if there is a v in V^* that satisfies: $\forall u \in V^* - \{v\}, \langle v, u \rangle \in E^*$ or $\langle u, v \rangle \in E^*$. G^* is called the star-shaped subgraph of G, and v is called the central node of G^*.

Definition 5 (Star-shaped subgraph matching) Given a star-shaped graph $G^* = (V^*, E^*, L^*, \varphi^*)$ and data graph $G = (V, E, L, \varphi)$, the subgraph matching mapping $f: V^* \rightarrow V$ of the star graph are defined as follows: For the star-center node v_s, there is $L^*(v_s) = L(f(v_s))$. For the neighboring point of the star-center, there is $|adj(v_s, \varphi)| \leq |adj(u, \varphi)|$. Where f represents the mapping from the nodes of G^* to the nodes of G, $L^*(v_s)$ is the label representing the center of the star in the star pattern, $|adj(v_s, \varphi)|$ represents the number of adjacent points of the star-center v_s with a label of 1.

3.2 Node Cost Model

In the RDF star graph, the node cost model of the central node v is specifically defined:

$$W(v) = \alpha \times deg(v) + t \times NE(v). \tag{1}$$

Where, α takes a value of [0-1], α refers to the appearance frequency of all out-degrees of the star S in the data graph. The higher the frequency of occurrence, the closer to 1, whereas the closer to 0. $deg(v)$ represents the number of nodes that all out-degrees of star S with v as the center node coincided with all out-degrees of another node at the same time in the data graph. t is the weight, $t \in [0,1]$, $NE(v)$ represents the edges (including in-degree and out-degree) owned by the node matching the query result of star s' in the last round, which are consistent with star s.

The node cost value model follows the following rules:

1) First consider the star pattern with the highest degree of the central node v. If there are multiple star patterns with the same out-degree, then follow (2). The higher the out-degree of the central node v, the stronger the constraint of node v, the smaller the candidate set, and the lower the query cost;

2) Select the star with the largest $W(v)$. The greater the $deg(v)$ and $NE(v)$, the greater the $W(v)$. This indicates that the star s with v as the central node in the data graph has a larger number of nodes that match all the out-degrees of the star s, the more the same number of edges as the previous matching star S', the greater the constraint, the fewer the number of nodes dominated, the lower the query cost, and the fewer intermediate results.

3.3 Star Decomposition of RDF Graph

In the previous section, a node cost value model was proposed. Using this model to sort the matching order for RDF subgraphs that have passed star decomposition can effectively reduce the generation of intermediate result sets, and improve query efficiency.

Figure 1 shows the main steps of the star decomposition algorithm:

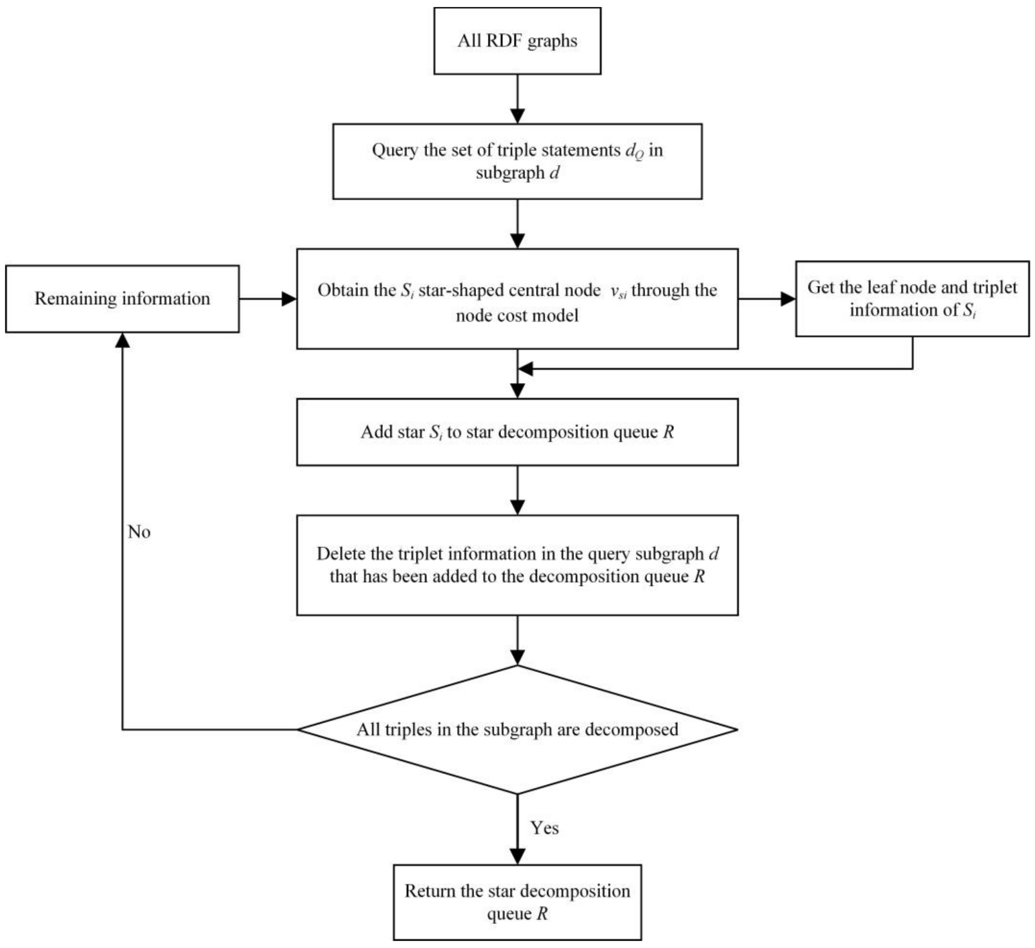

All RDF graphs
Query the set of triple statements d_Q in subgraph d
Remaining information
Obtain the S_i star-shaped central node v_{si} through the node cost model
Get the leaf node and triplet information of S_i
Add star S_i to star decomposition queue R
No
Delete the triplet information in the query subgraph d that has been added to the decomposition queue R
All triples in the subgraph are decomposed
Yes
Return the star decomposition queue R

Figure 1. The main steps of star decomposition

The main idea of star decomposition is as follows:

1) Query the set of triple statements d_Q from subgraph d, and select node c as the first star-shaped center node v_{s1} by the node cost model. Unselected nodes will remain in subgraph d, waiting for the next query;

2) Let the star S_1 take v_{s1} as the central node, find the corresponding edges and leaf nodes in the corresponding query subgraph d and get all the triples of S_1, and then add S_1 to the star decomposition queue R. Delete the triplet information in the query subgraph d that has been added to R;

3) Check whether all triple information in d has been decomposed, if decomposed, proceed to step 5), if not decomposed, keep the rest of the remaining information in d for the next lookup, proceed to step 4);

4) Continue to obtain the central node in the remaining subgraphs through the node cost model, then repeat steps 1), 2), and 3);

5) Return the star decomposition queue R.

For example, we perform a star decomposition query in the data graph G_0 shown in Figure 2 and the RDF subgraph d_0 shown in Figure 3. The star decomposition method is described in detail below. The star patterns S_1, S_2, and S_3 as shown in Figure 4 will be obtained. If these star patterns are respectively queried on the data graph G_0, the following results will be obtained:

The query results of S_1 are: $\{A = a, B = b, C = c, D = d\}, \{A = l, B = m, C = n, D = 0\}$

