World Wide Web (2023) 26:751–771
https://doi.org/10.1007/s11280-022-01075-6

Optimizing subgraph matching over distributed knowledge graphs using partial evaluation

Yanyan Song[1] · Yuzhou Qin[1] · Wenqi Hao[1] · Pengkai Liu[1] · Jianxin Li[2] · Farhana Murtaza Choudhury[3] · Xin Wang[1] · Qingpeng Zhang[4]

Received: 28 February 2022 / Revised: 12 May 2022 / Accepted: 2 June 2022 /
Published online: 8 July 2022

Abstract
The *partial evaluation and assembly* framework has recently been applied for processing subgraph matching queries over large-scale knowledge graphs in the distributed environment. The framework is implemented on the master-slave architecture, endowed with outstanding scalability. However, there are two drawbacks of partial evaluation: if the volume of intermediate results is large, a large number of repeated partial matches will be generated; and the assembly computation handled by the master would be a bottleneck. In this paper, we propose an optimal partial evaluation algorithm and a filter method to reduce partial matches by exploring the computing characteristics of partial evaluation and assembly framework. (1) An index structure named *inner boundary node index* (IBN-Index) is constructed to prune for graph exploration to improve the searching efficiency of the partial evaluation phase. (2) The boundary characteristics of local partial matches are utilized to construct a *boundary node index* (BN-Index) to reduce the number of local partial matches. (3) The experimental results over benchmark datasets show that our approach outperforms the state-of-the-art methods.

Keywords Partial evaluation · Subgraph matching · RDF graph

1 Introduction

Knowledge graphs have become the important cornerstone of the research and development of artificial intelligence technology. In recent years, the scale of knowledge graphs has increased at an unprecedented rate, and data processing with millions of vertices (10^6) and hundreds of millions of edges (10^8) has become commonplace [1]. Therefore, it is necessary to consider how to perform distributed query processing to cope with the growing demand for knowledge graphs.

This article belongs to the Topical Collection: *Special Issue on Web Information Systems Engineering 2021*
Guest Editors: Hua Wang, Wenjie Zhang, Lei Zou, and Zakaria Maamar

✉ Xin Wang
wangx@tju.edu.cn

Extended author information available on the last page of the article

In the Semantic Web community, the *Resource Description Framework* (RDF) has become a de-facto standard format for knowledge graphs and has been extensively applied [1]. An RDF dataset consists of a set of triples $\langle s, p, o \rangle$ and can be transformed into a graph where the resources denoted by s and o are vertices, and the attributes denoted by p are labeled edges. SPARQL [2] is the standard graph query language on RDF graphs. A SPARQL query can be regarded as the subgraph homomorphism problem [3], which is recognized as an NP-complete problem [4, 5].

The efficient processing of subgraph matching queries over large-scale RDF graphs in a distributed setting is a challenging problem. Recently, the *partial evaluation* technique [6] has been applied to solve the problem of regular path queries on distributed environment [7–9]. The queries Q are partially evaluated in parallel to obtain partial results on each fragment of data F_i on each site S_i, then all the partial results are transmitted to a master site. Finally, assemble these partial results to get the final results of Q.

Based on partial evaluation technique, the partial evaluation and assembly framework has been proposed to answer SPARQL queries [10]. However, a large number of partial results can be generated during the partial evaluation phase, making the assembly phase a computational bottleneck. To improve the efficiency of assembly phase, Peng et al. [11] proposed the LEC feature-based optimization strategy to prune some unpromising intermediate results. However, existing works only focus on the assembly phase of query processing, while ignoring the partial evaluation phase, and efficient index-based methods are not effectively utilized by the partial evaluation phase to speed up the search process. The following example demonstrates the drawback of computing partial matching results by the method in [11].

Example 1 As shown in Fig. 1a, given a distributed RDF graph G_0, and a query $Q = (?a, \text{spouse}, ?s) \wedge (?a, \text{director}, ?b) \wedge (?b, \text{country}, ?c) \wedge (?c, \text{capital}, ?d)$, the query engine traverses the dataset on F_i according to the query graph to obtain all the candidate sets of query variables. In fragment F_i, the candidate sets of query is denoted as B_{g_i}, where the subscript g_i is used to distinguish among different fragments. Based on the candidate generation strategy, the candidates of internal query nodes of query Q are $B_{g_1} = \{\langle ?a, (v_2, v_5)\rangle, \langle ?b, (v_3, v_4, v_{21})\rangle, \langle ?c, (v_8)\rangle\}$ and $B_{g_2} = \{\langle ?a, (v_{11}, v_{15}, v_{22})\rangle, \langle ?b, (v_{13}, v_{14})\rangle, \langle ?c, (v_9)\rangle\}$. The internal query nodes denote the query nodes that have more than one edge (the nodes

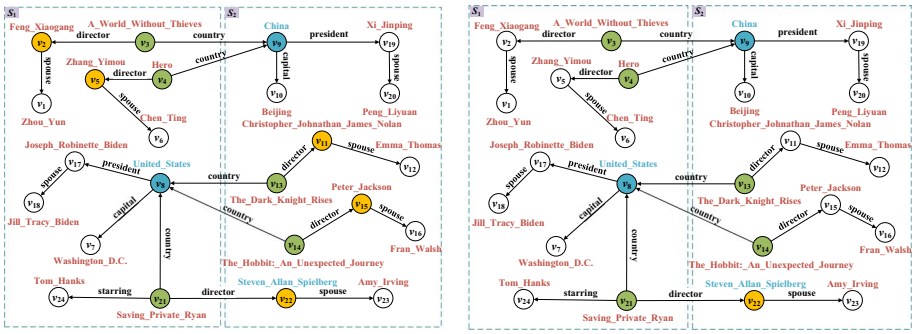

Fig. 1 A distributed RDF graph G_0 with candidates of query Q_0 colored. (**a**) Before filtering candidates, (**b**) After filtering candidates

?s and ?d are not internal query nodes because there only exists one edge connected with them in query graph Q). The candidate vertices are in yellow, green, and blue, respectively, in Fig. 1a. When computing local partial matches on each site, each candidate vertex will start a graph exploration, so that the search process will iterate six times on the first site and six times on the second. To further explore, we can find that a graph exploration from either v_2 or from v_3 can obtain the local partial match ($\langle ?s, v_1\rangle$, $\langle ?a, v_2\rangle$, $\langle ?b, v_3\rangle$, $\langle ?c, v_9\rangle$, $\langle ?d, v_{10}\rangle$) on F_1. As a result, this strategy can generate a large number of repeated local partial results, leading to a degradation in the performance of partial evaluation.

To handle this problem, we propose an effective optimization strategy to accelerate the partial evaluation phase by a constructed index named inner boundary node index which exploits characteristics of partial evaluation and assembly framework (the formal definition will be explained in detail in Section 4). For the example shown in Fig. 1, the following candidate sets are generated after filtered by IBN-Index: $B_{g_1} = \{\langle ?a, NULL\rangle, \langle ?b, (v_3, v_4, v_{21})\rangle, \langle ?c, (v_8)\rangle\}$, $B_{g_2} = \{\langle ?a, (v_{22})\rangle, \langle ?b, (v_{13}, v_{14})\rangle, \langle ?c, (v_9)\rangle\}$. These candidate vertices of ?a, ?b and ?c are colored yellow, green and blue respectively in Fig. 1b. The size of the whole candidate sets is much smaller than the candidates mentioned in Example 1. Furthermore, the local partial match ($\langle ?s, v_1\rangle$, $\langle ?a, v_2\rangle$, $\langle ?b, v_3\rangle$, $\langle ?c, v_9\rangle$, $\langle ?d, v_{10}\rangle$) will be only generated once which is started from v_3 on fragment F_1.

Since the growing number of partial matches heavily influence the assembly stage, filtering out part of the partial results becomes an effective way to speed up the query. Therefore, we propose another method that utilizes constructed boundary node index (the formal definition will be explained in detail in Section 5) to filter out part of the false local partial matches in advance, reducing the cost of communication and centralized computation.

