ISSN No: 2454-9614

Issues for Graph Pattern Queries using Views

S. Jagadheesh Ram

Department of Computer Science and Engineering, K.S.Rangasamy College of Technology, Namakkal, Tamilnadu, India.

*Corresponding Author: S.Jagadheesh Ram

E-mail: jaggu.11892@gmail.com

Received: 10/11/2015, Revised: 14/12/2015 and Accepted: 10/03/2015

Abstract:

Graph queries using views has proven effective for querying relational and semi structured data. This paper investigates this issue for graph pattern queries based on graph simulation. We propose a notion of pattern containment to characterize graph pattern matching using graph pattern views. We show that a pattern query can be answered using a set of views if and only if it is contained in the views. Based on this characterization, we develop efficient algorithms to answer graph pattern queries. We also study problems for determining (minimal, minimum) containment of pattern queries. We establish their complexity (from cubic-time to NP complete) and provide efficient checking algorithms (approximation when the problem is intractable). In addition, when a pattern query is not contained in the views, we study maximally contained rewriting to find approximate answers; we show that it is in cubic-time to compute such rewriting, and present a rewriting algorithm. We experimentally verify that these methods are able to efficiently answer pattern queries on large real-world graphs.

Keywords: Graph pattern queries, pattern containment, views.

Reviewed by* **ICETSET'16 *organizing committee*

1. Introduction

Graph queries using views has been extensively studied for relational data and semi-structured data. Given a query Q and a set V¼ fV1; . . . ;of views, the idea is to find another query A such that A is equivalent to Q, and A only refers to views in V. If such a query A exists, then given a database D, one can compute the answer Q to Q in D by using A, which uses only the data in the materialized views, without accessing D. This is particularly effective when D is "big" and/or distributed. Indeed, views have been advocated for scale independence, to query big data "independent of" the size of the underlying data. They are also useful in data integration, data warehousing, semantic caching, and access control. The need for studying this problem is even more evident for graph pattern queries (a.k.a. graph pattern matching). Graph pattern queries have been increasingly used in social network analysis, among other things. Real-life social graphs are typically large, and are often distributed. For example, Facebook has more than 1:26 billion users with 140 billion links, and the data is geo-distributed to various data centres.

One of the major challenges for social network analysis is how to cope with the sheer size of real life social graphs when evaluating graph pattern queries. Graph pattern matching using views provides an effective method to query such big data. For example, a fraction of a recommendation network is depicted as a graph G , where each node denotes a person with name and job title (e.g., project manager (PM), database administrator (DBA), programmer (PRG), business analyst (BA) and software tester (ST)); and each edge indicates collaboration/recommendation relation, e.g., (Bob, Dan) indicates that Dan worked well with Bob, on a project led by Bob.

To build a team, one issues a pattern query Qs depicted, to find a group of PM, DBA and PRG. It requires thatDBA1 and PRG2 worked well under the project manager PM; and each PRG (resp. DBA) had been supervised by a DBA (resp. PRG), represented as a collaboration cycle in Qs. For pattern matching based on graph simulation, the answer to Qs in G can be denoted as a set of pairs, such that for each pattern edge e in Qs, Se is a set of edges (a match set) for e in G. For example, pattern edge and has a match set Se ¼ , in which each edge satisfies the node labels and connectivity constraint of the pattern edge. It is known that it takes OðjQsj2 time to compute it, where it is the size of G (resp. Qs). For example, to identify the match set of each pattern edge (for i2 ½1; 2_), each pair of (DBA, PRG) in G has to be checked, and moreover, a number of join operations have to be performed to eliminate invalid matches.

This is a daunting cost when G is big. One can do better by leveraging a set of views. Suppose that a set of views V ¼ fV1; V2g is defined, materialized and cached (¼ fV1ðGÞ; V2ðGÞg) and it will be shown later to compute it, we only need to visit views in without accessing the original big graph G; and it can be efficiently computed by "merging" views in Indeed, the views already contains partial answers to Qs in G: for each pattern edge e in Qs, the matches of e (e.g., ðDBA1;) are contained either in V1ðGÞ or V2ðGÞ (e.g., the matches of e3 in V2). These partial answers can be used to construct the complete match. Hence, the cost of computing and is in quadratic time in it and it is where it is much smaller than |G|.

