Research Article

Subgraph-Indexed Sequential Subdivision for Continuous Subgraph Matching on Dynamic Knowledge Graph

Yunhao Sun, Guanyu Li, Mengmeng Guan, and Bo Ning

Faculty of Information Science and Technology, Dalian Maritime University, Dalian 116026, China

Correspondence should be addressed to Guanyu Li; rabitlee@163.com

Received 22 September 2020; Revised 24 October 2020; Accepted 9 November 2020; Published 22 December 2020

Academic Editor: Weitong Chen

Copyright © 2020 Yunhao Sun et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Continuous subgraph matching problem on dynamic graph has become a popular research topic in the field of graph analysis, which has a wide range of applications including information retrieval and community detection. Specifically, given a query graph \( q \), an initial graph \( G_0 \), and a graph update stream \( \Delta G_i \), the problem of continuous subgraph matching is to sequentially conduct all possible isomorphic subgraphs covering \( \Delta G_i \) of \( q \) on \( G_i (=G_0 \oplus \Delta G_i) \). Since knowledge graph is a directed labeled multigraph having multiple edges between a pair of vertices, it brings new challenges for the problem focusing on dynamic knowledge graph.

One challenge is that the multigraph characteristic of knowledge graph intensifies the complexity of candidate calculation, which is the combination of complex topological and attributed structures. Another challenge is that the isomorphic subgraphs covering a given region are conducted on a huge search space of seed candidates, which causes a lot of time consumption for searching the unpromising candidates. To address these challenges, a method of subgraph-indexed sequential subdivision is proposed to accelerate the continuous subgraph matching on dynamic knowledge graph. Firstly, a flow graph index is proposed to arrange the search space of seed candidates in topological knowledge graph and an adjacent index is designed to accelerate the identification of candidate activation states in attributed knowledge graph. Secondly, the sequential subdivision of flow graph index and the transition state model are employed to incrementally conduct subgraph matching and maintain the regional influence of changed candidates, respectively. Finally, extensive empirical studies on real and synthetic graphs demonstrate that our techniques outperform the state-of-the-art algorithms.

1. Introduction

The problem of subgraph matching is one fundamental issue in graph search, which is NP-Complete problem [1]. Specifically, given a query graph \( q \) and a large data graph \( G \), the problem of subgraph matching is to extract all isomorphic subgraphs of \( q \) on \( G \). In real world, data is usually emerged as a streamlined feature in social networks, which is formed as a graph stream. Recently, continuous subgraph matching on dynamic graph has become a popular research topic in the field of graph analysis, which has a wide range of applications including query answering [2], information retrieval [3, 4], and community detection [5, 6]. Specifically, given a query graph \( q \), an initial graph \( G_0 \), and a graph update stream \( \Delta G_i \), the problem of continuous subgraph matching is to sequentially conduct all possible isomorphic subgraphs covering \( \Delta G_i \) of \( q \) on \( G_i (=G_0 \oplus \Delta G_i) \). In this paper, we study the continuous subgraph matching on a special graph structure of knowledge graph (KG-CSM).

Despite the complex multigraph characteristic of knowledge graph and the polynomial-time complexity of continuous subgraph matching [1], recent existing research studies have made significant advances in developing computational paradigm of KG-CSM.

One aspect is to storing and indexing RDF triple data based on relational approaches. Weiss et al. [7] and Pérez et al. [8] employed an index-based solution to storing triples directly in an index of B*-tree over multiple redundant permutations. Abadi et al. [9] vertically partitioned the RDF triples into a set of tables bounded by the labels of patterns and used an index structure on top of it to locate the required tables. Broekstra et al. [10] were based on the idea of
complexity of candidate calculation. Three empirical studies motivate us to develop an efficient subgraph matching method on dynamic knowledge graph. The first empirical study demonstrated [5, 14] that the tree-based index can reduce the noncandidates of dynamic graph by the influenced analysis of anchored and followed relationships. The second empirical study [15] demonstrated that the sequential technology can effectively limit the search space of graph update stream. The third empirical study [16] was our prior research of subgraph index on static knowledge graph, which demonstrated that the subgraph index can effectively accelerate the subgraph matching on static knowledge graph. In this paper, we propose a method of subgraph index-based sequential subdivision to accelerating the continuous subgraph matching on dynamic knowledge graph. Our contributions are described as follows:

1. Challenge 1: Multigraph Characteristic of Knowledge Graph Intensifies the Complexity of Candidate Calculation. Knowledge graph is a directed labeled multigraph having multiple edges between a pair of vertices, each vertex represents an entity with attributes and each edge denotes an interentity relationship. Considering the model of knowledge multigraph in Figure 1, it is composed of attributed and topological structures. The attributed structure describes the attribute and type of entity, where attribute is taken as the label of edge coupled with a value and type is taken as the label of entity. The topological structure describes the relationship between a pair of entities and some relationships are coexistent, e.g., partnerships and couple relationship between persons. The multigraph characteristic of knowledge graph leads to a more dense adjacent structure than general graph, and it brings a new challenge to the research of KG-CSM problem. Furthermore, KG-CSM problem still contains the traditional challenge on general graph.

1.2. Challenge 2: Subgraph Isomorphic Mappings Covering a Given Region Are Conducted on a Huge Search Space of Seed Candidates. The traditional challenge on general graph is that the isomorphic subgraphs covering a given region are conducted on a huge search space of seed candidates, which causes a lot of time consumption for searching the unpromising candidate. Considering query graph \( q \) and data graph \( G \) in Figure 2, an edge \( (v_{10}, v_{11}) \) is inserted into \( G \). An isomorphic subgraph is defined as a subgraph isomorphic mapping and conducted as \( (\langle v_1, u_1 \rangle, \langle v_4, u_2 \rangle, \langle v_{10}, u_3 \rangle, \langle v_5, u_4 \rangle, \text{and} \langle v_{11}, u_5 \rangle) \). The basic strategy is to search the global space of \( G \) without the reduction of unpromising vertices \( v_2, v_3, v_6, v_7, \text{and} v_9 \).

1.3. Contributions. Three empirical studies motivate us to develop an efficient subgraph matching method on dynamic knowledge graph. The first empirical study demonstrated [5, 14] that the tree-based index can reduce the noncandidates of dynamic graph by the influenced analysis of anchored and followed relationships. The second empirical study [15] demonstrated that the sequential technology can effectively limit the search space of graph update stream. The third empirical study [16] was our prior research of subgraph index on static knowledge graph, which demonstrated that the subgraph index can effectively accelerate the subgraph matching on static knowledge graph. In this paper, we propose a method of subgraph index-based sequential subdivision to accelerating the continuous subgraph matching on dynamic knowledge graph. Our contributions are described as follows:

1) We develop a flow graph index to pruning the noncandidates of query vertices on topological knowledge graph. The flow graph index is defined as a flow graph (FG), which is a directed multigraph, constructed from the initial data graph \( G_0 \) and guided by a matching order of query graph. Each vertex of FG denotes a candidate of one query vertex, which is taken as the label of candidate. Each edge of FG corresponds to the relationships of nodes in the matching tree of query graph. The flow graph index can effectively reduce the scale of original data graph.

2) We design an adjacent index to accelerate the identification of candidate activation states on attributed knowledge graph. The three benefits are discovered from our adjacent index. The first benefit is that the adjacent index can improve the time-efficiency of comparison of the inclusion relationships of node pair. The second benefit is that the adjacent index can quickly verify the transformed state of seed candidate as graph update stream is incrementally inserted. The third benefit is that the adjacent index can quickly search the adjacent candidate region.

3) We propose a sequential subdivision technology of the flow graph to limit the search derivation of graph update stream. The sequential numbers of root candidates are assigned to the vertices of subdivided flow graphs and limit the search space of originating changed candidate of FG.

4) We design a state transition model to describe the transition states of changed candidates, which consists of three states and six transition rules. Based on the state transition model, we analyze the influence of changed candidates to the adjacent region and design our incremental maintenance strategy.

5) We design an incremental subgraph matching algorithm based on the sequential subdivided flow graph. The consistency of subgraph matching is guaranteed by two verifications of selected candidates, relational verification and sequential verification. The relational and sequential verifications are used to verify the local isomorphism and the
equivalence of sequential numbers between one local subgraph mapping and selected candidates, respectively. The isomorphic subgraphs are incrementally and effectively conducted with the aid of relational and sequential verifications.

