Bounded Pattern Matching Using Views

Xin Wang[1], Yang Wang[1], Ji Zhang[2], Yan Zhu[3]

[1] Southwest Petroleum University, China
email:xinwang.ed@gmail.com, wangyang@swpu.edu.cn
[2] University of Southern Queensland, Australia email:Ji.Zhang@usq.edu.au
[3] Southwest Jiaotong University, China email:yzhu@swjtu.edu.cn

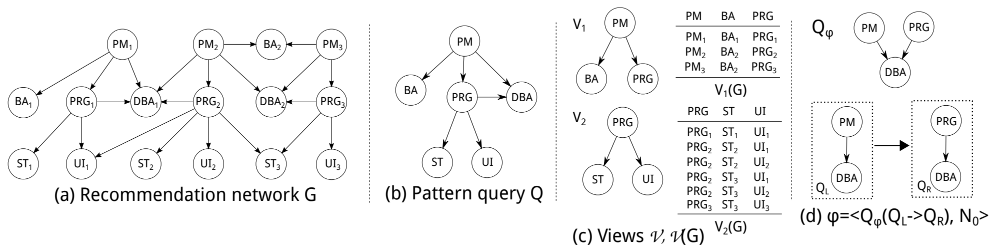
Abstract. Bounded evaluation using views is to compute the answers $Q(\mathcal{D})$ to a query Q in a dataset $\mathcal{D}$ by accessing only cached views and a small fraction D_Q of $\mathcal{D}$ such that the size $|D_Q|$ of D_Q and the time to identify D_Q are independent of $|\mathcal{D}|$, no matter how big $\mathcal{D}$ is. Though proven effective for relational data, it has yet been investigated for graph data. In light of this, we study the problem of *bounded pattern matching using views*. We first introduce *access schema* $\mathcal{C}$ for graphs and propose a notion of *joint containment* to characterize *bounded pattern matching using views*. We show that a pattern query Q can be boundedly evaluated using views $\mathcal{V}(G)$ and a fraction G_Q of G *if and only if* the query Q is *jointly contained* by $\mathcal{V}$ and $\mathcal{C}$. Based on the characterization, we develop an efficient algorithm as well as an optimization strategy to compute matches by using $\mathcal{V}(G)$ and G_Q. Using real-life and synthetic data, we experimentally verify the performance of these algorithms, and show that (a) our algorithm for joint containment determination is not only effective but also efficient; and (b) our matching algorithm significantly outperforms its counterpart, and the optimization technique can further improve performance by eliminating unnecessary input.

1 Introduction

With the advent of massive scale data, it is very urgent to have effective methods for query evaluation on large scale data. One typical solution is by means of *scale independence* [7, 8], whose idea is to compute the answers $Q(\mathcal{D})$ to a query Q in a dataset $\mathcal{D}$ by accessing a small fraction D_Q of $\mathcal{D}$ with bounded size, no matter how big the underlying $\mathcal{D}$ is. Following the idea, [12, 13] show that nontrivial queries can be *scale independent* under a set $\mathcal{C}$ of *access constraints*, a form of cardinality constraints with associated indices, and refer to a query Q as *boundedly evaluable* if for all datasets $\mathcal{D}$ that satisfy $\mathcal{C}$, $Q(\mathcal{D})$ can be evaluated from a fraction D_Q of $\mathcal{D}$, and the time for identifying and fetching D_Q and the size $|D_Q|$ of D_Q are independent of $|\mathcal{D}|$. Still, many queries are not *boundedly evaluable*, hence *bounded evaluation with views* was proposed by [9], which is to select and materialize a set $\mathcal{V}$ of small views, and answer Q on $\mathcal{D}$ by using cached views $\mathcal{V}(\mathcal{D})$ and an additional small fraction D_Q of $\mathcal{D}$. Then, the queries that are not *boundedly evaluable* can be efficiently answered with views and a small fraction of original data with bounded size.

Bounded evaluation with views have proven effective for querying relational data [10], but the need for studying the problem is even more evident for graph pattern matching (GPM), since (a) GPM has been widely used in social analysis [27] which is becoming increasingly important nowadays; (b) it is a challenging task to perform graph pattern matching on real-life graphs due to their sheer size; and (c) view-based matching technique is often too restrictive. Fortunately, *bounded pattern matching using views* fills this critical void. Indeed, cardinality constraints are imposed by social graphs, *e.g.,* on LinkedIn, a person can have at most 30000 connections and most of people have friends less than 1000 [3]; on Facekbook, a person can have no more than 5000 friends [6], etc. Given the constraints and a set of well-chosen views $\mathcal{V}$ along with their caches $\mathcal{V}(G)$ on graphs G, GPM can be evaluated by using the views plus a small fraction G_Q of G of bounded size, no matter how large G is.

Example 1. A fraction of a recommendation network G is shown in Fig. 1(a), where each node denotes a person with job title (*e.g.,* project manager (PM), business analyst (BA), database



PM	BA	PRG
PM_1	BA_1	PRG_1
PM_2	BA_2	PRG_2
PM_3	BA_2	PRG_3

$V_1(G)$

PRG	ST	UI
PRG_1	ST_1	UI_1
PRG_2	ST_2	UI_1
PRG_2	ST_2	UI_2
PRG_2	ST_3	UI_1
PRG_2	ST_3	UI_2
PRG_3	ST_3	UI_3

$V_2(G)$

Fig. 1. Graph G, pattern Q, views $\mathcal{V}$, $\mathcal{V}(G)$ and access schema φ

administrator (DBA), programmer (PRG), user interface designer (UI) and software tester (ST)); and each edge indicates collaboration, *e.g.,* $(\mathsf{PM}_1, \mathsf{PRG}_1)$ indicates that PRG_1 worked well with PM_1 on a project led by PM_1.

To build a team for software development, one issues a pattern query Q depicted in Fig. 1(b). The team members need to satisfy the following requirements: (1) with expertise: PM, BA, DBA, PRG, UI and ST; (2) meeting the following collaborative experience: (i) BA, PRG and DBA worked well under the project manager PM; and (ii) DBA, ST and UI have been supervised by PRG, and collaborated well with PRG. It is then a daunting task to perform graph pattern matching since it takes $O(|G|!|G|)$ time to identify all the isomorphic matches of Q in G [11], where $|G| = |V| + |E|$ indicates the size of G.

While one can do better by *bounded pattern matching using views.* Suppose that (1) a set of views $\mathcal{V} = \{\mathsf{V}_1, \mathsf{V}_2\}$ is defined and cached ($\mathcal{V}(G) = \{\mathsf{V}_1(G), \mathsf{V}_2(G)\}$) as shown in Fig. 1(c), and (2) there exists an *access constraint* $\varphi = \langle Q_\varphi(Q_L \to Q_R), N_0\rangle$ (Fig. 1(d)) that G satisfies. Here φ states that for each DBA that has been supervised by a PM, he can be supervised by at most N_0 distinct PRG. One may associate φ with an index $\mathcal{I}_\varphi$ for fast access. As will be seen shortly, with index $\mathcal{I}_\varphi$, a fraction G_Q of G of bounded size can be efficiently constructed, and $\mathsf{Q}(G)$ can be answered by using $\mathcal{V}(G)$ and G_Q. Since $\mathcal{V}(G)$ already contains partial answers to Q in G, and $\mathcal{V}(G)$ and G_Q are often much smaller than G, thus the cost for computing $\mathsf{Q}(G)$ can be substantially reduced. □

This example suggests that we perform pattern matching by using views $\mathcal{V}$, $\mathcal{V}(G)$ and a fraction G_D of original graph G of bounded size. In doing so, two key issues have to be settled. (1) How to decide whether a pattern query Q is *boundedly evaluable with views*? (2) How to efficiently compute $\mathsf{Q}(G)$ with $\mathcal{V}(G)$ and G_Q?

Contributions. This paper investigates the aforementioned questions for *bounded pattern matching using views*. We focus on graph pattern matching with *subgraph isomorphism* [11].

(1) We introduce *access schema* defined on graph data (Section 2.2), and propose a notion of *joint containment* (Section 3) for determining whether a pattern query is *boundedly evaluable with views*. Given a pattern query Q, a set of views $\mathcal{V}$ and access schema $\mathcal{C}$ with indexes associated, we show that Q is *boundedly evaluable with views* if and only if Q is *jointly contained* by $\mathcal{V}$ and $\mathcal{C}$.

(2) We provide an algorithm, that works in $O((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|!|\mathsf{Q}|)$ time to determine *joint containment*, where $||\mathcal{V}||$ and $||\mathcal{C}||$ refer to the cardinality of $\mathcal{V}$ and $\mathcal{C}$, respectively, and $|\mathsf{Q}|$ indicates the size of Q. As the cost of the algorithm is dominated by $||\mathcal{V}||$, $||\mathcal{C}||$ and $|\mathsf{Q}|$, which are often small in practice, the algorithm hence performs very efficiently.

(3) Based on *joint containment*, we develop an algorithm to evaluate graph pattern matching by using $\mathcal{V}(G)$ and G_Q (Section 4). Given a pattern query Q, a set of views $\mathcal{V}$ and its extension $\mathcal{V}(G)$ on a graph G, and an access schema $\mathcal{C}$ that G satisfies, the algorithm computes $\mathsf{Q}(G)$ in $O((|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||}(|\mathsf{Q}|^{|\mathsf{Q}|} \cdot N_m)^{||\mathcal{C}||})$ time, *without accessing* G at all, when Q is *jointly contained* in $\mathcal{V}$ and $\mathcal{C}$. It is far less costly than the algorithm [11] that takes $O(|G|!|G|)$ time to evaluate Q directly on G, since $|\mathsf{Q}|$, $|\mathcal{V}|$, $||\mathcal{V}||$, $||\mathcal{C}||$ are very small, and $|\mathcal{V}(G)|$ is typically *much*

smaller than $|G|$ in practice. We also study the *minimum containment problem*, which is to find a pair of subsets $\langle \mathcal{V}', \mathcal{C}' \rangle$ of $\mathcal{V}$ and $\mathcal{C}$ such that Q is jointly contained by $\langle \mathcal{V}', \mathcal{C}' \rangle$ and moreover, the input used by the matching algorithm can be dramatically reduced.