This example suggests that we conduct graph pattern matching by capitalizing on available views. To do this, several questions have to be settled. How to decide whether a pattern query Qs can be answered by a set V of views? If so, how to efficiently compute it from there. If not, how to find approximate answers to QsðGÞ by using. In both cases, which views in V should we choose to answer Qs?-Contributions. This paper investigates these questions for graph pattern queries using graph pattern views. We focus on graph pattern matching defined in terms of graph simulation, since it is commonly used in social community detection, biological analysis **[20],** and mobile network analyses. While conventional sub graph isomorphism often fails to capture meaningful matches, graph simulation fits into emerging applications with its "many-to-many" matching semantics. Moreover, it is more challenging since graph simulation is "recursively defined" and has poor data locality.

1) Relational data. Query processing using views has been extensively studied for relational data. It is known that for SPC (conjunctive) queries, query graph and rewriting using views are intractable. For the containment problem, the well-known homomorphism theorem shows that an SPC query is contained in

another if and only if there exists a homomorphism between the tableaux representing the queries, and it is NP-complete to determine the existence of such a homomorphism. Moreover, the containment problem is un-decidable for relational algebra.

2) XML. There has also been a host of work on processing XML queries using views. In, the containment of simple X-Path queries is shown co NP-complete. When disjunction, DTDs and variables are taken into account, the problem ranges from co NP-complete to EXPTIME-complete to un-decidable for various X Path classes. In containment and query rewriting of XML queries are studied under constraints expressed as a structural summary. For tree pattern queries (a fragment of X-Path), they have studied maximally contained rewriting.
3) Semi-structure data. Views defined in Lorel are studied in, which are quite different from graph patterns considered here. View-based query rewriting for regular path queries (RPQs) is shown PSPACE complete and an EXPTIME rewriting algorithm. The containment problem is shown un-decidable for RPQs under constraints and for extended conjunctive RPQs.
4) RDF. An EXPTIME query rewriting algorithm is given for SPARQL. It is shown that query containment is in EXPTIME for PSPARQL, which supports regular expressions. There has also been work on evaluating SPARQL queries on RDF based on cached query results. Our work differs from the prior work in the following, we study query graph using views for graph pattern queries via graph simulation, which are quite different from previous settings, from complexity bounds to processing techniques. We show that the containment problem for the pattern queries is in PTIME, in contrast to its intractable counterparts for e.g., SPC, XPath, RPQs and SPARQL. We study a more general form of query containment between a query Qs and a set of queries, to identify an equivalent query for Qs that is not necessarily a pattern query. The high complexity of previous methods for query graph using views hinders their applications in the real world. In contrast, our algorithms have performance guarantees and yield a practical method for querying real-life social networks.

Our work differs from the prior work in the following,

(1) We study query graph using views for graph pattern queries via graph simulation, which are quite different from previous settings, from complexity bounds to processing techniques.

(2) We show that the containment problem for the pattern queries is in PTIME, in contrast to its intractable counterparts for e.g., SPC, XPath, RPQs and SPARQL.

(3) We study a more general form of query containment between a query Qs and a set of queries, to identify an equivalent query for Qs that is not necessarily a pattern query.

2. Existing System

2.1 Graphs, Patterns, and Views

We first review pattern queries and graph simulation. We then state the problem of pattern matching using views.

2.1.1 Data Graphs and Graph Pattern Queries

Data graphs. A data graph is a directed graph G = ðV; E;LÞ, where V is a finite set of nodes; E _ V _ V, in which ðv; v0Þ denotes an edge from node v to v0; and L is a function such that for each node v in V , LðvÞ is a set of labels from an alphabet S. Intuitively, L specifies the attributes of a node, e.g., name, keywords, blogs and social roles.