Extensive empirical studies on real and synthetic graphs demonstrate that our techniques outperform the state-of-the-art algorithms.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries about problem definitions and related works. Section 3 provides a flow graph index of knowledge graph, including the definition and construction of the flow graph index. Section 4 presents an incremental subgraph matching on the flow graph, including the sequential division technology, incremental maintenance, and incremental subgraph matching on graph update stream. Experimental results are reported in Section 5. A conclusion is given in Section 6.

2. Preliminaries

In this section, the definitions of knowledge graph and subgraph matching are first given. Then, the related research studies are introduced.

2.1. Problem definition. Knowledge graph KG is a directed labeled multigraph having multiple edges between a pair of vertices. The labels of KG are extracted from RDF information. Resource Description Framework (RDF) [17] is a standard semantic model designed by W3C group 2, which is represented by a set of triples \( \langle S, P, O \rangle \). Each triple \( \langle s, p, o \rangle \) consists of three components: a subject, a predicate, and an object. Furthermore, a triple \( \langle s, p, o \rangle \) is formed as \( I \times I \times L \), where \( I \) denotes an IRI (Internationalized Resource Identifier) and \( L \) represents a literal. Through the extension of RDF triple model with timestamp, the model can be used to represent RDF stream, denoted as \( \langle s, p, o \rangle : t \) [18]. Here, \( \langle s, p, o \rangle \) is an RDF triple and \( t \) is a timestamp.

The labels of KG are classified as instance-label, relation-label, attribute-label, and type-label according to the resource and inter-resource relationship of RDF data. Considering an RDF triple \( \langle s, p, o \rangle \), \( o \) is named as type-label if and only if both \( s \) and \( o \) are IRIs and \( p \) is a typed predicate, e.g., \( rdf:type \) and \( rdf:subclassof \). \( s \) and \( o \) are called instance-label and \( p \) is named as relation-label if and only if both \( s \) and \( o \) are IRIs and \( p \) is not a typed predicate, and \( p \) is called as attribute-label if and only if \( o \) is a literal.

**Definition 1** (knowledge graph). A knowledge graph is a directed labeled multigraph, formed as \( G(V, E, L) \). Here, \( V \) is
a set of vertices, \( E \subseteq V \times V \) is a set of directed edges; \( L = \{L_v \cup L_e\} \) is a labeling function, \( L_v \) assigns type-labels and attribute-labels to vertices, and \( L_e \) assigns relation-labels to edges.

In a KG, each vertex can be assigned by multiple labels and each edge can also be assigned by multiple labels. Considering a vertex \( v_1 \) and an edge \((v_2, v_3)\) of KG in Figure 3(b), type-labels or attribute-labels \( A \) and \( B \) are assigned to \( v_1 \) and relation-labels \( a \) and \( e \) are assigned to \((v_2, v_3)\) and relation-label \( a \) is assigned to \((v_1, v_2)\). It can be found that KG has a denser intervertex relation than the general graph.

2.1.1. Subgraph Matching. The problem of subgraph matching is to search all possible subgraphs of data graph \( G \) that are isomorphic to query graph \( q \). The subgraph matching is formally defined as a problem of subgraph isomorphism, as described in Definition 2.

**Definition 2 (subgraph isomorphism).** Given a data graph \( G(V, E, L) \) and a query graph \( q(V_q, E_q, L) \), \( q \) is subgraph isomorphic to \( G \) if and only if there exists a bijection \( M \) from \( V_q \) to \( V \) such that \( \forall u \in V_q, \exists M(u) \in V: L_v[M(u)] = L_v(u) \) and \( \forall u, u' \in V_q, \exists M(u), M(u') \in E_q: (M(u), M(u')) \in E \). \( q \) is a subgraph isomorphic mapping from \( q \) to \( G \).

A query graph \( q \) is subgraph isomorphic to a data graph \( G \) if there exists a subgraph isomorphic mapping \( q \) from \( q \) to \( G \). Simply, considering the labeled query graph and data graph in Figures 3(a) and 3(b), respectively, \( \{A, B, C\} \) is a set of vertex-labels. \( q \) is subgraph isomorphic to \( G \) since there exist subgraph isomorphic mappings \( M_1: \{v_1, u_1, v_2, u_2, v_3, u_3, v_4, u_4\} \) and \( M_2: \{v_2, u_4, v_3, u_3, v_4, u_4\} \).

In this paper, our research of KG-CSM problem focuses on constructing an effective lightweight index to arrange the search space of seed candidates. Then, a region-limited technology is used to constrain the derivation of search spatial scale of graph update stream.

In this paper, we focus on a directed labeled graph \( G(V, E, L) \). Here, \( V \) is a set of vertices, \( E \subseteq V \times V \) is a set of edges, and \( L = L_v \cup L_e \) is a labeling function which assigns a label or multiple labels to vertex and edge. Both \( q \) and \( G \) are directed labeled graphs, and the directed or undirected edges cannot affect the execution scheduling of subgraph matching. The detailed notations and meanings are described in Table 1.

2.2. Related Works. In this section, we mainly review the related works on index and subgraph matching algorithms of knowledge graph and general graph and then outline their limitations.

2.2.1. Subgraph Matching of Knowledge Graph. The storage structure of RDF data should be introduced before discussing the index of the knowledge graph. The knowledge graph is modeled by Resource Description Framework (RDF), which is a standard semantic data model designed by W3C group. The storage structure of RDF data generally accepted by research studies mainly includes relational store. The relational store is involved in many systems, i.e., SW-store [9], Sesame [10], Jena [19], RDF-3X [20], etc. The relational approach can be classified as vertical representation and horizontal representation.

In the vertical representation approach, RDF data is conceptually stored in a single table over the relational schema. Due to the large size of RDF data and the potential large number of self-joins required to answer queries, it must be taken to devise an efficient physical layout with suitable indexes to support query answering. However, there are a mount of overlapped copies of RDF triples. For avoiding storing multiple copies of RDF triples, there are many triple stores [21, 22] that aggressively store the triple table in multiple sorted orders. A clustered B-tree is constructed and the desired triple ordering is available in the leaves of B-Tree. However, the approaches of B-Tree are more demanding in terms of storage space because the effective query answering should support the availability of various sorted recording for fast merge joins.

In the horizontal representation approach, RDF data is stored in one or more wide tables by interpreting predicate as column name. To minimize the storage overhead caused by empty cells, property table approaches [23, 24] were proposed and concentrated on dividing the wide table in multiple smaller tables containing related predicates. However, this approach actually creates many small tables, which are harmful on query evaluation performance. Specialiy, a vertically partitioned approach was proposed and the decomposition was taken to its extreme. Both the issues of empty cell and the multiple objects are solved at the same time. Abadi et al. [9] and Sidirourgos et al. [25] noted that the performance of this approach is best when sorting the binary tables lexicographically to allow fast joins.

2.2.2. Subgraph Matching on RDF Stream. Most studies on index and pattern matching of streaming RDF data are designed to leverage existing solutions for nonstreaming RDF data. The standardization of streaming RDF data is still an ongoing debate, and W3C RSP community group3 is an important initiative. These studies utilized various custom query languages that are extended from SPARQL to answer the queries. Additionally, these studies still followed a relational approach to storing and indexing RDF data.

For instance, C-SPARQL [3] used the underlying Jena architecture to store and index triples within property tables, whereas CQELS employed index-based solutions by storing
triples directly in asc-trees over multiple redundant permutations and used Eddy operators and query optimizations to index triples. SparkWave [4] used the RETE network to determine the set of triggers to fire when a new triple arrives, and it materialized intermediate results to reduce the amount of work that is required for each update. Those systems stored RDF data in relational tables and process queries using relational operators, such as scan and join operators.

However, the relational stores need too many join operations for evaluating the queries, especially those queries having complex and large graph patterns. Meanwhile, the index-based relational approaches represent progress towards the more dynamic environments by allowing continuous monitoring and periodically evaluating the index design. The majority of these approaches employed the re-evaluation strategies for optimizing execution plan, which requires an incremental indexing technique to maintain intermediate results automatically and incrementally.

The majority of RDF Stream Processing (RSP) systems are based on a recalculated model. The recalculcation of matches can result in unnecessary utilization of computational resources once the data are updated within a window. For instance, Eddy operators [26] that were employed by CQELS [18] resulted in expensive computations and continuous usage of resources to explore all plans, thus requiring a fully pipelined execution for RDF streams. Furthermore, caching the statistical measure of triples and choosing the correct order for every triple update causes considerable overhead.