(4) Using real-life and synthetic graphs, we experimentally verify the performances of our algorithms (Section 5). We find that (a) our algorithm for joint containment checking is very efficient, *e.g.,* taking only 145.5 milliseconds to determine whether a pattern query is contained by a set of views; (b) our view-based matching algorithm is efficient: it is 9.7 times faster than conventional method on *Youtube* [5] with 1.6 million nodes and 4.5 million edges; (c) our optimization technique is effective: it can reduce the size of input, *i.e.,* $|\mathcal{V}(G)|$ and $|G_Q|$ by 75% and improve the efficiency by 143%, on average, over real-life graphs; and (d) our matching algorithm scales well with the data size.

In a summary of the scientific contributions, this work gives a full treatment for *bounded pattern matching using views*, for pattern queries defined in terms of subgraph isomorphism. It introduces methods to efficiently determine whether a pattern query can be *boundedly evaluable with views*; provides effective techniques to evaluate pattern matching using $\mathcal{V}(G)$ and G_Q which is of bounded size. In contrast with prior works, this work fills one critical void for evaluating graph pattern matching on big graphs, and yields a promising approach to querying "big" social data. All the proofs, algorithms and complexity analyses can be found in [2].

Related work. We next categorize related work as follows.

Query answering using views. Answering queries using views has been well studied for relational data (see [17, 21] for surveys), XML data [16, 23, 25], and [15]. This work differs from them in the following aspects: (i) we adopt subgraph isomorphism as the semantic of pattern matching, instead of graph simulation [19] and bounded simulation [14], that are applied by [15]; (ii) we study a more practical problem to answer graph pattern matching using available views and a small fraction of graph G with bounded size; and (iii) we also investigate the problem of view selection, and provide effective technique for this problem.

Scale independence. The idea of scale independence, *i.e.,* querying dataset $\mathcal{D}$ by accessing only a bounded amount of data in $\mathcal{D}$, is proposed by [7–9]. Extending the idea with *access schema*, [12, 13] introduced *bounded evaluation*. To cope with nontrivial queries, [9] proposed *bounded evaluation with views*, *i.e.,* evaluating queries that are not boundedly evaluable on a dataset $\mathcal{D}$ by accessing not only cached views $\mathcal{V}(\mathcal{D})$ but also a small fraction of $\mathcal{D}$ with bounded size. Furthermore, [10] explored fundamental problems of *bounded evaluation with views*. This work differs from [9, 10] in that the query semantics are different, and we not only conduct static analysis for fundamental problems, but also provide effective technique for matching evaluation.

Organization. The remainder of the paper is organized as follows. Section 2 reviews the notions of data graphs, pattern queries, graph pattern matching, and views, graph pattern matching using views. Section 3 introduces pattern containment problem and provides algorithm for pattern containment checking. Section 4 provides view based matching algorithm, and optimization techniques to further improve matching evaluation. Extensive experimental studies are conducted in Section 5. Section 6 summarizes the main results of the paper and identifies open issues.

2 Preliminaries

In this section, we first review data graphs, pattern queries and graph pattern matching. We then introduce the problem of *bounded pattern matching using views*.

2.1 Basic Definitions

We start with basic notations, *i.e.,* data graphs, pattern queries and graph pattern matching.

Data graphs. A *data graph* is a node-labeled, directed graph $G = (V, E, L)$, where (1) V is a finite set of data nodes; (2) $E \subseteq V \times V$, where $(v, v') \in E$ denotes a *directed* edge from node v to v'; and (3) $L(\cdot)$ is a function such that for each node v in V, $L(v)$ is a label from an alphabet Σ. Intuitively, $L(\cdot)$ specifies *e.g.,* job titles, social roles, ratings, etc [20].

Pattern queries. A *pattern query* (or shortened as pattern) is a directed graph $\mathsf{Q} = (V_p, E_p, f_v)$, where (1) V_p is the set of *pattern nodes*, (2) E_p is the set of *pattern edges*, and (3) $f_v(\cdot)$ is a function defined on V_p such that for each node $u \in V_p$, $f_v(u)$ is a label in Σ.

Subgraphs & Sub-patterns. A graph $G_s = (V_s, E_s, L_s)$ is a subgraph of $G = (V, E, L)$, denoted by $G_s \subseteq G$, if $V_s \subseteq V$, $E_s \subseteq E$, and moreover, for each $v \in V_s$, $L_s(v) = L(v)$. Similarly, a pattern $\mathsf{Q}_s = (V_{p_s}, E_{p_s}, f_{v_s})$ is *subsumed by* another pattern $\mathsf{Q} = (V_p, E_p, f_v)$, denoted by $\mathsf{Q}_s \subseteq \mathsf{Q}$, if Q_s is a subgraph of Q, *i.e.,* $V_{p_s} \subseteq V_p$, $E_{p_s} \subseteq E_p$ and for each $u \in V_p$, $f_{v_s}(u) = f_v(u)$. We say Q_s a sub-pattern of Q when $\mathsf{Q}_s \subseteq \mathsf{Q}$.

Graph pattern matching [11]. A *match* of Q in G via *subgraph isomorphism* is a subgraph G_s of G that is isomorphic to Q, *i.e.,* there exists a *bijective function h* from V_p to the node set V_s of G_s such that (1) for each node $u \in V_p$, $f_v(u) = L(h(u))$ $(h(u) \in V_s)$; and (2) (u, u') is an edge in Q if and only if $(h(u), h(u'))$ is an edge in G_s.

We also use the following notations. (1) The match result of Q in G, denoted as $\mathsf{Q}(G)$, is a set consisting of all the matches of Q in G. (2) For a pattern edge $e = (u, u')$, we derive a set $S(e)$ from $\mathsf{Q}(G)$ by letting $S(e) = \{(v, v')|v = h(u), v' = h(u'), h \in \mathsf{Q}(G), (v, v') \in E\}$, and denote $S(e)$ as the match set of e. (3) We use $\mathsf{Q} \sim G_s$ to denote that G_s is a match of Q. (4) We denote $|V_p| + |E_p|$ as the size $|\mathsf{Q}|$ of Q and $|V| + |E|$ as the size $|G|$ of G.

2.2 Problem Formulation

We next formulate the problem of *bounded pattern matching using views*. We start from notions of views and view extensions, followed by access schema and problem statement.

Views. A *view* (*a.k.a.* view definition) V is also a pattern query. Its match result $\mathsf{V}(G)$ in a data graph G is denoted as *view extension*, or *extension* when it is clear from the context [18]. As shown in Fig. 1(c), a set of views $\mathcal{V} = \{\mathsf{V}_1, \mathsf{V}_2\}$ are defined, with extensions $\mathcal{V}(G) = \{\mathsf{V}_1(G), \mathsf{V}_2(G)\}$ on G cached.

Access schema. Extended from [10], we define *access schema* on graphs as follows. An *access schema* $\mathcal{C}$ is defined as a set of *access constraints* $\varphi = \langle Q_\varphi(Q_L \to Q_R), N\rangle$, where Q_φ is a pattern query, Q_L and Q_R are sub-patterns of Q_φ such that the union of edge sets of Q_L and Q_R equals to the edge set of Q_φ, and N is a natural number.

Given a graph G, a pattern Q_φ and its sub-pattern Q_L, a match G_L (resp. G_R) of Q_L (resp. Q_R) is denoted as a Q_L-value (resp. Q_R-value) of Q_φ in G. Then, we denote by $G_{[Q_\varphi:Q_R]}(Q_L \sim G_L)$ the set $\{G_R|G_R \subseteq G_s, G_R \in Q_R(G), G_L \subseteq G_s, G_s \in Q_\varphi(G)\}$, and write it as $G_{Q_R}(Q_L \sim G_L)$, when Q_φ is clear from the context.

A graph G *satisfies* the access constraint φ, if

- ○ for any Q_L-value G_L, $|G_{Q_R}(Q_L \sim G_L)| \leq N$; and
- ○ there exists a function (referred to as an *index*) that given a Q_L-value G_L, returns $\{G_s|G_s \in Q_\varphi(G), G_L \subseteq G_s\}$ from G in $O(N)$ time.

Intuitively, an *access constraint* φ is a combination of a cardinality constraint and an index $\mathcal{I}_\varphi$ on Q_L for Q_R. It tells us that given any Q_L-value, there exist at most N distinct Q_R-values, and these Q_R-values can be efficiently fetched by using $\mathcal{I}_\varphi$. By using indices, we can also construct a fraction G_Q of G, whose size is bounded by $\Sigma_{\varphi_i \in \mathcal{C}} N_i \cdot |Q_{\varphi_i}|$. We refer to the maximum cardinality of access constraints in an access schema $\mathcal{C}$ as N_m. A graph G satisfies *access schema* $\mathcal{C}$, denoted by $G \models \mathcal{C}$, if G satisfies all the access constraints φ in $\mathcal{C}$.