We summarize the contributions of the paper with the following three aspects:

- Based on partial evaluation and assembly framework, we propose an inner boundary node index (IBN-Index) and a partial evaluation algorithm that utilizes IBN-Index to filter the candidate sets of query nodes, which can significantly speed up partial evaluation phase.
- To reduce the number of local partial matches, the boundary node index (BN-Index) is constructed by exploiting the characteristics of local partial matches. Furthermore, a BN-Index-based filter algorithm is proposed.
- Extensive experiments on benchmark datasets have been conducted to verify the efficiency and scalability of our method. The experimental results show that our method outperforms the state-of-the-art method.

The rest of this paper is organized as follows. Section 2 reviews the related work. In Section 3, we present the preliminaries and problem definition. An overview of the methods is depicted in Section 4. In Section 5 and Section 6, we propose the inner boundary node index and the boundary node index, with their corresponding algorithm, respectively. Finally, experimental evaluations are presented in Section 7 and we draw conclusions in Section 8.

2 Related work

Due to performance, confidentiality, and security factors [12, 13], the cluster-based distributed data management architecture has become the inevitable research trend to deal with the knowledge graph. In this section, we will review several distributed subgraph matching

research on large-scale RDF graphs, which can be classified into two categories, including MapReduce-based graph systems, and specialized RDF systems. Furthermore, existing works on partial evaluation and assembly and graph indexing methods are summarized.

2.1 MapReduce-based graph systems

SHARD [14], a MapReduce-based triple store for RDF graphs, is able to process SPARQL queries, which decomposes the query graph into a set of triples (a triple containing variables) and binds variables to the vertices of the data graph by iterating on the triple patterns. Meanwhile, it is necessary to satisfy all the constraints in the query. For example, each round of the MapReduce operation adds only one query clause through the join operation. Likewise, the smallest decomposition unit of the query graph in HadoopRDF [15] is also the triple pattern, and it also utilizes the MapReduce framework to divide the RDF triples into multiple small files based on the predicate. However, both methods mentioned above ignore the structural information of the query graph, require multiple MapReduce iterations, and require a large number of join transactions, resulting in high query cost.

2.2 Specialized RDF systems

Trinity.RDF [16], a distributed in-memory key-value store, stores RDF data in the native form, with vertex identifiers as keys and adjacent lists of vertices as values. Trinity.RDF finds the optimal exploration plan and reduces the number of intermediate results using the graph exploration instead of join operations, while the final results need to be obtained using a single thread on the master node. In addition, systems based on partial evaluation and assembly framework have also been extensively and deeply studied in recent years.

The method of partial evaluation and assembly is first applied in distributed XML data management by Peter et al. [17]. The key idea is to transmit the whole query graph to each site that is partially evaluated in parallel, and after each node computes the partial results of the query, the results are transmitted as compact Boolean functions to the master node, which are combined to obtain the result. Fan et al. [7, 18, 19] subsequently propose a series of algorithms based on partial evaluation and assembly framework to deal with XQuery on distributed XML data, reachability query and graph simulation on distributed graph. Peng et al. [10, 11] design a subgraph matching query algorithm based on partial evaluation and assembly framework to process SPARQL queries on distributed RDF data and propose relevant optimization strategies.

However, the huge overhead caused by repeated partial matches during partial evaluation is not handled in aforementioned methods. Furthermore, when the number of intermediate results obtained in the local computation phase is extensive, the aforementioned methods may suffer from a performance bottleneck in the assembly phase.

2.3 Graph indexing

As a classic space-for-time strategy, graph indexing has been researched extensively and deeply in the past years. Graph indexing methods can be classified into value-based indexing and structure-based indexing.

Through value-based methods, a graph index is usually constructed on one or more properties of an entity. SB-Tree [20] is a variant of B-Tree, which has a better performance on dynamic data. Hexastore [21] and RDF-3X [22] index the RDF data in six possible ways. Based on S-Tree [23], gStore [24] proposed VS-Tree to prune the search space efficiently.

Structure-based methods focus on mining the features of a graph, such as a path, subtree, or other substructure, indexing them to filter the search space and accelerate the query process. Closure-Tree [25] proposed graph closure, which is a generalized graph that represents several graphs, and based on it to organize graphs as a tree. Both subgraph queries and similarity queries can benefit from this method. SPath [26] takes the shortest paths around the vertex neighborhood as the basic process unit and decomposes the query into a set of shortest paths to exploit its indexing. K-path-bisimulation [27] is a path-based index, the path with identical length and the same edge label sequence is divided into the same catalog, and the query is decomposed into a set of paths.

In this paper, the features of partial evaluation and assembly framework are exploited profoundly, and two kinds of indexes are constructed to compensate for the shortcomings of previous methods. Although the proposed index-based methods are value-based, the structural information of the graph is also included in the indexes.

3 Preliminaries

Let U and L be the disjoint infinite sets of URIs and literals. Then, a tuple in the form of $\langle s, p, o\rangle \in U \times U \times (U \cup L)$ is called an RDF triple, where s is the subject, p the predicate, and o the object. Given an RDF dataset as a finite set of triples, it can be converted to its corresponding RDF graph. In this paper, we focus on the problem of subgraph matching query over a distributed RDF graph. This section will present preliminaries for distributed RDF graphs and subgraph matching queries. Table 1 lists the notations frequently used in this paper.

Definition 1 (RDF Graph) Given an RDF dataset as a finite set of triples in the form $\langle s, p, o\rangle$, its corresponding RDF graph is $G = (V, E, \Sigma)$, where the set of vertices V is the union of all s and o. For each $\langle s, p, o\rangle$, there is a directed edge $e \in E$ from the vertex s to the vertex o, where p is the label of that edge e. Here, Σ is the set of all labels, i.e., $\Sigma = \{p \mid \langle s, p, o\rangle \in G\}$.

Table 1 Frequently used notations

Notation	Description	Notation	Description
Q and G	Query graph and data graph	V_i^e	The set of boundary nodes on F_i
V, E, Σ	Vertex set, edge set, label set	S_i	Computing node
F_i	A fragment of graph G	P_Q	The set of results of query Q
$\mathcal{E}_i$	A disjoint entity set of graph G	PM_i	A local partial match on F_i
N_i^c	The rest of F_i except $\mathcal{E}_i$	I_i^{IBN}	IBN-Index on F_i
D_i	Inner boundary node set of F_i	I_i^{OUT}, I_i^{IN}	BN-Index on F_i

Definition 2 (Distributed RDF Graph) RDF graph G is partitioned into n disjoint 'entity sets' $\{\mathcal{E}_1, ..., \mathcal{E}_n\}$, where each $\mathcal{E}_i = (V_i, E_i, \Sigma_i)$. Here, (1) for each $i \in \{1,...,n\}$, $\mathcal{E}_i$ is a subset of G, where $V_i \subseteq V$, $E_i \subseteq E$, and $\Sigma_i \subseteq \Sigma$; (2) for each $i, j \in \{1,...,n\} \wedge i \neq j$, there is $\mathcal{E}_i \cap \mathcal{E}_j = \varnothing$; and (3) $\bigcup_{i=1}^{n} \mathcal{E}_i = G$.

To ensure the integrity and consistency of the RDF graph when partitioned in a distributed system, each computing node needs to store some copies of the edges that cross between different entity sets. Let the copy of the associated edges with other partition be denoted as N_i^c.