A prominent contribution for RDF stream is the study entitled SPECTRA [27], in which a set of vertically partitioned views was used to collect the summarized data from each event and sibling lists were employed to incrementally index the joined triples between views. The matched results were shown in a set of final views, thus enabling an incremental evaluation with the arrival of new events. Although the combination of RSP and incremental algorithms improve the execution time efficiency for streaming RDF data, however, the relational approach with a higher joined depth and a greater focus on independent events was presented by SPECTRA, which provides a motivation for our study. We argue that the incremental evaluations can greatly reduce the computation tasks and improve the execution performance.

2.2.3. Subgraph Matching on General Graph. The problem of pattern matching for RDF graph is similar as the problem of subgraph isomorphism for the general graph. Ullmann [28] proposed a backtracking algorithm that significantly reduces the size of search space. VF2 [29] was a well-known state-of-the-art algorithm, which proposed a state space representation to deal with different exact graph matching problem: each state is a partial mapping between two given graphs, while goal states are

![Figure 3: Knowledge multigraphs. (a) Query multigraph. (b) Data multigraph.](image)

Table 1: Notations and meanings.

| Notations | Meanings |
|-----------|----------|
| q = (Vq, Eq, L) | A directed query multigraph with vertex set Vq, edge set Eq and labeling function L |
| G = (V, E, L) | A directed data multigraph with Vertex set V, edge set E and labeling function L |
| qT = (VT, ET, LT) | A matching tree of q with node set VT and edge set ET |
| FG = (VF, EF, LF) | A flow graph index of q on G vertex set VF, edge set EF, and labeling function LF |
| ΔFGq | A flow graph index on the initial data graph Gq |
| ΔFGi | A flow graph index maintained by graph update stream ΔGi |
| np: (v, u) | A query-data vertex pair with v ∈ V and u ∈ Vq |
| State (np) | The state of node pair np. State (np) = CS (np) ∧ FS (np) |
| CS (np), FS (np) | The candidate and following states of np |
| SG | A subgraph isomorphic mapping of q on G |
| M | A set of subgraph isomorphic mappings Ms of q on G |
| M̂ = {⟨v1, u1⟩, ⟨v2, u2⟩, ..., ⟨vn, un⟩} | A candidate set of u, v ∈ Vq |
| CR(u, v) | A candidate region of u and adjacent to v, u ∈ Vq, v ∈ V |

![Notations Table](image)
complete mappings consistent with the problem constraints. Hence, the search space was explored through a depth-first strategy with backtracking, which is driven by a set of feasibility rules to prune unfruitful search paths. SPath [30] implemented path-at-a-time pattern during the searching process. It decomposed the query graph into several paths and found the embeddings of each path which would be joined later. TurboISO [31] and BoostISO [32] tried to find greater matching order to make subgraph matching more efficient, where the graph-compressed method is implemented to reduce the space complexity. For reducing the duplicate adjacent candidates, a compact path index was proposed by CFLMatch [33]. The compact path index is a multipath index induced by a spanning tree of query graph, and it is composed of multiple clusters and intercluster relations. Each cluster collects the candidates of one query vertex. A data-centric path index was proposed by TurboFlux [13], which can further eliminate the storage of duplicate candidates. The data-centric path index attaches query vertices as a label set to data vertices. TurboFlux employed a data-centric path index to accelerate the continuous subgraph matching on dynamic graph. However, the algorithms of subgraph isomorphism hardly migrate to the problem of patterning matching on RDF graph. Therefore, the algorithms of subgraph isomorphism on RDF graph, which was proved by [34], due to the unsymmetrical structural characteristic of RDF graph essentially.

The indexing and machine learning technologies employ semantic and structural characteristics to enhance the semantic equivalence of KG. S2R-tree [35] was a pivot-based hierarchical indexing structure to integrate spatial and semantic information in a seamless way, which used a space mechanism to transform the high-dimensional semantic vectors to a low-dimensional space. A predictive model of future star was proposed by FS-ELM [36], which studied a rising star evaluation by exploiting social topology characteristics and user behavior patterns in geo-social networks. MTLM [37] proposed a multistage learning model for traversal time estimation, which first recommended the appropriate transportation mode for users and then estimated the related traversal time of path in tree pattern. RQL [38] designed a reinforcement learning-based algorithm for the dynamic bipartite graph matching problem, which made near-optimal decisions on batch splitting with a constant competitive ratio. Gao et al. [39] proposed a novel framework to achieve the privacy-preserving subgraph pattern matching in cloud. The framework used a label-generated privacy model to protect and label the potential privacy in both data graphs and pattern graphs.

2.2.4. Subgraph Matching on Dynamic Graph. A dynamic graph is modeled as a graph, whose edges are activated by sequences of time-dependent elements. Wang et al. [6] discussed the definition and topological structure of time-dependent graphs, as well as models for their relationship to dynamic systems. In addition, they reviewed some classic problems on time-dependent graphs and studied the weight-constrained route planning problem over a large time-dependent graph coupled with continuous time and weight functions [40]. Choudhury et al. [41] provided a subgraph selectivity approach to determine subgraph search strategies and used a subgraph tree structure to decompose the query graph into smaller subgraphs, which are responsible for storing partial results. However, retaining and querying thousands of edges within a large window requires considerable amount of space and computational resources. Moreover, this approach only supports simple path-based queries, and it is optimized for homogeneous graphs using an edge stream model. Fan et al. [12] presented algorithms for graph pattern matching over evolving graphs by employing a repeated search strategy to calculate matches until a fixed point reached with each graph is updated and removed. However, the repeated search strategy can enlarge the time consumption of subgraph matching.

The more related technologies employed semantic and structural characteristics to improve the performance of dynamic problem. INC-GPM [42] built an index to incrementally record the shortest path length range between different label types and then identified the affected parts of graph update stream. DCSGR [43] exploited the connections between group users in community detection and proposed an aggregation function to integrate the recommended media lists of all interest subgroups as the final group recommendation results.

3. Flow Graph Index of Knowledge Graph

In this section, a flow graph index (FG) of the knowledge graph is proposed to arrange the search space of seed candidates. Before the introduction of FG, our solution for KG-CSM problem is first given to clarify the core role of FG in our algorithm (Algorithm 1).

A pseudocode of continuous subgraph matching is described in Algorithm 1, named as incremental pattern matching algorithm (iPM). A matching tree orchestrates a matching order to iteratively conduct subgraph mappings (Line 1). In this paper, we employ the matching order generated by a depth-first traversal without considering the calculated paradigms of near-optimal matching order because we are committed to the incremental calculated paradigms of graph update stream. The flow graph index FG of the knowledge graph is constructed by sequential and mapping relationships of $q_T$ on $G_0$ (Line 3 and Section 3.2) and incrementally maintained by a graph update stream $\Delta G_i$ (Line 5 and Section 4.1). Then, all subgraph mappings covering $\Delta G_i$ are directly conducted by the iterative traversal on FG (Line 6 and Section 4.2).

The core role of FG in our algorithm consists of three parts, described as follows. The first part is the initial construction of FG, defined as $\Delta FG_0$, which is guided by a matching tree $q_T$ on the initial data graph $G_0$. The second part is the maintenance of $\Delta FG_0$ adapting to graph update stream $\Delta G_i$, defined as $\Delta FG_i$. The third part is the incremental matching of $\Delta FG_i$ in the adaptive matching order $\Delta q_T$. Thus, FG is the core role of our designed approach, introduced in Section 3.1.

3.1. Data Index of Knowledge Graph. Knowledge graph is a directed labeled multigraph, which is the combination of complex topological and attributed structures. The data
Complexity

index of knowledge graph is composed of flow graph index and adjacent index. The flow graph index is constructed from the topological structure of knowledge graph, which is used to arrange the search space of seed candidates.

3.1.1. Flow Graph Index of Topological Knowledge Graph. The flow graph index is defined as flow graph (FG), which is designed to arrange the binary relationships between a pair of data vertices in G. The binary relationship of FG follows the parent-child relationship of spanning tree of q. We divide the edges of q into tree edge and non-tree edge according to the parent-child relationship of spanning tree of q. A spanning tree containing both tree edges and non-tree edges is called as matching tree, formed as QT. Considering a query graph in Figure 2(a), a matching tree is described in Figure 4(a), which is ordered by a depth-first traversal on q.

The solid line denotes the tree edge and dotted line indicates the non-tree edge. Regarding a tree edge \( (u_1, u_2) \), u_1 is a parent of u_2, described as \( u_2.p = u_1 \). A flow graph is constructed in the guide of matching tree, described in Definition 4.

**Definition 4** (flow graph). A flow graph is a directed labeled multigraph, formed as \( FG_q(q_T, E_F, L_F) \). Here, \( V_F \) is a set of vertices, \( E_F \) is a set of edges, and \( L_F \) is a labeling function that assigns one or multiple labels to vertices.