Symbols	Notations
$G_s \subseteq G$ (resp. $\mathsf{Q}_s \subseteq \mathsf{Q}$)	G_s (resp. Q_s) is a subgraph (resp. sub-pattern) of G (resp. Q)
$\mathsf{Q}(G)$	match result of Q in G
$S(e)$	match set of pattern edge e in G
$\mathcal{V} = \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$	a set of view definitions $\mathsf{V}_i = (V_{\mathsf{V}_i}, E_{\mathsf{V}_i}, f_{\mathsf{V}_i})$
$\mathcal{V}(G) = \{\mathsf{V}_1(G), \ldots, \mathsf{V}_n(G)\}$	a set $\mathcal{V}(G)$ of view extensions $\mathsf{V}_i(G)$
$\varphi = \langle Q_\varphi(Q_L \to Q_R), N\rangle$	access constraint
$\mathcal{C} = \{\varphi_1, \cdots, \varphi_n\}$	access schema
N_m	maximum cardinality of access constraints in $\mathcal{C}$
$\mathsf{Q} \sqsubseteq \mathcal{V}$	Q is contained in $\mathcal{V}$.
$\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$	Q is jointly contained in $\mathcal{V}$ and $\mathcal{C}$.
Q_g	a containing rewriting of Q
$\bar{E}_g$	"uncovered edges" of Q_g in Q
$\mathsf{H}_{\mathsf{V}}^{\mathsf{Q}}$	the shadow of a view V in Q
$\lvert\mathsf{Q}\rvert$ (resp. $\lvert\mathsf{V}\rvert$)	size (total number of nodes and edges) of a pattern Q (resp. view definition V)
$\lvert\mathsf{Q}(G)\rvert$ (resp. $\lvert\mathsf{V}(G)\rvert$)	total size of matches of Q (resp. V) in G
$\lvert\mathcal{V}\rvert$ (resp. $\lvert\mathcal{Q}\rvert$)	total size of view definitions in $\mathcal{V}$ (resp. pattern queries in $\mathcal{Q}$)
$\lvert\lvert\mathcal{V}\rvert\rvert$ (resp. $\lvert\lvert\mathcal{C}\rvert\rvert$, $\lvert\lvert\mathcal{Q}\rvert\rvert$)	the number of view definitions in $\mathcal{V}$ (resp. access constraints in $\mathcal{C}$, pattern queries in $\mathcal{Q}$)
$\lvert\mathcal{V}(G)\rvert$	total size of matches in $\mathcal{V}(G)$

Table 1. A summary of notations

Example 2. An *access schema* with a single *access constraint* φ is shown in Fig. 1(d). By definition, pattern Q_φ takes two edges that are from its sub-patterns Q_L and Q_R, respectively. Assume that an index $\mathcal{I}_\varphi$ is constructed on G (Fig. 1(a)), then given a Q_L-value $(\mathsf{PM}_2, \mathsf{DBA}_1)$, one can fetch from $\mathcal{I}_\varphi$ a set of matches of Q_φ, *i.e.,* $\{(\mathsf{PM}_2, \mathsf{DBA}_1, \mathsf{PRG}_1), (\mathsf{PM}_2, \mathsf{DBA}_1, \mathsf{PRG}_2)\}$. One may further verify that $|G_{Q_R}(Q_L \sim G_L)| = 2$. □

Bounded pattern matching using views. Given a pattern query Q, a set $\mathcal{V}$ of view definitions and an *access schema* $\mathcal{C}$, *bounded pattern matching using views* is to find another query $\mathcal{A}$ such that for any graph G that satisfies $\mathcal{C}$,

- ○ $\mathcal{A}$ is equivalent to Q, *i.e.,* $\mathsf{Q}(G) = \mathcal{A}(G)$; and
- ○ $\mathcal{A}$ only refers to views $\mathcal{V}$, their extensions $\mathcal{V}(G)$ in G and G_Q only, without accessing original graph G. Here G_Q is a fraction of G and can only be constructed with indexes $\mathcal{I}_\varphi$, that are associated with access constraints $\varphi \in \mathcal{C}$, such that the time for generating G_Q is in $O(\Sigma_{\varphi_i \in \mathcal{C}} N_i)$, and the size $|G_Q|$ of G_Q is bounded by $O(\Sigma_{\varphi_i \in \mathcal{C}} N_i \cdot |Q_{\varphi_i}|)$.

If such an algorithm $\mathcal{A}$ exists, we say that pattern Q is *boundedly evaluable with views*, and can be evaluated using $\mathcal{V}(G)$ and a fraction G_Q of G of bounded size, no matter how big G is.

Remark. To make practical use of *bounded pattern matching using views*, it is critical to identify a set of views and *access schema*. There exist a host of works on view selection based on query logs. For *access schema*, its identification problem still needs further investigation, while some heuristic methods can be applied. To be more concentrated, this paper mainly foucses on the matching evaluation by using views and *access schema*.

3 Characterization for Bounded Pattern Matching using Views

We propose a characterization for *bounded pattern matching using views*, *i.e.,* a sufficient and necessary condition for deciding whether a pattern query is *boundedly evaluable with views*.

3.1 Joint Containment Problem

We first introduce the notion of *joint containment*.

Joint containment. A pattern query Q with edge set E_p is *jointly contained* by a set of views $\mathcal{V} = \{\mathsf{V}_1, \cdots, \mathsf{V}_n\}$, and a set of access constraints $\mathcal{C} = \{\varphi_1, \cdots, \varphi_k\}$, denoted by Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$, if for any graph G that satisfies $\mathcal{C}$, there exist a pair of mappings $\langle\lambda, \rho\rangle$, such that

- ○ E_p is divided into two disjoint parts E_c and E_u;
- ○ E_c is mapped via λ to powerset $\mathcal{P}(\bigcup_{i\in[1,n]} E_{\mathsf{V}_i})$, and $S(e) \subseteq \bigcup_{e'\in\lambda(e)} S(e')$ for any $e \in E_c$; and moreover,
- ○ E_u is mapped via ρ to powerset $\mathcal{P}(\bigcup_{j\in[1,k]} E_{\varphi_j})$, and $S(e) \subseteq \bigcup_{e'\in\rho(e)} S(e')$ for any edge $e \in E_u$,

where E_{V_i} refers to the edge set of the i-th view definition V_i in $\mathcal{V}$ and E_{φ_j} indicates the edge set of Q_{φ_j} of the j-th access constraint φ_j in $\mathcal{C}$.

Intuitively, Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$ indicates that Q can be divided into two disjoint parts, that take edge sets E_c and E_u, respectively; and moreover, there exist mappings λ and ρ, that map E_c to edges in $\mathcal{V}$ and E_u to edges in $\mathcal{C}$, respectively, such that match set $S(e)$ can be derived from either $\mathcal{V}(G)$ or G_Q, for any e in Q, without accessing original graph G.

Example 3. Recall G, Q, $\mathcal{V}$ and $\mathcal{C}$ in Fig. 1. One may verify that Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$, since the edge set of Q can be divided into two parts $E_c = \{(\mathsf{PM}, \mathsf{BA}), (\mathsf{PM}, \mathsf{PRG}), (\mathsf{PRG}, \mathsf{ST}), (\mathsf{PRG}, \mathsf{UI})\}$ and $E_u = \{(\mathsf{PM}, \mathsf{DBA}), (\mathsf{PRG}, \mathsf{DBA})\}$, that are mapped via mappings λ and ρ to the sets of edges in $\mathcal{V}$ and $\mathcal{C}$, respectively. In any graph G, one may verify that for any edge e of Q, its match set $S(e)$ must be a subset of the union of the match sets of the edges in $\lambda(e)$ or $\rho(e)$, *e.g.,* $S(\mathsf{PM}, \mathsf{DBA})$ in G is $\{(\mathsf{PM}_1, \mathsf{DBA}_1), (\mathsf{PM}_2, \mathsf{DBA}_1), (\mathsf{PM}_3, \mathsf{DBA}_2)\}$, which is contained in the match set of Q_L of Q_φ in G. □

Theorem 1. *Given a set of views $\mathcal{V}$ and an access schema $\mathcal{C}$, a pattern query* Q *is boundedly evaluable with views if and only if* Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.

Proof. We now prove Theorem 1.
(I) We first prove the **only if** condition, *i.e.,* if Q is *boundedly evaluable with views* then Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$. We show this by contradiction. Assume that Q is *boundedly evaluable with views*, while Q $\not\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$. By the definition of *joint containment*, when Q $\not\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$, there must exist some data graph G_o, such that for all possible mapping pairs $\langle\lambda, \rho\rangle$, there always exists at least one edge e in Q, such that either (a) $S(e) \not\subseteq \bigcup_{e'\in\lambda(e)} S(e')$ when e is mapped via λ to $\mathcal{V}$ or (b) $S(e) \not\subseteq \bigcup_{e'\in\rho(e)} S(e')$ when e is mapped via ρ to $\mathcal{C}$. Then, there must exist an edge e_g in G_o such that $e_g \in S(e)$ but $e_g \notin \bigcup_{e'\in\lambda(e)} S(e')$ for case (a), or $e_g \notin \bigcup_{e'\in\rho(e)} S(e')$ for case (b). No matter which case happens, it contradicts to the assumption that Q can be answered by using $\mathcal{V}(G)$ and G_Q, since at least for data graph G_o, one match of Q with edge e_g can not be found from $\mathcal{V}(G)$ and G_Q. Thus, Q is *boundedly evaluable with views only if* Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.
(II) We next show the **if** condition, also by contradiction. Assume that Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$, but Q is not *boundedly evaluable with views*. Then there must exist some data graph G_o such that Q(G_o) contains a match G_s, that can not be derived from $\mathcal{V}(G)$ and G_Q. As a result, there must exist an edge e_g in G_s, that is not in $S(e')$ for any e' that is mapped either via λ or via ρ from E_p. This contradicts the assumption, as Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$ (by assumption) already indicates that there exist a pair of mappings $\langle\lambda, \rho\rangle$ such that $S(e)$ is a subset of $\bigcup_{e'\in\lambda(e)} S(e')$ or $\bigcup_{e'\in\rho(e)} S(e')$ for any e in E_p.