Definition 3 (Fragment) Graph G is partitioned into n fragments $\mathcal{F} = \{F_1, ..., F_n\}$, such that $F_i = \mathcal{E}_i \cup N_i^c$. In other words, G can be considered as a distributed RDF graph w.r.t. $\mathcal{F}$, such that:

1) For each $\mathcal{E}_i = (V_i, E_i, \Sigma_i)$, V_i, E_i, and Σ_i represent the set of internal vertices, the set of edges, and the set of labels in F_i, respectively. Formally, $V_i = \{s \mid \langle s, p, o\rangle \in \mathcal{E}_i\} \cup \{o \mid \langle s, p, o\rangle \in \mathcal{E}_i\}$, $E_i \subseteq V_i \times V_i$, and $\Sigma_i = \{p \mid \langle s, p, o\rangle \in \mathcal{E}_i\}$;
2) $N_i^c = (V_i^e, E_i^c, \Sigma_i^c)$, where E_i^c is the set of crossing edges between F_i and other fragments. If an internal vertex of F_i has a direct edge with any vertex v in F_j, where $i \neq j$, then $v \in V_i^e$. Formally, $E_i^c \subseteq V_i \times V_j$, $\Sigma_i^c = \{p \mid \langle s, p, o\rangle \in E_i^c\}$, the set of boundary vertices between F_i and F_j is $V_i^e = \{s \mid \langle s, p, o\rangle \in E_i^c \wedge s \in V_j\} \cup \{o \mid \langle s, p, o\rangle \in E_i^c \wedge o \in V_j\}$, $i, j = 1,2,...,n \wedge i \neq j$;

Let $\mathcal{S} = \{S_0, S_1, \cdots, S_n\}$ be a set of $n+1$ computing nodes, i.e., *sites*, in a cluster. Without loss of generality, each fragment F_i is stored at a slave site S_i for $i \in \{1, \cdots, n\}$.

Example 2 As shown in Fig. 2, given a distributed RDF graph G_1 extracted from the DBpedia dataset, G_1 can be divided into four parts $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$, which are respectively

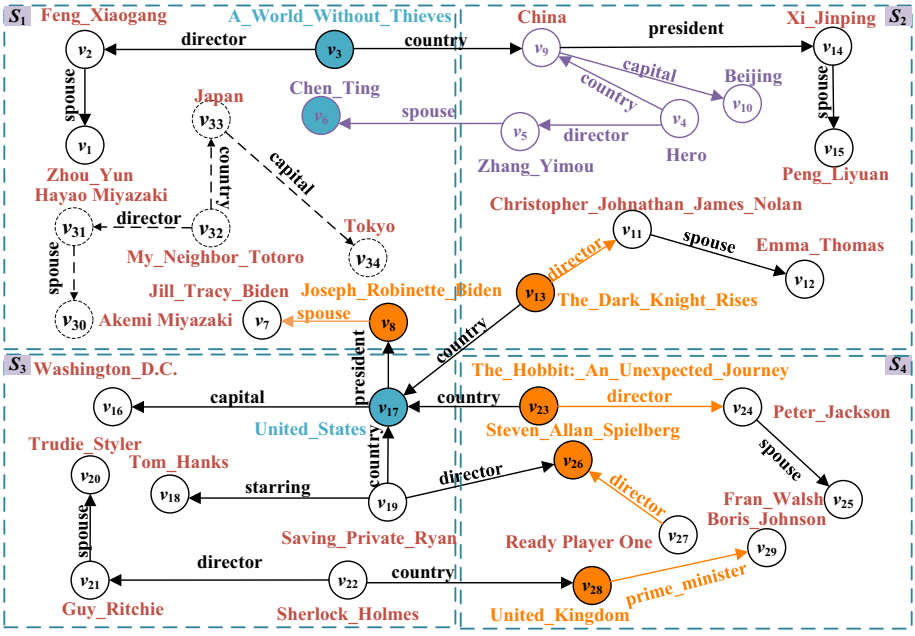

Fig. 2 A distributed RDF graph G_1

stored on the corresponding sites $\{S_1, S_2, S_3, S_4\}$ in the cluster. For a fragment $F_2 = \mathcal{E}_2 \cup N_2^c$, the partition $\mathcal{E}_2 = (V_2, E_2, \Sigma_2)$, and $V_2 = \{v_4, v_5, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}$, $E_2 = \{(v_4, v_5), (v_9, v_{10}), (v_9, v_{14}), (v_{14}, v_{15}), (v_{13}, v_{11}), (v_{11}, v_{12})\}$. The copy between F_2 and other fragments $N_2^c = (V_2^e, E_2^c, \Sigma_2^c)$, where $V_2^e = \{v_3, v_6, v_{17}\}$, $E_2^c = \{(v_3, v_9), (v_5, v_6), (v_{13}, v_{17})\}$. In particular, we colored the *incoming* and *outgoing* vertices of fragment F_2 in blue.

Given an RDF graph G and a query graph Q as a set of triple patterns, a subgraph matching problem is to find the subgraphs over G that satisfy all the triple patterns in Q. Such a subgraph matching problem is a conjunctive query (CQ) on G, which is the focus of this paper. In the following, we formally present the query graphs and the other necessary definitions adapted from [28].

A query graph includes m triple patterns $\langle s_r, p_r, o_r \rangle$, where the value of each s_r, o_r can either be a member of V, or 'not labeled'. If a s_r or o_r is 'not labeled', s_r or o_r belongs to a special set *Var*, and the name of each element in *Var* starts with the character '?'. Similarly, the value of each p_r can either be a member of Σ, or *Var*.

Definition 4 (Query Graph) Given an RDF graph G, a CQ Q over G is defined as: $Q(z_1, \cdots, z_t) \leftarrow \bigwedge_{1 \le r \le m} tp_r$, where $tp_r = \langle s_r, p_r, o_r \rangle$ is a triple pattern. $s_r, o_r \in V \cup Var$, z_l is a variable and $z_l \in \{s_r | 1 \le r \le m\} \cup \{o_r | 1 \le r \le m\}$. A CQ Q is also referred to as a query graph.

Before defining subgraph matching, we recapitulate certain definitions of the *mapping*. For a mapping μ, $dom(\mu)$ is its domain. Two mappings μ_1 and μ_2 are compatible, i.e., $\mu_1 \sim \mu_2$, iff for every element $v \in dom(\mu_1) \cap dom(\mu_2)$, it holds that $\mu_1(v) = \mu_2(v)$. Furthermore, the set-union of two compatible mappings, i.e., $\mu_1 \cup \mu_2$, is also a mapping.

Definition 5 (Subgraph Matching) The semantics of a CQ Q over an RDF graph G is defined as:

1) μ is a mapping from the vertices in Q to the vertices in V, i.e., mapping from $\overline{s} = \{s_1, \ldots s_m\}$ and $\overline{o} = \{o_1, \ldots o_m\}$ to the vertices in V;
2) $(G, \mu) \vDash Q$ iff $\langle \mu(s_r), \mu(p_r), \mu(o_r) \rangle \in E$ and the labels of s_r, p_r and o_r are the same as that of $\mu(s_r)$, $\mu(p_r)$, and $\mu(o_r)$, respectively, if $s_r, p_r, o_r \notin Var$;
3) P_Q is the set of all results, where each result satisfies the subgraph matching query Q over G.

Problem statement Consider a distributed RDF graph G, w.r.t., a fragmentation $\mathcal{F} = \{F_1, \ldots, F_n\}$, and let F_i be stored in the cluster $\mathcal{S} = \{S_0, S_1, \ldots S_n\}$. For simplicity, we assume that each site S_i hosts one fragment F_i. Given a query graph Q, the problem is to find all subgraph matching results P_Q of Q in G.

Example 3 Given a CQ, $Q = (?a, \text{spouse}, ?s) \wedge (?a, \text{director}, ?b) \wedge (?b, \text{country}, ?c) \wedge (?c, \text{capital}, ?d)$. Q consists of five query vertices and its semantic is to find the films directed by a person with his spouse, the film's country and the capital of the country. The corresponding query graph is shown in Fig. 3, with one of the query results being highlighted in purple in Fig. 2.

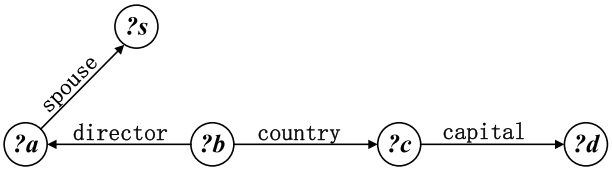

Fig. 3 A query graph Q with 5 query variables

4 Overview

The partial evaluation and assembly framework is extended to answer SPARQL queries over a distributed RDF graph G, as shown in Fig. 4. In the execution model, there are two phases: the partial evaluation phase and the assembly phase. In addition, two optimization strategies are designed and embedded in this framework.