Here, each vertex of \( V_F \) refers to a query-data vertex pair (node pair for short) of q on G. Regarding a node pair \( (v, u) \), satisfying \( v \in V \) and \( u \in V_q \), then v is a vertex labeled by u in FG. Each edge of \( E_F \) indicates the tree edge or the non-tree edge similar as the matching tree. Regarding node pairs \( (v, u) \) and \( (v', u') \), satisfying \( u.p = u' \) and \( v \) is a neighbor of \( v' \), then v is a parent of \( v' \), formed as \( v.p = v' \). Considering the data graph in Figure 2(b) and the matching tree in Figure 4(a), a flow graph is described in Figure 4(b). Regarding vertices \( v_1 \) and \( v_3 \), which are labeled by \( u_1 \) and \( u_2 \), respectively, satisfying \( v_4.p = v_1 \), because \( u_2.p = u_1 \) and \( v_4 \) is a neighbor of \( v_1 \).

The unconstrained quantity of node pairs may cause a huge space scale of vertices in FG. Considering a query graph of size n and a data graph of size m, the quantity of node pairs are calculated as \( n \times m \). The two strategies are used to solving the unconstrained quantity of node pairs. One strategy is to employ a labeling function that assigns multiple labels to vertices, which avoids the repeated storage of vertices in FG. Another strategy is to design the constraint rules of node pairs. The constraint rules are denoted in the definition of candidate verification, as described in Definition 5.

**Definition 5** (candidate verification). Given a node pair \( (v, u) \), data vertex v is the candidate of query vertex u if and only if it satisfies the following constraints: (1) \( L_V(u) \subseteq \) \( L_V(v) \), (2) \( \forall u' \in N(u), \exists v' \in N(v) : L_v(u') \subseteq L_V(v') \), and (3) \( \forall (u, u') \in E_q, (v, v') \in E : E_{L_G}(u, u') \subseteq L_E(v, v') \).

Here, \( L_V \) and \( L_E \) denote the labeling functions of vertex and edge, respectively. The constraints of candidate verification can effectively reduce the scale of node pairs. A node pair is deleted if it does not satisfy constraint (1). Furthermore, we divide the node pair as positive and negative node pair according the relax and strict constraints. Considering a node pair \( np.(v, u) \), np is a negative node pair if and only if it satisfies constraint (1), and np is a positive node pair if and only if it satisfies constraints (1), (2), and (3). Considering the flow graph in Figure 4(b), solid cycle denotes the positive candidate and dotted cycle indicates the negative candidate.

The negative candidate may be changed as a positive one when graph update stream is inserted into FG. To intuitively express the transformed state, a node pair state is defined to denoting the active and silent states of node pairs, formed as \( State((v, u)) \). Node pair np.(v, u) satisfies the relax and strict constraints, formed as \( State(np) = 0 \) and \( State(np) = 1 \), respectively, then np is encapsulated into a labeled vertex in FG, otherwise it is pruned.

A node pair state is composed of candidate state \( CS \) and following state \( FS \), denoted as \( State(np) = CS(np) \land FS(np) \). The candidate state \( CS \) describes the negative and positive node pairs. A node pair \( CS \) is positive if \( CS(np) = 1 \). The following state \( FS \) is used to describe the candidate states of followers. We define the descendants of query vertex u as \( Des(u) \). Given a node pair np.(v, u), a candidate \( v' \) of \( Des(u) \) is the follower of np if it is reachable from v, then v is named as the dominator of u. Regarding a node pair np.(v, u), \( FS(np) = 1 \) if it satisfies the condition \( \exists v' \in Des(u) : v' \in Des(np) \). Here, AL, OEL, and IEL denote the labels of adjacent vertex, inner edge, and outer edge of query or data vertices, respectively.

3.1.2. Adjacent Index of Attributed Knowledge Graph. In this paper, we focus on the problem of continuous subgraph matching on a special knowledge graph. Knowledge graph (KG) is a directed labeled multigraph having multiple edges between a pair of vertices. The labels of knowledge graph can be classified as type label and attribute label. Actually, the vertex of KG can be coupled with one or multiple labels.

To deal with the challenge of multigraph characteristic, the adjacent indexes of query and data vertices are proposed to accelerating the time-efficiency of candidate verification between initial data graph and graph update stream. Considering query and data multigraphs in Figure 3, the adjacent indexes of query vertex \( u_1 \) and data vertex \( v_2 \) are described in Table 2. Here, AL, OEL, and IEL denote the labels of neighbors, inner edge, and outer edge of query or data vertex, respectively.

The first benefit is that adjacent indexes can improve the calculated time efficiency of the inclusion relationships of node pair. Considering the common adjacent label B of \( u_1 \) and \( v_1 \), the counts can be used to quickly calculate the inclusion relationships of \( u_1 \) and \( v_1 \) with \( O(1) \) time complexity. The adjacent label B of \( u_1 \) is included by the one candidate of \( u_1 \) if the adjacent label count of \( v_1 \) is not less than the count of \( u_1 \). Thus, the verification of inclusion relationship of node pair is \( O(n) \) time-complexity, where \( n \) is the count of unique number about adjacent vertex, inner and outer edge labels.
The second benefit is that adjacent indexes can quickly verify the transformed state of node pair as graph update stream is inserted. The candidate state of node pair \( np \) can be calculated as the intersection operation of the three label index, formed as \( \text{CS} (np) = \text{AL}(np) \land \text{OEL}(np) \land \text{IEL}(np) \). The candidate state of node pair is verified with \( O(1) \) time-complexity if one label index of node pair is changed.

The third benefit is that adjacent indexes can quickly search the adjacent candidate region. Through the verification of common adjacent label \( B \) of \( u_1 \) and \( v_2 \), it can be found that \( v_1 \) and \( v_4 \) are the negative candidates of \( u_2 \), that satisfy constraint (1) in Definition 5. Furthermore, the final adjacent candidate region can be reduced through the intersection operation of common neighbor of different unique labels.

### 3.1.3. Time and Space Complexity of Knowledge Graph Index

The data index of knowledge graph is composed of flow graph index and adjacent index.

The time and space of flow graph index is described as follows. The worst-case space-complexity of FG is \( O(2|V| + (2 + |V_q|) \cdot |V| \cdot (|V| − 1)) \). The first reason is that the size of vertices in FG is at most \( |V| \) when each vertex is the candidate data of one query vertex. The second reason is that the size of edges in FG is at most \( 2 \cdot |V| \cdot (|V| − 1) \), when each vertex pair has an edge relationship coupled with two tree edges and two nontree edges. The third reason is that the size of edge labels in FG is at most \( |V_q| \cdot |V| \cdot (|V| − 1) \) when each tree-edge is assigned by all query vertices. Actually, the tree edge-labels can be encoded by a \( |V_q| \)-bit string.

The worst-case time complexity is \( O(|V_q| + |E| − 1) \) about insertion and deletion of one vertex on FG. Regarding

---

**Algorithm 1: Incremental pattern matching (iPM).**

**Table 2: Adjacent indexes of query and data vertices.**

| Adjacent Label Index of \( u_1 \) | Adjacent Label Index of \( v_2 \) |
|-----------------------------------|-----------------------------------|
| **AL Count** | **N(\( u_1 \))** | **AL Count** | **N(\( u_1 \))** |
| \( u_1 \) | \( B \) | 1 | \( u_2 \) | \( B \) | 1 | \( u_3 \) | \( A \) | 1 |
| \( u_1 \) | \( C \) | 1 | \( v_2 \) | \( C \) | 1 |

| Outer edge labels index of \( u_1 \) | Outer edge label index of \( v_2 \) |
|-----------------------------------|-----------------------------------|
| **OEL Count** | **N(\( u_1 \))** | **OEL Count** | **N(\( u_1 \))** |
| \( u_1 \) | \( A \) | 1 | \( u_3 \) | \( a \) | 2 |
| \( u_1 \) | \( E \) | 1 | \( v_2 \) | \( c \) | 2 |
| \( v_2 \) | \( IEL \) | Count | \( N(\( u_1 \)) \) | \( v_1 \) |

---

**Input:** a query graph \( q \), an initial data graph \( G_0 \) and a graph update stream \( \Delta G_i \)

**Output:** the set \( \mathcal{M} \) of all subgraph mappings of \( q \) in \( G_0 \oplus \Delta G_i \)

1. \( q_T \leftarrow q \)-Generation \( q \);
2. If \( i = 0 \) then \( \Delta FG_0 \leftarrow FG(q_T, G_0) \space{2cm} \text{(2)} \)
3. Else \( \Delta FG_i \leftarrow \text{incrementalMaintenance} (\Delta FG_0, \Delta G_i) \space{2cm} \text{(4)} \)
4. \( \mathcal{M} \leftarrow \text{IncrementalMatching} (\Delta FG_i, \Delta q_T) \space{2cm} \text{(6)} \)
5. Return \( \mathcal{M} \);

**Figure 4:** (a) Matching tree. (b) Flow graph.
vertex \( v \in V_F \), if \( v \) has the worst-case time complexity, it should satisfy that \( v \) is the common candidate of all query vertices in \( q \). \( v \) is connected with all other vertices in \( V_F \), and \( |V_F| = |V| \).

