Research Article

Incremental Graph Pattern Matching Algorithm for Big Graph Data

Lixia Zhang¹ and Jianliang Gao²

¹College of Mathematics and Computer Science, Key Laboratory of High Performance Computing and Stochastic Information Processing, Ministry of Education of China, Hunan Normal University, Changsha 410081, China

²School of Information Science and Engineering, Central South University, Changsha 410083, China

Correspondence should be addressed to Jianliang Gao; gaojianliang@csu.edu.cn

Received 19 October 2017; Accepted 20 December 2017; Published 22 January 2018

Academic Editor: Longxiang Gao

Copyright © 2018 Lixia Zhang and Jianliang Gao. 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.

Graph pattern matching is widely used in big data applications. However, real-world graphs are usually huge and dynamic. A small change in the data graph or pattern graph could cause serious computing cost. Incremental graph matching algorithms can avoid recomputing on the whole graph and reduce the computing cost when the data graph or the pattern graph is updated. The existing incremental algorithm PGC_IncGPM can effectively reduce matching time when no more than half edges of the pattern graph are updated. However, as the number of changed edges increases, the improvement of PGC_IncGPM gradually decreases. To solve this problem, an improved algorithm iDeltaP_IncGPM is developed in this paper. For multiple insertions (resp., deletions) on pattern graphs, iDeltaP_IncGPM determines the nodes' matching state detection sequence and processes them together. Experimental results show that iDeltaP_IncGPM has higher efficiency and wider application range than PGC_IncGPM.

1. Introduction

Graph pattern matching is to find all the subgraphs that are the same or similar to a given pattern graph P in a data graph G . It is widely used in a number of applications, for example, web document classification, software plagiarism detection, and protein structure detection [1–3].

With the rapid development of Internet, huge amounts of graph data emerge every day. For example, the Linked Open Data Project, which aims to connect data across the Web, has published 149 billion triples until 2017 [4]. In addition, real-world graphs are dynamic [5]. It is often cost-prohibitive to recompute matches starting from scratch when G or P is updated. An incremental matching algorithm is needed, which aims to minimize unnecessary recomputation by analyzing and computing the changes of matching result in response to updates ΔG (resp., ΔP) to G (resp., P).

For example, Figure 1(a) is a pattern graph P and Figure 1(b) is a data graph G . The subgraph which is composed of A_1, B_1, C_1, D_1, E_1 , and the edges between them (for simplicity, denoted as $\{A_1, B_1, C_1, D_1, E_1\}$) is the only matching subgraph. Assuming that (B, E) and (C, D) are

removed from the pattern graph, the traditional recomputing algorithm will compute the matches for the new pattern graph on the whole data graph. It is time consuming. The incremental algorithm will just check a part of nodes in G , that is, B_2, B_3, C_2, C_3, A_2 , and A_3 , and add new matching subgraphs $\{A_2, B_2, C_2, D_2, E_2\}$, $\{A_3, B_3, C_3, D_3, E_3\}$ to the original matching result.

At present, the study of incremental graph pattern matching is still in its infancy and existing work [6–12] mainly focuses on the updates of data graphs. In our previous study, we proposed an incremental graph matching algorithm named PGC_IncGPM, which can be used in scenarios where data graphs are constant and pattern graphs are updated [13]. PGC_IncGPM can effectively reduce the runtime of graph matching as long as the number of changed edges is less than the number of unchanged edges in P . However, the improvement effect of PGC_IncGPM gradually decreases as the number of changed edges increases. In this paper, the bottleneck of PGC_IncGPM is further analyzed. An optimization method of nodes' matching state detection sequence is proposed, and a more efficient algorithm called iDeltaP_IncGPM is designed and implemented.

FIGURE 1: An example of incremental graph pattern matching.