Pattern queries: A graph pattern query, denoted as Qs, is a directed graph Qs = ðVp; Ep; fvÞ, where Vp and Ep are the set of pattern nodes and the set of pattern edges, respectively; and fv is a function defined on Vp such that for each node uVp, fvðuÞ is a label in S. We remark that fv can be readily extended to specify search conditions in terms of Boolean predicates. Graph pattern matching. We say that a data graph G ¼ ðV; E;LÞ matches a graph pattern query Qs ¼ ðVp; Ep; fvÞ via simulation, denoted by Qs E G, if there exists a binary relation S _ Vp_ V , referred to as a match in G for Qs, such that_ for each pattern node u 2 Vp, there exists a node v 2 V such that ðu; vÞ 2 S, referred to as a match of u; and_ for each pair ðu; vÞ 2 S, LðvÞ; and moreover, for each pattern edge e ¼ ðu; u0Þ in Ep, there exists an edge ðv; v0Þ in E, referred to as a match of e in S, such that ðu0; v0Þ S, i.e., v0 is a match of u0.When Qs E G, there exists a unique maximum match So in G for Qs.

We derive fðe;SeÞ j e Epg from So, where Se is the set of all matches of e in So, referred to as the match set of e. Here Seisnonempty for all e 2 Ep. We define the result of Qs in G, denoted as QsðGÞ, to be the unique maximum set fðe; SeÞ j e Epgif Qs E G, and let QsðGÞ¼ ; otherwise, We define the size of query Qs, denoted by jQsj, to be the total number of nodes and edges in Qs; we define the size jQsðGÞj of QsðGÞ to be the total edge number of sets Se for all edges in Qs.

2.1.2 Graph Pattern Matching Using Views

We next formulate the problem of graph pattern matching using views. We study views V defined as a graph pattern query, and refer to the query result VðGÞ in a data graph G as the view extension for V in G or simply as a view.

Given a pattern query Qs and a set V ¼ fV1; . . . ; Vng of view definitions, graph pattern matching using views is to find another query A such that for all data graphs G,A only refers to views Vi 2 V and their extensions VðGÞ ¼ fV1ðGÞ; VnðGÞg in G, and A is equivalent to Qs. If such a query A exists, we say that Qs can be answered using views V.

In contrast to query rewriting using views, here A isnot required to be a pattern query,depicts a set V ¼ fV1; V2g of view definitions and their extensions VðGÞ ¼ fV1ðGÞ; V2ðGÞg. To answer the query Qs,we want to find a query A that computes QsðGÞ by using only V and VðGÞ, where A is not necessarily a graph pattern.

2.2 Characterization

A characterization of graph pattern matching using views, i.e., a sufficient and necessary condition for deciding whether a pattern query can be answered by using a set of views. We also provide a quadratic-time algorithm for graph pattern queries using views.

2.2.1 Pattern containment

We first introduce a notion of pattern containment, by extending the traditional notion of query containment to a set of views. Consider a pattern query Qs ¼ ðVp; Ep; fvÞand a set V ¼ fV1; . . . ; Vngof view definitions, where Vi¼ ðVi; Ei; fiÞ. We say that Qs is contained in V, denoted by Qs v V, if there exists a mapping _ from Ep to power set PðS i2½1;n_EiÞ, such that for all data graphs G, the match set Se _ S e02_ðeÞSe0 for all edges e 2 Ep. The analysis involves query Qs and view definitions V, independent of data graphs G and view extensions VðGÞ.

This suggests an approach to graph pattern queries, as follows. Given a pattern Qs and a set V of views, we first efficiently determine whether Qs v V(by using the algorithm);if so, for all (possibly big) graphs G, we compute QsðGÞ by using VðGÞ instead of G, in quadratic-time in size of VðGÞ, which is much smaller than G. Here we prove Theorem 2,

I) we first prove the Only If condition, i.e., if Qs can be answered using V, then Qs v V. We show this by contradiction. Assume that Qs can be answered using V, while Qs 6v V. By the definition of containment, there must exist some data graph Go such that for all the possible mappings _, there always exists at least one edge e in Qs such that Se 6_ S e02_ðeÞSe0. Consider the following two cases.