The time and space of the adjacent index is described as follows. The space complexity is equivalent to the one of doubly linked list. The worst-case time complexity is \( O(|E| - 1) \) about the insertion and deletion of one vertex on \( FG \) if the vertex is connected with all other vertices in \( V_F \) and \( |V_F| = |V| \).

In this paper, our research of KG-CSM problem focuses on constructing an effective lightweight index to search space of seed candidates. Through the analysis of time and space complexities, our knowledge graph index is a linear consumption and it is beneficial to indexing the single large-scale data graph.

### 3.2. Construction of Flow Graph

In this section, our construction algorithm of the flow graph is introduced in Algorithm 2. The inputs are an initial data graph \( G_0 \), a query graph \( q \), and its matching tree \( q_T \). The output is the subgraph index of flow graph \( D\bar{F}G_0 \). A matching tree \( q_T \) orchestrates a matching order \((u_0, u_1, \ldots, u_{|V_F|-1})\) and the parent-child relationships of query vertices in \( q \). The construction algorithm of the flow graph contains three modules (Algorithm 2).

The first module is used to verify the candidate state of node pair with the aid of adjacent indexes (Lines 3–6). The following and candidate states of node pair are initialized as 0 and \(-1\), respectively. Regarding a node pair \( np: \langle v, u \rangle \), \( v \) is negative candidate of \( u \) if \( np \) satisfies constraint (1) in Definition 5, then the candidate state of \( np \) is marked as 0 (Line 4). If \( np \) satisfies constraints (1), (2), and (3) in Definition 5, \( v \) is positive candidate of \( u \), then the candidate state of \( np \) is marked as 1 (Line 5). Considering a node pair \( np: \langle v, u \rangle \), satisfying \( u \) is a leaf node in \( q_T \) and \( v \) is positive candidate of \( u \), then the following state of \( np \) is marked as 1 (Line 6) because there is not a descendant can be included into leaf nodes. All negative and positive candidates are added into candidate set and vertex set \( V_F \) (Lines 4–5), and \( C(u) \) denotes a set of candidates of query vertex \( u \).

The second module is used to verify the node pair state through the calculation of following state in bottom-up matching order (Lines 7–14). Regarding node pairs \( \langle v, u \rangle \) and \( \langle v', u' \rangle \), satisfying \( u', p = u \) and \( v', v' \) are the positive of \( u \) and \( u' \), respectively, then \( v.FS(u) = v'.FS(u) \lor v.FS(u) \lor v'.FS(u) = v'.State(u') \lor v.FS(u) \) denotes that \( v.FS(u) \) is true once the boolean of one descendant of \( v, u \) is true. \( v'.State(u') = v'.State(u) \land v.FS(u) \) refers that \( v'.State(u') \) is true if and only if both \( v.FS(u) \) and \( v.FS(u) \) are true.

The third module is used to insert the edges to \( FG \) in top-down matching order. Considering node pairs \( \langle v, u \rangle \) and \( \langle v', u' \rangle \), satisfying \( u, p = u \) and \( v \) is a candidate of \( u \), then \( (v, v') \) is a tree edge and inserted into \( FG \). Otherwise, \( (v', v) \) is a nontree edge and inserted into \( FG \). The function \( (v, v') \rightarrow s_{nontree}E_p \) denotes that \( (v', v) \) is inserted into \( E_p \) and the minimum sequential number of \( v' \) is assigned to \( v \), which is described clearly in Section 4. The tree edge and nontree edge are used to distinguish the operations of node pair in continuous subgraph matching.

### 3.2.1. Example for Algorithm 2

Considering the matching tree and flow graph in Figure 4, an example of FG construction algorithm is described in Table 3. The matching order of \( q_T \) is orchestrated as a sequence of query vertices \( u_1, u_2, u_3, u_4, \) and \( u_5 \) and the query vertex is marked as 1 if it is visited. In the first module of Algorithm 2, the following and candidate states of node pair are initialized as 0 and \(-1\), respectively. Since \( \langle v_7, u_2 \rangle \) is a negative node pair, \( CS(\langle v_7, u_2 \rangle) \) is marked as 0. The candidate states of other node pairs are marked as 1. In the second module, the node pair state is verified through the calculation of candidate and following states in the bottom-up matching order. Regarding node pair \( \langle v_7, u_2 \rangle \), satisfying \( u_1, p = u_2 \), it cannot find a candidate of \( u_1 \) that is adjacent to \( v_7 \), then the following state \( FS(\langle v_7, u_2 \rangle) \) is marked as 0. In the third module, the edges are inserted into \( FG \) and minimum sequential numbers of parent-nodes are assigned to child-nodes in top-down matching order. Regarding node pair \( \langle v_{11}, u_2 \rangle \), satisfying \( u_5, p = u_2 \) and \( C(u_2) = \{v_9, v_{10}, v_8\} \), the minimum numbers of \( v_5, v_6, \) and \( v_9 \) are transformed to \( v_{11} \), thus the sequential number of \( v_{11} \) are 1, 2, and 3. The detailed description of sequential number is described in Section 4.

### 4. Incremental Subgraph Matching on Flow Graph Index

In this section, a sequential subgraph matching approach of flow graph is first given to limit the search derivation of graph update stream. Then, the strategies of incremental subgraph matching and incremental maintenance are proposed based on the divided flow graph.

#### 4.1. Sequential Subdivision of Flow Graph Index

The sequential subdivision of flow graph divides a flow graph into multiple flow subgraphs and sequentially encodes the vertices of flow subgraphs. The flow graph is divided on the basic of candidates of root node in \( q_T \), described as \( FG(\forall v) \) and \( v \in C(u_0) \). Here, \( u_0 \) is an originating node of query vertices in matching order \( q_T \) and \( v \) is named as root candidate of root node \( u_0 \) in \( q_T \). Considering the flow graph in Figure 4(b), the divided flow subgraphs are denoted in Figure 5(a), described as \( FG(v_1), FG(v_2), \) and \( FG(v_3) \).

All subgraph mappings can be conducted on the traversal of flow subgraphs, defined in Theorem 1.

**Theorem 1.** All subgraph mappings of \( q \) on \( G \) must be included into one flow subgraph.

**Proof.** For Theorem 1, regarding a subgraph mapping \( M = \{\langle v_1, u_1 \rangle, \langle v_2, u_2 \rangle, \ldots, \langle v_m, u_m \rangle\} \), it must be found in a flow subgraph \( FG(v_1) \).

In the subdivision of flow graph, the root candidates of \( FG \) are sequentially arranged to identify the relationship of flow subgraphs through a unique encoding technology. The unique encoding of vertices can effectively avoid the
Lemma 1. Given an inserted node pair \((v, u')\) of FG, if \(v\) can conduct the new subgraph mappings, it should satisfy the condition, \(\forall u' \in \text{Des}(u), \exists v' \in F(v): \text{CS}(v', u') = 1\).

Proof. For Lemma 1, considering a subgraph mapping \(\{(v_0, u_0), (v_1, u_1), \ldots, (v_n, u_n)\}\) originating from candidate \(v_0\) of query vertex \(u_0\), if exists a descendant \(u'\) of \(u_0\), such that it cannot find a positive candidate of \(u'\), then a subgraph mapping cannot be conducted by node pair of \(u'\).

Benefit from sequential subdivision of FG, the first aspect is that it can avoid the repeated encoding of vertices in the following region. Regarding the sequential flow subgraphs \(FG(v_2)\) and \(FG(v_3)\), the follower \(v_{12}\) does not need to be encoded in \(FG(v_3)\) because \(v_{12}\) is a follower of \(v_2\) that has been encoded with a new number 3.