Before the query starts, the entire RDF graph G is divided into multiple fragments according to a certain partitioning strategy, and an index named BN-Index is built on each fragment. The fragments and corresponding BN-Index are then transmitted to each site, and an index named IBN-Index is further constructed on each site locally. When querying, the master node sends the entire query graph to all slave nodes, and the subsequent partial evaluation phase can be summarized into three processes. (1) Each site S_i first receives the complete query graph Q and finds all candidate sets of the query graph variables. (2) The query engine uses the IBN-Index to filter the candidates, and executes the graph exploration algorithm according to the filtered candidates to find local matches. (3) Finally, BN-Index is utilized to filter the local partial matches after graph exploration.

The local partial matches are then sent to the master site to compute the complete SPARQL matches, which is called the assembly stage. Benefiting from the filtering effect of BN-Index, the number of partial matches is drastically reduced, which alleviates the assembly bottleneck problem to a certain extent.

To better illustrate the effect of IBN-Index, it is necessary to explain the partial evaluation process of gStoreD in detail. After each site S_i receives the complete query graph, the candidate set of each query variable is obtained according to the predicates connected with the variables. Specifically, the vertices in the candidate set can be classified into internal candidate vertices and boundary candidate vertices. The internal candidate vertices denote the vertices contained in the subgraph F_i allocated on S_i, while the boundary candidate vertices denote those vertices connected with F_i but do not belong to it. After obtaining the

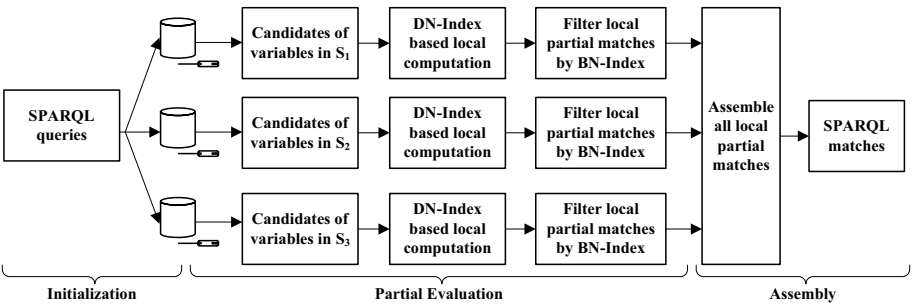

Fig. 4 Overview of the optimized partial evaluation and assembly method

candidate sets, all the sites transmit the internal candidate vertices to the master site, and the collection of all internal candidates is resent to all sites.

In order to find partial results on F_i, graph exploration starts with each internal candidate vertex of the internal query variables. Specifically, for a query graph, the query variables can be classified into internal query variables and satellite variables, depending on the edges connected with the query node. If a query node only has one edge, it is denoted as a satellite variable.

The reasons why we choose these special candidate vertices as the starting vertices are considered from two aspects. (1) A boundary vertex is also an internal vertex on another site simultaneously so that it will be set as a starting vertex on that site, and the path connected with it will not be lost. (2) If a partial match on S_i only matches a satellite node, it will also be found on other sites. Take the partial match in purple on S_1 in Fig. 2 as an example, the partial match will be found on S_2 and connected with v_6 as a complete SPARQL match. Therefore, only starting with internal query variables will not lose any partial results.

At each expansion step of graph exploration, query engine judges whether the matching node belongs to the collection of all the internal candidates. The graph exploration process will not stop until all maximal partial matches are obtained.

5 Inner boundary node-based algorithm

Recall that in partial evaluation and assembly framework, given a distributed RDF graph G, each site S_i receives a part of the graph F_i and constructs IBN-Index according to the subgraph. Then, when answering query Q, each site S_i computes local partial matches utilizing the constructed index. In this section, the structure of the IBN-Index is defined, and the IBN-Index construction algorithm is introduced. Then we present the subgraph matching algorithm utilizing IBN-Index on each site and give the complexity analysis of the proposed method.

5.1 Inner boundary node

The local partial match computation algorithm based on partial evaluation and assembly framework have been proposed in [10]. First, an in-depth analysis of the performance problem of existing work in computing local matches during partial evaluation is carried out, and on this basis, an optimization using the inner boundary node index is proposed.

When computing local partial matches, each internal candidate vertex of the candidate vertices sets starts the graph exploration to find the local partial matches. It should be noted that it is not necessary to traverse all internal nodes as the starting point of graph exploration. The reasons can be considered from the following two aspects.

(1) Intuitively, a complete SPARQL match only needs to be traversed from one vertex so that the candidate vertices in other candidate sets can be discarded. (2) Besides, as mentioned in [10], a local partial match is the overlapping part of an unknown crossing match and a fragment F_i. There must be a crossing edge derived from an internal vertex connected with other fragments. Therefore, only searching from the vertices connected with other fragments can get all the partial matches. These kinds of vertices are defined as inner boundary nodes, and the formal definition is given as follows:

Definition 6 (Inner Boundary Node.) Given a distributed RDF graph Q and a fragmentation $\mathcal{F} = \{F_1,...,F_n\}$. In fragment $F_i = \mathcal{E}_i \cup N_i^c$, if an internal vertex v of F_i has a direct edge with any vertex u in F_j, where $i \neq j$, then v is an inner boundary node.

Based on inner boundary nodes, the internal entity set V_i of fragment F_i can be divided into two mutually exclusive subsets, pure internal node set P_i and inner boundary node set D_i. Formally, $V_i = P_i \cup D_i$, where $D_i = \{s \mid \langle s,p,o\rangle \in E_i^c \wedge o \in V_j\} \cup \{o \mid \langle s,p,o\rangle \in E_i^c \wedge s \in V_j\}$, $i,j = 1,2,...,n \wedge i \neq j$. The definition of the inner boundary node index is presented as follows:

Definition 7 (Inner Boundary Node Index.) Given a fragment F_i of RDF graph G, the inner boundary node index, i,e., IBN-Index of F_i is a key-value map I^{IBN} where

1) for any tuple $(v, tag) \in I^{IBN}$, the key is a vertex $v \in V_i$, and the value tag is a Boolean value denoting if v is an inner boundary node or not; if a vertex v is an inner boundary node, its corresponding *tag* will be set to *True*, otherwise it will be set to *False*;
2) for any vertex $v \in V_i$, there exists a unique tuple in I^{IBN} with v as the key and a Boolean *tag* as the value.

Example 4 As shown in Fig. 2, on site S_1, the inner vertices (inner boundary nodes) connected with vertices on other sites (S_2 and S_3) are v_3, v_6, and v_8. And on site S_2, the inner boundary nodes are v_9, v_5, and v_{13}. As a result, the inner boundary node index on site s_1 is $I_1^{IBN} = \{(v_3, True), (v_6, True), (v_8, True), (v_1, False), (v_2, False), (v_7, False), (v_{30}, False), (v_{31}, False), (v_{31}, False), (v_{32}, False), (v_{33}, False), (v_{34}, False)\}$. And the IBN-Indexes on site S_2 is $I_2^{IBN} = \{(v_5, True), (v_9, True), (v_{13}, True), (v_4, False), (v_{10}, False), (v_{11}, False), (v_{12}, False), (v_{14}, False), (v_{15}, False)\}$.

Algorithm 1 Constructing inner boundary node index.

Input: Fragment F_i
Output: The set of IBN-Index in F_i, denoted as I_i^{IBN}

```
1  I_i^IBN ← ∅, D_i ← ∅;
   // initialize the set of I_i^IBN and D_i
2  for each ⟨s, p, o⟩ ∈ F_i do
3      if ⟨s, p, o⟩ ∈ E_i^c ∧ s ∈ V_i (or o ∈ V_i) then
4          D_i ← D_i ∪ {s} (or D_i ← D_i ∪ {o});
5  for each v ∈ V_i do // add index for each inner boundary node
6      if v ∈ D_i (or v ∉ D_i) then
7          I_i^IBN.put(v, True) (or I_i^IBN.put(v, False));
8  return I_i^IBN;
```

To improve space efficiency of IBN-Index, the strategy of dictionary encoding is adopted, such that each vertex is encoded into an integer by hash operation.