(1) When QsðGoÞ¼ ;.

By Lemma1, for all e in Qs, Se ¼ ; in Go and hence it contradicts to the assumption that Se 6_ S e02_ðeÞSe0 .

(2) When QsðGoÞ6¼ ;.

If so, there must exist at least one edge eo in Go such that eo is in Se for some edge e in Qs, but it is not in Se0 for any e0 2 _ðeÞ. That is, eo cannot be included in Se0 for any e0 2 _ðeÞ, for all possible .This contradicts the assumption that Qs can be answered using only V and VðGoÞ, since at least the edge eo is missing from VðGoÞ for some graph Go, no matter how _ is defined. Therefore, Qs can be answered using V only if Qs vV.

II) We next show the If condition of Theorem 2**(1)** with a constructive proof: we give an algorithm to evaluate Qs using VðGÞ, if Qs v V. We verify theorem 2 by showing that the algorithm is in OðjQsjjVðGÞj+ jVðGÞj2Þ time.

2.3 Algorithm

We next present the algorithm that evaluates Qs using V. The algorithm, denoted as MatchJoin,it takes as input (1) a pattern query Qs and a set of view definitions V ¼ fVij i2 ½1; n_g, (2) a mapping _ for Qs v Vview extensions VðGÞ ¼ fViðGÞ j i2 ½1; n_g. The merge process iteratively identifies and removes those edges that are not matches of Qs, until a fixpointis reached and QsðGÞis correctly computed.More specifically, MatchJoinworks as follows. It starts with empty match sets Se for each pattern edge e.MatchJoinsets Se asS e02_ðeÞSe0 , where Se0 is extracted from VðGÞ,following the definition of _ðeÞ. It then performs a fix point computation to remove all invalid matches from Se. For each pattern edge ep ¼ ðu; u0Þ with its match set Sep changed, it checks whether the change propagates to the “parents” (i.e., u00 with edge ðu00; uÞ) of u.

2.4 Determining Pattern Containment

We prove Theorems 4, 6 and 7 by providing effective (approximation) algorithms for checking pattern containment, minimal containment and minimum containment.

2.4.1 Pattern Containment

With a proof of Theorem 4, i.e., whether Qs v V can be decided in time. To do this, we first propose a sufficient and necessary condition to characterize pattern containment. We then develop a cubic time algorithm based on the characterization. Sufficient and necessary condition. To characterize pattern containment, we introduce a notion of view matches. Consider a pattern query Qs and a set V of view definitions. For each V 2 V, let VðQsÞ ¼ fðeV; SeVÞ j eV 2 Vg, by treating Qs as a data graph. Obviously, if V E Qs, then Se V is the nonempty match set of eV for each edge eV of V. We define the view match from V to Qs, denoted by MQs V, to be the union of SeV for all eV in V. The result below shows that view matches yield a characterization of pattern containment.

2.4.2 Minimal Containment Problem

We now prove Theorem 6 by presenting an algorithm that, given Qs and V, finds a minimal subset V0 of V containing Qs in jQsjjVjÞ time if Qs v V.

2.4.3 Algorithm

The algorithm, denoted as minimal. Given a pattern query Qs and a set V of view definitions, it returns either a nonempty subset V0 of V that minimally contains Qs, or; to indicate that Qs 6v V.

Algorithm minimal initializes **(1)** an empty set V0 for selected views**, (2)** an empty set S for view matches of V0, and (3) an empty set E for edges in view matches. It also maintains an index M that maps each edge e in Qs to a set of views (line 1). Similar to algorithm contain, minimal first computes MQs Vi for all Vi 2 V (lines 2-7).

In contrast to contain that simply merges the view matches, it extends S with a new view match MQsVi only if MQsVi contains a new edge not in E, and updates M accordingly (lines 4-7). The for loop stops as soon as E ¼ Ep (line 7), as Qs is already contained in V0. If E 6¼ Ep after the loop, it returns ;(line 8), since Qs is not contained is V (Proposition 8). The algorithm then eliminates redundant views, by checking whether the removal of Vj causes MðeÞ¼ ; for some e 2 MQs Vj(line 10). If no such e exists, it removes Vj from V0. After all view matches are checked, minimal returns V0.