The second aspect is that it can previously verify the common flow subgraphs of inserted edges. Regarding an inserted edge \((v_6, v_4)\) in Figure 5(a), the incremental subgraph matching of \((v_6, v_4)\) does not need to be executed because \(v_6\) and \(v_4\) are included into different flow subgraphs, where \(v_6\) is located into \(FG(v_2)\) and \(FG(v_3)\) and \(v_4\) is located into \(FG(v_1)\).
verification of common flow subgraph of inserted edge is defined in Lemma 2.

Lemma 2. Given vertices \( v \) and \( v' \) of FG if \( v \) and \( v' \) are the followers of a common dominator, then \( v \) and \( v' \) are included into a common flow graph.

Proof. For Lemma 2, according to Theorem 1, all subgraph mappings of \( q \) on \( G \) must be included into one of flow subgraphs. Since a flow subgraph is composed of a root candidate and its followers, a flow subgraph at least contains one dominator, that is, root candidate. Thus, vertices \( v \) and \( v' \) are not included into a subgraph mapping if \( v \) and \( v' \) have not a common domination vertex.

The third aspect is that the deleted edges can block the dominating relationships of flow subgraphs. Regarding a deleted vertex \( v_4 \) in Figure 4(a), it cannot find a subgraph mapping consisting of \( v_5 \) after vertex \( v_4 \) is deleted. The influence of deleted vertex on dominating relationship is described in Lemma 3.

Lemma 3. Given a deleted vertex \( v' \) of flow subgraph FG (v), \( v' \) may block the dominating relationships of FG (v).

Proof. For Lemma 3, we prove the influence of deleted vertex on dominating relationship through an argument of contradiction. If the domination relationship can be remained, it must find a subgraph mapping composed of other vertices instead of the deleted vertex. However, a subgraph mapping consisting of a unique in subgraph matching and an answer containing a unique vertex are a common phenomenon in subgraph matching.

Benefit from Lemmas 1–3, the algorithm of incremental subgraph matching and incremental maintenance on FG are designed in Sections 4.2 and 4.3.

4.2. Incremental Maintenance. The incremental maintenance of FG employs a state transition model to effectively identify the influence of update candidate state on subgraph index of the flow graph and contribute to the incremental subgraph matching on the flow graph.

The state transition model consists of three candidate states (−1, 0, 1) and six transition rules (Transitions 1–6), which demonstrates the adjacent influence of changed candidate from one state to another one. The vertices of states 0 and 1 indicate the negative and positive candidates respectively that are included into previous FG. A vertex of state −1 denotes an inserted negative or positive candidate, which is not included into previous FG. The six transition rules describe the state changing of candidates as vertices are inserted and deleted in the graph update stream. Figure 6(a) describes the state transition model, where the solid lines denote the transition rules (Transitions 1–3) of deleted vertices and the dashed lines indicate the transition rules (Transitions 4–6) of inserted vertices.

Furthermore, we analyze the influence of six transition rules on the adjacent region of changed candidate state in FG. In order to reflect the impact of changed candidate state on structural characteristic of our flow
In this section, the incremental subgraph matching is given to conduct the subgraph mappings of $q$ on the initial graph $G_0$ and graph update stream $\Delta G_t$.

Two matching orders are designed to orchestrate the traversal sequence of node pairs in the initial graph $G_0$ and the graph update stream $\Delta G_t$. The matching order $q_t$ of initial graph is used to orchestrate the traversal order of flow graph construction and matching sequence of subgraph mappings. The matching order $q_t$ is fixed in the subgraph matching of initial graph, and the originating nodes of $q_t\triangleq$ are the candidates of root node in $q_t$.

The matching order $\Delta q_T$ of graph update stream is used to orchestrate matching sequence of subgraph mappings on the incremental maintenance of graph update stream. The matching order $\Delta q_T$ is changed by each active candidate in the subgraph matching of graph update stream. Given an active node pair $\langle v, u_2 \rangle$, it first traces the root candidate of $q_T$ in backward order and then traverses the nodes of other paths in forward order. Regarding an active node pair $\langle v, u_2 \rangle$ by transition rules 2 and 3, $\Delta q_T = \{u_2, u_1, u_4, u_5, v_3\}$ in Figure 4(a) and $u_1$ is the root of $q_T$.

The consistency of subgraph matching is guaranteed by two verifications of inserted node pairs, relational verification and sequential verification. The relational verification is to verify the local isomorphism of local subgraph mapping and selected node pair, as described in Verification 1.

**Verification 1 (relational consistency).** Given a partial subgraph mapping $\tilde{M}_{t-1}$ and a selected node pair $\langle v, u_2 \rangle$, data vertex $v$, is the relational consistency with query vertex $u_2$ if and only if it satisfies the following constraint: (1) State $\langle \langle v, u_2 \rangle \rangle = 1$, (2) $\forall u \in N(u_i) \cap \tilde{M}_{t-1}, \exists v \in N(v_i) \cap \tilde{M}_{t-1}$: $v \in C(u)$, and (3) $L_E(u, u_i) \leq L_E(v, v_i)$ (or $L_E(u, u_i) \leq L_E(v, v_i)$).

Here, $\langle \langle v, u_2 \rangle \rangle$ denotes the state of the node pair $\langle v, u_2 \rangle$. State $\langle \langle v, u_2 \rangle \rangle = 1$ is true if and only if $CS (\langle v, u_2 \rangle)$ is true and $\forall u' \in Des (u_i), \exists v' \in F (v_i)$: $CS (\langle v', u' \rangle) = 1$, according to Definition 5 and Lemma 1, that is, $State (\langle v, u_2 \rangle) = CS (\langle v, u_2 \rangle) \land FS (\langle v, u_2 \rangle)$. A subgraph mapping is composed of multiple node pairs, formed as $M$ and the number of node pairs is equivalent to the number of query vertices, denoted as $|M| = |V_q|$. Given a subgraph mapping.
\[ M = \{ \langle v_1, u_1 \rangle, \langle v_2, u_2 \rangle, \ldots, \langle v_n, u_n \rangle \} \] a partial subgraph mapping is a subset of sequential node pairs, defined as \[ \bar{M}_i = \{ \langle v_1, u_1 \rangle, \langle v_2, u_2 \rangle, \ldots, \langle v_i, u_i \rangle \}, \quad i \leq n \quad \text{and} \quad \bar{M}_i \subseteq M. \]

Here, \( v_i \) denotes a sequential vertex in matching tree \( q_T \).

The sequential verification is to verify the equivalence of sequential numbers between local subgraph mapping and selected node pair, as described in Verification 2.

**Verification 2** (sequential consistency). Given a partial subgraph mapping \( \bar{M}_{i-1} \) and a selected node pair \( \langle v_i, u_i \rangle \), data vertex \( v_i \) is the sequential consistency with query vertex \( u_i \) if and only if it satisfies the following constraint: \( \exists v \in N(v_i) \cap \bar{M}_{i-1}: SN(v_i) \cap SN(v) \neq \emptyset \).

Here, \( SN(v_i) \) refers to the sequential numbers of \( v_i \) and sequential numbers are transitively assigned by the number of root candidate. Regarding the sequential number 2 of \( v_{11} \) in Figure 5(b), there is a reachable path from \( v_{27} \) to \( v_{11} \).

Given vertices \( v \) and \( v' \) of \( FG \) if \( v \) and \( v' \) are the followers of different common dominators, then \( v \) and \( v' \) cannot conduct the subgraph mappings according to Lemma 2. Regarding sequential number 1 of \( v_{10} \) and sequential numbers 2 and 3 of \( v_{7} \) in Figure 5(b), it cannot find a subgraph mapping conducted by \( v_{7} \) and \( v_{10} \) because \( v_{7} \) and \( v_{10} \) are located into different sequential flow subgraphs \( FG \) (Algorithm 4).

An incremental subgraph matching is described in Algorithm 4. The inputs are a merged flow graph \( \Delta FG \) and a matching tree \( \Delta q_T \). The outputs are the subgraph mapping \( \mathcal{M} \) of \( \Delta q_T \) on \( FG \). The subgraph mappings are iteratively conducted if and only if \( i = |VT| \) (Lines 1–2).

One module is to iteratively conduct all subgraph mappings of initial graph (Lines 5–7). Another module is to iteratively conduct all subgraph mappings of graph update stream (Lines 11–14). \( SNValid(v, u) \) and \( RCVValid(v, u) \) are used to verify the relational and sequential consistencies of selected node pairs (Lines 6 and 12). The selected node pairs are inserted into the subgraph mapping \( M \) if the verifications of selected node pairs are valid (Lines 7 and 13). \( FG(u,successor) \) and \( \Delta FG(u,successor) \) are used to acquire the successor of \( u \) in \( q_T \) and \( \Delta q_T \) for traversing the node pairs of query vertices in forward order. \( FG(u,precursor) \) and \( \Delta FG(u,precursor) \) are used to backtrack the precursor of \( u \) in the backward order of \( q_T \) and \( \Delta q_T \), respectively.

