electronics

MDPI

Article

Optimized Distributed Subgraph Matching Algorithm Based on Partition Replication

Ling Yuan, Jiali Bin and Peng Pan *

School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China; cherryyuanling@hust.edu.cn (L.Y.); binj@hust.edu.cn (J.B.)

* Correspondence: panpeng@hust.edu.cn

Received: 7 December 2019; Accepted: 2 January 2020; Published: 18 January 2020

Abstract: At present, with the explosive growth of data scale, subgraph matching for massive graph data is difficult to satisfy with efficiency. Meanwhile, the graph index used in existing subgraph matching algorithm is difficult to update and maintain when facing dynamic graphs. We propose a distributed subgraph matching algorithm based on Partition Replica (noted as PR-Match) to process the partition and storage of large-scale data graphs. The PR-Match algorithm first splits the query graph into sub-queries, then assigns the sub-query to each node for sub-graph matching, and finally merges the matching results. In the PR-Match algorithm, we propose a heuristic rule based on prediction cost to select the optimal merging plan, which greatly reduces the cost of merging. In order to accelerate the matching speed of the sub-query graph, a vertex code based on the vertex neighbor label signature is proposed, which greatly reduces the search space for the subquery. As the vertex code is based on the increment, the problem that the feature-based graph index is difficult to maintain in the face of the dynamic graph is solved. An abundance of experiments on real and synthetic datasets demonstrate the high efficiency and strong scalability of the PR-Match algorithm when handling large-scale data graphs.

Keywords: subgraph matching; graph indexing; distributed computing; graph partition

1. Introduction

A graph is a semi-structured data represented by vertices and edges, which is usually represented as $G(V, E)$, where V represents the set of vertices and E the set of edges between vertices. In the data analysis area, vertices are usually used to represent objects while edges reflects relationships among objects. For example, a protein can be viewed as the vertex of a graph and the interaction of a protein can be regarded as the edge of a graph, while a protein interaction network can be structured as a STRING.

Graph theories have been applied to many scenarios such as optimal transport path, semantic web analysis, social network analysis, community discovery, knowledge question and answer, and so on, which can use an unsupervised embedding learning feature representation scheme by deep Siamese neural networks [1] for dimension reduction. The graph theories can be applied to many other fields, like merging with sensory data using the artificial neural networks for prediction [2] or merging with a Markov Random Field to model the spatial correlation among data [3] for more accurate map matching, which can use the Gaussian kernel-based method [4] for dimension reduction. Subgraph matching is a very important research topic in graph theory which can help users to extract valuable information from graph datasets. There are many important applications of the subgraph matching in the real world. First, a protein molecular structure can be found to match the virus antibody in a biological protein network, and knowledge retrieval with subgraph matching can be completed by transforming the user description into the corresponding subgraph matching template. Also, subgraph matching

can be used to detect similar organizations by finding the subgraph characteristics of certain groups in social networks, thus providing useful information for public security and to investigate and solve incidents. All in all, the subgraph matching is a very meaningful research field that has been used in all walks of life, whose range of field is from biomedicine to artificial intelligence.

Subgraph matching has been proven as an NP complete problem by Ullmann [5]. With the explosive growth of data scale in the big data era, the problem of subgraph matching for the massive graph data has brought more challenges. The worst time complexity of the traditional subgraph matching algorithm is close to the exponential level, which is very weak in the face of large-scale data, and the traditional subgraph matching algorithms are mostly for the small-scale graphs of the single-computer version, which do not consider the problem of the data graph split and distributed query when the computing power and storage capacity of the single computer are insufficient. At present, most of the subgraph matching algorithms utilize recursive backtracking methods to match the query vertex continuously, and filter the query candidates with the graph index. However, most of the algorithms lack extensibility and scalability for large-scale dynamic graph data. Accompanying the increment of graph size, a single machine has difficulty in holding and computing a large graph, therefore, distributed graph storage and matching present the significance and necessity for massive graph datasets. Importantly, a large graph means that both the the numbers of vertices and edges are large, and usually contain hundreds of millions of vertices and billions of edges; large graph data partition is a very important research direction in the distributed graph data processing. Existing distributed subgraph matching mainly uses an RDF graph engine and map-reduce computing framework, which can hardly achieve satisfying efficiency.

To solve these problems, we propose a distributed subgraph matching algorithm based on Partition Replica (noted as PR-Match) to process the subgraph matching of large-scale data graph. The PR-Match algorithm mainly consists of four stages: data graph partitioning and storage, query graph splitting, subquery matching, and subquery matching result merging. For the proposed PR-Match algorithm, we design a large-scale data graph partition and storage scheme based on the theory of equilibrium separation of large graphs, develop a high efficient vertex code index to process fast updating and maintenance on dynamic graphs, and establish the heuristic rules based on the prediction overhead to determine the merging sequence of subquery matching results.

We conduct abundant evaluations and comparative experiments on the proposed PR-Match algorithm. We choose a proper data cleaning strategy based on the relationship of data volume, completeness, time-dependence and correctness [6], and take the privacy of datasets into account [7]. The experimental results and analyses demonstrate that the PR-Match algorithm has good scalability with different sizes of graph datasets. In addition, the performance of the PR-Match algorithm is greatly improved when the vertex average degree of the data graph is large and the lables are more signed, which shows the feasibility and efficiency of the vertex code index. At the same time, the query response time of the PR-Match algorithm can be only slowly increasing with the size of graph datasets. Accordingly, the PR-Match algorithm has obvious advantages over the high performance graph database Neo4j, which shows that the PR-Match algorithm can be competitive for large-scale graph data matching.

The remaining parts of this paper are organized as follows. Section 2 discusses the related work. Section 3 illustrates problem definitions about subgraph matching. The proposed PR-Match algorithm is elaborated in Section 4. The experiments are elaborated in Section 5. Section 6 concludes the paper.

2. Related Work

Subgraph matching usually uses the index strategy which establishes the inverted index according to some features in the graph data to reduce the search space. iGraph [8] has made a related summary and introduction to the index-based subgraph matching algorithm. iGraph divides the graph index set into a mining index [9] part and a non-mining index [10] part. Mining index uses the high frequency subgraph mining algorithm to find high frequency subgraphs, subtree, path and so on as the key

of the index. The Tree+Delta and SwiftIndex [10] take subtree as index structure. Tree+Delta uses the ring structure in the graph query as the online index and obtains a good retrieval effect for the graph query containing the ring. Non-mining index uses the graph inherent structure information to establish the index structure. C-Tree [10,11] puts forward the concept of the closure graph, and uses this concept to build a hierarchical tree. Moreover, the equivalent vertices can be found by a two-time sequential search in a graph [12] and large data graphs can be partitioned based on structures and labels [13]. iGraph has concluded that (1) although gIndex is the oldest index method, it has the best index effect (the pruning ability and I/O overhead in query); (2) the index effect is better in the dense data graph with fewer lables; (3) the results of C-Tree is a little poor in most cases; (4) the complex query Tree+delta on dense data graph is the best.