The query results of S_2 are: $\{E = e, A = a, F = f\}$

The query results of S_3 are: $\{E = f, G = g\}, \{F = j, G = h\}$

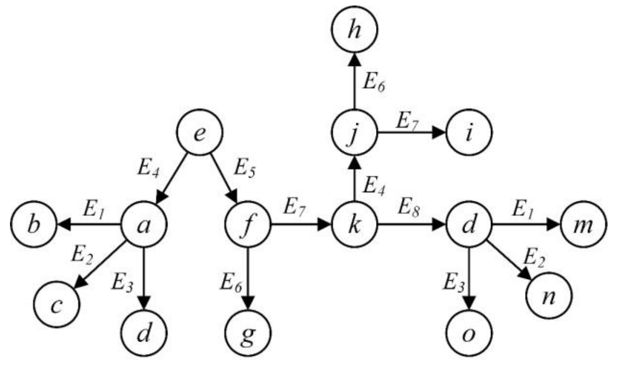


Figure 2. RDF data graph G_0

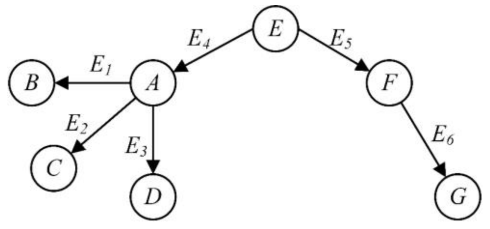


Figure 3. RDF subgraph d_0

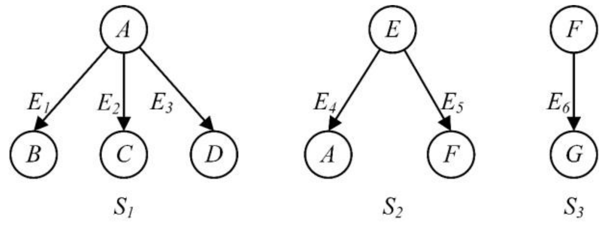


Figure 4. Stars S_1, S_2 and S_3

The query order for these star types is not random, and the number of intermediate result sets often depends on the arrangement order. S_3 is not associated with S_1, if the matching order is S_1, S_3, and S_2, four matching intermediate results will be generated. According to the designed node cost value model, firstly, the star S_1 with the most out-degree is considered, and S_1 is ranked first in the query sequence. Secondly, compare the node cost value of S_2 and S_3, and put the node with the largest node cost value in the second place. According to the node cost value model formula, the value of t is 0.5, and the node cost value of the central node of star S_2 can be obtained as $W(v_{s2}) = \frac{3}{14} \times 1 + 0.5 \times 1 = \frac{5}{7}$, the node cost of the central node of star S_3 is $W(v_{s3}) = \frac{2}{14} \times 2 + 0.5 \times 0 = \frac{2}{7}$. Because S_2 has the largest value, it is used as the priority query matching object, and the query order (star decomposition queue) is S_1, S_2, S_3. Querying in this order will only produce one intermediate result set. Therefore, finding an optimal matching query order is very important to reduce the generation of intermediate result sets.

3.4 Star Query Algorithms for RDF Subgraph

When performing query matching operations on the data graph, in order to obtain the optimal execution order, it is necessary to perform star decomposition on each RDF subgraph to obtain a star queue. When matching, the matching order for the stars was arranged through the node cost value model.

Definition 6 (Partial query subgraph) Obtain a subgraph g in the RDF graph G, use the star decomposition algorithm for g to obtain the star decomposition queue P, and give the matching order S_1, S_2, …, S_n, which satisfies $\cup_{1\le i\le j} V(S_i) \cap V(S_j) \neq \emptyset, 1 \le j \le n$. P_j is a sequential partial query subgraph, $1 \le j \le n$ is a subgraph of subgraph g, and satisfies the following conditions:

(1) $V(P_j) = \cup_{1\le i\le j} V(S_i)$

(2) $Y(P_j) = \cup_{1\le i\le j} V(S_i)$

Where, $V(P_j)$ represents the node of the local subgraph P_j, and $Y(P_j)$ represents the edge of the local subgraph P_j.

StarMatching algorithm is mainly based on the idea as follows:

1) Input the star S and the adjacency list $N(v)$;

2) Query all the node v sets in the data graph that can be matched with the star-shaped center node v_s;

3) Obtain all the candidate sets $A(l_i)$ of nodes l_i that can match each leaf node $S.Leaf$ of star S in the data graph;

4) Do the Cartesian product operation on the candidate set $A(l_i)$ of the star S, we can get the matching result $\Omega_v(S)$ of the star S on the node v. The union of the matching result sets $\Omega_v(S)$ of all nodes v is also called the matching result $\Omega(S)$ of star S.

Figure 5 shows the main steps of the StarMatching algorithm:

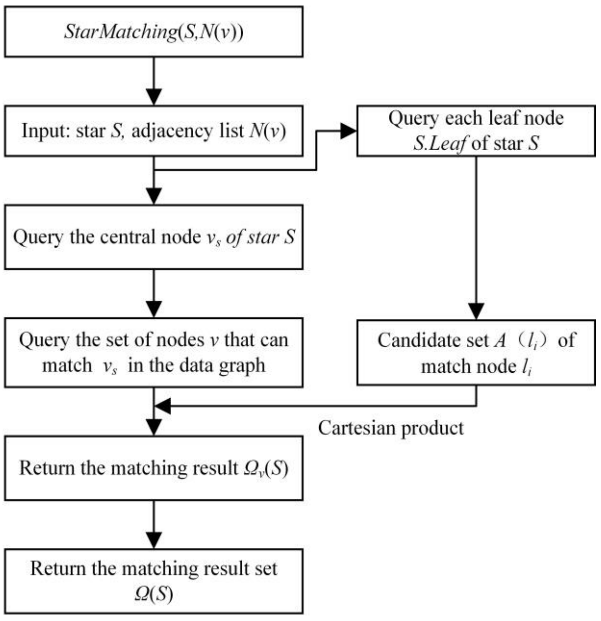

StarMatching(S,N(v))
Input: star S, adjacency list N(v)
Query each leaf node S.Leaf of star S
Query the central node v_s of star S
Query the set of nodes v that can match v_s in the data graph
Candidate set A（l_i）of match node l_i
Cartesian product
Return the matching result $\Omega_v(S)$
Return the matching result set $\Omega(S)$