### 5. Experimental Evaluation

We conduct extensive performance studies to evaluate our incremental subgraph matching (iPM). All the experiments are performed on an Intel Xeon E7520 processor with 12 MB of L3 cache. The system is equipped with 32 GB of main memory and it runs a 64 bit Linux 3.13.0 kernel.

5.1. Experimental Settings. The performance evaluation of algorithm mainly depends on two aspects, query graph and sliding window. The influencing factors of algorithm include Query Factor (QF), Query Shape Factor (QSF), and Data Window Factor (DWF).

Since flow graph index employs the structural feature and semantic label of query graph to pruning the non-candidates of the data graph, it is closely related to the size of the flow graph index. The size of the query graph is denoted as Query factor QF. The factor of query shape QSF refers to the shape of query graph, such as chain, star, cyclic and chain-star shapes [44], as described in Figure 7. The QSF is closely related to the density of adjacent structure. Considering the chain and star queries, the adjacency structure of star-shape query is more dense than the chain-shape query. Then, the star-shape query can prune the more non-candidates than chain-shape query because the query within denser structural feature can prune the more non-candidates than simple one in the original data graph.

Data Window Factor DWF illustrates the influence of sliding window on conducting subgraph mappings. Sliding window is associated with the changed size of graph update stream locked in the quantified size of initial data graph.

The impact factors of iPM are described in Table 4. The initial data graph contains the RDF data of size \( 1.0 \times 10^4 \), which is encapsulated into the quantified window in the initial traversal processing. The query graphs of different shapes (star, chain, and cycle) and scales (1–22 triple patterns) are used to evaluate the influence of query factors on conducting subgraph mappings. The analysis of initial data graph and subgraph mappings are illustrated in Figure 8.

5.1.1. DataSet. A real-world dataset and a synthetic one are used in this paper.

- (1) The NY Taxi Dataset 4 is a publicly available real-world dataset with total of 1 billion taxi related RDF stream data. The dataset contains 27 different measurement values for taxi fares, locations, trip distance, trip time, etc. A query graph can be corresponded to at most 24 triple patterns.

- (2) The Social Network Benchmark (SNB) [22] is a synthetic dataset, which contains social data distributed into streams of GPS, posts, comments, photos, and user profiles. The dataset collects the information of persons about their friendship network and content data of messages between persons, e.g., posts, comments, and likes. We generated a total of 50 million RDF triples containing data for 30,000 users. The scale of data graphs is extended from
**Input:** a merged flow graph $\Delta FG_i$ and a matching tree $\Delta q_T$.

**Output:** the subgraph mappings $M$ of $\Delta q_T$ on $\Delta FG_i$.

1. if $i \in |V_T|$ then
2. Output $M \rightarrow M$;
3. else
4. if $i = 0$ then
5. for each $\langle v, u \rangle \in C(u)$ do
6. if $SNValid (v, u)$ and $RCValid (v, u)$ then
7. $\langle v, u \rangle \rightarrow M$;
8. IncMatching ($FG (u.successor)$);
9. IncMatching ($FG (u.precursor)$);
10. else
11. for each $\langle v, u \rangle \in \Delta C (u)$ do
12. if $SNValid (v, u)$ and $RCValid (v, u)$ then
13. $\langle v, u \rangle \rightarrow M$;
14. IncMatching ($\Delta FG (u.successor)$)
15. IncMatching ($\Delta FG (u.precursor)$)

**Algorithm 4:** IncMatching ($\Delta FG_i, \Delta q_T$).

**Figure 7:** Different query shapes: (a) chain queries, (b) star queries, (c) cyclic queries, and (d) chain star queries.

**Table 4:** Impact factors.

| QS   | QSF (query edges) | DWF (RDF triples) |
|------|-------------------|-------------------|
| Chain, star, cyclic | 1–22             | 10,000            |

**Figure 8:** Analysis of (a) data graphs and (b) subgraph results.
Figure 9: The performance evaluation on SNB dataset. (a) Total matching time of star query. (b) Total matching time of chain query. (c) Total matching time of cycle query. (d) Matching time of star query. (e) Matching time of chain query. (f) Matching time of cycle query.
10,000 sets of data graphs, which are extracted randomly from SNB dataset and the size of data graphs are illustrated in Figure 8(a).

5.1.2. Query Graphs. The query graphs of different shapes (star, chain, and cycle) and scales (1–22 triple patterns) are designed to evaluate the matching influence of query on data graph.

(1) Query-Star. A star query refers to the graph containing an instance node with multiple attributes. Thus, the scale of star query depends on the number of attributes and the performance evaluation of query-star is presented in Figures 9(a) and 9(d).

(2) Query-Chain. A chain query refers to the graph containing multiple instance nodes linked in a line. Thus, the scale of chain query depends on the number of instance nodes and the performance evaluation of query-chain is presented in Figures 9(b) and 9(e).

(3) Query-Cycle. A cycle query refers to the graph containing multiple instance nodes linked in the form of cycles. The smallest cycle contains at least three instance nodes and three undirected edges. The processing of continuously embedding a query vertex and two edges into the cycle query is used to increasing the scale of query graph. And the performance evaluation of star-chain-cycle queries is presented in Figures 9(c) and 9(f).

5.2. Analysis of Algorithms. In this section, we mainly look at the total execution and traversal time-efficiency of iPM.

In the experimental evaluation, we focus on the count-based sliding window since it can be adapted to a time-based one using a simple transformation. The initial sliding window contains RDF data of size 10,000 which slides one data at a time. The performance evaluation of iPM is executed on the dataset containing 0.5 million RDF data (About 10,000 data graphs).

The compared algorithm of SPECTRA [27] is chosen to evaluating the experimental performance with our algorithm iPM because SPECTRA is a competitor of our methods, which employ a set of vertically partitioned views to collect the summarized data from each event, and sibling lists are employed to incrementally index the joined triples between views. The matched results are shown in a set of intermediate view for ease of enabling an incremental evaluation with the arrival of new events.

SPECTRA [27] is a competitor for comparison experiments with our methods. A prominent contribution for PM-S is the study entitled SPECTRA, in which a set of vertically partitioned views is used to collect the summarized data from each event and sibling lists are employed to incrementally index the joined triples between views. The matched results are shown in a set of final views, thus enabling an incremental evaluation with the arrival of new events.

The ten thousand data graphs are extracted from SNB dataset, which is described in Figure 8(a). The scales of most data graphs are located in the range of 15 to 20 RDF triples.

The performance evaluation of subgraph results is presented in Figure 8(b). The quantity of subgraph result is evaluated within a sliding window designed as the sliding interval of 500 RDF triples. In the trend of experimental graphs, the results are incremental increasing before \( x = 1 \) because RDF data is constantly filled into the fixed window in the initial execution processing. Then, the experimental graph is presented by a wavy line because the subgraph results are incrementally produced with graph stream updates.

The total matching time of star, chain, and cycle queries are measured through different quantities of triple patterns, which are presented in Figures 9(a)–9(c), respectively. In the trend of experimental graphs, our methods (iPM) have a...
more significant advantage than SPECTRA (SPE) in star and cycle queries. As the quantity of triple patterns increases, iPM approximates a linear growth trend, while SPE is closer to the exponential growth trend.

The traversal time of star, chain, and cycle queries are measured through different quantities of triple patterns, which are presented in Figures 9(d) – 9(f), respectively. In the trend of experimental graphs, the total matching time of iPM increases first and then decreases as the scale of query graph enlarges, while SPE is closer to the linear or exponential growth trend. The variant traversal time indicates that massive RDF triples are filtered through candidate verification. Thus, the structure and label of query graph are beneficial to reduce the noncandidates in the flow graph index.

The performance evaluations on NY taxi dataset are described in Figures 10(a) and 10(b). Figure 10(a) depicts the trend of different sliding size on intermediate results. The influenced trend can find a most suitable sliding size for continuous subgraph matching. In NY taxi dataset, the most suitable sliding size is 200.

Figure 10(b) depicts the matching time with different query scales coupled with a most suitable sliding size. In the trend of experimental graphs, the matching time is increasing first and then decreasing as the quantity of triple patterns enlarges. Intuitively, the massive RDF data is filtered after $x = 18$.

The experimental results show that our methods are able to address the complex graph (i.e., star and cycle queries) and large datasets. Meanwhile, our method also provides better benefits with chain query.