The construction approach of the IBN-Index is shown in Algorithm 1. First, the inner boundary node set and IBN-Index are initialized by the fragment identifier F_i (line 1). Then, for each triple $\langle s, p, o \rangle$, if it is a crossing edge and the subject s (or object o) is an internal vertex, the s (or o) is put into the inner boundary node set D_i (lines 2-6). For each node v in the internal node set V_i, if it is also an inner boundary node in D_i, a mapping (v, *True*) will be put into I_i^{IBN}; otherwise, a mapping (v, *False*) will be put into I_i^{IBN} (lines 7-11).

5.2 IBN-index based partial evaluation

In order to answer query Q, each site S_i computes the local partial matches based on the known fragment F_i. The formal definition of local partial match was defined in [10]. Intuitively, a local partial match PM_i is an overlapping part between a crossing match M and fragment F_i.

Algorithm 2 Computing local partial matches.

Input: A fragment F_i, a query graph Q and IBN-Index I_i^{IBN}
Output: The set of all local maximal partial matches in F_i, denoted as $\Omega(F_i)$

1 $\mathcal{B} \leftarrow \emptyset$, $\Omega(F_i) \leftarrow \emptyset$;
// initialize candidate set and result set
2 **for** *each* $v \in Var$ **do**
3 $B \leftarrow$ GenerateCandidatesSet(v);
4 $\mathcal{B} \leftarrow \mathcal{B} \cup \{B\}$;
// generate candidate set for each variable node
5 $B_{min} \leftarrow$ SelectMinimalSize($\mathcal{B}$);
6 $\mathcal{B} \leftarrow \mathcal{B} \setminus \{B_{min}\}$;
7 **for** *each* $u \in B_{min}$ **do**
8 $f \leftarrow$ InitializeFunction($\langle v, u \rangle$);
9 ComParMatch(f);
// handle B with the minimal size and delete it from $\mathcal{B}$
10 **for** *each* $B \in \mathcal{B}$ **do**
11 **for** *each* $u \in B$ **do**
12 **if** $I_i^{IBN}.valueOf(u)$ **then**
13 InitializeFunction($\langle v, u \rangle$);
14 ComParMatch(f);
// for each inner boundary node, start the graph exploration process
15 $\mathcal{B} \leftarrow \mathcal{B} \setminus \{B\}$;
16 **return** $\Omega(F_i)$;

Algorithm 2 describes the local partial match computation process utilizing the IBN-Index. The key idea is to use the inner boundary node index to filter the candidate sets, thereby reducing the search space when finding local partial results. Since the result of partial evaluation can be divided into complete SPARQL matches and local partial matches, the correctness of Algorithm 2 can be considered from the following two aspects. (1) Since

some complete SPARQL matches contain only pure internal vertices (e.g., the dashed partial match on S_1 in Fig. 2), i.e. they do not have any inner boundary nodes, in order to ensure that all complete SPARQL matches are obtained, we keep a complete candidate set in which all vertices will start graph exploration (line 8). Here a greedy strategy is applied, selecting the candidate set with the smallest size as the reserved set (line 6). (2) In other candidate sets, only the candidate vertex judged as an inner boundary node will start the graph exploration (lines 13-19).

Function 3 ComParMatch(f).

```
1  if ∀v ∈ Var ∧ HaveMachedInFunction(v, f) then
2      return;
       // return if all vertices of query Q have been matched
          in the function f
3  v' ← SelectUnmatchedAjacent(v, Q);
   // Select an unmatched v' adjacent to a matched vertex v
      in the function f
4  for each u' ∈ GetCandidate(v') do
5      f' ← f ∪ ⟨v', u'⟩;
6      if FailToMatch(f', Q) then
7          continue;
8      if NotAMaximalMatch(f', Q) then
9          ComParMatch(f');
           // explore further if f' is not a maximal local
              partial match
10     else
11         Ω(F_i) ← Ω(F_i) ∪ {f};
           // f specifies a local partial match that will be
              inserted into the answer set Ω(F_i)
```

Example 5 We take site S_1 as an example. As shown in Fig. 2, for query Q, the candidate sets of each internal query variables on site S_1 are $B_{g1} = \{\langle ?a, (v_2, v_{31}, v_{v_8})\rangle, \langle ?b, (v_3, v_{32})\rangle, \langle ?c, (v_{33})\rangle\}$. Firstly, the candidate sets are sorted to find the candidate set with the smallest size and the set $\langle ?c, (v_{33})\rangle$ is reserved. In the candidate set of $?a$, the nodes v_2 and v_{31} are not inner boundary nodes according to the IBN-Index, so they are all filtered out. As for the candidate set of $?b$, v_3 will not be filtered, while v_{32} will be discarded. The filtered set of candidate sets is $B_{g1} = \{\langle ?a, (v_8)\rangle, \langle ?b, (v_3)\rangle, \langle ?c, (v_{33})\rangle\}$. As a result, graph exploration can find all partial matches on S_1 starting only from v_8, v_3, and v_{33}. Likewise, the candidate sets on other sites are also filtered out of a large number of candidate vertices using the same method.

Space complexity of IBN-Index For each fragment F_i, each vertex corresponds to a tag indicating whether it is an inner boundary node. The extra space is bounded with $O(|V_i| + |V_i^e|)$, where V_i is the set of internal nodes of fragment F_i and V_i^e is the set of boundary nodes of fragment F_i.

6 Filter local partial matches with boundary node index

As shown in Fig. 5, since the RDF data graph is distributed and stored on multiple sites, the boundary node on each site becomes a bridge connecting any two sites. Node 1 (in blue) on F_1 is a boundary node for F_2, while it is an internal node in the view of F_1. As a result, it is named an inner boundary node on F_1, while it is a boundary node for F_2. In partial evaluation and assembly framework, after the local partial matches are obtained, all of them are sent to the master site uniformly. However, not all the partial results can continue to be joined to form a complete SPARQL match on the master site. Therefore, it is unnecessary to transmit local partial matches, which are not associated with partial matches on other sites, to the master site. Based on the above problem, this paper proposes an optimization strategy for pre-judging edge labels from boundary nodes to reduce the communication overhead between local sites and the master site.

The definition of boundary nodes (see the definition of V_i^e) has been given in Section 3, which means vertices that belong to other fragments but are directly connected to internal vertices in F_i. For an RDF graph G, when dividing the data, we record each boundary node's out-edge and in-edge information on the fragment F_i as boundary node index. The formal definition of boundary node index is as follows:

Definition 8 (Boundary Node Index) . Given a fragment F_i of RDF graph G, the boundary node index, i,e., BN-Index of F_i is a key-value map I_i^{BN} where

1) $I_i^{BN} = I_i^{Out} \cup I_i^{In}$;
2) for any tuple $(v, v.Out) \in I_i^{Out}$, the key is a vertex $v \in V_i^e$, and the value $v.Out = \{(p_1, p_2, ..., p_n) \mid i \neq j \wedge \langle v, p_l, u\rangle \in F_j \wedge l \in \{1,...,n\}\}$;
3) for any tuple $(v, v.In) \in I_i^{In}$, the key is a vertex $v \in V_i^e$, and the value $v.In = \{(p_1, p_2, ..., p_n) \mid i \neq j \wedge \langle u, p_l, v\rangle \in F_j \wedge l \in \{1,...,n\}\}$.

Algorithm 4 Constructing boundary node index.