Using Figure 1 as an example, suppose (B, E) and (C, D) are deleted from the pattern graph. PGC_IncGPM algorithm will first consider the deletion of (B, E) , that is, checking B_2 , A_2 , B_3 , and A_3 , and then consider the deletion of (C, D) , that is, checking C_2 , B_2 , A_2 , C_3 , B_3 , and A_3 . Thus B_2 , A_2 , B_3 , and A_3 are all checked twice. iDeltaP_IncGPM considers the two deletions together; C_2 , B_2 , A_2 , C_3 , B_3 , and A_3 are all checked only once.

The remainder of this paper is organized as follows. In Section 2, related work is reviewed. The model and definition are described in Section 3. In Section 4, our algorithm is presented. Section 5 is experimental results and comparison, and Section 6 presents the conclusion.

2. Related Work

We surveyed related work in two categories: graph pattern matching models and incremental algorithms for graph matching on massive graphs.

Graph pattern matching is typically defined in terms of subgraph isomorphism [14, 15]. However, subgraph isomorphism is an NP-complete problem [16]. In addition, subgraph isomorphism is often too restrictive because it requires that the matching subgraphs have exactly the same topology as the pattern graph. These hinder its applicability in emerging applications such as social networks and crime detection. Thus, graph simulation [17] and its extensions [18–22] are adopted for pattern matching. Graph simulation preserves the labels and the child relationship of a graph pattern in its match. In practical applications, graph simulation is so loosely that it may produce a large number of useless matches, which can flood useful information. Dual simulation [18] enhances graph simulation by imposing an additional condition, to preserve both child and parent relationships (downward and upward mappings). Due to the good balance and high practical value of dual simulation in response time and effectiveness, graph pattern matching is defined as dual simulation in this paper.

At present, the study of incremental graph pattern matching is still in its infancy; existing work [6–12] mainly focuses on the updates of data graphs. Fan et al. proposed the incremental graph simulation algorithm IncMatch [6, 7]. Sun et al. studied the Maximal Clique Enumeration problem on

dynamic graph [8]. Stotz et al. studied incremental inexact subgraph isomorphic problem [9]. Wang and Chen proposed an incremental approximation graph matching algorithm, which transformed the approximate subgraph search into vector space relation detection [10]. When inserting or deleting on the data graph, the vectors of relevant nodes are modified and whether the new vectors still contain the vector of the pattern graph is rechecked. Choudhury et al. developed a fast matching system StreamWorks for dynamic graphs [11]. The system can real-time detect suspicious pattern graphs and early warn high-risk data transfer modes on constantly updated network graphs. Semertzidis and Pitoura proposed an approach to find the most durable matches of an input graph pattern on graphs that evolve over time [12]. In [13], an incremental graph matching algorithm was proposed for updates of pattern graphs.

In big data era [23], graph computing is widely used in different fields such as social networks [24], sensor networks [25, 26], internet-of-things [27, 28], and cellular networks [29]. Therefore, there is urgent demand for improving the performance of big graph processing, especially graph pattern matching.

3. Model and Definition

For graph pattern matching, pattern graphs and data graphs are directed graphs with labels. Each node in graphs has a unique label, which defines the attribute of the node (such as keywords, skills, class, name, and company).

Definition 1 (graph). A node-labeled directed graph (or simply a graph) is defined as $G = (V, E, L)$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, and L is a function that map each node u in V to a label $L(u)$; that is, $L(u)$ is the attribute of u .

Definition 2 (graph pattern matching). Given a pattern graph $P = (V_p, E_p, L_p)$ and a data graph $G = (V, E, L)$, P matches G if there is a binary relation $R \subseteq V_p \times V$, such that

- (1) if $(u, v) \in R$, then $L_p(u) = L(v)$;
- (2) $\forall u \in V_p$, there exists a node v in G such that $(u, v) \in R$ and (a) $\forall (u, u') \in E_p$, there exists an edge $(v, v') \in E$

such that $(u', v') \in R$; (b) $\forall (u'', u) \in E_p$, there exists an edge $(v'', v) \in E$ such that $(u'', v'') \in R$.

Condition (2)(a) ensures that the matching node v keeps the child relationship of u ; condition (2)(b) ensures that v maintains the parent relationship of u .