Figure 5. The main steps of star matching

RNSV-SQ mainly follows the following steps:

1) Input the RDF subgraph d, the star decomposition queue R, and the adjacency list $N(v)$ of the data graph G;

2) Obtain the star S_1 in the star decomposition queues R, and match the star S_1 on the adjacency list $N(v)$, which is the result $\Omega(P_1)$ of the local subgraph P_1;

3) Continue to acquire new star patterns until all acquisitions are over to get the result $\Omega(P_{t-1})$;

4) Obtain the matching result of the star S_t through the star matching algorithm;

5) Combine the matching result of the partial subgraph

P_{t-1} obtained in the last round and the matching result of the star-shape S_t to obtain the result $\Omega(P_t)$;

6) Traverse all stars $R = \{S_1, S_2, \ldots, S_n\}$ in the queue, the matching result T in the data graph is the union of $\Omega(P_t)$, and output the result set T.

The time complexity of RNSV-SQ is $O(V^s \cdot maxN^{s \cdot maxM})$, where s is the number of stars, V is the number of nodes in the RDF subgraph d, $maxN$ is the maximum number of outgoing-degrees owned by nodes in data graph G, and $maxM$ is the maximum number of outgoing-degrees owned by nodes in subgraph d. According to the process of the algorithm, the time complexity of star matching is the total time consumed by s stars in matching all nodes V of the data graph, which is $\sum_1^s O\left(\sum_{v\in V}\left(N(v) + \Omega(S_t)\right)\right)$, the time complexity of the star pattern of the connection matching with the local query graph is $\sum_1^s O\left(\sum_{v\in V}\left(\Omega(P_{t-1}) \times \Omega(S_t)\right)\right)$. In the matching of s star patterns, $maxN$, the maximum out-degree owned by the node in the data graph G, and $maxM$, the maximum out-degree owned by the node in the subgraph d, are taken as the maximum time consumption. Although this algorithm will consume a certain amount of time in the process of decomposing the query graph and obtaining the matching sequence of the star pattern. However, according to the obtained matching order, the generation of intermediate results can be greatly reduced, and the time consumption in the query process can be reduced, which still shows better advantages.

4 Experimental Analysis

4.1 Experimental Environment and Dataset

The experimental platform uses Linux Ubuntu operating system, using Intel(R) Core(TM)i7-9700@3.0GHz eight-core processor, the memory is 16GB, and the hard disk size is 1T.

The algorithm development environment is Eclipse 2018, and the development language is Java, JDK1.8.

This experiment uses two standard RDF datasets, DBpedia2015A [1] dataset and WatDiv [2] dataset. The DBpedia2015A dataset collects data related to sports content, while WatDiv is a dataset used to describe e-commerce, it contains information about some products sold and retailers, as well as information about users who purchase goods.

Both of these datasets allow users to generate datasets of appropriate scale according to the required dataset size by using the method described in [21]. We generated a DBpedia2015A-1M dataset, which contains 90512 nodes and 270745 edges. For WatDiv dataset, we generated 1M, 10M, and 100M, of which 1M contains 132478 nodes and 384015 edges, 10M contains 1102045 nodes and 2097786 edges, and 100M contains 9145251 nodes and 24774138 edges.

4.2 Comparison of Query Efficiency

RNSV-SQ is compared with the graph matching algorithms GADDI [5], SPath [6], and VF3 [7] proposed in recent years. Compare and analyze the performance of these four algorithms in terms of database pre-creation time, data storage memory size, and query time.

The basic test provided by the WatDiv dataset contains four query template categories, namely linear query, star query, snowflake query, and complex query. These query templates are generated by representing the data as a model of basic graphics and performing a random traversal of the data in the query pool. There are some given basic query templates on the WatDiv dataset. 12 basic query templates were selected from the following four categories: linear (L), star (S), snowflake (F), and complex (C). These query templates were L1, L2, L3, S1, S2, S3, F1, F2, F3, C1, C2, and C3. Due to the lack of a given query template on the DBpedia2015A dataset, query graph templates containing the above four query types were imitated and designed on the DBpedia2015A dataset, denoted as L11, S11, F11, and C11.