Input: RDF graph G, fragmentation $\mathcal{F} = \{F_1, ..., F_n\}$
Output: The set of BN-Index in F_i, denoted as I_i^{Out} and I_i^{In}

```
1 I_i^Out ← ∅, I_i^In ← ∅;
2 for each ⟨s, p, o⟩ ∈ G do
3     if s ∈ V_i^c ∧ ⟨s, p, o⟩ ∈ F_j ∧ i ≠ j then
4         I_i^Out.put(s, p);
5     if o ∈ V_i^c ∧ ⟨s, p, o⟩ ∈ F_j ∧ i ≠ j then
6         I_i^In.put(o, p);
7 return I_i^Out, I_i^In;
```

The construction approach of boundary node index is shown in Algorithm 4. First, the BN-Index is initialized by the identifier F_i (line 1). Then, for each triple $\langle s, p, o\rangle$ in RDF graph G, if s (or o) is a boundary node in F_i and the triple $\langle s, p, o\rangle \notin F$, the the (s, p) (or (o, p)) will be put into I_i^{Out} (or I_i^{In}) (lines 2-8). Algorithm 4 will iterate over each triple until there is no triple left.

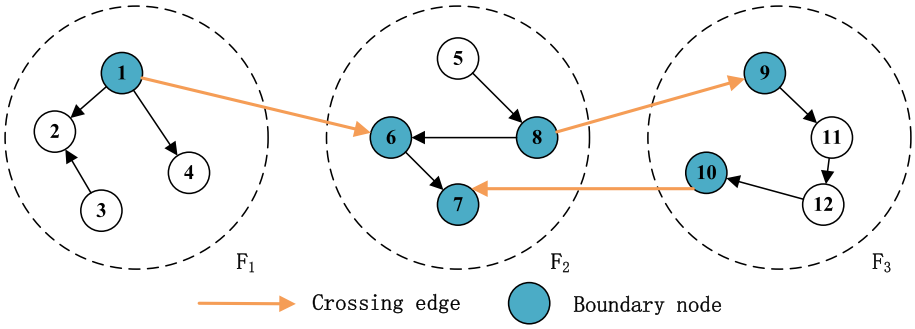

Fig. 5 Fragmentation of a data graph

Example 6 As shown in Fig. 2, on site S_3, the boundary nodes (with their predicates in orange) are $V_i^e = \{v_8, v_{13}, v_{23}, v_{26}, v_{28}\}$. And the corresponding boundary node index is $I_3^{Out} = \{\langle v_8, (spouse)\rangle, \langle v_{13}, (director)\rangle, \langle v_{23}, (director)\rangle, \langle v_{28}, (prime\_minister)\rangle, \langle v_{26}, NULL\rangle\}$, $I_3^{In} = \{\langle v_{26}, (director)\rangle, \langle v_8, NULL\rangle, \langle v_{13}, NULL\rangle, \langle v_{23}, NULL\rangle, \langle v_{28}, NULL\rangle\}$.

Example 7 As shown in Fig. 6, the local partial matches on S_3 is $PM_3 = \{(\langle ?s, null\rangle, \langle ?a, null\rangle, \langle ?b, v_{13}\rangle, \langle ?c, v_{17}\rangle, \langle ?d, v_{16}\rangle), ((\langle ?s, null\rangle, \langle ?a, null\rangle, \langle ?b, v_{23}\rangle, \langle ?c, v_{17}\rangle, \langle ?d, v_{16}\rangle), (\langle ?s, null\rangle, \langle ?a, v_{26}\rangle, \langle ?b, v_{19}\rangle, \langle ?c, v_{17}\rangle, \langle ?d, v_{16}\rangle), (\langle ?s, v_{20}\rangle, \langle ?a, v_{21}\rangle, \langle ?b, v_{22}\rangle, \langle ?c, v_{28}\rangle, \langle ?d, null\rangle))\}$. According to the boundary node index, the last two local partial matches could not constitute any final results. Then these two matching pairs will not be sent to the master node as a message.

The filtering partial matches process is presented briefly in Algorithm 5. On site S_i, each partial match is iterated to judge on the boundary nodes (line 3). Only when the value (also known as the predicates belong to other fragments) of BN-Index of the boundary node contains the unmatched predicates on the query graph, can the partial match be further matched and reserved to transmit to the master site.

Algorithm 5 Filtering local partial matches.

Input: The set of partial matching results in fragment F_i, denoted as $P_{Q,i}$
Output: The set of partial matching results in fragment F_i after filter

1 **for** *each* $\mu_p \in P_{Q,i}$ **do**
2 **for** *each* $\mu_p(v_r) \in \mu_p$ **do**
3 **if** $\mu_p(v_r) \in V_i^I \wedge \exists\langle v_r, p_r, o_r\rangle \in Q \wedge p_r \notin I_i^{Out}$.`valueOf`$(\mu_p(v_r))$ **then**
4 $P_{Q,i} \leftarrow P_{Q,i} \setminus \{\mu_p\}$;
5 **if** $\mu_p(v_r) \in V_i^I \wedge \exists\langle s_r, p_r, v_r\rangle \in Q \wedge p_r \notin I_i^{In}$.`valueOf`$(\mu_p(v_r))$ **then**
6 $P_{Q,i} \leftarrow P_{Q,i} \setminus \{\mu_p\}$;
7 **return** $P_{Q,i}$;

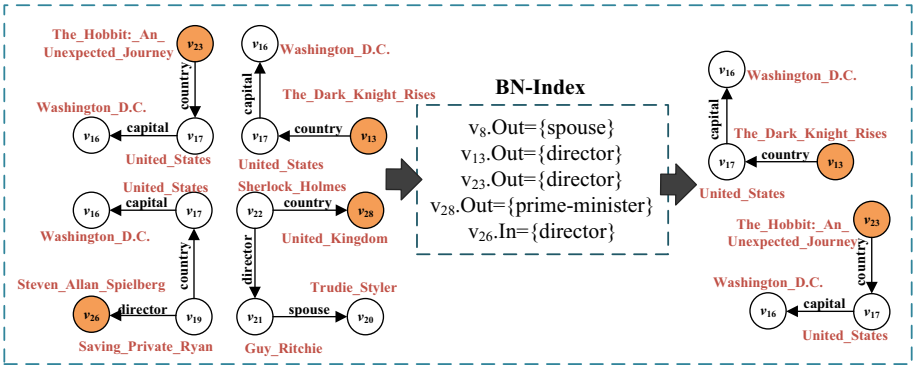

Fig. 6 Filtering partial matches with BN-index on site S_3 on graph G_1

Space complexity of BN-Index For each fragment F_i, the number of the BN-Index is $O(|V_i^e|)$ at most. As a result, the extra space of BN-Index is bounded with $O(|V_i^e|)$, where V_i^e is the set of crossing edges.

7 Experimental evaluation

In order to verify the effectiveness and efficiency of the IBN-Index method and BN-Index filtering method under the partial evaluation and assembly framework, a comparative experiment with gStoreD [10, 11] was performed over several benchmark RDF datasets. The proposed algorithm is implemented on top of gStoreD, and is deployed on a 3-node cluster, of which each node is in a Docker. The three dockers are all deployed on a machine with 16 cores Intel Xeon Silver 4216 2.10 GHz processors, 512 GB of RAM, and 1.92 TB SSD, running the 64-bit CentOS 7.7 operating system.

7.1 Datasets and queries

In this experiment, the proposed method and gStoreD are evaluated using the LUBM [29] synthetic benchmark dataset of different scales. The statistics of the datasets are shown in Table 2. We need to compare the query efficiency of the method based on IBN-Index, the method based on BN-Index and the combination of the two, and the original gStoreD on different queries. It is necessary to gradually change the number of intermediate results that the query satisfies while limiting the basic structure of the query. Therefore, we choose benchmark datasets rather than real-world datasets to keep the dataset size positively correlated with the number of intermediate results. In addition, to eliminate the impact of the partitioning strategy on query performance, we use a random partitioning method to divide each dataset into four fragments.

As shown in Table 3, eight complex queries of different scales on the LUBM dataset are presented, i.e., $Q_1 \sim Q_8$. To exhibit the effect of IBN-Index and BN-Index, the proposed queries may generate large amount of intermediate results in the partial evaluation phase, as showed in Fig. 7a.