Previous algorithms based on the recursive backtracking and graph index target at small scale graph datasets, while large-scale social networks and bioinformatics network are common at present, a lot of researchers study the matching problem of super large-scale graphs. The Turboiso [14] algorithm proposes a plan merging candidate region detection and joint arrangement that makes the algorithm applicable to various query graphs with different structures and distribution. Similar paper [15] avoids the Descartes Cartesian product by means of using paths. Liang et al. [16] design a very clever index structure, which can take advantage of the powerful anti-monotone pruning, horizontal pruning and vertical pruning of the index structure to greatly reduce the candidate set. On this basis, a subgraph matching algorithm SMS2 is developed to handle subgraph queries over tens of millions of vertices. Splitting a query graph into multiple sub query graphs accelerates the query procedure. Based on that, SGMatch [17] realizes high performance large-scale subgraph matching by optimizing the query graph decomposition and prediction-based subquery sequence.

Cloud computing and distributed computing are becoming more and more popular. For example, the CAGW_PD dynamic replication strategy was applied to reduce file access time [18] in the distributed environment, FD was applied to some distributed applications for improving QoS [19], MR-M was applied to achieve the intrasession fairness and intersession fairness [20], and cloud computing brought great convenience for smart energy management [21], and so on. More and more large-scale data processing and analyses are moving to the distributed environment which makes the research on distribution subgraph popular. There are two main patterns of graph expression, simple graph and RDF graph, which can be transformed into each other although each of them has different data presentation. Distributed RDF graph matching methods are divided into three categories according to [22]: a cloud-based method [23–25] , a partition-based method [26–28] and a joint-based method [29]. Cloud computing methods mostly use map-reduce computing framework and HDFS-like distributed storage systems. Partition-based methods divide an RDF graph into multiple subgraphs and each subgraph is maintained by a cluster node. When an SPARQL query is proposed, it is split into multiple subqueries, and then the sub results are merged to obtain the complete matching result. GraphPartition [28] splits the data graph by a hash partition algorithm. The subgraph is extended by n-hop so that each subquery does not need to communicate with other cluster nodes. The joint-based method needs to get metadata for each RDF graph endpoint, which is suitable for data sharing among multiple organizations. When choosing a cloud-based method, multi-clouds can perform better than a single cloud under a loose deadline with MCPCPP algorithms [30]. Furthermore, when network congestion occurrs in a distributed environment, the NPD-RED [31] whose core idea is probability for packet dropping is a useful choice.

3. Problem Definition

We consider the subgraph matching problem on the labeled undirected graphs, and the relevant problem definitions are presented as follows:

Definition 1. ***Label graph*** *A labeled graph is a quadruple of the form $G(V, E, L, F)$, where V is a set of vertices, $e(u_i, u_j) \in E$ is a set of edges, L is a set of label on vertices and edges, and F is a labeling function of the form $F : V \cup E \rightarrow L$, such that it gives a label to each vertex and edge.*

Definition 2. ***Subgraph*** *Given graphs $G_1 < V_1, E_1, L_1, F_1 >$ and $G_2 < V_2, E_2, L_2, F_2 >$, graph G_1 is the subgraph of graph G_2 if and only if:*

(1) $V_1 \subseteq V_2, E_1 \subseteq E_2, L_1 \subseteq L_2$;
(2) $\forall v \in V_1, F_1(v) = F_2(v)$;
(3) $\forall e(v_1, v_2) \in E_1, F_1(e) = F_2(e)$.

Definition 3. ***Subgraph matching*** *Given a query graph $G_q < V_q, E_q, L_q, F_q >$ and a graph database $D = \{G_1, G_2 \dots G_n\}$, subgraph matching problem or subgraph isomorphism problem is to find all data graphs or subgraphs isomorphism of query graph G_q in graph database D.*

Therefore, Subgraph matching aims to find the graph which is isomorphic to the query graph in the graph datasets. The graph isomorphism definition is defined as follows:

Definition 4. ***Graph isomorphism*** *Given graphs $G_1 < V_1, E_1, L_1, F_1 >$ and $G_2 < V_2, E_2, L_2, F_2 >$, G_1 and G_2 is graph isomorphism if and only if there is an injective function $f : V_1 \rightarrow V_2$ such that the conditions hold:*

(1) $\forall v \in V_1, F_1(v) = F_2(f(v))$;
(2) $\forall e_1(v_1, v_2) \in E_1, \exists e_2(f(v_1), f(v_2)) \in E_2, F_1(e_1) = F_2(e_2)$.

For convenience, we use Q to represent the query graph, D to represent the graph database, G to represent a data graph, u to represent the vertex in the query graph, and v to represent the vertex in the graph data.

4. Proposed PR-Match Algorithm

This section gives a detailed illustration about our proposed PR-Match algorithm, mainly involving data graph partition and storage, query graph split, subquery match, and intermediate results merge.

4.1. Graph Data Partition

For the graph data partitions which are stored on multiple machines, the time cost of each subgraph matching is different. Considering the storage space overhead and the characteristics of the data access, we choose the neighbor vertex replication strategy to reduce the cost of subgraph matching. The graph data is firstly divided into the hash partition graph according to the specific vertex information. Then, the neighbor vertices of the core vertices in the partitioned graph and the direct adjacency edges are copied to the current partition graph. A distributed graph definition is given below:

Definition 5. ***Distributed graph*** *A distributed graph $G < V, E, L, F >$ consists of a set of partitions $F = \{F_1, F_2, \cdots, F_k\}$, where each F_i is specified by $< V_c^i \cup V_e^i, E_c^i \cup E_e^i, L_i, F_i >$ $(i \in 1, 2, \cdots, k)$ such that :*

(1) $V_c^1, V_c^2, \cdots, V_c^k$ *is a partition of V, $\forall i, j \in 1, 2, \cdots, k, i \neq j, V_c^i \cap V_c^j = \emptyset$, and $U_{i \in 1,2,\cdots,k} V_c^i = V$, V_c^i is called as core vertex of F_i;*
(2) $e(v_1, v_2) \in E_c^i$, *where $v_1 \in V_c^i, v_2 \in V_c^i$, E_c^i is called as core edge of F_i;*
(3) E_e^i *is a set of crossing edges between F_i and other partitions, E_e^i is called as extended edge of F_i;*
(4) $e(v_1, v_2) \in E_e^i$, *where $v_1 \in V_c^i, v_2 \in V_c^j$ and $i \neq j$, V_e^i is called as extended vertex of F_i.*

The data graph is divided into multiple partitions after being hashed. Each machine stores a partition graph. A specific description of the hash partition and vertex neighbor replication on graph data is described in Algorithm 1.