For any P and G , there exists a unique maximum matching relation R_M . Graph pattern matching is to find R_M , and the result graph G_r is a subgraph of G that can represent R_M .

Considering a real-life example, a recruiter wants to find a professional software development team from social network. Figure 2(a) is the basic organization graph of a software development team. The team consists of the following staffs with identity: project manager (PM), database engineer (DB), software architecture (SA), business process analyst (BA), user interface designers (UD), software developer (SD), and software tester (ST). Each node in the graph represents a person, and the label of node means the identity of person. The edge from node A to node B means that B works well under the supervision of A. A social network is shown in Figure 2(b). In this example, R_M is $\{(DB, DB_1), (PM, PM_1), (SA, SA_1), (BA, BA_1), (UD, UD_1), (SD, SD_1), (SD, SD_2), (ST, ST_1), (ST, ST_2)\}$. Because BA_2 does not have a child matching UD and SA_2 does not have a parent matching DB, PM_2 does not keep the child relationship of PM. For the same reason, SD_3 (resp., ST_3) does not match SD (resp., ST).

Definition 3 (incremental graph pattern matching for pattern graph changing). Given a data graph G and a pattern graph P , the matching result in G for P is $M(P, G)$. Assuming that P changes ΔP , the new pattern graph is expressed as $P \oplus \Delta P$. As opposed to batch algorithms that recompute matches starting from scratch, an incremental graph matching algorithm aims to find changes of ΔM to $M(P, G)$ in response to ΔP such that $M(P \oplus \Delta P, G) = M(P, G) \oplus \Delta M$.

When ΔP is small, ΔM is usually small as well, and it is much less costly to compute than to recompute the entire set of matches. In other words, this suggests that we compute matches once on the entire graph via a batch-matching algorithm and then incrementally identify new matches in response to ΔP without paying the cost of the high complexity of graph pattern matching.

In order to get ΔM quickly, indexes can be prebuilt based on the selected data features of graphs to reduce the search space during incremental matching. The more indexes, the shorter the time to get ΔM and the larger the space to store indexes. For large-scale data graphs, both response time and storage cost are needed to be reduced. Considering the balance of storage cost and response time, in this paper, three kinds of sets generated in the process of graph matching are used as index. (1) First are candidate matching sets $cand(\cdot)$; for each node u in P , $cand(u)$ includes all the nodes in G which only have the same label with u . The nodes in $cand(\cdot)$ are called c -nodes. (2) The second are child matching sets $sim(\cdot)$; for each node u in P , $sim(u)$ includes all the nodes in G which preserve the child relationship of u . The nodes in $sim(\cdot)$ are called s -nodes. (3) The third are complete matching sets $mat(\cdot)$; for each node u in P , $mat(u)$ includes all the nodes in

G which preserve both the child and parent relationship of u . The nodes in $mat(\cdot)$ are called m -nodes.

The symbols used in this paper are shown in Notions Section.

4. iDeltaP_IncGPM Algorithm

In this section, we propose the improved incremental graph pattern matching algorithm for pattern graph changing (ΔP).

4.1. The Idea of PGC_IncGPM Algorithm. The basic framework of PGC_IncGPM [13] is shown in Figure 3.

The graph pattern matching algorithm (GPMS) is first performed on the entire data graph G for the pattern graph P . It computes the matching result graph G_r and creates the index needed for subsequent incremental matching. ΔP may include edge insertions (E^+) and edge deletions (E^-). Incremental graph pattern matching algorithm PGC_IncGPM first calls the subalgorithm AddEdges for E^+ to get G'_r and $index'$ and then calls the subalgorithm SubEdges for E^- to get G''_r and $index''$. G'_r is the new matching result $M(P \oplus \Delta P, G)$, and $index''$ is the new index that can be used for subsequent incremental matching if the pattern graph changes again.

Edge insertions (resp., edge deletions) in ΔP are processed one by one by AddEdges (resp., SubEdges). For example, when deleting multiple edges from P , the processing of PGC_IncGPM is as follows.