1) Algorithm minimal repeats the for loop at most card ðVÞ times, and in each iteration it computes view matches and adds a view definition Vi to a result set V0. It then performs the redundant checking (lines 9-11) to remove all redundant view definitions, if there exists any. As V0 is a finite set,and its size is monotonically decreasing, the algorithm always terminates.
2) We show that minimal only removes “redundant” view definitions. (a) Each time it computes the view match for a view definition Vi, and it adds Vi to V0 only if the corresponding match set of Vi can cover edges in Qs that have not been covered yet (line 4). Hence when the for loop terminates, one can verify that either the union of the view matches from V0 covers Ep, which indicates that V0 contains Qs, or Qs 6v V, following Proposition 8. (b) A view definition Vj is removed from V0 only when there already exist other view definitions in V0 “covering” every pattern edge e 2 MQsVj. Thus, minimal only removes redundant view definitions.

3) When algorithm minimal terminates with Qs v V, for any view definition V in V0, there exists at least an edge e that can only be introduced by V to cover Ep. By Proposition 8, this indicates that Qs 6v V nfVg for any V 2 V. Thus minimal returns a minimal set that contains Qs.

2.4.4 Maximally Contained Rewriting

When a pattern query Qs is not contained in a set V of views, we want to identify a maximal part Qs0 of Qs that can be answered by using V, referred to as a maximally contained rewriting of Qs using V. As will be seen shortly, given a graph G, Qs0 helps us approximately answer Qs in G, or compute exact answers QsðGÞ by additionally accessing a small fraction of the data in G.

A pattern query Qs0 is a sub query of Qs, denoted as Qs0 _ Qs, if it is an edge induced sub-graph of Qs, i.e., Qs0 is a sub graph of Qs consisting of a subset of edges of Qs, together with their endpoints as the set of nodes. Query Qs0 is called a contained rewriting of Qs using a set V of view definitions if,

- Qs0 _ Qs, i.e., Qs0 is a sub query of Qs, and
- Qs0 v V, i.e., Qs0 can be answered using V.

Acc¼ 2 _ ðrecall_ precisionÞ=ðrecallþ precision where recall = #true matches found#true matches , and precision = #true matches found #matches .

Here #matches is the number of all (edge) matches found by Qs0ðGÞ using views, #true matches is the number of all matches in QsðGÞ; and #true matches found is the number of all the true matches in both Qs0ðGÞ and QsðGÞ.

Initially, a high precision means that many matches in Qs0ðGÞ are true matches, and a high recall means Qs0ðGÞ contains most of the true matches in QsðGÞ. The larger Acc that can be induced by Qs0, the better. If Qs0 is equivalent to Qs, i.e., Qs0ðGÞ = QsðGÞ for all G, Acc takes the maximum value 1:0. Observe that for any edge e in Qs, if e is covered by Qs0, then for any G, the match set Se of e in QsðGÞ is a subset of the match set S0eof e in Qs0ðGÞ; that is, Qs0ðGÞ finds all candidate matches of e in G.

Computing maximally contained rewriting. It is known that finding maximally contained rewriting is intractable for SPC queries. In contrast, maximally contained rewriting can be efficiently found for graph pattern queries.

3. Experimental Evaluation

Using real-life data, we conducted four sets of experiments to evaluate the efficiency and scalability of algorithm Match Join for graph pattern matching using views; the effectiveness of optimization techniques for MatchJoin; the efficiency and effectiveness of (minimal, minimum)containment checking; and the efficiency, accuracy and scalability of our query-driven approximation scheme, using maximally contained rewriting.