Algorithm 1: Graph Data Partition Algorithm

input : a data graph $G < V, E >$; the number of subgraphs k
output: a set of Partitions $F = \{F_1, F_2, \cdots, F_k\}$

```
for v_i ∈ V do
    p = hash(v_i) mod k + 1;
    V_c^p = V_c^p ∪ {v_i};
for e_i(v_m, v_n) ∈ E do
    if hash(v_m) = hash(v_n) then
        p = hash(v_m) mod k + 1;
        E_c^p = E_c^p ∪ {e_i(v_m, v_n)};
    else
        x = hash(v_m) mod k + 1;
        y = hash(v_n) mod k + 1;
        E_e^x = E_e^x ∪ {e_i(v_m, v_n)}, E_e^y = E_e^y ∪ {e_i(v_m, v_n)};
        V_e^x = V_e^x ∪ {v_n}, V_e^y = V_e^y ∪ {v_m};
```

Firstly, Algorithm 1 determines which partition the current vertex v_i belongs to according to the vertex hash value, at the same time, the vertex v_i is the core vertex of the partition F_i. Secondly, Algorithm 1 obtains the edge set of the partition. If the vertices of the current edge $e_i(v_m, v_n)$ are both in the same partition F_i, add the edge e_i to the partition F_i as the core edge of this partition. If the vertices v_m and v_n of the current edge belong to different partitions F_i and F_j, then the edge is added to the partition F_i and F_j as their extended edges. Furtherrmore, Algorithm 1 adds v_m to partition F_j as extended vertex of partition F_j, adds v_n to partition F_i as extended vertex of partition F_i. Also, both the number of the vertices and edges are very large, whose numerical units are both 100 million. The time complexity of Algorithm 1 is $O(|V| + |E|)$, where $|V|$ is the number of vertices in the graph data, and $|E|$ is the number of edges in the graph data.

4.2. Query Decomposition

After partitioning the big graph data into the cluster environment, a subdgraph on a cluster node is incomplete. If the query request is sent to a cluster node directly, the information of other subgraphs should be obtained from other cluster nodes, which would increase the communication overhead among the cluster nodes. According to the neighbor replication strategy, each cluster node has all the direct neighbor information of its core nodes. Based on this knowledge, the query graph can be decomposed into several subquery graphs, so that each subquery can be independently calculated on each cluster node to reduce the communication overhead. Firstly, we give the relevant definitions.

Definition 6. ***Hopping number*** *Given a graph $G < V, E, L, F >$, the hop number between vertex v_i and vertex v_j is denoted as $hop(v_i, v_j)$, which is the minimum distance between v_i and v_j in the graph. Similarly the hop number between vertex v and edge e is denoted as $hop(v, e(v_i, v_j))$, which is "$min(hop(v, v_i), hop(v, v_j)) + 1$" meaning the minimal number of crossing edges that v_i reaching to e.*

We specify $hop(v_i, v_j) = 0$ when $v_i = v_j$, otherwise, when v_i and v_j are not reachable, $hop(v_i, v_j) = \infty$.

Definition 7. ***Star graph*** *Graph $G < V, E, L, F >$ is called as a star graph, if and only if:*

(1) $\exists v_0 \in V, \forall v \in V$ *when* $v \neq v_0, hop(v_0, v) = 1$, v_0 *is called as the center point of graph G;*
(2) $\forall e \in E, hop(v_0, e) = 1$ *where v_0 is the center point of the graph G.*

We use G^* to represent the star graph. From the Definition 6, we can find that the star graph G^* is a graph composed of a center point and all its direct neighbor vertices, and the edges of a center point

to its neighbor vertices. To be convenient, the star graph is represented as $G^* < v_0 \cup N(v_0), E, L, F >$, where v_0 is the center point of G^*, and $N(v_0)$ is the neighbor vertex of v_0 .

Theorem 1. *A data graph G is partitioned into $F = F_1, F_2, \cdots, F_k$, if the query graph Q is a star graph, query graph Q can be answered independently on each partitioned graph F_i.*

Proof. When the query graph Q is matched with the partition graph F_i, the starting query vertex is bound to the center point of the partition graph F_i. Because all the neighbors and direct adjacency edge information of the center point exist in the partition graph F_i, all the match results of the query graph Q with the partition graph F_i can be obtained. □

To avoid the communication overhead between cluster nodes during the process of subgraph matching, the original query graph is split into several star subgraphs according to the Definition 7. Although it is an NP-hard problem to split a query graph into multiple star graphs [32], there still remains a variety of resolution schemes to be obtained. To choose the best decomposition solution, some conditions should be taken into consideration. One is that the lower the number of subqueries, the less calculation cost of subgraph matching. Another condition is that the fewer candidate results for each subquery, the less cost of intermediate result merging. According to these conditions, we define the center point selection function for a star query graph, which is represented as Selectivity:

$$Selectivity(u_1) = \frac{degree(u_1)}{freq(u_1.label) * \min\limits_{e(u_1,u_2)\in E} freq(u_2.label)}$$

Where the $degree(u)$ represents the degree of vertex u, and $freq(u.label)$ indicates the number of labels of the vertex u appearing in the graph data. We choose a vertex with bigger degree and fewer candidate sets as the center point of the query graph. Based on the selection function, we propose a query graph decomposition algorithm, as shown in Algorithm 2.

Algorithm 2: Query Graph Decomposition Algorithm

input : a query graph $Q < V, E, L, F >$
output: a set of star graph T

while $E \neq \emptyset$ **do**
  $S_{max} \leftarrow 0, u \leftarrow null$;
  for $u_i \in V$ **do**
    $S_{tmp} = Selectivity(u_i)$;
    if $S_{tmp} \geq S_{max}$ **then**
      $S_{max} = S_{tmp}$;
      $u = u_i$;
  $T = T \cup \{G^*(u)\}$;
  $E = E / \{e | e \in G^*(u)\}$;
  $V = u / \{v | v = u, or \, \mathtt{deg}(v) = 0\}$;

In Algorithm 2, the input is the original query graph, and the output is the decomposition result of the query graph. Because subquery decomposition is an edge coverage problem, our algorithm utilizes the edge traversal method to generate a subquery. The algorithm firstly finds the highest selective vertex, then constructs the star query graph with the current vertex as the center point of the current subquery. $G^*(u)$ is a star graph with vertex u as its center point. Next, the algorithm deletes the center point of the current subquery and the involved edges, as well as the vertex with 0 indegree. This algorithm is terminated when all the edges of the query graph are covered by the subquery. The time complexity of Algorithm 2 is $O(|V| \cdot \overline{deg(u)})$, where $|V|$ represents the number of the vertex in the query graph, and the $\overline{deg(u)}$ represents the average degree of the vertex in the query graph.

4.3. Subquery Matching

After the original query graph is decomposed into sevaral star query graphs, the subquery matching requests are distributed to all the nodes in the cluster; therefore, the cluster nodes can complete the subquery matching according to the local data. Since the subgraph matching is an NP-complete problem, most algorithms use the "filter-refining" framework to accelerate the response time of subgraph matching. Firstly, the candidates which cannot satify the conditions are removed by the pre-designed filtering strategy. Secondly, the subgraph isomorphism test is applied to the remaining candidate sets. With the thought of the graph index, we design a vertex code to reduce the search space of the subquery.