6. Conclusions

In this paper, a flow graph index is first proposed to pruning the noncandidates of query vertices. The flow graph FG is a directed multigraph, which is constructed from the initial data graph $G_0$ and guided by a matching order of query graph. Then, a sequential subdivision technology of the flow graph is employed to limit the search derivation of incremental subgraph matching. The sequential numbers of root candidates are assigned to the vertices of divided flow graphs and limit the search space of originating changed candidate of FG. For incrementally conducting the subgraph mappings, a state transition model is first used to illustrate the transition state of changed candidates, which consists of three states and six transition rules. Based on the state transition model, we analyze the influence of changed candidates to adjacent region and design our incremental maintenance strategy. Then, an incremental subgraph matching algorithm is executed on the sequential divided flow graph. The consistency of subgraph matching is guaranteed by two verifications of selected candidates, relational and sequential verifications. Finally, extensive empirical studies on real and synthetic graphs demonstrate that our techniques outperform the state-of-the-art algorithms.

Data Availability

The NY Taxi data used to support the findings of this study have been deposited in the repository http://chriswhong.com/open-data. Previously reported Social Network Benchmark (SNB) data were used to support this study and are available at DOI: 10.1145/2723372.2742786. These prior studies are cited at relevant places within the text as references [22, 27].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant no. 61976032.

References

[1] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, New York, NY, USA, 1979.
[2] M. S. Islam, C. Liu, and J. Li, "Efficient answering of why-not questions in similar graph matching," IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 10, pp. 2672–2686, 2015.
[3] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, “c-sparql: a continuous query language for rdf data streams,” International Journal of Semantic Computing, vol. 4, no. 1, pp. 3–25, 2010.
[4] S. Komazec, D. Cerri, and D. Fensel, “Sparkwave: continuous schema-enhanced pattern matching over RDF data streams,” in Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, pp. 58–68, Berlin, Germany, July 2012.
[5] T. Cai, J. Li, A. S. Mian, R. Li, T. Sellis, and J. X. Yu, “Target-aware holistic influence maximization in spatial social networks,” IEEE Transactions on Knowledge and Data Engineering, p. 1, 2020.
[6] Y. Wang, Y. Yuan, Y. Ma, and G. Wang, “Time-dependent graphs: definitions, applications, and algorithms,” Data Science and Engineering, vol. 4, no. 4, pp. 352–366, 2019.
[7] C. Weiss, P. Karras, and A. Bernstein, “Hexastore,” Proceedings of the VLDB Endowment, vol. 1, no. 1, pp. 1008–1019, 2008.
[8] J. Pérez, M. Arenas, and C. Gutierrez, “Semantics and complexity of SPARQL,” ACM Transactions on Database Systems, vol. 34, no. 3, pp. 16–45, 2009.
[9] D. J. Abadi, A. Marcus, S. R. Madden, K. Hollenbach, and “ SW-Store, “SW-Store: a vertically partitioned DBMS for Semantic Web data management,” The VLDB Journal, vol. 18, no. 2, pp. 385–406, 2009.
[10] J. Broekstra, A. Kampman, and F. V. Harmelen, “Sesame: a generic architecture for storing and querying rdf and rdf schema,” in Proceedings of the International Semantic Web Conference, pp. 54–68, Sardinia, Italy, June 2002.
[11] G. Ramalingam and T. W. Reps, “A categorized bibliography on incremental computation,” in Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 502–510, Charleston, CA, USA, January 1993.
[12] W. Fan, X. Wang, and Y. Wu, “Incremental graph pattern matching,” ACM Transactions on Database Systems, vol. 38, no. 3, pp. 18–47, 2013.
18 Complexity

J. R. Ullmann, "An algorithm for subgraph isomorphism," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1820–1834, 2016.

X. Song, J. Li, S. Sun et al., "SEPN: a sequential engagement based academic performance prediction model," *IEEE Intelligent Systems*, vol. 99, 2020.

Y. Sun, G. Li, J. Du, B. Ning, and H. Chen, "A subgraph matching algorithm based on subgraph index for knowledge graph," *Frontiers of Computer Science*, 2020.

J. Z. Pan, "Resource description framework," *Handbook on Ontologies*, pp. 71–90, Springer, Berlin, Germany, 2009.

D. L. Phuoc, J. X. Parreira, M. Hausenblas, and M. Hauswirth, *Continuous Query Optimization and Evaluation over Unified Linked Stream Data and Linked Open Data*, Technical Report, DERI, Galway, Ireland, 2010.

J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th International Conference on World Wide Web*, pp. 74–83, New York, NY, USA, May 2004.

T. Neumann and G. Weikum, "RDF-3X: a RISC-style engine for RDF," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 647–659, 2008.

E. I. Chong, S. Das, G. Eadon, and J. Srinivasan, "An efficient SQL-based RDF querying scheme," in *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 1216–1227, Trondheim, Norway, August 2005.

O. Erling, A. Averbuch, J. Larriba-Pey et al., "The LDBC social network benchmark: Interactive workload," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 619–630, Melbourne, Australia, May 2015.

J. J. Levandoski and M. F. Mokbel, "RDF data-centric storage," in *Proceedings of the IEEE International Conference on Web Services*, pp. 911–918, Los Angeles, CA, USA, July 2009.

M. Sintek and M. Kiesel, "RDFBroker: a signature-based high-performance RDF store," in *Proceedings of the European Semantic Web Conference*, pp. 363–377, Budva, Montenegro, June 2006.

L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold, "Column-store support for RDF data management," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1553–1563, 2008.

R. Avnur and J. M. Hellerstein, "Eddies: continuously adaptive query processing," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 261–272, Dallas, TX, USA, May 2000.

S. Gillani, G. Picard, and F. Laforest, "SPECTRA: continuous query processing for RDF graph streams over sliding windows," in *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, pp. 1–17, Budapest, Hungary, July 2016.

J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, 1976.

L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

P. Zhao and J. Han, "On graph query optimization in large networks," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 340–351, 2010.

W. S. Han, J. Lee, J. H. Lee, and "Turboiso," "Towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 337–348, New York, NY, USA, June 2013.

X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 617–628, 2015.

F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1199–1214, San Francisco, CA, USA, June 2016.

J. Kim, H. Shin, W.-S. Han, S. Hong, and H. Chafi, "Taming subgraph isomorphism for RDF query processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1238–1249, 2015.

X. Chen, J. Xu, R. Zhou et al., "S2R-tree: a pivot-based indexing structure for semantic-aware spatial keyword search," *GeoInformatica*, vol. 24, no. 1, pp. 3–25, 2020.

Y. Ma, Y. Yuan, G. Wang, X. Bi, Z. Wang, and Y. Wang, "Rising star evaluation based on extreme learning machine in geo-social networks," *Cognitive Computation*, vol. 12, no. 1, pp. 296–308, 2020.

S. Xu, R. Zhang, W. Cheng, and J. Xu, "MTLM: a multi-task learning model for travel time estimation," *GeoInformatica*, 2020.

Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach," in *Proceedings of the IEEE 35th International Conference on Data Engineering*, pp. 1478–1489, Macao, China, April 2019.

J.-R. Gao, W. Chen, J.-J. Xu et al., "An efficient framework for multiple subgraph pattern matching models," *Journal of Computer Science and Technology*, vol. 34, no. 6, pp. 1185–1202, 2019.

Y. Yuan, X. Lian, G. Wang, L. Chen, Y. Ma, and Y. Wang, "Weight-constrained route planning over time-dependent graphs," in *Proceedings of the IEEE 35th International Conference on Data Engineering*, pp. 914–925, Maco, China, April 2019.

S. Choudhury, L. B. Holder, K. Agarwal, and J. Feo, "A selectivity based approach to continuous pattern detection in streaming graphs," in *Proceedings of the 18th International Conference on Extending Database Technology*, pp. 157–168, Brussels, Belgium, March 2015.

G. Sun, G. Liu, Y. Wang, M. A. Orgun, and X. Zhou, "Incremental graph pattern based node matching," in *Proceedings of the IEEE 34th International Conference on Data Engineering*, pp. 281–292, Paris, France, April 2018.

D. Qin, X. Zhou, L. Chen, G. Huang, and Y. Zhang, "Dynamic connection-based social group recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 3, pp. 453–467, 2020.

A. Maduko, K. Anyanwu, A. P. Sheth, and P. Schiekelman, "Estimating the cardinality of RDF graph patterns," in *Proceedings of the 16th International Conference on World Wide Web*, pp. 1233–1234, Banff, Canada, May 2007.