Putting these together, Theorem 1 follows. □

Theorem 1 shows that *joint containment* determines whether a pattern query is *boundedly evaluable with views*. This further motivates us to study the *joint containment* (JPC) problem, which is to determine, given a pattern query Q, a set of views $\mathcal{V}$ and an access schema $\mathcal{C}$, whether Q $\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.

Remarks. (1) A special case of *joint containment* is the pattern containment [26]. Indeed, when access schema $\mathcal{C}$ is an empty set, *joint containment* problem becomes pattern containment problem. (2) When Q is not contained in $\mathcal{V}$ (denoted as $\mathsf{Q} \not\sqsubseteq \mathcal{V}$), one may find another pattern query $\mathsf{Q}_g = (V_g, E_g)$, referred to as a *containing rewriting* of Q *w.r.t.* $\mathcal{V}$, such that $\mathsf{Q}_g \subseteq \mathsf{Q}$ and $\mathsf{Q}_g \sqsubseteq \mathcal{V}$, compute matches $\mathsf{Q}_g(G)$ of Q_g from $\mathcal{V}(G)$ and treat $\mathsf{Q}_g(G)$ as "approximate matches" [26]. While, we investigate more practical but nontrivial cases, *i.e.,* $\mathsf{Q} \not\sqsubseteq \mathcal{V}$, and advocate to integrate access schema into view-based pattern matching such that exact matches can be identified by using a small portion of additional data of bounded size.

3.2 Determination of Joint Containment

To characterize *joint containment*, a notion of *shadow*, which is introduced in [26] is required. To make the paper self-contained, we cite it as follows (rephrased).

Given a pattern query Q and a view definition V, one can compute $\mathsf{V}(\mathsf{Q})$ by treating Q as data graph, and V as pattern query. Then the *shadow* from V to Q, denoted by $H_{\mathsf{V}}^{\mathsf{Q}}$, is defined to be the union of edge sets of matches of V in Q.

To ease the presentation, we denote by $\bar{E}_g = E_p \setminus E_g$ as the "uncovered edges" of Q_g (a *containing rewriting* of Q) in Q, where E_p and E_g are the edge sets of Q and Q_g, respectively.

The result below shows that *shadow* yields a characterization of *joint containment*.

Proposition 1. *For a pattern* Q*, a set of view definitions* $\mathcal{V}$ *and an access schema* $\mathcal{C}$*,* $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$ *if and only if there exists a containing rewriting* Q_g *of* Q *such that* $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H_{Q_\varphi}^{\mathsf{Q}}$.

Proof. (I) We first show the **if** condition by contradiction. Assume that there exists a containing rewriting Q_g of Q with $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H_{Q_\varphi}^{\mathsf{Q}}$, but $\mathsf{Q} \not\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$. Then by the definition of *joint containment*, there must exist a data graph G_o with edge $e_g = (v, v')$, such that e_g is in $S(e)$ but not in $\bigcup_{e' \in \lambda(e)} S(e')$ when e is mapped via λ to $\mathcal{V}$, or $\bigcup_{e' \in \rho(e)} S(e')$ when e is mapped via ρ to $\mathcal{C}$, for an edge $e = (u, u')$ of Q.
(1) If e is mapped via λ to $\mathcal{V}$, then one may verify that v, v' are matches of u, u', respectively, while there does not exist a view definition V in $\mathcal{V}$ such that v and v' can match u_v and u_v', simultaneously, where (u_v, u_v') is an edge in V. This indicates that there does not exist a containing rewriting Q_g of Q *w.r.t.* $\mathcal{V}$.
(2) When e is mapped via ρ to an edge e_φ in Q_φ of φ, it can be easily verified that e_g is not $S(e_\varphi)$. While, by the assumption that $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H_{Q_\varphi}^{\mathsf{Q}}$ and the definition of *shadow*, for each edge $e_s = (u_s, u_s')$ in $\bar{E}_g$, there must exist at least one φ such that u_s, u_s' are the matches of u_v, u_v', respectively, where (u_v, u_v') is an edge in Q_φ of φ. Accordingly, one may further verify that nodes v, v' in G, that are matches of u, u', must also be matches of u_v, u_v' by the semantic of subgraph isomorphism. Thus, e_g must be in $\bigcup_{e' \in \rho(e)} S(e')$, which contradicts the assumption.

Putting these together, the contradiction is incorrect.

(II) For the **Only If** condition, we assume by contradiction that $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$, but there does not exist a containing rewriting Q_g of Q with $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H_{Q_\varphi}^{\mathsf{Q}}$. By the assumption that there does not exist a containing rewriting Q_g of Q with $\bar{E}_g = \bigcup_{\varphi \in \mathcal{C}} H_{Q_\varphi}^{\mathsf{Q}}$, either (a) there does not exist any containing rewriting Q_g of Q *w.r.t.* $\mathcal{V}$, or (b) $\bar{E}_g = \bigcup_{\varphi \in \mathcal{C}} H_{Q_\varphi}^{\mathsf{Q}}$. For case (a), one may verify that there does not exist a mapping from E_c to $\mathcal{V}$ satisfying $S(e) \subseteq \bigcup_{e' \in \lambda(e)} S(e')$ for any edge $e \in E_c$; for case (b), it can also be verified that no mapping from E_u to $\mathcal{C}$ with $S(e) \subseteq \bigcup_{e' \in \rho(e)} S(e')$ for any edge $e \in E_u$ exists. Either of two cases leads to the contradiction of $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.

These complete the proof of Proposition 1. $\square$

Based on the characterization, we show below that the JPC problem is nontrivial. Despite hardness, we also provide an efficient algorithm for the determination of *joint containment*.

Input: A pattern $\mathsf{Q} = (V_p, E_p)$, views $\mathcal{V}$ and access schema $\mathcal{C}$.
Output: A boolean value ans that is true if and only if $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.

```
1. boolean ans := false; set $E_c := \emptyset$, $E_u := \emptyset$, $\mathcal{C}' := \emptyset$;
2. for each view definition $V_i \in \mathcal{V}$ do
3.     compute $H^{Q}_{V_i}$; $E_c := E_c \bigcup H^{Q}_{V_i}$;
4. $E_u := E_p \setminus E_c$;
5. for each access constraint $\varphi = \langle Q_\varphi(Q_L \to Q_R), N\rangle$ in $\mathcal{C}$ do
6.     if $H^{Q}_{Q_\varphi} \cap E_u \neq \emptyset$ then
7.         $\mathcal{C}' := \mathcal{C}' \cup \{\varphi\}$; $E_u := E_u \setminus H^{Q}_{Q_\varphi}$;
8.     if $E_u = \emptyset$ then
9.         ans := true; break ;
10.return ans;
```

Fig. 2. Algorithm JCont

Theorem 2. *(1) The* JPC *problem is* NP*-hard. (2) It is in* $O((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|!|\mathsf{Q}|)$ *time to decide whether* $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$*, and if so, to compute associated mappings from* Q *to* $\mathcal{V}$*,* $\mathcal{C}$*.*

Proof. We first show Theorem 2(1). We then show Theorem 2(2) by presenting an algorithm as a constructive proof.

(I) We show NP-hardness of the JPC problem by reduction from the NP-complete *subgraph isomorphism problem* (ISO) [11].

An instance of ISO consists of a graph $G_1 = (V_1, E_1)$ and a pattern $\mathsf{Q}_1 = (V_{p_1}, E_{p_1})$, ISO is to decide whether there exists a subgraph G_s in G_1 that is isomorphic to Q_1. Given such an instance of ISO, we construct an instance of JPC problem as follows: (a) we create a pattern query Q that takes the same node and edge set as G_1; (b) we create a view set $\mathcal{V}$ that takes a single view definition V and let V equal to Q_1; and (c) we create an access schema $\mathcal{C}$ that only includes a single access constraint φ and let Q_φ take edges in Q but not in V. The construction is obviously in PTIME. We next verify that there exists a subgraph G_s of G_1 that is isomorphic to Q_1 if and only if $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H^{\mathsf{Q}}_{Q_\varphi}$.

Assume that there exists a subgraph G_s as a match of Q_1 in G_1, one can verify that $\bar{E}_g$ consists of edges in G_1 but not G_s, and $H^{\mathsf{Q}}_{Q_\varphi}$ includes edges of $E_1 \setminus E_{p_1}$. Thus $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H^{\mathsf{Q}}_{Q_\varphi}$. Conversely, if $\bar{E}_g \subseteq \bigcup_{\varphi \in \mathcal{C}} H^{\mathsf{Q}}_{Q_\varphi}$, it can be easily verified that a pattern with edge set $E_1 \setminus \bar{E}_g$ is a match of Q_1.

As ISO is NP-complete, so JPC problem is NP-hard.

(II) We show Theorem 2(2) with an algorithm and its analysis.