As shown in Figure 6, we can easily see the experimental comparison results on two datasets, VF3 and RNSV-SQ are better than GADDI and SPath in the time of database pre-creation. During the database creation process, the database creation time gap between VF3 and RNSV-SQ on these datasets is very small, and the time required is relatively short compared with the other two algorithms, this is because VF3 does not need to extract auxiliary structures for the data graph, and the preprocessing requires a short time. RNSV-SQ takes some time to save node adjacency structure information. SPath needs to calculate and save the k-order neighbor structure, which takes a long time. GADDI needs to calculate the NDS distance between nodes in the data preprocessing stage, so compared with several other algorithms, GADDI database creation takes the longest time.

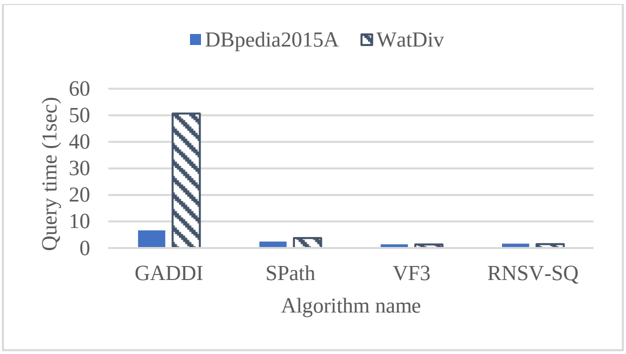


Figure 6. Database pre-creation time

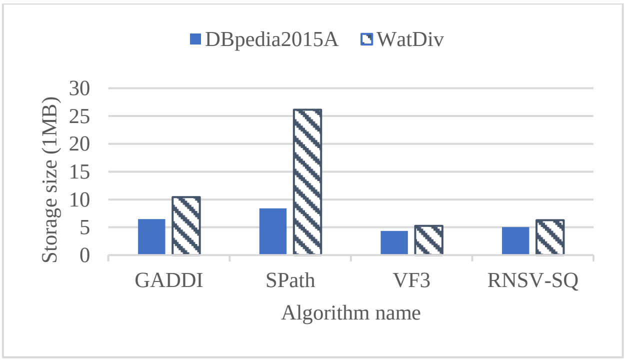


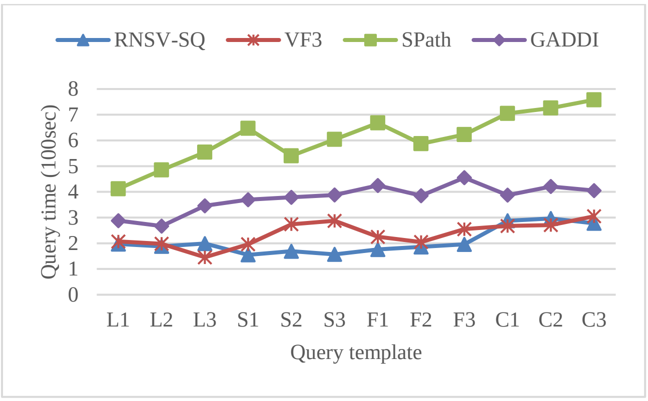
Figure 7. Data storage size

Figure 7 shows the comparison of storage space size. It can be seen that SPath has the largest storage space. Because SPath is aimed at the more complex adjacency structure of the graph. It proposes the width-first traversal centered on nodes, and the adjacencies information about nodes is represented by triples. Therefore, in addition to the data graph nodes, the k-order

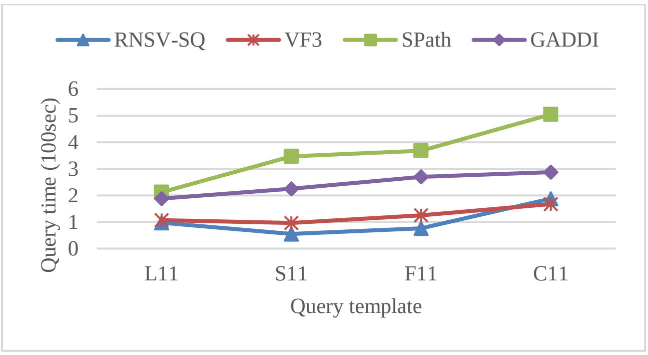
[1] DBpedia2015A [DB]: http://downloads.dbpedia.org/2015-04/

[2] WatDiv [DB]: http://dsg.uwaterloo.ca/watdiv/

adjacency information of the nodes needs to be stored. Additionally, the storage space required by SPath is about 1.6-4.1 times that of RNSV-SQ, while GADDI needs to store the NDS distance between two nodes and adjacent nodes, and the storage space is 1.2-1.6 times that of RNSV-SQ. RNSV-SQ performs slightly better than VF3 in terms of storage space, because RNSV-SQ makes better use of the structural information on adjacent nodes to reduce storage space.