In the first step, the following operations are performed for each deleted edge (u, u') : for each $v \in cand(u)$, whether v keeps the child relationship of u in $P \oplus \Delta P$ is checked. If v keeps the child relationship of u , then v is removed from $cand(u)$ to $sim(u)$ and the parents of v in $cand(\cdot)$ are also processed.

In the second step, each node in $sim(\cdot)$ is repeatedly filtered according to its parents and children; the new generated m -nodes are added to $mat(\cdot)$.

In the first step, when deleting (u, u') from P , some nodes in $cand(u)$ and $cand(u'')$ (u'' is an ancestor of u) may change from c -nodes to s -nodes. So when a c -node becomes an s -node, a bottom-up approach is used to find its parents and ancestors from $cand(\cdot)$. If (u_1, u'_1) and (u_2, u'_2) are deleted, and u_1 and u_2 have a common ancestor u' , then $cand(u')$ will be visited twice. In summary, there is a bottleneck of PGC_IncGPM for multiple deleted edges. There is the same problem for multiple inserted edges.

4.2. Optimization for Matching State Detection Sequence. Since PGC_IncGPM deals with edge insertions (resp., deletions) one by one, the efficiency of it gradually decreases as the number of changed edges increases. To overcome the bottleneck of PGC_IncGPM, multiple edge insertions (resp., deletions) should be considered together. In this paper, the optimization method for nodes' matching state detection sequence is proposed. The optimization can be applied to both insertions and deletions on P .

Taking SubEdges as an example, the optimization method is as follows.

FIGURE 2: An example of graph pattern matching.

FIGURE 3: Basic framework of PGC_IncGPM algorithm.

First, analyze all edges deleted from P to determine which nodes' candidate matching sets may change. If $cand(u)$ may change, then u is added to $filterorder^-$ set.

Secondly, $filterorder^-$ is sorted by the inverse topological sequence of P . There may be some strong connected components in P . In this case, we first find out all the strong connected components in P and, then, converge each strong connected component into a node to get a directed acyclic graph P' and find the inverse topological sequence of P' ; finally, we replace the strong connected component convergence node with the original node set. Thus, the approximate inverse topological sequence of P is obtained.

Finally, for each u in $filterorder^-$, $cand(u)$ is processed in turn. Depending on whether there is a deleted edge from u , two different filtering methods are used: (1) if u has at least one out-edge to be deleted, then each node in $cand(u)$ is likely to keep the child relationship of u now. So whether they keep the child relationship of u should be checked; (2) if u does not have an out-edge be deleted, then only part of the nodes in $cand(u)$ are needed to be checked. That is, a node in $cand(u)$ will be checked only if it has at least one child which changes from c -node to s -node.

The visited times of some candidate matching sets can be reduced through the above optimization.

4.3. iDeltaP_IncGPM Algorithm. Based on the optimization method proposed in Section 4.2, iDeltaP_IncGPM is proposed. It uses the optimized method for both multiple inserted edges and multiple deleted edges. The optimization algorithm for edge deletions is shown in Algorithm 1. In Algorithm 1, $nodes^-$ contains all the nodes which have out-edge deleted. For a node u in P , if the changes of P may

result in some nodes in $cand(u)$ becoming s -nodes, then $u \in filterorder^-$. $Filterorder^-$ is sorted by the inverse topological sequence of P (lines (1)–(5)). If u has an out-edge removed, that is, $u \in nodes^-$, then all the nodes in $cand(u)$ need to be checked whether they keep the child relationship of u (lines (7)–(12)). If $u \in filterorder^-$ and u is not in $nodes^-$, then only part of nodes in $cand(u)$ are checked. That is, if w has a child w' and w' is moved from $cand(u')$ to $sim(u')$ ($w' \in snw(u')$), then whether w is still an s -node will be checked (lines (14)–(20)).

Here we use an example to illustrate the implementation process of PGC_IncGPM and iDeltaP_IncGPM. The pattern graph P is shown in Figure 4, assuming that (E, H), (G, I), and (C, G) are deleted from P .