Algorithm. The algorithm, denoted as JCont, takes Q, $\mathcal{V}$ and $\mathcal{C}$ as input, and returns true if and only if $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$. The algorithm works in three stages. In the first stage, it initializes a boolean variable ans, and three empty sets $\mathsf{E_c}$, $\mathsf{E_u}$ and $\mathcal{C}'$, to keep track of "covered edges", "uncovered edges", and selected access constraints, respectively (line 1). In the second stage, it identifies an edge set $\mathsf{E_c}$ such that the sub-pattern of Q induced with E_c is *contained by* $\mathcal{V}$. Specifically, it (1) computes shadow $H^{\mathsf{Q}}_{\mathsf{V}_i}$ for each V_i in $\mathcal{V}$, by invoking the revised subgraph isomorphism algorithm, which finds all the matches of V_i in Q with algorithm in [11], and then merges them together; (2) extends $\mathsf{E_c}$ with $H^{\mathsf{Q}}_{\mathsf{V}_i}$ (lines 2-3). After all the shadows are merged, JCont generates an edge set $\mathsf{E_u} = E_p \setminus \mathsf{E_c}$ (line 4). In the last stage, JCont verifies the condition for *joint containment* as follows. It checks for each access constraint φ whether its Q_φ can *cover* a part of $\mathsf{E_u}$, *i.e.,* $H^{\mathsf{Q}}_{Q_\varphi} \cap \mathsf{E_u} \neq \emptyset$ (line 6). If so, JCont enlarges $\mathcal{C}'$ with φ and updates $\mathsf{E_u}$ with $\mathsf{E_u} \setminus H^{\mathsf{Q}}_{Q_\varphi}$ (line 7). When the condition $\mathsf{E_u} = \emptyset$ is encountered, the variable ans is changed to true, and the **for** loop (line 5) immediately terminates (line 9). JCont finally returns ans as result (line 10).

Example 4. Consider Q, $\mathcal{V} = \{\mathsf{V}_1, \mathsf{V}_2\}$ and $\mathcal{C} = \{\varphi\}$ in Fig. 1. JCont first computes shadows for each $\mathsf{V}_i \in \mathcal{V}$ and obtains $H^{\mathsf{Q}}_{\mathsf{V}_1} = \{(\mathsf{PM}, \mathsf{BA}), (\mathsf{PM}, \mathsf{PRG})\}$ and $H^{\mathsf{Q}}_{\mathsf{V}_2} = \{(\mathsf{PRG}, \mathsf{ST}), (\mathsf{PRG}, \mathsf{UI})\}$. Then E_u includes $\{(\mathsf{PM}, \mathsf{DBA}), (\mathsf{PRG}, \mathsf{DBA})\}$. It next computes $H^{\mathsf{Q}}_{Q_\varphi} = \{(\mathsf{PM}, \mathsf{DBA}), (\mathsf{PRG}, \mathsf{DBA})\}$, which exactly covers E_u. Finally, JCont returns true indicating that Q is *boundedly evaluable with views.* □

Correctness. The correctness is ensured by that when JCont terminates, JCont correctly identifies a part Q_c (with edge set $\mathsf{E_c}$) of Q that can be answered by using $\mathcal{V}$; and in the meanwhile, the remaining part Q_u (with edge set $\mathsf{E_u}$) of Q can also be *covered* by $\mathcal{C}$. To see this, observe the following. (1) JCont always terminates, since two **for** loops (lines 2-3, 5-9) execute $||\mathcal{V}||$ and $||\mathcal{C}||$ times, respectively. (2) When JCont terminates, (a) Q_c consists of edges of shadows from each view definition to Q, *i.e.,* $\bigcup_{\mathsf{V}_i \in \mathcal{V}} H^{\mathsf{Q}}_{\mathsf{V}_i}$. Thus, Q_c is contained by $\mathcal{V}$ and can be answered using $\mathcal{V}(G)$. (b) $\mathsf{E_u}$ includes edges of shadows from each Q_φ to Q, showing how many *uncovered* edges of Q can be covered by using $\mathcal{C}$. When $\mathsf{E_u}$ turns to an empty set, *i.e.,* all the *uncovered* edges can be covered, JCont changes ans to true, indicating that $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.

Complexity. The initialization of JCont is in constant time. For the second stage, JCont iteratively computes shadow $H^{\mathsf{Q}}_{\mathsf{V}_i}$ for each $\mathsf{V}_i \in \mathcal{V}$. As it takes $O(|\mathsf{Q}|!|\mathsf{Q}|)$ time to compute shadow from V_i to Q for a single iteration, and the **for** loop repeats $||\mathcal{V}||$ times, thus, it is in $O(||\mathcal{V}|||\mathsf{Q}|!|\mathsf{Q}|)$ time for the second stage. In the last stage, JCont computes shadow from Q_φ to Q for each access constraint φ in $\mathcal{C}$, which is in $O(|\mathsf{Q}|!|\mathsf{Q}|)$ time for a single iteration as well. As the iteration executes $||\mathcal{C}||$ times, it hence takes JCont $O(||\mathcal{C}|||\mathsf{Q}|!|\mathsf{Q}|)$ time. Putting these together, JCont is in $O((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|!|\mathsf{Q}|)$ time.

Remarks. (1) There already exist techniques, *e.g.,* [26] to answer graph pattern matching using views only. Indeed, after containment checking (lines 2-4 of JCont), if $\mathsf{E_u}$ becomes an empty set, then pattern Q is contained in $\mathcal{V}$, indicating that pattern Q can be answered by using views only, without access constraints. While, in this paper, we are focusing on nontrivial pattern queries Q, *i.e.,* $\mathsf{Q} \not\sqsubseteq \mathcal{V}$, then $\mathsf{E_u}$ is enforced to be a nonempty set after containment checking. (2) Algorithm JCont can be easily adapted to return a pair of mappings $\langle \lambda, \rho \rangle$ that serve as input for the matching algorithm (will be given in Section 4).

4 Matching Evaluation

In this section, we study how to evaluate pattern matching using views $\mathcal{V}(G)$ and a fraction G_Q of G. We first develop an algorithm to find matches from $\mathcal{V}(G)$ and G_Q. We next study the *minimum containment problem*, to optimize the matching evaluation.

4.1 An Matching Algorithm

Along the same line as pattern matching using views, on a graph G that satisfies access schema $\mathcal{C}$, a pattern query Q can be answered with $\mathcal{V}(G)$ and G_Q as following: (1) determine whether $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$ and compute a pair of mappings $\langle \lambda, \rho \rangle$ with revised algorithm of JCont; and (2) compute $\mathsf{Q}(G)$ with an matching algorithm that takes λ, ρ, $\mathcal{V}$, $\mathcal{V}(G)$ and $\mathcal{C}$ as input, if $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$. We next show that such an matching algorithm indeed exists.

Theorem 3. *For any graph G that satisfies access schema $\mathcal{C}$, a pattern* Q *can be answered by using $\mathcal{V}$, $\mathcal{V}(G)$ and G_Q in $O((|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||}(|\mathsf{Q}|^{|\mathsf{Q}|} \cdot N_m)^{||\mathcal{C}||})$ time, if* $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$.

Proof. We next provide an algorithm with analyses as a constructive proof of Theorem 3.

Algorithm. The algorithm, denoted as BMatch, is shown in Fig. 3. It takes a pattern Q, a set of views $\mathcal{V}$ and $\mathcal{V}(G)$ and mappings λ and ρ, as input, and works in two stages: "merging" views

Input: Pattern Q, views $\mathcal{V}$, $\mathcal{V}(G)$, access schema $\mathcal{C}$, mappings λ, ρ.
Output: The query result M as $\mathsf{Q}(G)$.

```
1.  initialize an empty pattern $Q_o$; set $\mathsf{M} := \emptyset$, $\mathsf{M}_1 := \emptyset$;
2.  for each view $\mathsf{V}_i$ that is mapped via $\lambda$ from $E_p$ do
3.      for each $G_s \in \lambda^{-1}(\mathsf{V}_i)$ do
4.        $Q_o \oplus G_s$;
5.        for each $m_1 \in \mathsf{M}$ and each $m_2 \in \mathsf{V}_i(G)$ do
6.          if $m_1$ and $m_2$ can be merged then
7.            $m := m_1 \oplus m_2$; $\mathsf{M}_1 := \mathsf{M}_1 \cup \{m\}$;
8.        update $\mathsf{M}$, $\mathsf{M}_1$;
9.  for each access constraint $\varphi$ in $\rho(E_p)$ do
10.     $\langle Q_o, \mathsf{M}\rangle := \mathsf{Expand}(Q_o, \mathsf{M}, \varphi, \rho)$;
11. return M;
```

Fig. 3. Algorithm BMatch

$\mathsf{V}_i(G)$ following the mapping λ; and "expanding" partial matches with G_Q under the guidance of mapping ρ.

More specifically, BMatch starts with an empty pattern query Q_o, an empty set M to keep track of matches of Q_o and another empty set M_1 for maintaining intermediate results (line 1). It then iteratively "merges" $\mathsf{V}_i(G)$ following mapping λ (lines 2-8). Specifically, for each view definition V_i that is mapped via λ from edge set E_p (line 2) and each match G_s of V_i in Q (line 3), BMatch expands Q_o with G_s (line 4), and iteratively expands each match m_1 of Q_o with each match m_2 of V_i if they can be merged in the same way as the merging process of Q_o and G_s, and includes the new match m in M_1 (lines 5-7). When a round of merging process for matches of Q_o and V_i finished, BMatch updates M and M_1 by letting $\mathsf{M} := \mathsf{M}_1$ and $\mathsf{M}_1 := \emptyset$ (line 8). When the first stage finished, Q_o turns to a containing rewriting Q_g of Q and M includes all the matches of Q_g in G. In the following stage, BMatch iteratively invokes Procedure Expand to "expand" Q_o and its matches under the guidance of mapping ρ (lines 9-10). It finally returns M as matches of $\mathsf{Q}(G)$ (line 11).

Procedure Expand. Given mapping ρ, access constraint φ, pattern Q_o and its match set M, Expand (not shown) expands Q_o and M as follows. It first initializes an empty set M_1. For each match G_s of Q_φ in Q, Expand first extends Q_o with G_s; it next expands a match m_1 of Q_o with a match G_L of Q_L if m_1 has common nodes or edges with G_L and further expands m_1 with each Q_R-value G_R of G_L, for each match m_1 of Q_o and each match G_L of Q_L. The new matches are maintained by the set M_1. After all the G_s are processed, Expand returns updated Q_o and its match set M_1 as final result.