3.1 Experimental setting

We used four real-life graphs: (a) Amazon, aproduct co-purchasing network with 548 K nodes and1:78 M edges. Each node has attributes such as title, group and sales-rank, and an edge from product x to y indicates that people who buy x also buy y. (b) Citation **[49],** a DAG (directed acyclic graph) with 1:4 M nodes and 3 M edges, in which nodes represent papers with attributes such as title, authors, year and venue, and edges denote citations. YouTube **[50],** a recommendation network with 1:6 M nodes and 4:5 M edges. Each node is a video with attributes such as category, age and rate, and each edge from x to y indicates that y is in the related list of x. (d) Web Graph, a web graph including 118:1Mnodes and 1:02 B edges, where each node represents a web page with id and domain. Pattern and view generator. We implemented a generator for graph pattern queries, controlled by three parameters: the number jVpj of pattern nodes, the number jEpj of pattern edges, and label fv from an alphabet S of labels taken from corresponding real-life graphs. We use ðjVpj; jEpjÞ to denote the size of a pattern query. We generated a set of 12 view definitions for each real-life dataset. For Amazon, we generated 12 frequent patterns following, where each view extension contains on average

5 K nodes and edges. The views take 14:4 percent of the space of the Amazon dataset. For Citation, we designed 12 views to search for papers and authors in computer science. The view extensions account for 12 percent of the Citation graph. We generated 12 views, to find videos on Youtube, where each node is associated with a Boolean condition, specified by e.g., age (A), length (L),category (C), rate (R) and visits (V). Each view extension has about 700 nodes and edges, accounting for 4 percent of Youtube. On Web Graph, we designed 12 views to search Web pages, where the view extensions account for 11 percent of Web Graph.

4. Implementation

We implemented the following algorithms, all in Java:contain, minimum and minimal for checking pattern containment; maximal for finding the maximally contained rewriting; Match, MatchJoinmin and Match Joinmnl for computing matches of patterns in a graph, where Match is the matching algorithm without using views and MatchJoinmin

(resp. MatchJoinmnl) revises MatchJoin by using a minimum (resp. minimal) set of views; an algorithm Match Joinmax for approximately graph pattern queries, which invokes MatchJoin to evaluate maximally contained rewriting using views and a version of MatchJoinmin without using the ranking optimization, denoted by MatchJoinnopt.

4.1 Scalability

Using Web Graph, we evaluated the scalability of MatchJoinmin, MatchJoinmnl and Match. Fixing jQsj ¼ ð4; 6Þ, we varied jGj by using scale factors from 0:1(0:1 times of original graph size) to 1:0. The results are reported, from which we can see the following MatchJoinmin scales best with jGj, and is 1.73 times faster than MatchJoinmnl. This verifies that evaluating pattern queries by using less view significantly reduces computation time. The results are consistent.

5. Optimization techniques

Varying the size of DAG(resp. cyclic) patterns, we evaluated the effectiveness of the optimization strategy, by comparing the performance of MatchJoinmin and MatchJoinnopt on Citation (resp. WebGraph).The MatchJoinmin is more efficient than MatchJoinnopt for all the patterns. For example, MatchJoinmin is 1.46 (resp. 1.66) times faster than MatchJoinnopt for DAG (resp. cyclic) patterns on average. The improvement becomes more substantial when jQsj gets larger. This is because for larger patterns, the bottom-up strategy used in MatchJoinmin can eliminate redundant matches more quickly. The optimization strategy works even better on denser big graphs, since more invalid matches can be removed by the strategy. This explains why MatchJoinmin works better on Web Graph than on Citation, since Web Graphis denser than Citation.

5.1 Query containment

We evaluated the efficiency of pattern containment checking w.r.t. query complexity.

5.2 Accuracy

We report the accuracy of MatchJoinmax on Web Graph and Youtube, respectively. We found the following. MatchJoinmax finds approximate answers with high accuracy. The Acc is 0.73 (resp. 0.65) on Web Graph(resp. Youtube) on average. The accuracy of MatchJoinmax is not sensitive to the pattern size; instead, it is determined by how much a maximally contained rewriting "covers" the pattern query. For example, we found that the accuracy of MatchJoinmax is on average 0.63 when the rewriting "missed" two edges in the pattern query, and it increases to 0.82 when only one query edge is missed.