The process of PGC_IncGPM is as follows. (1) the deletion of (E, H) is processed, and each w in $cand(E)$ is checked whether it keeps the child relationship of u in $P \oplus \Delta P$. If w keeps the child relationship of u , then its parents founded from $cand(B)$ (resp., $cand(C)$) are checked. If these nodes keep the child relationship of B (resp., C), then they are removed to $sim(B)$ (resp., $sim(C)$). After that, their parents founded from $cand(A)$ are checked; (2) the deletion of (G, I) is processed, and the nodes in $cand(G)$, $cand(C)$, $cand(D)$, and $cand(A)$ are checked in turn; (3) the deletion of (C, G) is processed, and the nodes in $cand(C)$ and $cand(A)$ are checked in turn. From the above steps, it can be seen that $cand(C)$ and $cand(A)$ are visited three times, $cand(G)$, $cand(D)$, $cand(E)$, and $cand(B)$ are visited once.

The process of iDeltaP_IncGPM is as follows: because of the deletion of (E, H), (G, I), and (C, G), some nodes in $cand(E)$, $cand(G)$, $cand(C)$, $cand(B)$, $cand(D)$, and $cand(A)$ may become s -nodes. The nodes in $cand(\cdot)$ are checked by


```

(1)  $nodes^- = \Phi$ ;  $filterorder^- = \Phi$ ;
(2) for each deleted edge  $e = (u, u')$  do
(3)    $nodes^- = nodes^- \cup \{u\}$ ;
(4)    $filterorder^- = filterorder^- \cup \{u \text{ and all ancestor nodes of } u\}$ ;
(5)   sort  $filterorder^-$  according to the inverse topological of  $P$ ;
(6)   for each node  $u$  in  $filterorder^-$  do
(7)     if  $u \in nodes^-$  then
(8)       for each  $w \in cand(u)$  do
(9)         check if  $w$  keeps the child relationship of  $u$ ;
(10)        if  $w$  keeps the child relationship of  $u$  then
(11)           $sim(u) = sim(u) \cup \{w\}$ ;  $cand(u) = cand(u) \setminus \{w\}$ ;
(12)           $snew(u) = snew(u) \cup \{w\}$ ;
(13)        else
(14)          for each  $w \in cand(u)$  do
(15)            if there exist  $w'$  which is a child of  $w$  such that  $(u, u') \in E_p, w' \in snew(u')$  then
(16)              check if  $w$  keeps the child relationship of  $u$ ;
(17)              if  $w$  keeps the child relationship of  $u$  then
(18)                 $sim(u) = sim(u) \cup \{w\}$ ;
(19)                 $cand(u) = cand(u) \setminus \{w\}$ ;
(20)                 $snew(u) = snew(u) \cup \{w\}$ ;
(21) repeatedly filter  $sim(\cdot)$  according to the parent and child relationships of nodes in the
    subgraph constructed by  $sim(\cdot)$  to get added  $mat(\cdot)$  and updated  $sim(\cdot)$  and  $cand(\cdot)$ ;

```

ALGORITHM 1: iDeltaP_IncGPM for edge deletions.

FIGURE 4: An example for pattern graph changing.

the order $\{G, E, D, C, B, A\}$. That is, the nodes in $cand(G)$ are checked first, and the nodes in $cand(A)$ are checked at last. E, G, and C all have out-edges deleted, so all the nodes in their candidate matching sets are checked. For the nodes in $cand(B)$, $cand(D)$, and $cand(A)$, only if they have a child changing from c -node to s -node, they will be checked. Therefore, $cand(C)$, $cand(D)$, $cand(A)$, $cand(G)$, $cand(E)$, and $cand(B)$ are only visited once. In other words, the optimized scheme reduces the visited times of $cand(\cdot)$.

For multiple edges inserted to the pattern graph, the similar optimization method is adopted. $nodes^+$ contains all the source nodes of inserted edges. If some nodes in $sim(u)$ may become c -nodes because of edge insertions, then u is in $filterorder^+$. $filterorder^+$ is ordered by the reverse topological sequence of the pattern graph. $nodes^+$ and $filterorder^+$ are used to reduce the visited times of $sim(\cdot)$ and $mat(\cdot)$.