Example 5. Consider Q, $\mathcal{V}$, $\mathcal{V}(G)$ and φ shown in Fig. 1. BMatch first merges "partial matches" in $\mathcal{V}(G)$ following the guidance of mapping λ. Specifically, it first initializes Q_o and M with V_1 and $\mathsf{V}_1(G)$ (as shown in Fig. 1(c)); it next expands Q_o with V_2, and merges each match in M with each match in $\mathsf{V}_2(G)$. After "partial matches" are merged, Q_o includes all the edges of V_1 and V_2, and set M contains following matches.

id	PM	BA	PRG	ST	UI	id	PM	BA	PRG	ST	UI
m_1	PM_1	BA_1	PRG_1	ST_1	UI_1	m_2	PM_2	BA_2	PRG_2	ST_2	UI_1
m_3	PM_2	BA_2	PRG_2	ST_2	UI_2	m_4	PM_2	BA_2	PRG_2	ST_3	UI_1
m_5	PM_2	BA_2	PRG_2	ST_3	UI_2	m_6	PM_3	BA_2	PRG_3	ST_3	UI_3

Guided by mapping ρ, BMatch expands Q_o and its matches via Procedure Expand. Expand first merges Q_o with Q_φ. Then, it first expands m_1 with G_L (with edge set $\{(\mathsf{PM}_1, \mathsf{DBA}_1)\}$), and then merges m_1 with G_R (with edge set $\{(\mathsf{PRG}_1, \mathsf{DBA}_1)\}$). The above merge process repeats another 5 times for each m_i ($i \in [2, 6]$). Finally, BMatch returns a set of 6 matches that are grown from $m_1 - m_6$, respectively. □

Correctness. The correctness is guaranteed by the following three invariants: (1) BMatch correctly merges Q_o (resp. M) with views V_i (resp. $\mathsf{V}_i(G)$); (2) procedure Expand correctly expands Q_o and M using indexes associated with access schema; and (3) when BMatch terminates, Q_o (resp. M) is equivalent to Q (resp. $\mathsf{Q}(G)$). To see these, observe the following:
(1) Mappings λ and ρ essentially split pattern Q into two parts: Q_c with edge set E_c and Q_u with edge set E_u. These two parts are "covered " by $\mathcal{V}$ and $\mathcal{C}$, respectively.
(2) A view V_i can be mapped to different parts of Q_c, via λ^{-1}. BMatch hence merges Q_o with each match G_s of V_i in Q (lines 3-4). This merge process also guides the merge of matches of Q_o with matches of V_i as follows: if a match m_2 of V_i can be merged with a match m_1 of Q_o, along the same line as the merge of Q_o and G_s, then m_1 and m_2 are merged as a whole (lines 5-7). These guarantee that when a new round merging process terminates, a new pattern query, that expands Q_o with the shadow of V_i in Q_c, is formed, and the set M includes all the matches of the newly formed pattern query.
(3) Guided by ρ^{-1}, Q_φ in φ can also be mapped to various parts G_s of Q. Thus Expand iteratively expands Q_o with those G_s. When the outmost loop terminates (line 2), Q_o has been enlarged with the entire shadow of Q_R in Q_u. During the expansion of Q_o, Expand first fetches a set of matches G_L of Q_L from index $\mathcal{I}_\varphi$. For each match m_1 of Q_o and each match G_L of Q_L, if they overlap each other; Expand merges m_1 with G_L first and then expands m_1 for each Q_R-value of G_L. Thus, it can be easily verified that each match G_L of Q_L and the corresponding Q_R-value of G_L will not be missed, during expansion, which ensures that M_1 correctly maintains matches of Q_o.
(4) When two **for** loops (lines 2-7 and 8-9) of BMatch terminate, Q_o must be equivalent to Q, as edges in E_c and edges in E_u are all covered, in the meanwhile, M must contain a complete set of matches of Q_o, guaranteed by observations given above.

Complexity. We give a detailed complexity analysis as below.
(I) BMatch iteratively merges Q_o and M with view V_i and $\mathsf{V}_i(G)$, respectively. For a single iteration, it takes BMatch $|\lambda^{-1}(\mathsf{V}_i)||\mathsf{M}||\mathsf{V}_i(G)|$ time for the "merge" task. As in the worst case, (1) there may exist $|V_{\mathsf{V}_i}|^{|\mathsf{Q}_c|}$ matches of V_i in Q_c, where V_{V_i} denotes the node set of V_i and $|\mathsf{Q}_c|$ is bounded by $|\mathsf{Q}|$, hence $|\lambda^{-1}(\mathsf{V}_i)|$ is bounded by $|V_{\mathsf{V}_i}|^{|\mathsf{Q}|}$; (2) $|\mathsf{M}|$ is bounded by $\prod_{i\in[1,k-1]} |V_{\mathsf{V}_i}|^{|\mathsf{Q}|}|\mathsf{V}_i(G)|$ before the k-th iteration; and (3) the iteration repeats at most $||\mathcal{V}||$ times, hence the first stage is bounded by $\prod_{i\in[1,||\mathcal{V}||]} |V_{\mathsf{V}_i}|^{|\mathsf{Q}|}|\mathsf{V}_i(G)|$, which is in $O((|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||})$ time.
(II) BMatch repeatedly invokes Procedure Expand to process expansion of Q_o and M with access constraint φ. For a single process, it takes Expand $|\rho^{-1}(Q_R)||\mathsf{M}||\mathcal{I}_\varphi(G_L)|$ time. Note that (1) $|\rho^{-1}(Q_R)|$ is bounded by $|V_R|^{|\mathsf{Q}_u|}$ (V_R refers to the node set of Q_R), which is further bounded by $|V_R|^{|\mathsf{Q}|}$, as $|\mathsf{Q}_u|$ is bounded by $|\mathsf{Q}|$; (2) $|\mathsf{M}|$ is bounded by $(|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||} \cdot \prod_{i\in[1,k-1]} |V_{R_i}|^{|\mathsf{Q}|} \cdot N_i$, before the k-th iteration; and (3) $|\mathcal{I}_\varphi(G_L)| \leq N_k$ at the k-th iteration, thus, Expand is in $O(|V_R|^{|\mathsf{Q}|} \cdot (|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||} \prod_{i\in[1,k]}(|V_{R_i}|^{|\mathsf{Q}|} \cdot N_i))$ time. As the iteration repeats at most $||\mathcal{C}||$ times, the second stage is hence in $O(|\mathsf{Q}|^{|\mathsf{Q}|}(|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||}(|\mathsf{Q}|^{|\mathsf{Q}|} \cdot N_m)^{||\mathcal{C}||})$ time. Putting these together, BMatch is in $O((|\mathcal{V}|^{|\mathsf{Q}|}|\mathcal{V}(G)|)^{||\mathcal{V}||}(|\mathsf{Q}|^{|\mathsf{Q}|} \cdot N_m)^{||\mathcal{C}||})$ time, where N_m is the maximum cardinality of access schema $\mathcal{C}$. □

4.2 Optimization Strategy

As the cost of BMatch is partially determined by $|\mathcal{V}(G)|$ and $|G_Q|$, it is hence beneficial to reduce the size of the parameters. This motivates us to study the *minimum containment problem*.

Minimum Containment Problem. Given a pattern query Q, a set of view definitions $\mathcal{V}$ with each V_i associated with weight $|\mathsf{V}_i(G)|$, and an access schema $\mathcal{C}$, the problem, denoted as MCP, is to find a subset $\mathcal{V}'$ of $\mathcal{V}$ and a subset $\mathcal{C}'$ of $\mathcal{C}$, such that (1) $\mathsf{Q} \sqsubseteq_J [\mathcal{V}', \mathcal{C}']$, and (2) for any subset $\mathcal{V}''$ of $\mathcal{V}$ and any subset $\mathcal{C}''$ of $\mathcal{C}$, if $\mathsf{Q} \sqsubseteq_J [\mathcal{V}'', \mathcal{C}'']$, then $|\mathcal{V}'(G)| + \Sigma_{\varphi_i \in \mathcal{C}'} N_i \cdot |Q_{\varphi_i}| \leq |\mathcal{V}''(G)| + \Sigma_{\varphi_j \in \mathcal{C}''} N_j \cdot |Q_{\varphi_j}|$.

As will be seen in Section 5, MCP is effective: it can eliminate redundant views (as well as their corresponding extensions), and reduce the size of G_Q thereby improving the efficiently of BMatch. However, MCP is nontrivial, its decision problem is NP-hard. Despite of this, we develop an algorithm for MCP, which is approximable within $O(\log |\mathsf{Q}|)$. That's, the algorithm can identify a subset $\mathcal{V}'$ of $\mathcal{V}$ and a subset $\mathcal{C}'$ of $\mathcal{C}$ when $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$, such that $\mathsf{Q} \sqsubseteq [\mathcal{V}', \mathcal{C}']$ and $|\mathcal{V}'(G)| + \Sigma_{\varphi_i \in \mathcal{C}'} N_i \cdot |Q_{\varphi_i}|$ is guaranteed to be no more than $\log(|\mathsf{Q}|) \cdot (|\mathcal{V}_{\mathsf{OPT}}(G)| + \Sigma_{\varphi_j \in \mathcal{C}_{\mathsf{OPT}}} N_j \cdot |Q_{\varphi_j}|)$, where $\mathcal{V}_{\mathsf{OPT}}$ and $\mathcal{C}_{\mathsf{OPT}}$ are the subsets of $\mathcal{V}$ and $\mathcal{C}$, respectively, and moreover, $\mathsf{Q} \sqsubseteq_J [\mathcal{V}_{\mathsf{OPT}}, \mathcal{C}_{\mathsf{OPT}}]$ and $|\mathcal{V}_{\mathsf{OPT}}(G)| + \Sigma_{\varphi_j \in \mathcal{C}_{\mathsf{OPT}}} N_j \cdot |Q_{\varphi_j}|$ is minimum, among all possible subset pairs of $\mathcal{V}$ and $\mathcal{C}$.