The vertex candidate set of the query vertex u named $C(u)$ in graph database D consists of all vertices which contains $F_v(u)$ labels in the graph database. If the vertex v of the data graph matches the vertex u of the query graph, $|N(v)|$ is bigger than $|N(u)|$, where $|N(u)|$ is the number of neighbors of the vertex u.

Definition 8. ***Neighbor label signature*** *The neighbor label signature of vertex v is denoted by $Sig(v)$, which is represented by a tuple $< P_n(v), P_e(v) >$, where $P_n(v)$ is a label of multiple sets of all its neighbor vertices, $P_e(v)$ is a label of multiple sets of edges between vertex and its neighbors, that is:*

(1) $I \in P_n(v) \Rightarrow \exists v^{'} \in N(u), I = L_v(v)$;
(2) $I \in P_e(v) \Rightarrow \exists v^{'} \in N(v), e(v, v^{'}) \in E, I = L_e(e)$.

Theorem 2. *Given graphs Q and G. Under the bijective function f, Q is isomorphic to G. For any vertex u in graph Q, the neighbor label signature of vertex u is signed to be $Sig(u) =< P_n(u), P_e(u) >$. If $v = f(u)$ and its neighbor label signature is signed to be $Sig(v) =< P_n(v), P_e(v) >$, then they should satisfy:*

$$(1)\ P_n(u) \subseteq P_n(v);$$
$$(2)\ P_e(u) \subseteq P_e(v).$$

then, the label of vertex v covers that of the vertex u.

The Theorem 2 clearly states that vertex neighbor label signature contains the label information of the vertices around the vertex and their rough structure information; thus, the candidate nodes can be filtered with the vertex label signature. In order to update and verify the signature information of a vertex label, we map the signature information of the vertex neighbor label to the numerical space.

Definition 9. ***Label code*** *Given a label l, the number of non-negative hash functions m, the label code of label l is denoted by $Encode(l)$ which is a binary string I with a length of K, where I is initialized to 0, and each of the values satisfies the following formula: where $I[j]$ represents the value of the j_{th} bit in the binary string I.*

$$I[hash_i(label)] \bmod K + 1 = 1, i \in 1, 2, \cdots, k;$$

Definition 10. ***Vertex code*** *Given a vertex v, the neighbor label signature of point v is signed to $Sig(v) =< P_n(v), P_e(v) >$, and the vertex code of vertex v is denoted by $Encode(v) = p \diamond q$, where p is a counting string of all labels encoded in $P_n(v)$, and q is a counting string of all labels encoded in $P_e(v)$. $\diamond$ is a join operation for counting strings, and $|Encode(v)| = 2k$, that is:*

$$(1)\ p[i] = \sum_{l \in P_n(v)} Encode(l)[i], i \in 1, 2, \cdots, k$$
$$(2)\ q[i] = \sum_{l \in P_e(v)} Encode(l)[i], i \in 1, 2, \cdots, k$$

Theorem 3. *Given graphs Q and G, under the bijective function f, Q is isomorphic to G. For any vertex u in graph Q, the vertex code of vertex u is signed to be $Encode(u) = p_1 \diamond q_1$, if $v = f(u)$ and its vertex code is signed to be $Encode(v) = p_2 \diamond q_2$, then they should satisfy:*

(1) $p_1[i] \leq p_2[i]$;
(2) $q_1[i] \leq q_2[i]$.

According to the above definitions and theorems, the subquery matching algorithm is presented in Alogirthm 3.

Algorithm 3: Subquery Matching

input : star query graph $Q^* < u_0 \cup N(u_0), E_q, L_q, F_q >$, partition graph F_i on clusters i
output: a set of matching graph PM_i^q graph

```
PM_i^q ← ∅ ;
get candidate vertices C(u_0) of query vertex u_0 ;
for v_0 ∈ C(u_0) do
    if |N(v_0)| ≤ |N(u_0)| then
        Break ;
    if v_0 ∈ V_c^i then
        Get vertex code of vertex v_0 as Encode(v_0) ;
        if Encode(v_0)[i] ≥ Encode(u_0)[i], i ∈ {1, 2, 3, ··· , 2k} then
            for u_m ∈ N(u_0) do
                for v_n ∈ N(v_0) do
                    if F_q(u_m) ⊆ F_i(v_n) and F_q(u_0, u_m) ⊆ F_i(v_0, v_n) then
                        S_m → S_m ∪ {(u_m, v_n)};
            PM_i^q = PM_i^q ∪ {{(u_0, v_0)} × S_1 × S_2 × S_3 ··· S_p} ;
```

The star graph matching mainly includes two processes: namely, off-line operation and online operation. In off-line operation, vertex code is generated for each vertex in the graph database D. The online operation is divided into two stages: candidate filtering and subgraph connectivity testing. The Algorithm 3 firstly obtains the candidate set of a center point of star query graph based on the label, then removes the vertex candidates which are not the center point of the partition F_i. Next, the pruning operation is carried out according to the vertex degree and the vertex code, then the connectivity test of the star graph is terminated. Finally, the Cartesian product of the neighbor matching vertices of the center points is computed and all the matching results are expanded. The worst time complexity of Algorithm 3 is $O(n \cdot (m-1)!)$, where n represents the number of candidate sets of the center point of the star query graph in the graph database D_i, and m represents the number of vertices contained in the query graph.

4.4. Intermediate Result Merge

After completing the previous work, the matching results of the subqueries of the original query graph are obtained. In order to achieve the result of the original query graph, it is necessary to merge the intermediate results of the subquery. The merge operation of the subquery matching results is a time-consuming task and a large number of previous work [14,33–35] shows that the matching order of processing units and the merging order of subquery matching results have a very significant impact on the performance of subgraph matching. This section mainly discusses the optimization of the merging order of subquery matching results.

Definition 11. ***Merge plan*** *The partition result of the query graph Q is $T = q_1, q_2, \cdots, q_n$, and its matching result on all cluster nodes is $M = M_1, M_2, \cdots, M_n$, and $\Omega = M_{s1} \bowtie M_{s2} \bowtie \cdots \bowtie M_{sn}$ represents a merge plan for the matching result of subquery. The star graph corresponding to the M_{si} has intersecting vertices with a subquery graph before M_{si} in the merge plan sequences. $M_{si} \in M$, $\bowtie$ represents the merge operation.*

In the Definition 11, the subquery located in the merge plan is intersected with a certain subquery before its location, which can make sure that each merge performs a connectivity check and avoids the invalid merge overhead.