5. Experiments and Results Analysis

The following experiments evaluate our proposed algorithm. Runtime is used as a key assessment of algorithms. In addition, in order to show the effectiveness of incremental algorithms visually, improvement ratio (IR) is proposed, which is the ratio of runtime saved by incremental matching algorithms to the runtime of ReComputing algorithm. Two real data sets (Epinions and Slashdot [30]) are used for experiments. The former is a trust network with 75879 nodes and 508837 edges. The latter is a social network with 82168 nodes and 948464 edges. In previous work, we experimented with normal size and large size pattern graphs, respectively, and the results show that the complexity and effectiveness of incremental matching algorithm are not affected by the size of pattern graph. Therefore, in this paper, by default, the number of nodes in P ($|V_p|$) is 9, the original number of edges in P ($|E_p|$) is 8 (resp., 16) for insertions (resp., deletions) and 9 for both insertions and deletions.

In order to evaluate the improvement of our proposed algorithm, iDeltaP_IncGPM, PGC_IncGPM, and ReComputing are all performed on Epinions and Slashdot under different settings. Each experiment was performed 5 times with different pattern graphs, and the average results are reported here. The experimental results are shown in Figure 5. The histogram represents the runtime of algorithm, and the line chart represents the improvement ratio of iDeltaP_IncGPM and PGC_IncGPM to ReComputing.

Figure 5(a) (resp., Figure 5(b)) shows the runtime of three algorithms over Epinions (resp., Slashdot) for insertions on pattern graphs. The X-axis represents the number of insertions on P , “+2” represents that two edges are inserted to P , “+4” represents four edges are inserted

FIGURE 5: The runtime of different algorithms when pattern graph changed.

to P , and so on. The figure tells us the following: (a) when insertions are no more than 10, the runtime of PGC_IncGPM and iDeltaP_IncGPM is significantly shorter than that of ReComputing, and iDeltaP_IncGPM has the shortest runtime; (b) when insertions are 12 (new inserted edges account for 60% of the edges in $P \oplus \Delta P$), the runtime of PGC_IncGPM is longer than that of ReComputing, while iDeltaP_IncGPM still gets the shortest runtime; (c) the improvement ratio of iDeltaP_IncGPM and PGC_IncGPM decreases with the increase of edge insertion, but the decrease of iDeltaP_IncGPM is smaller. The more inserted edges, the better iDeltaP_IncGPM than PGC_IncGPM. When 12 edges are inserted to P , the IR of iDeltaP_IncGPM is 40% on average, and the IR of PGC_IncGPM is 33% on average. Therefore, iDeltaP_IncGPM is better than PGC_IncGPM. The

reason is that PGC_IncGPM processes the inserted edges one by one. Therefore, as insertion increases, its runtime grows almost linearly. However, iDeltaP_IncGPM integrates all the inserted edges, analyzes which matching sets are affected, and processes them in the appropriate order. This will prevent some matching sets to be processed repeatedly, which will shorten the running time.

Figure 5(c) (resp., Figure 5(d)) shows the runtime of three algorithms over Epinions (resp., Slashdot) for deletions on pattern graph. The X-axis represents the number of deletions on P , “-2” represents that two edges are deleted from P , “-4” represents four edges are deleted from P , and so on. It can be seen that (a) when deletion changes from 2 to 12, the runtime of all three algorithms increases, and iDeltaP_IncGPM always has the shortest runtime; (b) as the deletion increases, the IR

of PGC_IncGPM decreases and the IR of iDeltaP_IncGPM slowly increases. For 12 deletions, the IR of PGC_IncGPM decreases to 7% on average, while the IR of iDeltaP_IncGPM increases to 78% on average. The reason is that as the deletion increases, the runtime of ReComputing increases dramatically, while the runtime of iDeltaP_IncGPM increases a little. iDeltaP_IncGPM is better than PGC_IncGPM because it compositely processes deleted edges and its runtime does not increase linearly as the number of deleted edges increases.