(a) Query time of different templates on WatDiv-10M



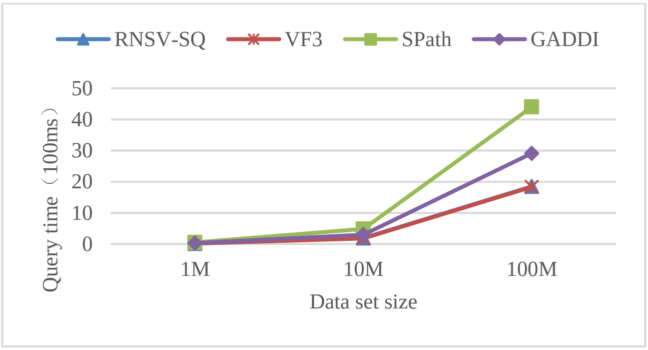
(b) Query time of different templates on DBpedia2015A-1M

Figure 8. Query time of different query templates

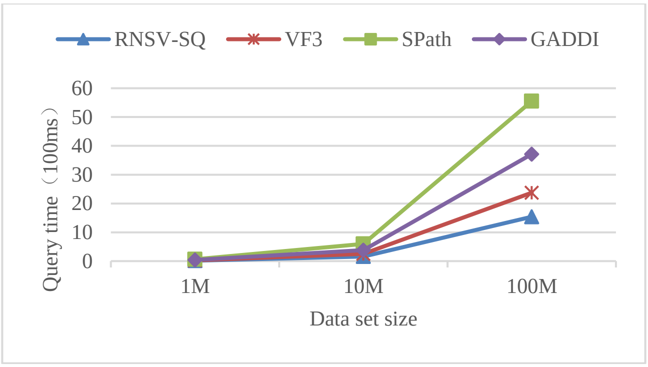
In the WatDiv-10M dataset, as shown in Figure 8(a), RNSV-SQ has a relatively good performance in the compared query template, and the required query time is lower. The queries efficiency of VF3 and RNSV-SQ is not much different. Among the query templates for L1, C1, and C2, VF3 has the fastest matching speed, while SPath and GADDI perform poorly. SPath has the longest query time for the four types of query templates, mainly due to the need to match nodes in the triplet when filtering candidate nodes, and the number of nodes is too large, which leads to a long time. Moreover, GADDI has to calculate the adjacency distance between any two nodes in the query graph during execution, which leads to excessive time consumption.

On the DBpedia2015A-1M dataset, the change from linear to complex can be obtained from Figure 8(b). As the complexity of the query graph structure increases, the rising trend of query time for VF3 and RNSV-SQ is much smaller than that of GADDI and SPath. The main reason is that SPath needs to calculate and merge label paths around nodes, resulting in high query response time for complex structure graphs, while GADDI lacks an effective node merging sequence.

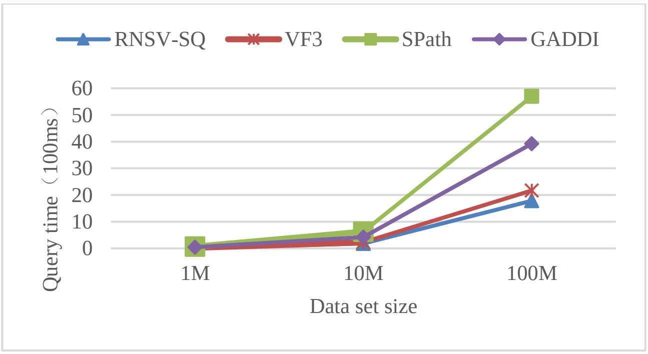
We also tested the effect of dataset size on the performance of the algorithms. Through experiments on 1M, 10M, and 100M of WatDiv dataset, 12 query templates of the above four types are queried respectively, and the average query time of RNSV-SQ, GADDI, SPath, and VF3 on the four types is calculated.