Table 2 Datesets

Dateset	#edges	#vertices
LUBM10	1,316,700	314,852
LUBM20	2,782,126	663,647
LUBM30	3,890,992	841,108

7.2 Experimental results

Table 3 Queries

#Query vertices	Queries
4	Q_1, Q_2
5	Q_3, Q_4
6	Q_5, Q_6
7	Q_7, Q_8

To verify the effectiveness of the IBN-Index and BN-Index based partial evaluation algorithm, extensive experiments were conducted.

Exp 1. Number of partial matching results. To make it more intuitive to observe and evaluate the performance of the partial evaluation algorithm with different queries and datasets, the number of partial matching results and complete SPARQL matches of $Q_1 \sim Q_8$ over LUBM10, LUBM20, and LUBM30 is recorded. The maximal total number of all partial evaluation results (including complete SPARQL matches and local partial matches) generated from each site are depicted in Fig. 7a, which determines the time consumption of the partial evaluation and influences assembly phases. As shown in Fig. 7a, for Q_1, the number of partial evaluation results increases approximately linearly with the size of the datasets. For the query $Q_2 \sim Q_8$, their partial evaluation results on LUBM20 are the most, which is affected by the partitioning strategy.

Exp 2. The construction time and space occupied by IBN-Index and BN-Index. Figure 8a shows the largest time overhead and space occupation of the IBN-Index among all slave nodes on different datasets. It can be observed that the time to construct the IBN-Index and the size of the IBN-Index are positively correlated with the size of the graph. Figure 8b presents the construction time and space of the BN-Index, which have similar trends to that of the IBN-Index.

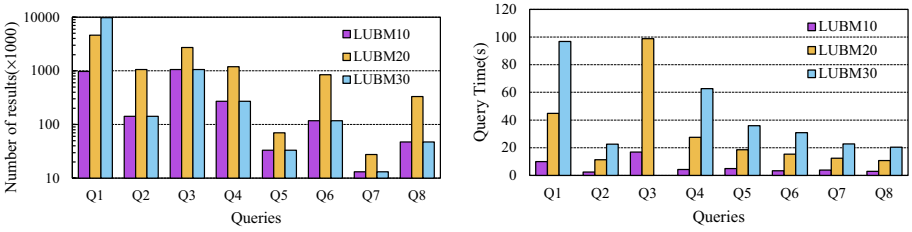

Fig. 7 Query evaluation on LUBM datasets. (**a**) Partial evaluation results on LUBM datasets, (**b**) Query evaluation time of Q1 ~ Q8

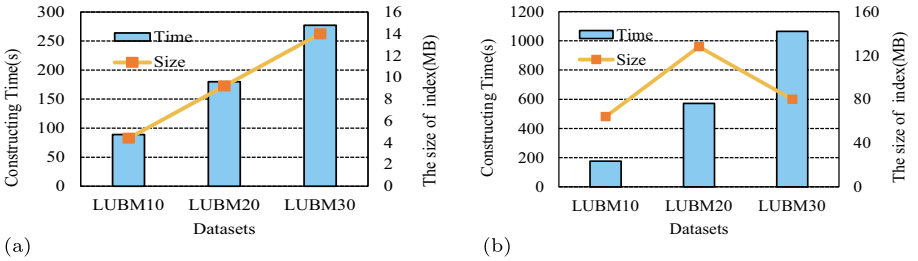

Fig. 8 The constructing time and size of IBN-Index and BN-Index. (**a**) The constructing time and size of IBN-Index, (**b**) The constructing time and size of BN-Index

For IBN-Index, since the value of each node is only a Boolean value, the space required for the index is small, which guarantees the time and space complexity of the proposed method. As for BN-Index, the space required is correlated with the number of boundary nodes, which depends on the graph partitioning strategy. Although the random partitioning method we use will produce a large number of intermediate results, the experimental results prove the time and space complexity of BN-Index. Overall, the size of the BN-index is proportional to the size of the graph except that on LUBM20. The reason is that the number of crossing edges on LUBM20 is even more than that on LUBM30, which can be verified by the partial evaluation results depicted in Fig. 7a.

Exp 3. Efficiency of IBN-Index Based Optimization. A measurement of the statistical consequences of graph exploration time in the partial evaluation phase of $Q_1 \sim Q_8$ over different datasets is shown in Fig. 9. It can be observed that the partial evaluation method based on IBN-Index outperforms gStoreD on all queries and can improve query performance by 1.64 times in the best case. Furthermore, the method combining IBN-Index and BN-Index (the grey lines) improves the query efficiency by 1.79 times in the best case. As the size of the dataset increases, the proportion of time reduction also increases.

Interestingly, the optimization becomes more significant as the number of query nodes increases. The reasons can be summarized in two aspects. (1) When dealing with more

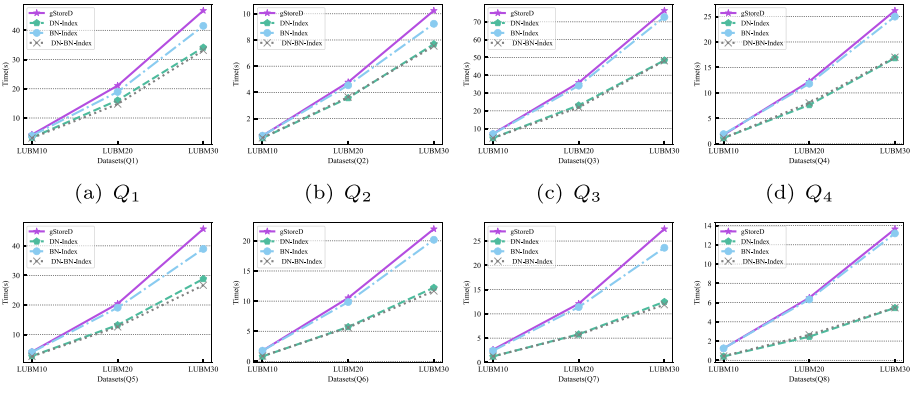

Fig. 9 The graph exploration in partial evaluation on LUBM datasets

query nodes, the number of candidate sets also rises, leading to more repetitive partial evaluation results. Therefore, the IBN-Index can be affected on more candidate sets resulting in a better pruning efficiency. (2) As the length of the result of partial evaluation expands, the candidate nodes are more likely to be pure internal nodes that can be filtered by IBN-Index.

Exp 4. Efficiency of BN-Index Based Optimization. Although gStoreD already has a partial match filtering strategy, the BN-Index-based method could have the same filtering effect and even higher efficiency than gStoreD. As shown in Fig. 9, the efficiency of graph exploration during the partial evaluation of the BN-Index-based method (the blue lines) exceeds gStoreD in most cases, and the strength is expanded as the scale of datasets grows. The reason is that when dealing with a large number of candidate vertices, gStoreD collects all the internal candidate vertices from all slave sites and transmits them back to all sites, which enlarges the searching space of graph exploration seriously. However, BN-Index will not enlarge the candidate sets.

Exp 5. Scalability. To prove the scalability of the IBN-Index and BN-Index based methods, the whole query times of the improved partial evaluation and assembly method over eight queries are presented in Fig. 7b. It is obvious that the query time of our method is nearly linear with the scale of the datasets. Unfortunately, all runs of query Q_3, including gStoreD and the IBN-Index and BN-Index based methods, are failed on the LUBM30 dataset due to the limited memory.

8 Conclusion

In this paper, we proposed an inner boundary node index-based method and a boundary node index-based method to improve the computing efficiency of the subgraph matching queries in distributed settings based on the partial evaluation and assembly framework. Moreover, we also proved that the IBN-Index and BN-Index are both time-efficient and space-effective. The extensive experimental results on benchmark datasets verified the efficiency and scalability of the proposed method, which clearly outperforms gStoreD when large-scale intermediate results need to be processed.