Definition 12. ***Merge cost*** *A graph database D has been stored on m machines, the query graph Q is decomposed into n star query graphs, the matching results of subquery q_i on the node k is PM_k^i, then the merge overhead of the merge plan Ω is:*

$$Cost(\Omega) = O(\prod_{i=1}^{n}(\sum_{j=1}^{m}(|PM_j^i| + 1)))$$

Definition 13. ***Optimal merge plan*** *Given the matching results of all subqueries on the partition, the merge plan is the optimal merge plan if and only if for any merge plan $\Omega^{'}$, $Cost(\Omega) \leq Cost(\Omega^{'})$.*

Since finding the optimal merge plan is an NP-complete problem, many researcheres have used dynamic programming and greedy strategy to obtain the suboptimal merge plan. The methods to obtain the suboptimal merge plan are mainly divided into two categories. The first one is to determine the merge sequence before the actual merge conduction accroding to a static overhead prediction model, so the merge sequence will not be modified during the merge process. Actually, a well performed static overhead prediction is a key point in this method. The other one is to firstly choose an initial matching set, and the next matching set is dynamically selected according to the current merged state. This method requires a dynamic merging cost calculation model. Although the dynamic methods have better merging performance, the static methods have better results for specific datasets or specific query graphs. In this paper, we use the static method to determine the merging order of the subquery matching results, and design a static cost prediction function called $P - Cost$. Based on the above, this paper is different from other papers that also use the Partition Replication, such as [36].

Definition 14. ***Prediction merge cost*** *A graph database D has been stored on m machines, the partition result of the query graph Q is $T = q_1, q_2, \cdots, q_n$, and its matching result on all cluster nodes is $M = M_1, M_2, \cdots, M_n$, and $\Omega = M_{s1} \bowtie M_{s2} \bowtie \cdots \bowtie M_{sn}$ represents a merge plan for the matching result of subquery, then the prediction merge cost of the merge plan Ω such that:*

(1) The prediction merge cost of the matching result M_{si} and the matching result M_{sj} is:

$$P - Cost(M_{si} \bowtie M_{sj}) = (\sum_{i=1}^{m}(|PM_i^{si}| + 1)) \times (\sum_{i=1}^{m}(|PM_i^{sj}| + 1))$$

(2) The prediction merge cost of merging operation O_i and matching result M_{si+1} is:

$$P - Cost(O_i \bowtie M_{si+1}) = p - Cost(O_i \times (\sum_{i=1}^{m}(|PM_i^{si}| + 1)) \times (\frac{1}{2})^{\alpha})$$

(3) The prediction merge cost of merge plan Ω is:

$$P - Cost(\Omega) = \sum_{i \in 1,2,\cdots,n-1}(P - Cost(O_i \bowtie M_{i+1}))$$

α is the number of intersecting vertices of the subquery q_{si+1} and the merging result O_i, represents the matching results of subquery q_i on the node k, and the $M_{si} \in M$, $\bowtie$ represents the merge operation.

Since the size of the query graph is small, we can list all possible merging plans in a reasonable time; then, we can choose the merging plan with the lowest prediction cost as the optimal merger plan.

After the merging sequence is determined, we can use the nested loop to complete merging of the matching results of the subquery. The illustration about the intermediate result merge is presented in Algorithm 4. The Algorithm 4 mainly completes the merge process of the matching results of the subquery by calling the merge subroutine *recusiveJoin* which is presented in Algorithm 5 based on the depth first traversal. The Algorithm 5 firstly checks whether the current depth has reached the

maximum depth (that is, the merge result is the matching graph of the original query graph). If the maximum depth is reached, the current result is added to the final result set, otherwise, a connectable test will be conducted between the current depth subquery matching results and the merged results. The merge state will be updated when one of the current subgraph matching results is connectable with the merged result; we then proceed to the next layer's recursive merge operation. Note that after each merge is completed, the state that needs to be restored before the merge would be returned. The worst time complexity of the matching result merge is $O(\prod_{i=1}^{m}(|M_{s_i}|)$, where$|M_{s_i}|$ represents the number of matching graphs of the subquery q_{s_i} on all partition graphs.

Algorithm 4: Subquery Matching Result Merge

input : optimal merge plan $\Omega = M_{s_1} \bowtie M_{s2 \bowtie \cdots \bowtie M_{sn}}$, the matching results of all subqueries on all partitions $PM = \{\cup PM_i^{s_1}, \cup PM_i^{s_2}, \cdots, \cup PM_i^{s_n}\}$

output: all the matching subgraph MG of original query graph on graph database D

1 $MG \leftarrow \emptyset, M \leftarrow \emptyset$, curDepth $\leftarrow 1$, maxDepth $\leftarrow n+1$;
2 Call `recusiveJoin` (curDepth, maxDepth, M, MG) ;
3 return MG;

Algorithm 5: Merge Subroutine recursiveJoin

input : curDepth, current recursive depth; maxDepth, the max recursive depth, M, intermediate matching result; MG, matching subgraph

output:

1 **if** curDepth == maxDepth **then**
2 $\quad MG \leftarrow MG \cup M$;
3 *Get the subquery matching results of* q_{curDepth} *as* $\cup PM_i^{\text{curDepth}}$;
4 **for** $G_i^* \in \cup PM_i^{\text{curDepth}}$ **do**
5 $\quad$ **if** G_i^* *is joinable with* M **then**
6 $\quad\quad$ Merge G_i^* with intermediate result M, as $M \leftarrow M \bowtie G_i^*$;
7 $\quad\quad$ `recursiveJoin` (curDepth + 1, maxDepth, M, MG) ;
8 $\quad\quad$ Remove G_i^* from M, and restore the state before merge ;

5. Experiments

The proposed PR-Match algorithm runs in a distributed cluster with six machines, each of the machines is configured with 8G DDR3 memory, an Intel i5-4590 CPU of 3.3 GHz, four cores per CPU; the network adapter is the RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller with a capacity of 1Gbps, a ST1000DM003-1ER162 disk with a capacity of 1TB. All the codes of the algorithm are implemented by Java, the operating system of the machine is Ubuntu Linux, and the version number of the neo4j graph database used by the comparison experiment is neo4j-community-3.1.7. The experiments datasets are illustrated as below:

(1) The subgraph matching on the small graph set uses the AIDS real data and the synthesized dataset generated by GraphGen;
(2) The subgraph matching on a single large graph uses the US Patents [37] real dataset and the synthesized dataset generated by R-Mat [38] .

5.1. Subgraph Matching on Small Graphs

For the subgraph matching experiment of small graph sets, we use the AIDS dataset and the GraphGen synthesis dataset. GraphGen synthetic dataset contains 20,000 data graphs, 100 vertex labels and 100 edge labels. The experimental results are shown in Figures 1 and 2. It is found that the efficiency of the subgraph matching of PR-Match algorithm is similar to that of the Neo4j. When the

query graph is small, the PR-Match even performs worse than the Neo4j, because Neo4j is a centralized single machine matching and the PR-Match algorithm is a distribution pattern matching.

Figure 1. Subgraph matching on a set of small graphs-AIDS.

Figure 2. Subgraph matching on a set of small graphs-GraphGen.