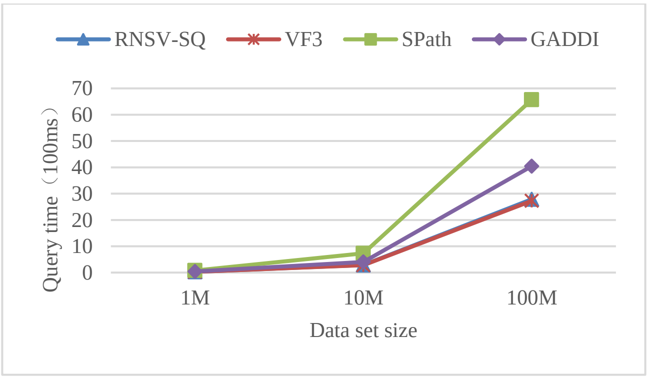
(a) Query time of type L on 1M, 10M, and 100M of WatDiv dataset



(b) Query time of type S on 1M, 10M, and 100M of WatDiv dataset



(c) Query time of type F on 1M, 10M, and 100M of WatDiv dataset



(d) Query time of type C on 1M, 10M, and 100M of WatDiv dataset

Figure 9. Query time of different dataset sizes and query types

From the experimental results in Figure 9, it can be observed that when the dataset size increases from 1M to 100M, the query time of the four algorithms increases, and the query time of SPath and GADDI increases greatly, especially when the dataset increases from 10M to 100M. The query time of SPath and GADDI is more obvious than that of VF3 and RNSV-SQ, while the query time of RNSV-SQ and VF3 increases less, they are more stable in different query types. It can be seen from the query templates S and F that RNSV-SQ has less query time than VF3, so RNSV-SQ has high query efficiency in some cases. Therefore, as the size of the dataset increases, RNSV-SQ is still effective and feasible.

In sum, experiments prove that RNSV-SQ proposed in this paper has better advantages in database pre-creation, storage memory size, and execution efficiency compared with GADDI, SPath, and VF3.

5 Conclusions

This paper proposes RNSV-SQ to decompose the RDF graph into a star-shape, and calculates the query sequence of generating these star-shaped subgraphs by using a custom node cost model. Based on the generated sequential query, the generation of intermediate result sets can be effectively reduced, and query performance can be improved. Experiments have shown that compared with other algorithms, the algorithm proposed in this paper has better advantages in execution efficiency and other aspects.

The algorithm proposed in this paper has a good performance in querying RDF subgraphs, but it also has its shortcomings. There may be many directions for improvement in future related research. The current optimal query order can be obtained through the custom node cost model, and in the future, we can try to obtain a better query order to reduce the intermediate result set. In terms of datasets, we can also add further experiments on actual large real datasets or further query and research on datasets with fixed frequency change and update. In addition to star decomposition, other improved graph structures decomposition methods can be introduced to the intermediate result processing method, such as linear type, snowflake type, etc.

Acknowledgements

This research was supported by the National Natural Science Foundation of China under Grant No. 61862010; “BAGUI Scholar” Program of Guangxi Zhuang Autonomous Region of China; Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing; Guangxi Natural Science Foundation under Grant No. 2018GXNSFBA281086; Innovation Project of Guangxi Graduate Education No. YCSW2021283.

References

[1] K. Lee, H. Jung, J. S. Hong, W. Kim, Learning Knowledge Using Frequent Subgraph Mining from Ontology Graph Data, Applied Sciences, Vol. 11, No. 3, Article No. 932, February, 2021.

[2] S. U. Rehman, S. Asghar, A-RAFF: A Ranked Frequent Pattern-growth Subgraph Pattern Discovery Approach, *Journal of Internet Technology*, Vol. 20, No. 1, pp. 257-267, January, 2019.

[3] W. Zhang, W. K. Chan, Subgraph Isomorphism Building on A Hierarchical Query Graph, *5th International Conference on Compute and Data Analysis (ICCDA)*, Sanya, China, 2021, pp. 107-111.

[4] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, A (Sub)graph Isomorphism Algorithm for Matching Large Graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 26, No. 10, pp. 1367-1372, October, 2004.

[5] S. Zhang, S. Li, J. Yang, GADDI: Distance Index Based Subgraph Matching in Biological Networks, *12th International Conference on Extending Database Technology (EDBT)*, Saint Petersburg, Russia, 2009, pp. 192-203.

[6] P. Zhao, J. Han, On Graph Query Optimization in Large Networks, *Proceedings of the VLDB Endowment*, Vol. 3, No. 1-2, pp. 340-351, September, 2010.