Theorem 4. *The* MCP *is (1)* NP*-hard (decision problem), but (2) there exists an algorithm for* MCP *that finds a subset $\mathcal{V}'$ of $\mathcal{V}$ and a subset $\mathcal{C}'$ of $\mathcal{C}$ with $\mathsf{Q} \sqsubseteq_J [\mathcal{V}', \mathcal{C}']$ and $|\mathcal{V}'(G)| + \Sigma_{\varphi_i \in \mathcal{C}'} N_i \cdot |Q_{\varphi_i}| \leq \log(|\mathsf{Q}|) \cdot (|\mathcal{V}_{\mathsf{OPT}}(G)| + \Sigma_{\varphi_j \in \mathcal{C}_{\mathsf{OPT}}} N_j \cdot |Q_{\varphi_j}|)$ in $O(||\mathcal{V}|||\mathsf{Q}|!|\mathsf{Q}| + (||\mathcal{V}|||\mathsf{Q}|)^{3/2})$ time.*

Proof. We now prove Theorem 4 (1) and (2), respectively.

(I) We prove NP-hardness of MCP by showing NP-hardness of its special case, *i.e.,* when access schema $\mathcal{C}$ is an empty set. The decision problem of the special case is to determine, given a pattern query Q, a set of view definitions $\mathcal{V} = \{\mathsf{V}_1, \cdots, \mathsf{V}_n\}$ with each V_i taking integer weight $|\mathsf{V}_i(G)|$ ($i \in [1, n]$), and an integer B, whether there exists a subset $\mathcal{V}'$ of $\mathcal{V}$ such that $\mathsf{Q} \sqsubseteq \mathcal{V}'$ and $|\mathcal{V}'(G)| \leq B$.

We show that this problem is NP-hard by reduction from the *set cover problem* (SCP) [24], which is known NP-complete. An instance of SCP consists of a set U, a collection of subsets of U, $\mathcal{S} = \{S_1, \cdots, S_n\}$, where each subset S_i takes integer weight $\omega(S_i)$, and an integer K, SCP is to decide whether there exists a subset $\mathcal{S}'$ of $\mathcal{S}$ that covers U, *i.e.,* $\bigcup_{S_i \in \mathcal{S}'} S_i = U$, and has total weight no more than K. Given such an instance of SCP, we construct an instance of MCP as follows: (a) for each $x_i \in U$, we create a unique edge e_{x_i} with two distinct nodes u_{x_i} and v_{x_i}; (b) we define a pattern query Q as a graph consisting of all edges e_{x_i} defined in (a); (c) we construct a set $\mathcal{V}$, and define each view definition V_j in $\mathcal{V}$ taking edges e_{x_i} and weight $\omega(S_j)$ from S_j, for each subset $S_j \in \mathcal{S}$ and $x_i \in S_j$; and (d) we set $K = B$.

The construction is obviously in PTIME. We next verify that there exists a subset $\mathcal{S}'$ of $\mathcal{S}$ with total weight no more than K if and only if there exists a subset $\mathcal{V}'$ of $\mathcal{V}$ such that Q is contained in $\mathcal{V}'$ and has total weight $\Sigma_{\mathsf{V}_j \in \mathcal{V}'} |\mathsf{V}_j(G)|$ no larger than B.
(1) Assume that there exists a subset $\mathcal{S}'$ of $\mathcal{S}$ that covers U with total weight at most K. Let $\mathcal{V}'$ be the set of views V_j corresponding to $S_j \in \mathcal{S}'$. One can verify that $\mathsf{Q} \sqsubseteq \mathcal{V}'$ and $\Sigma_{\mathsf{V}_j \in \mathcal{V}'} |\mathsf{V}_j(G)| \leq B$, since the union of all the edges from these $\mathsf{S}^{\mathsf{Q}}_{\mathsf{V}_j}$ is E_q, and the weight $|\mathsf{V}_j(G)| = \omega(S_j)$ for each V_j in $\mathcal{V}'$.
(2) Conversely, if there exists a subset $\mathcal{V}'$ of $\mathcal{V}$ with $\mathsf{Q} \sqsubseteq \mathcal{V}'$ and $\Sigma_{\mathsf{V}_j \in \mathcal{V}'} |\mathsf{V}_j(G)| \leq B$, it is easy to see that the corresponding subset $\mathcal{S}'$ of $\mathcal{S}$ has total weight no more than K and covers U.

As SCP is NP-complete, so is NP-hardness of MCP.

(II) We next show Theorem 4(2) by providing an approximation algorithm as a constructive proof.

Algorithm. The algorithm, denoted as Minimum, is shown in Fig. 4. Given a pattern query Q, a set of view definitions $\mathcal{V}$ with each V_i in $\mathcal{V}$ taking a weight $|\mathsf{V}_i(G)|$, and an access schema $\mathcal{C}$, Minimum identifies a pair $\langle \mathcal{V}', \mathcal{C}' \rangle$ of subsets of $\mathcal{V}$ and $\mathcal{C}$ such that (1) $\mathsf{Q} \sqsubseteq_J [\mathcal{V}', \mathcal{C}']$ if $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$ and (2) $|\mathcal{V}'(G)| + \Sigma_{\varphi_i \in \mathcal{C}'} (N_i \cdot |Q_{\varphi_i}|) \leq \log(|\mathsf{Q}|) \cdot (|\mathcal{V}_{\mathsf{OPT}}(G)| + \Sigma_{\varphi_j \in \mathcal{C}_{\mathsf{OPT}}} (N_j \cdot |Q_{\varphi_j}|))$, where (a) $\log(|\mathsf{Q}|)$ is the approximation ratio, and (b) $\mathcal{V}_{\mathsf{OPT}}$, $\mathcal{C}_{\mathsf{OPT}}$ are the subsets of $\mathcal{V}$, $\mathcal{C}$, respectively, and moreover, $\mathsf{Q} \sqsubseteq_J [\mathcal{V}_{\mathsf{OPT}}, \mathcal{C}_{\mathsf{OPT}}]$ and $|\mathcal{V}_{\mathsf{OPT}}(G)| + \Sigma_{\varphi_j \in \mathcal{C}_{\mathsf{OPT}}} (N_j \cdot |Q_{\varphi_j}|)$ is minimum, among all possible subset pairs of $\mathcal{V}$ and $\mathcal{C}$.

```
Input: A pattern $\mathsf{Q} = (V_p, E_p)$, a set of views $\mathcal{V}$, an access schema $\mathcal{C}$.
Output: A subset $\mathcal{V}'$ of $\mathcal{V}$ and a subset $\mathcal{C}'$ of $\mathcal{C}$ such that $\mathsf{Q} \sqsubseteq_J [\mathcal{V}', \mathcal{C}']$.

1. initialize sets $\mathcal{V}' := \emptyset, \mathcal{C}' := \emptyset, \mathcal{F} := \emptyset$;
2. for each view definition $\mathsf{V}_i \in \mathcal{V}$ do
3.     compute $\mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i}$; $\mathcal{F} := \mathcal{F} \cup \{\langle \mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i}, |\mathsf{V}_i(G)|\rangle\}$;
4. for each access constraint $\varphi_j \in \mathcal{C}$ do
5.     compute $\mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}}$; $\mathcal{F} := \mathcal{F} \cup \{\langle \mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}}, N_j \cdot |Q_{\varphi_j}|\rangle\}$;
6. $\langle \mathcal{V}', \mathcal{C}' \rangle$:= PtnFinder(Q, $\mathcal{F}$);
7. return $\langle \mathcal{V}', \mathcal{C}' \rangle$;

Procedure PtnFinder
Input: A pattern query $\mathsf{Q} = (V_p, E_p)$, set $\mathcal{F}$.
Output: $\langle \mathcal{V}', \mathcal{C}' \rangle$.

1. initialize an empty pattern $Q_o = (V_o, E_o)$, set $\mathcal{V}' := \emptyset, \mathcal{C}' := \emptyset$;
2. while $\mathcal{F} \neq \emptyset$ do
3.     find an object obj with the least $\alpha(\cdot)$ value;
4.     if $(\mathsf{H}^{\mathsf{Q}}_{\mathsf{obj}} \setminus Q_o) = \emptyset$ then break ;
5.     update $\mathcal{F}$; $Q_o$ merges $\mathsf{H}^{\mathsf{Q}}_{\mathsf{obj}}$;
6.     if obj is a view $\mathsf{V}_i$ then $\mathcal{V}' := \mathcal{V}' \cup \{\mathsf{V}_i\}$;
7.     else if obj is an access constraint $\varphi_j$ then $\mathcal{C}' := \mathcal{C}' \cup \{\varphi_j\}$;
8. if $E_o \neq E_p$ then return $\emptyset$;
9. return $\langle \mathcal{V}', \mathcal{C}' \rangle$;
```

Fig. 4. Algorithm Minimum

In a nutshell, the algorithm applies a greedy strategy to find a views V_i in $\mathcal{V}$ or an access constraint φ_j from $\mathcal{C}$ that is considered "best" during the iteration. To measure the goodness of the views and access constraints, we define a metric $\alpha(\mathsf{V}_i) = \frac{|\mathsf{V}_i(G)|}{|\mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i} \setminus Q_o|}$ for a view V_i, and $\alpha(\varphi_j) = \frac{N_j \cdot |Q_{\varphi_j}|}{|\mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}} \setminus Q_o|}$ for an access constraint φ_j. Here, Q_o takes edges from shadows whose corresponding views (resp. access constraints) are chosen in $\mathcal{V}'$ (resp. $\mathcal{C}'$). Intuitively, $\alpha(\mathsf{V}_i)$ (resp. $\alpha(\varphi_j)$) indicates how costly it is to "cover" the remaining part $\mathsf{Q} \setminus Q_o$ of Q with $\mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i}$ (resp. $\mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}}$), hence a V_i or φ_j with the least $\alpha(\cdot)$ is favored in each round iteration.