5.2. Subgraph Matching on a Single Large Graph

"US Patents" is a patent reference network that recorded the reference relations of patents between 1963 and 1999 in the United States. It is used in [35] . We use the patent "NCLASS" domain as a patent label, then 714 labels of vertex in total. For the reason that there is only one reference relationship between the patents, to extend the edge relationship, considering a relationship edge, we use the sum of the end point patents release year as the edge label, so we get 630 edge labels in total. R-Mat is a large graph generating tool to simulate large-scale network graphs. The graph generated by R-Mat is unlabeled graph. For this reason, we randomly assign a label for each vertex and edge. Three types of query graphs for subgraph matching have been conducted on a single large graph: path query, clique query and random graph query.

5.2.1. Path Query

In the path query experiment, the query graph is a path consists of vertices. We give a path set which contains nine kinds of path, the vertex number of path range from 2 to 10. Experiment results on the US Patents dataset and the R-Mat synthetic dataset are shown in Figures 3 and 4. Accroding to the Figures 3 and 4, neo4j has advantages in path matching while the response time of PR-Match algorithm is a little high in the path matching. The main reason is that neo4j uses a unique physical storage mode and a powerful traverse framework, meanwhile, PR-Match increases the number of sub-queries in the path query and the pruning ability at the center vertex of the sub-query also decreases.

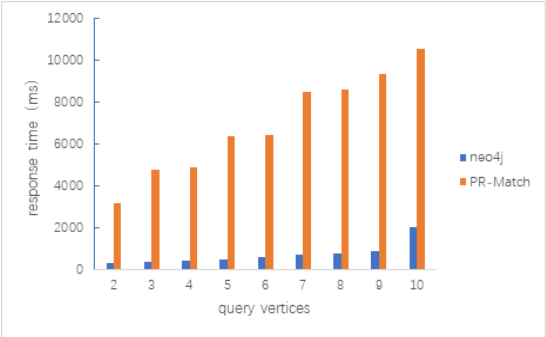

Figure 3. Path matching-US Patents.

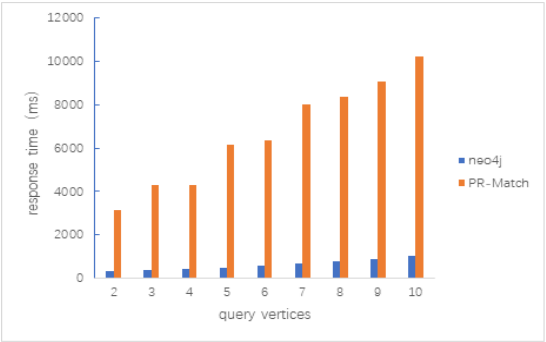

Figure 4. Path matching-R-Mat.

5.2.2. Clique Query

In clique query, each query graph is a complete graph, which indicates that the query graph contains $1/2|V|(|V|-1)$ edges. The clique query graph set contains six kinds of queries with the graph vertex number vary from 2 to 7. Experiment results on the US Patents dataset and the R-Mat synthetic dataset are shown in Figures 5 and 6. With the analysis, Neo4j query response time is rapidly increased when the vertex number of the query graph goes big. As a comparison, the PR-Match response time growth maintains as a stable rate. The query efficiency of neo4j is lower than PR-Match when the vertex number of a query graph is larger than 5. This is because as the query graph density increases, the number of query resolutions will not be too large but the neighbor label density vertex increases at the same time. The pruning ability of vertex neighbor label coding is highly improved.

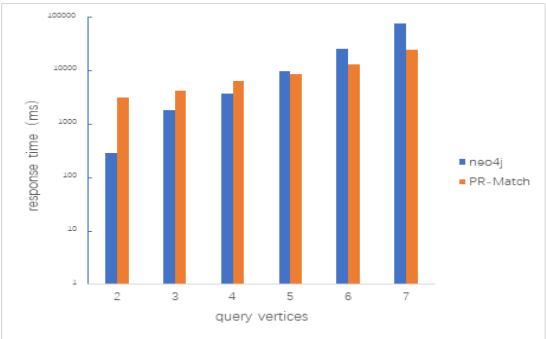

Figure 5. Clique matching-US Patents.

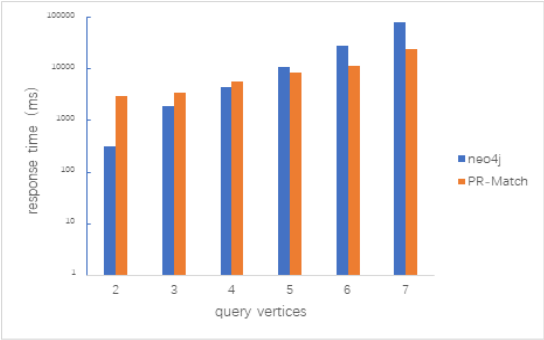

Figure 6. Clique matching-R-Mat.

5.2.3. Random Query

The vertices and edges of the query graph in the random graph query are randomly selected. The random query graph set contains a total of six kinds of queries with the number of vertices ranging from 2 to 7. Experiment results on the US Patents dataset and the R-Mat composite dataset are in Figures 7 and 8, the PR-Match algorithm does not have advantages over the neo4j in the small-scale query graph, but as the size of the query graph gradually increases, the gap between PR-Match and neo4j has been narrowed and surpassed.

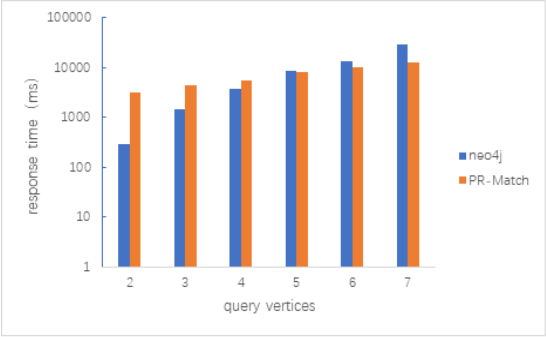

Figure 7. Random query-US Patents.

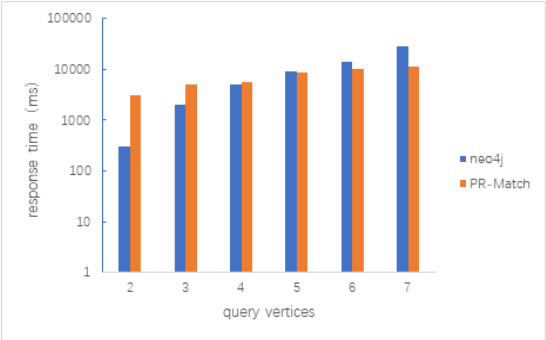

Figure 8. Random query-R-Mat.

5.3. Scalability Test of PR-Match Algorithm

Two sets of experiments are designed to test the scalability of subgraph matching. The first set is used to study the effect of data graph size on the efficiency of subgraph matching. The second set aims to study the influence of an average vertex degree of the query graph on the efficiency of subgraph matching.