Figure 5(e) (resp., Figure 5(f)) shows the runtime of three algorithms over Epionions (resp., Slashdot) for both insertions and deletions on pattern graph. The X -axis represents the number of insertions and deletions on P , “+2-2” means that two edges are inserted to P and the other two edges are removed from P , and so on. As shown in the figure, iDeltaP_IncGPM always has shorter runtime than the others do.

In conclusion, iDeltaP_IncGPM effectively improves the efficiency of PGC_IncGPM through the optimization strategy. For the same ΔP , the runtime of iDeltaP_IncGPM is shorter, and as $|\Delta P|$ increases, the runtime increases less; the decrease of IR is also more moderate. Therefore, iDeltaP_IncGPM can be applied to larger changes of the pattern graph, and it has a wider range of applications.

6. Conclusion

In this paper, we analyze PGC_IncGPM to find its efficiency bottleneck and propose a more efficient incremental matching algorithm iDeltaP_IncGPM. Multiple insertions (resp., deletions) are considered together and the optimization method for nodes’ matching state detection sequence is used. Experimental results on real data sets show that iDeltaP_IncGPM has higher efficiency and wider application range than PGC_IncGPM.

Next, we will study the distributed incremental graph matching algorithm. Real-life graphs grow rapidly in size and hyper-massive data graphs cannot be centrally stored in one data center and need to be distributed across multiple data centers. It is very worthy studying how to make efficient incremental matching on distributed large graphs.

Notations

P/G :	Pattern graph/data graph
u/u' :	Nodes in P
v/v' :	Nodes in G
ΔP :	Changes of P
ΔG :	Changes of G
$P \oplus \Delta P$:	New pattern graph
$cand(u)$:	Nodes in G that have same label with u but do not keep child relationship of u
$sim(u)$:	Nodes in G that only keep child relationship of u
$mat(u)$:	Nodes in G that keep child and parent relationships of u
index:	The sets including $cand(u)$, $sim(u)$ and $mat(u)$

$c/s/m$ -node: v in G such that

$$v \in cand(u)/sim(u)/mat(u)$$

$M(P, G)$: The maximum match in G for P

Gr : The result graph, a subgraph represents $M(P, G)$.

Conflicts of Interest

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

References

- [1] X. Ren and J. Wang, “Multi-query optimization for subgraph isomorphism search,” *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 121–132, 2016.
- [2] Z. Yang, A. W.-C. Fu, and R. Liu, “Diversified top-k subgraph querying in a large graph,” in *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data, SIGMOD 2016*, pp. 1167–1182, USA, July 2016.
- [3] J. Gao, B. Song, W. Ke, and X. Hu, “BalanceAli: Multiple PPI Network Alignment With Balanced High Coverage and Consistency,” *IEEE Transactions on NanoBioscience*, vol. 16, no. 5, pp. 333–340, 2017.
- [4] A. Jentzsch, “Linked Open Data Cloud,” in *Linked Enterprise Data*, X.media.press, pp. 209–219, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
- [5] Y. Hao, G. Li, P. Yuan, H. Jin, and X. Ding, “An Association-Oriented Partitioning Approach for Streaming Graph Query,” *Scientific Programming*, vol. 2017, pp. 1–11, 2017.
- [6] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu, “Incremental graph pattern matching,” in *Proceedings of ACM SIGMOD and 30th PODS 2011 Conference*, pp. 925–936, Greece, June 2011.
- [7] W. Fan, C. Hu, and C. Tian, “Incremental Graph Computations,” in *Proceedings of ACM International Conference*, pp. 155–169, Chicago, Illinois, USA, May 2017.
- [8] S. Sun, Y. Wang, W. Liao, and W. Wang, “Mining Maximal Cliques on Dynamic Graphs Efficiently by Local Strategies,” in *Proceedings of IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 115–118, San Diego, CA, USA, April 2017.
- [9] A. Stotz, R. Nagi, and M. Sudit, “Incremental graph matching for situation awareness,” in *Proceedings of 12th International Conference on Information Fusion, FUSION 2009*, pp. 452–459, usa, July 2009.
- [10] C. Wang and L. Chen, “Continuous subgraph pattern search over graph streams,” in *Proceedings of the 25th IEEE International Conference on Data Engineering, ICDE 2009*, pp. 393–404, China, April 2009.
- [11] S. Choudhury, L. Holder, G. Chin, A. Ray, S. Beus, and J. Feo, “Streamworks - A system for dynamic graph search,” in *Proceedings of ACM SIGMOD Conference on Management of Data, SIGMOD 2013*, pp. 1101–1104, USA, June 2013.
- [12] K. Semertzidis and E. Pitoura, “Durable Graph Pattern Queries on Historical Graphs,” in *Proceedings of International Conference on Data Engineering*, pp. 541–552, October 2016.
- [13] L. X. Zhang, W. P. Wang, J. L. Gao, and J. X. Wang, “Pattern graph change oriented incremental graph pattern matching,” *Journal of Software. Ruanjian Xuebao*, vol. 26, no. 11, pp. 2964–2980, 2015.