Appendix:: Workload Queries on LUBM

The query workload ($Q_1 \sim Q_8$) designed on LUBM are listed as follows:

PREFIX ub: ⟨ http://swat.cse.lehigh.edu/onto/univ-bench.owl#⟩
PREFIX rdf: ⟨ http://www.w3.org/1999/02/22-rdf-syntax-ns#⟩

Q1: SELECT ? X ? Y ? Z ?d WHERE{ ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:undergraduateDegreeFrom ? Y }
Q2: SELECT ? X ? Y ? Z ?d WHERE{ ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:mastersDegreeFrom ? Y . }

Q3: SELECT ? X ? Y ? Z ?d ?c WHERE{ ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:undergraduateDegreeFrom ? Y . ?d ub:takesCourse ?c . }
Q4: SELECT ? X ? Y ? Z ?d ?c WHERE{ ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:mastersDegreeFrom ? Y . ?d ub:teacherOf ?c . }
Q5: SELECT ? X ? Y ? Z ?d ?c ?t WHERE{ ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:undergraduateDegreeFrom ? Y . ?d ub:takesCourse ?c . ?t ub:teachingAssistantOf ?c . }
Q6: SELECT ? X ? Y ? Z ?d ?c ?t WHERE{ ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:mastersDegreeFrom ? Y . ?d ub:teacherOf ?c . ?t ub:teachingAssistantOf ?c . }
Q7: SELECT ? X ? Y ? Z ?d ?p ?c ?t WHERE{ ? X ub:advisor ?p . ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:mastersDegreeFrom ? Y . ?d ub:takesCourse ?c . ?t ub:teachingAssistantOf ?c . }
Q8: SELECT ? X ? Y ? Z ?d ?c ?t ?p WHERE{ ? X ub:advisor ?p . ? X ub:memberOf ? Z . ? Z ub:subOrganizationOf ? Y . ?d ub:mastersDegreeFrom ? Y . ?d ub:teacherOf ?c . ?t ub:teachingAssistantOf ?c . }

Acknowledgments This work is expanded on the optimal subgraph matching queries over distributed knowledge graphs based on partial evaluation [30], and is supported by National Key Research and Development Program of China (2019YFE0198600); the National Natural Science Foundation of China (61972275), partially supported by Australian Research Council Linkage Project (LP180100750).

Declarations

Conflict of Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Wang, X., Zou, L., Wang, C., Peng, P., Feng, Z.: Research on knowledge graph data management: a survey. J. Softw. **30**(7), 2140 (2019)
2. Consortium, W.W.W., et al.: Sparql 1.1 overview (2013)
3. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. In: International Semantic Web Conference, pp. 30–43. Springer (2006)
4. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, pp. 77–90 (1977)
5. Ren, X., Wang, J., Han, W.-S., Yu, J.X.: Fast and robust distributed subgraph enumeration. arXiv:1901.07747 (2019)
6. Jones, N.D.: An introduction to partial evaluation. ACM Computing Surveys (CSUR) **28**(3), 480–503 (1996)
7. Fan, W., Wang, X., Wu, Y.: Performance guarantees for distributed reachability queries. arXiv:1208.0091 (2012)

8. Wang, X., Wang, J., Zhang, X.: Efficient distributed regular path queries on rdf graphs using partial evaluation. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 1933–1936 (2016)
9. Wang, X., Wang, S., Xin, Y., Yang, Y., Li, J., Wang, X.: Distributed pregel-based provenance-aware regular path query processing on rdf knowledge graphs. World Wide Web **23**(3), 1465–1496 (2020)
10. Peng, P., Zou, L., Özsu, M.T., Chen, L., Zhao, D.: Processing sparql queries over distributed rdf graphs. The VLDB Journal **25**(2), 243–268 (2016)
11. Peng, P., Zou, L., Guan, R.: Accelerating partial evaluation in distributed sparql query evaluation. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 112–123. IEEE (2019)
12. Ge, Y.-F., Orlowska, M., Cao, J., Wang, H., Zhang, Y.: Mdde: multitasking distributed differential evolution for privacy-preserving database fragmentation. The VLDB Journal, pp. 1–19 (2022)
13. Ge, Y.-F., Yu, W.-J., Cao, J., Wang, H., Zhan, Z.-H., Zhang, Y., Zhang, J.: Distributed memetic algorithm for outsourced database fragmentation. IEEE Trans. Cybern. **51**(10), 4808–4821 (2020)
14. Rohloff, K., Schantz, R. E.: Clause-iteration with mapreduce to scalably query datagraphs in the shard graph-store. In: Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing, pp. 35–44 (2011)
15. 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. **23**(9), 1312–1327 (2011)
16. Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z.: A distributed graph engine for web scale rdf data. Proceedings of the VLDB Endowment **6**(4), 265–276 (2013)
17. Buneman, P., Cong, G., Fan, W., Kementsietsidis, A.: Using partial evaluation in distributed query evaluation. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 211–222 (2006)
18. Cong, G., Fan, W., Kementsietsidis, A., Li, J., Liu, X.: Partial evaluation for distributed xpath query processing and beyond. ACM Transactions on Database Systems (TODS) **37**(4), 1–43 (2012)
19. Ma, S., Cao, Y., Huai, J., Wo, T.: Distributed graph pattern matching. In: Proceedings of the 21st International Conference on World Wide Web, pp. 949–958 (2012)
20. O'Neil, P. E.: The sb-tree: an index-sequential structure for high-performance sequential access. Acta Informatica **29**(3), 241–265 (1992)
21. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. Proceedings of the VLDB Endowment **1** (1), 1008–1019 (2008)
22. Neumann, T., Weikum, G.: Rdf-3x: a risc-style engine for rdf. Proceedings of the VLDB Endowment **1**(1), 647–659 (2008)
23. Deppisch, U.: S-tree: a dynamic balanced signature index for office retrieval. In: Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 77–87 (1986)
24. Zou, L., Mo, J., Chen, L., Özsu, M.T.: Zhao, d.: gstore: answering sparql queries via subgraph matching. Proceedings of the VLDB Endowment **4** (8), 482–493 (2011)
25. He, H., Singh, A.K.: Closure-tree: An index structure for graph queries. In: 22nd International Conference on Data Engineering (ICDE'06), pp. 38–38. IEEE (2006)
26. Zhao, P., Han, J.: On graph query optimization in large networks. Proceedings of the VLDB Endowment **3**(1-2), 340–351 (2010)
27. Sasaki, Y., Fletcher, G., Onizuka, M.: Structural indexing for conjunctive path queries. arXiv:2003.03079 (2020)
28. Wang, X., Chai, L., Xu, Q., Yang, Y., Li, J., Wang, J., Chai, Y.: Efficient subgraph matching on large rdf graphs using mapreduce. Data Sci. Eng. **4**(1), 24–43 (2019)
29. Guo, Y., Pan, Z., Heflin, J.: Lubm: a benchmark for owl knowledge base systems. Journal of Web Semantics **3**(2-3), 158–182 (2005)
30. Xing, J., Liu, B., Li, J., Choudhury, F.M., Wang, X.: Optimal subgraph matching queries over distributed knowledge graphs based on partial evaluation. In: International Conference on Web Information Systems Engineering, pp. 274–289. Springer (2021)

Authors and Affiliations

Yanyan Song[1] · Yuzhou Qin[1] · Wenqi Hao[1] · Pengkai Liu[1] · Jianxin Li[2] · Farhana Murtaza Choudhury[3] · Xin Wang[1] · Qingpeng Zhang[4]

Yanyan Song
songyanyan1895@tju.edu.cn

Yuzhou Qin
yuzhou_qin@tju.edu.cn

Wenqi Hao
haowenqi@tju.edu.cn

Pengkai Liu
liupengkai@tju.edu.cn

Jianxin Li
jianxin.li@deakin.edu.cn

Farhana Murtaza Choudhury
farhana.choudhury@unimelb.edu.au

Qingpeng Zhang
qingpeng.zhang@cityu.edu.hk

1 College of Intelligence and Computing, Tianjin University, Tianjin, China

2 School of Information Technology, Deakin University, Melbourne, Australia

3 School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

4 School of Data Science, City University of Hong Kong, Hong Kong, China