5.3.1. Data Size

To study the effect of different scale data graphs on the efficiency of subgraph matching, we use GraphGen to generate five sets of small graphs, the graph numbers of each set are 10 K, 20 K, 30 K, 40 K and 50 K, respectively. We also get five large graphs generated by R-Mat, the number of vertices corresponding to the five large graphs are 2 million, 4 million, 6 million, 8 million and 10 million, respectively. The used query graph is a random graph. The query graph contains seven vertices. The experimental results are presented in Figures 9 and 10. Comparative analysis shows that PR-Match has obvious advantages in large-scale data graphs and query response time increases slowly with the increase of data graph size.

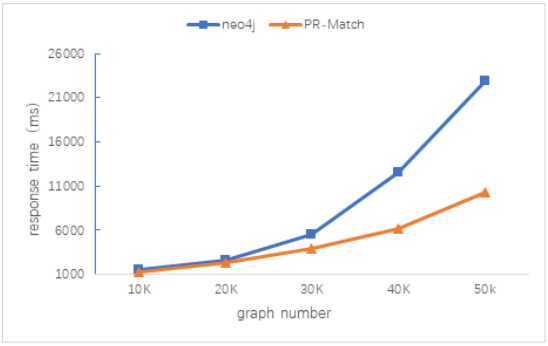

Figure 9. Data size-GraphGen.

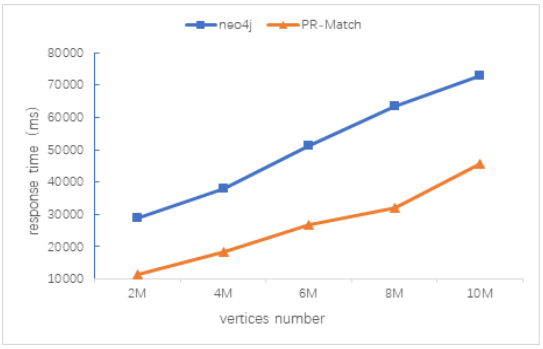

Figure 10. Data size-R-Mat.

5.3.2. Average Vertex Degree

In order to study the influence of the average vertex degree of the query graph on the subgraph matching algorithm, the experimental evaluation is carried out on the AIDS and US Patents datasets. The query graph is a random graph, and the query graph contains seven vertices with average vertex degrees increased from 2 to 6. Experiment results shows in Figures 11 and 12. Comparative analysis shows that on the small scale data graph AIDS, the performance of Neo4j and PR-Match is similar but the response time of the PR-Match algorithm increases slower than Neo4j when the average vertex number of query graph increased. On the large data graph US Patents, PR-Match is not only owns shorter response time but also has lower response time increament rate when the average vertex degree of the query graph grows compared with Neo4j. Therefore, we can demonstrate that the PR-Match algorithm has obvious advantages in a large-scale data graph and dense query graph.

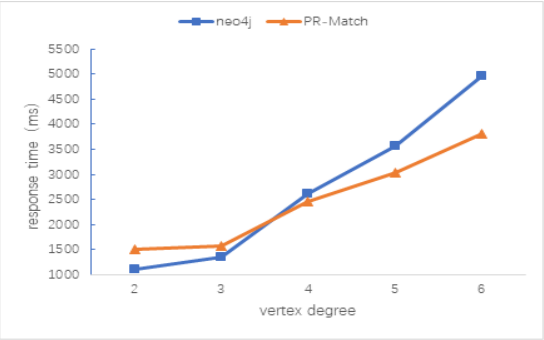

Figure 11. Average vertex degree-AIDS.

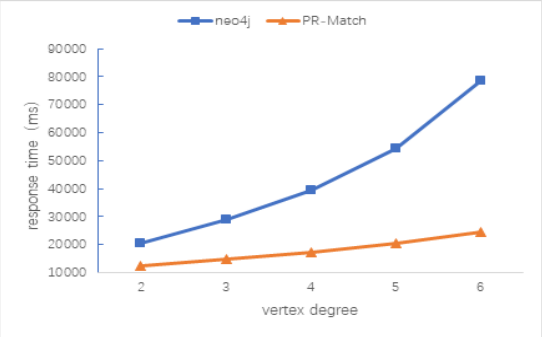

Figure 12. Average vertex degree-US Patents.

5.4. Experiment Summary

Based on the above extensive experimental evaluations and comparative analysis, we find that the PR-Match algorithm has good performance in two different application environments: small graph set and single large graph. The PR-Match has more applications and is compared with the previous subgraph matching algorithms in a single environment. In addition, the performance of the PR-Match algorithm is greatly improved when both the average vertex degree of the data graph and the number of graph labels are large, which indicates the feasibility and efficiency of vertex neighbor label coding in this paper.

6. Conclusions

Our paper proposes a PR-Match algorithm for subgraph matching on large-scale graph datasets.The main work involves that a vertex neighbor replication strategy is designed to consider the efficient graph data partition and query decomposition; a vertex code graph index of the vertex neighbor label is used to prune the query result candidate set; a combined order selection strategy based on the cost prediction is proposed to greatly reduce the cost of merging. Abundant experiments are conducted to domonstrate the efficiency and scalability of the proposed PR-Match algorithm. Meanwhile, the query response time of the PR-Match algorithm only increases slowly with the increase of data graph scale. Therefore, compared with Neo4j, a high-performance graph database at the present stage, it has significant advantages and it is capable of sub-graph matching tasks on large-scale data graphs.

Author Contributions: Methodology, L.Y.; Project administration, P.P.; Validation, J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by National Natural Science Fund of China under grants 61502185.