- [14] X. Ren and J. Wang, "Exploiting Vertex Relationships in Speeding up Subgraph Isomorphism over Large Graphs," in *Proceedings of the 3rd Workshop on Spatio-Temporal Database Management, Co-located with the 32nd International Conference on Very Large Data Bases, VLDB 2006*, pp. 617–628, kor, September 2006.
- [15] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing Cartesian products," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 1199–1214, USA, July 2016.
- [16] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, 1976.
- [17] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu, "Graph pattern matching," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 264–275, 2010.
- [18] W. Fan, "Graph pattern matching revised for social network analysis," in *Proceedings of the 15th International Conference on Database Theory, ICDT 2012*, pp. 8–21, deu, March 2012.
- [19] J. Gao, Q. Ping, and J. Wang, "Resisting re-identification mining on social graph data," *World Wide Web-Internet and Web Information Systems*, 2017.
- [20] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, "Capturing topology in graph pattern matching," *Proceedings of the VLDB Endowment*, vol. 5, no. 4, pp. 310–321, 2011.
- [21] A. Fard, M. U. Nisar, L. Ramaswamy, J. A. Miller, and M. Saltz, "A distributed vertex-centric approach for pattern matching in massive graphs," in *Proceedings of IEEE International Conference on Big Data, Big Data 2013*, pp. 403–411, USA, October 2013.
- [22] Y. Liang and P. Zhao, "Similarity Search in Graph Databases: A Multi-Layered Indexing Approach," in *Proceedings of IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 783–794, San Diego, CA, USA, April 2017.
- [23] X. Liu, Y. Liu, H. Song, and A. Liu, "Big Data Orchestration as a Service Network," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 94–101, 2017.
- [24] J. Gao, J. Wang, J. He, and F. Yan, "Against Signed Graph Deanonimization Attacks on Social Networks," *International Journal of Parallel Programming*.
- [25] Q. Zhang and A. Liu, "An unequal redundancy level-based mechanism for reliable data collection in wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, article 258, 2016.
- [26] J. Gao, J. Wang, P. Zhong, and H. Wang, "On Threshold-Free Error Detection for Industrial Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics*, pp. 1–11.
- [27] Y. Xu, A. Liu, and C. Changqin, "Delay-aware program codes dissemination scheme in internet of everything, mobile information systems," *Mobile Information Systems*, vol. 2016, Article ID 2436074, 18 pages, 2016.
- [28] X. Liu, S. Zhao, A. Liu, N. Xiong, and A. V. Vasilakos, "Knowledge-aware Proactive Nodes Selection approach for energy management in Internet of Things," *Future Generation Computer Systems*, 2017.
- [29] K. Zhou, J. Gui, and N. Xiong, "Improving cellular downlink throughput by multi-hop relay-assisted outband D2D communications," *EURASIP Journal on Wireless Communications and Networking*, vol. 2017, no. 1, 2017.
- [30] Stanford Large Network Dataset Collection, <http://snap.stanford.edu/data/index.html>.