6. Proposed System

We have proposed a notion of pattern containment to characterize what pattern queries can be answered using views, and provided such an efficient matching algorithm. We have also identified three fundamental problems for pattern containment, established their complexity, and developed effective (approximation) algorithms. When a pattern query is not contained in available views, we have developed efficient algorithms for computing maximally contained rewriting using views to get approximate answers. Our experimental results have verified the efficiency and effectiveness of our techniques. These results extend the study of query graph using views from relational and XML queries to graph pattern queries.

Techniques such as adaptive and incremental query expansion may apply. Another issue concerns view-based pattern matching via sub graph isomorphism. The third topic is to find a subset V0of V such that V0ðGÞ is minimum for all graphs G. Finally, to find a practical method to query "big" social data, one needs to combine techniques such as view-based, distributed, incremental, and compression methods.

7. Conclusion

We have studied graph simulation using views, from theory to algorithms. We have proposed a notion of pattern containment to characterize what pattern queries can be answered using views, and provided such an efficient matching algorithm. We have also identified three fundamental problems for pattern containment, established their complexity and developed effective (approximation) algorithms. When a pattern query is not contained in available views, we have developed efficient algorithms for computing maximally contained rewriting using views to get approximate answers. Our experimental results have verified the efficiency and effectiveness of our techniques. These results extend the study of query graph using views from relational and XML queries to graph pattern queries.

Our techniques can be readily extended to variants of graph simulation. Take strong simulation as example, MatchJoin only needs to check, for each pattern edge ðu0; uÞ and its match ðv0; vÞ in S, whether for each pattern edge ðu00; u0Þ, there is a match ðv00; v0Þ, with time complexity unchanged.

The study of graph pattern matching using views is still in its infancy. One issue is to decide what views to cache such that a set of frequently used pattern queries can be answered by using the views. Techniques such as adaptive and incremental query expansion may apply. Another issue concerns view-based pattern matching via Sub graph isomorphism. The third topic is to find a subset V0 of V such that V0ðGÞ is minimum for all graphs G. Finally, to find a practical method to query “big” social data, one needs to combine techniques such as view-based, distributed, incremental, and compression methods.

References

[1] A. Y. Halevy, “Graph queries using views: A survey,” VLDB J., vol. 10, no. 4, pp. 270–294, 2001.

[2] M. Lenzerini, “Data integration: A theoretical perspective,” in Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Principles DatabaseSyst., 2002, pp. 233–246.

[3] L.V.S. Lakshmanan, W. H. Wang, and Z. J. Zhao, “Graph tree pattern queries using views,” in Proc. 32nd Int. Conf. VeryLarge Data Bases, 2006, pp. 571–582.

[4] J. Wang, J. Li, and J. X. Yu, “Graph tree pattern queries using views: A revisit,” in Proc. 14th Int. Conf. Extending Database Technol., 2011, pp. 153–164.

[5] X. Wu, D. Theodoratos, and W. H. Wang, “Graph XML queries using materialized views revisited,” in Proc. 18th ACMConf. Inf. Knowl. Manag., 2009, pp. 475–484.

[6] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi,“View-based query processing and constraint satisfaction,” in Proc. 15th Annu. IEEE Symp. Logic Comput. Sci., 2000, p. 361.

[7] Y. Papakonstantinou and V. Vassalos, “Query rewriting for semistructureddata,”inProc. ACM SIGMOD Int. Conf. Manag. Data,1999, pp. 455–466.

[8] Y. Zhuge and H. Garcia-Molina, “Graph structured views and their incremental maintenance,” in Proc. 14th Int. Conf. Data Eng.,1998, pp. 116–125.

[9] M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D.A.Patterson, “Generalized scale independence through incremental precomputation,”in Proc. ACM SIGMOD Int. Conf. Manag. Data,2013, pp. 625–636.

[10] [10] W. Fan, F. Geerts, and L. Libkin, “On scale independence for querying big data,” in Proc. 33rd ACM SIGMOD-SIGACT-SIGARTSymp. Principles Database Syst., 2014, pp. 51–62.