[7] V. Carletti, P. Foggia, A. Saggese, M. Vento, Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3, *IEEE transactions on pattern analysis and machine intelligence*, Vol. 40, No. 4, pp. 804-818, April, 2018.

[8] X. Ren, J. Wang, Exploiting Vertex Relationships in Speeding up Subgraph Isomorphism over Large Graphs, *Proceedings of the VLDB Endowment*, Vol. 8, No. 5, pp. 617-628, January, 2015.

[9] J. Wang, X. Ren, S. Anirban, X.-W. Wu, Correct Filtering for Subgraph Isomorphism Search in Compressed Vertex-Labeled Graphs, *Information Sciences*, Vol. 482, pp. 363-373, May, 2019.

[10] K. M. Kyu, K. T. Yar, A. N. Oo, A Proposal of CS-index Approach for SPARQL Queries Considering Chain and Star Shaped Subgraphs, *Proceedings of the 10th International Conference on Advances in Information Technology (IAIT)*, Bangkok, Thailand, 2018, pp. 1-7.

[11] K. M. Kyu, A. N. Oo, Graph-based Indexing Method for Searching in RDF Data, *2019 International Conference on Advanced Information Technologies (ICAIT)*, Yangon, Myanmar, 2019, pp. 96-101.

[12] Y. Park, S. Ko, S. S. Bhowmick, K. Kim, K. Hong, W.-S. Han, G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching, *International Conference on Management of Data (SIGMOD/PODS’20)*, Portland OR, USA, 2020, pp. 1099-1114.

[13] F. Bi, L. Chang, X. Lin, L. Qin, W. Zhang, Efficient Subgraph Matching by Postponing Cartesian Products, *International Conference on Management of Data (SIGMOD/PODS'16)*, San Francisco, California, USA, 2016, pp. 1199-1214.

[14] S. U. Rehman, S. Asghar, S. J. Fong, Optimized and Frequent Subgraphs: How Are They Related?, *IEEE Access*, Vol. 6, pp. 37237-37249, June, 2018.

[15] S. U. Rehman, S. Asghar, S. Fong, An Efficient Ranking Scheme for Frequent Subgraph Patterns, *Proceedings of the 10th international conference on machine learning and computing (ICMLC)*, Macau, China, 2018, pp. 257-262.

[16] S. U. Rehman, K. Liu, T. Ali, A. Nawaz, S. J. Fong, A Graph Mining Approach for Ranking and Discovering the Interesting Frequent Subgraph Patterns,

International Journal of Computational Intelligence Systems, Vol. 14, No. 1, pp. 1-17, August, 2021.

[17] H. Guan, B. Zhu, G. Y. Li, L. Zhao, Efficient Subgraph Matching Method based on Structure Segmentation of RDF graph, *Journal of Computer Applications*, Vol. 38, No. 7, pp. 1898-1904+1909, July, 2018.

[18] B. Ning, Y. Sun, D. Zhao, W. Xing, G. Li, Dominance-Partitioned Subgraph Matching on Large RDF Graph, *Complexity*, Vol. 2020, pp. 1-18, December, 2020.

[19] G. Li, L. Yan, Z. Ma, An Approach for Approximate Subgraph Matching in Fuzzy RDF Graph, *Fuzzy Sets and Systems*, Vol. 376, pp. 106-126, December, 2019.

[20] S. Sakr, M. Wylot, R. Mutharaju, D. L. Phuoc, I. Fundulaki, *Centralized RDF Query Processing*, in: Linked Data, Springer, Cham, 2018, pp. 33-49.

[21] M. Spasić, M. Jovanovik, A. Prat-Pérez, An RDF Dataset Generator for the Social Network Benchmark with Real-World Coherence, *2016 Workshop on Benchmarking Linked Data*, Kobe, Japan, 2016, pp. 1-8.

Biographies

Mingyan Wang is currently a graduate student at the School of Physics and Electronics, Nanning Normal University, Nanning, China. His research interests include graph data management and graph mining.

Qingrong Huang is currently a graduate student at the School of Computer & Information Engineering, Nanning Normal University, Nanning, China. Her research interests include graph data management and graph mining.

Nan Wu is currently a graduate student at the School of Computer & Information Engineering, Nanning Normal University, Nanning, China. Her research interests include data mining and graph mining.

Ying Pan received her Ph.D. degree from Sun Yat-sen University in 2011. Currently, she is a professor at the School of Computer & Information Engineering, Nanning Normal University, China. Her research interests include graph databases, big data, and intelligent computing.