Acknowledgments: The author would like to thank the equipment support of Huazhong University of Science and Technology and the support of National Natural Science Fund of China.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Guo, W.; Shi, Y.; Wang, S.; Xiong, N. An unsupervised embedding learning feature representation scheme for network big data analysis. *IEEE Trans. Netw. Sci. Eng.* **2019**, *1*. [CrossRef]
2. Cheng, H.; Xie, Z.; Shi, Y.; Xiong, N. Multi-step data prediction in wireless sensor networks based on one-dimensional CNN and bidirectional LSTM. *IEEE Access* **2019**, *7*, 117883–117896. [CrossRef]
3. Cheng, H.; Su, Z.; Xiong, N.; Xiao, Y. Energy-efficient node scheduling algorithms for wireless sensor networks using Markov Random Field model. *Inf. Sci.* **2016**, *329*, 461–477. [CrossRef]
4. Zheng, H.; Guo, W.; Xiong, N. A kernel-based compressive sensing approach for mobile data gathering in wireless sensor network systems. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, 1–13. [CrossRef]
5. Ullmann, J.R. An algorithm for subgraph isomorphism. *J. ACM* **1976**, *23*, 31–42. [CrossRef]
6. Cheng, H.; Feng, D.; Shi, X.; Chen, C. Data quality analysis and cleaning strategy for wireless sensor networks. *Eurasip J. Wirel. Commun. Netw.* **2018**, *61*. [CrossRef]
7. Sang, Y.; Shen, H.; Tan, Y.; Xiong, N. Efficient protocols for privacy preserving matching against distributed datasets. In Proceedings of the International Conference on Information and Communications Security, Raleigh, NC, USA, 4–7 December 2006; pp. 210–227. [CrossRef]
8. Han, W.S.; Lee, J.; Pham, M.D.; Yu, J.X. iGraph: A framework for comparisons of disk-based graph indexing techniques. *Proc. Vldb Endow.* **2010**, *3*, 449–459. [CrossRef]
9. Shang, H.; Zhang, Y.; Lin, X.; Yu, J.X. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proc. Vldb Endow.* **2008**, *1*, 364–375. [CrossRef]
10. Zhang, S.; Hu, M.; Yang, J. TreePi: A novel graph indexing method. In Proceedings of the IEEE International Conference on Data Engineering, Istanbul, Turkey, 15–20 April 2007; pp. 966–975.
11. Jin, F.; Yang, Y.; Wang, S.; Xue, Y.; Yan, Z. TBSGM: A fast subgraph matching method on large-scale graphs. *Int. J. Data Warehous. Min. (IJDWM)* **2018**, *14*, 67–89. [CrossRef]
12. Chen, W.; Li, M.; Chen, Z. Efficient index construction algorithm for isomorphism of subgraphs. *J. Harbin Inst. Technol.* **2019**, *40*, 548–554.
13. Huang, Y.; Hong, J.; Jia, Z. Approximate subgraph matching based on double index. *Comput. Appl.* **2012**, *32*, 1994–1997.
14. Han, W.S.; Lee, J.; Lee, J.H. Turbo iso : towards ultrafast and robust subgraph isomorphism search in large graph databases. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 22–27 June 2013; pp. 337–348.
15. Bi, F.; Chang, L.; Lin, X.; Qin, L.; Zhang, W. Efficient subgraph matching by postponing cartesian products. In Proceedings of the 2016 International Conference on Management of Data, Pune, India, 11–13 March 2016; pp. 1199–1214.
16. Hong, L.; Zou, L.; Lian, X.; Yu, P.S. Subgraph matching with set similarity in a large graph database. *IEEE Trans. Knowl. Data Eng.* **2015**, *27*, 2507–2521. [CrossRef]
17. Rivero, C.R.; Jamil, H.M. *Efficient and Scalable Labeled Subgraph Matching Using SGMatch*; Springer: New York, NY, USA, 2016; pp. 1–27.
18. Wang, Z.; Li, T.; Xiong, N.; Pan, Y. A novel dynamic network data replication scheme based on historical access record and proactive deletion. *J. Supercomput.* **2012**, *62*, 227–250. [CrossRef]
19. Xiong, N.; Vasilakos, A.V.; Yang, L.T.; Song, L.; Pan, Y.; Kannan, R.; Li, Y. Comparative analysis of quality of service and memory usage for adaptive failure detectors in healthcare systems. *IEEE J. Sel. Areas Commun.* **2009**, *27*, 495–509. [CrossRef]
20. Xiong, N.; Jia, X.; Yang, L.T.; Vasilakos, A.V.; Li, Y.; Pan, Y. A distributed efficient flow control scheme for multi-rate multicast networks. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *21*, 1254–1266. [CrossRef]
21. Liu, Y.; Ota, K.; Zhang, K.; Ma, M.; Xiong, N.; Liu, A.; Long, J. QTSAC: An energy-efficient MAC protocol for delay minimization in wireless sensor networks. *IEEE Access* **2018**, *6*, 8273–8291. [CrossRef]
22. Peng, P.; Zou, L.; Chen, L.; Zhao, D. Processing SPARQL queries over distributed RDF graphs. *Vldb J. Int. J. Very Large Data Bases* **2016**, *25*, 243–268. [CrossRef]
23. Husain, M.; Mcglothlin, J.; Masud, M.M.; Khan, L.; Thuraisingham, B.M. Heuristics-Based query processing for large RDF graphs using cloud computing. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1312–1327. [CrossRef]

24. Papailiou, N.; Tsoumakos, D.; Konstantinou, I.; Karras, P.; Koziris, N. H_2 RDF+: An efficient data management system for big RDF graphs. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 22 June 2014.
25. Gao, J.; Lei, C.; Tian, L.; Ling, Y.; Chen, Z.; Song, B. Distributed Top-k subgraph matching in a big graph. In Proceedings of the 2018 IEEE International Conference on Big Data, Seattle, WA, USA, 10–13 December 2018; pp. 5325–5327.
26. Hose, K.; Schenkel, R. WARP: Workload-aware replication and partitioning for RDF. In Proceedings of the IEEE International Conference on Data Engineering Workshops, Brisbane, QLD, Australia, 8–12 April 2013; pp. 1–6.
27. Gurajada, S.; Seufert, S.; Miliaraki, I.; Theobald, M. TriAD: A Distributed Shared-Nothing RDF Engine Based on Asynchronous Message Passing. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 22 June 2014; pp. 289–300.
28. Lee, K.; Liu, L. Scaling queries over big RDF graphs with semantic hash partitioning. *Proc. Vldb Endow.* **2013**, *6*, 1894–1905. [CrossRef]
29. Schwarte, A.; Haase, P.; Hose, K.; Schenkel, R.; Schmidt, M. FedX: Optimization techniques for federated query processing on linked data. In Proceedings of the International Conference on the Semantic Web, Bonn, Germany, 23–27 October 2011; pp. 601–616.
30. Lin, B.; Guo, W.; Xiong, N.; Chen, G.; Vasilakos, A.V.; Zhang, H. A pretreatment workflow scheduling approach for big data applications in multi-cloud environments. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 1, [CrossRef]
31. Xiong, N.; Vasilakos, A.V.; Yang, L.T.; Wang, C.; Kannan, R.; Chang, C.; Pan, Y. A novel self-tuning feedback controller for active queue management supporting TCP flows. *Inf. Sci.* **2010**, *180*, 2249–2263. [CrossRef]
32. Nguyen, K. *Inverse Location Theory with Ordered Median Function and Other Extensions*; Epubli: Berlin, Germany, 2014.
33. He, H.; Singh, A.K. Query language and access methods for graph databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, 10–12 June 2008; pp. 405–418.
34. Lee, J.; Han, W.S.; Kasperovics, R.; Lee, J.H. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proc. Vldb Endow.* **2013**, *6*, 133–144. [CrossRef]
35. Sun, Z.; Wang, H.; Wang, H.; Shao, B.; Li, J. Efficient subgraph matching on billion node graphs. *Proc. Vldb Endow.* **2012**, *5*, 788–799. [CrossRef]
36. Huang, J.; Abadi, D. Leopard: lightweight edge-oriented partitioning and replication for dynamic graphs. *Proc. Vldb Endow.* **2016**, *9*, 540–551. [CrossRef]
37. Hall, B.H.; Jaffe, A.B.; Trajtenberg, M. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*; The MIT Press: Cambridge, MA, USA, 2001.
38. Chakrabarti, D.; Zhan, Y.; Faloutsos, C. R-MAT: A recursive model for graph mining. In Proceedings of the Siam International Conference on Data Mining, Lake Buena Vista, FL, USA, 22–24 April 2004.