The algorithm works in two stages.

(I) It initializes three empty sets $\mathcal{V}'$, $\mathcal{C}'$ and $\mathcal{F}$ (line 1), and computes the shadow $\mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i}$ for each $\mathsf{V}_i \in \mathcal{V}$ (resp. $\mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}}$ for each $\varphi_j \in \mathcal{C}$) and maintains a pair $\langle \mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i}, |\mathsf{V}_i(G)|\rangle$ (resp. $\langle \mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}}, N_j \cdot |Q_{\varphi_j}|\rangle$) in $\mathcal{F}$ (lines 2-5). Intuitively, $|\mathsf{V}_i(G)|$ (resp. $N_j \cdot |Q_{\varphi_j}|$) can be viewed as the "weight" of its corresponding V_i (resp. φ_j).

(II) Minimum invokes procedure PtnFinder to compute a pair $\langle \mathcal{V}', \mathcal{C}' \rangle$ of subsets of $\mathcal{V}$ and $\mathcal{C}$. Specifically, it first initializes an empty pattern query Q_o and two empty sets $\mathcal{V}'$, $\mathcal{C}'$ (line 1). It then iteratively selects an object obj, whose corresponding $\alpha(\cdot)$ is the least (line 3). Here, the chosen object obj is either a view definition V_i or an access constraint φ_j. Once there does not exist any obj whose corresponding shadow can expand Q_o, *i.e.*, $(\mathsf{H}^{\mathsf{Q}}_{\mathsf{obj}} \setminus Q_o) = \emptyset$, PtnFinder breaks the loop (line 4). Otherwise, it updates $\mathcal{F}$ by removing $\langle \mathsf{H}^{\mathsf{Q}}_{\mathsf{V}_i}, |\mathsf{V}_i(G)|\rangle$ (resp. $\langle \mathsf{H}^{\mathsf{Q}}_{Q_{\varphi_j}}, N_j \cdot |Q_{\varphi_j}|\rangle$), and expands Q_o with shadow $\mathsf{H}^{Q}_{\mathsf{obj}}$ (line 5). If obj is a view V_i, PtnFinder includes it in $\mathcal{V}'$ (line 6), otherwise, PtnFinder enriches $\mathcal{C}'$ with φ_j (line 7). After **while** loop terminates, if E_o is not equivalent to E_p, PtnFinder returns an empty set, since $\mathsf{Q} \not\sqsubseteq_J [\mathcal{V}, \mathcal{C}]$ and hence no subset pair exists (line 8). Otherwise, PtnFinder returns $\langle \mathcal{V}', \mathcal{C}' \rangle$ as final result (line 9).

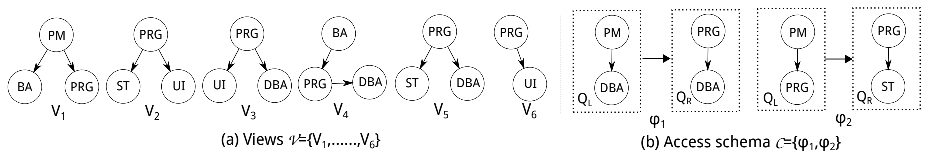


Fig. 5. Views $\mathcal{V}$ and access schema $\mathcal{C}$

We next provide detailed analysis for the algorithm Minimum.

Correctness. Observe that Minimum either finds a pair $\langle \mathcal{V}', \mathcal{C}' \rangle$ of subsets of $\mathcal{V}$ and $\mathcal{C}$ such that Q $\sqsubseteq_J$ $[\mathcal{V}', \mathcal{C}']$ or an empty set indicating Q $\not\sqsubseteq_J$ $[\mathcal{V}, \mathcal{C}]$. This is ensured by joint containment checking of the algorithm (line 8 of PtnFinder), by following Proposition 1. Moreover, PtnFinder identifies $\langle \mathcal{V}', \mathcal{C}' \rangle$ with a greedy strategy, which is verified to guarantee $\log(|\mathsf{Q}|)$ approximation ratio for *weighted set cover problem* [24].

Complexity. Algorithm Minimum computes "shadows" for each V_i in $\mathcal{V}$ and each φ_j in $\mathcal{C}$ in $O((||\mathcal{V}||+||\mathcal{C}||)|\mathsf{Q}|!|\mathsf{Q}|)$ time (lines 2-5). The procedure PtnFinder is in $O(((||\mathcal{V}||+||\mathcal{C}||)|\mathsf{Q}|)^{3/2})$ time, as the while loop is executed $\min\{(||\mathcal{V}|| + ||\mathcal{C}||), |\mathsf{Q}|\}$ times, which is bounded by $O(((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|)^{1/2})$ time, and each iteration takes $O((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|)$ time to find a view with least $\alpha(\cdot)$ [24]. Thus, Minimum is in $O((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|!|\mathsf{Q}| + ((||\mathcal{V}|| + ||\mathcal{C}||)|\mathsf{Q}|)^{3/2})$ time.

The analysis above completes the proof of Theorem 4. □

Example 6. Recall pattern query Q in Fig. 1 (b). Given a set of views $\mathcal{V} = \{\mathsf{V}_1, \cdots, \mathsf{V}_6\}$ and an access schema $\mathcal{C} = \{\varphi_1, \varphi_2\}$, as shown in Fig. 5, algorithm Minimum identifies $\langle \mathcal{V}', \mathcal{C}' \rangle$ from $\mathcal{V}$ and $\mathcal{C}$ as following. It first computes shadows for each V_i in $\mathcal{V}$ and each φ_j in $\mathcal{C}$, and initializes set $\mathcal{F}$ as table below. It next applies PtnFinder to iteratively select V_i (resp. φ_j) based on their $\alpha(\cdot)$ values. Assume N_1 of φ_1 is 6 and N_2 of φ_2 is 4 (their actual occurrences in G of Fig. 1 (a)). Then, PtnFinder successively selects V_2, φ_1 and V_1 to enrich $\mathcal{V}'$ and $\mathcal{C}'$, and terminates **while** loop. Minimum finally returns $\mathcal{V}' = \{\mathsf{V}_1, \mathsf{V}_2\}$ and $\mathcal{C}' = \{\varphi_1\}$ as result.

	shadow	coverage		shadow	coverage
V_1	{(PM, BA), (PM, PRG)}	$3 * 5$	V_2	{(PRG, ST), (PRG, UI)}	$6 * 5$
V_3	{(PRG, UI), (PRG, DBA)}	$4 * 5$	V_4	$\emptyset$	0
V_5	{(PRG, ST), (PRG, DBA)}	$4 * 5$	V_6	{(PRG, UI)}	$4 * 5$
φ_1	{(PM, DBA), (PRG, DBA)}	$N_1 * 5$	φ_2	{(PM, PRG), (PRG, ST)}	$N_2 * 5$

5 Experimental Evaluation

Using real-life and synthetic data, we conducted three sets of tests to evaluate (1) performances of algorithms for *joint containment* checking, *i.e.,* the efficiency of algorithm JCont compared with Minimum; and (2) performances of algorithms for *bounded pattern matching using views*, *i.e.,* the effectiveness, efficiency and scalability of algorithms BMatch, $\mathsf{BMatch}_{\mathsf{min}}$.

Experimental setting. We used the following data.

(1) Real-life graphs. We used four real-life graphs: (a) *Amazon* [4], a product co-purchasing network with 548K nodes and 1.78M 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* [1], a collaboration network with 1.4M nodes and 3M edges, in which nodes represent papers with attributes such as title, authors, year and venue, and edges denote citations. (c) *YouTube* [5], a recommendation network with 1.6M nodes and 4.5M 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.

(2) Synthetic graphs. We designed a generator to produce random graphs, controlled by the number $|V|$ of nodes, the number $|E|$ of edges, and an alphabet Σ for node labels. We enforced a set of access constraints during random generation.

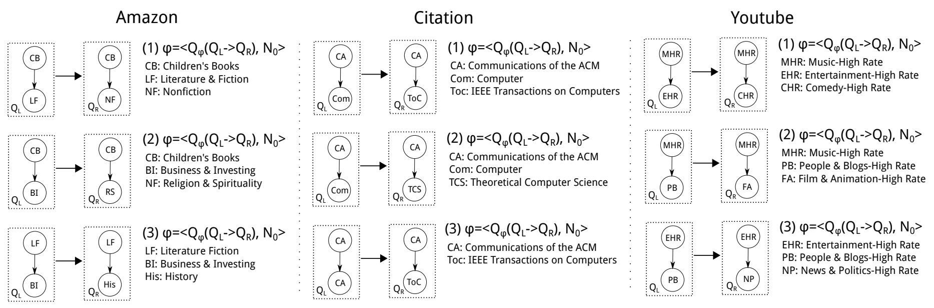


Fig. 6. Typical access constraints on real-life graphs

(3) Pattern queries. We implemented a generator for pattern queries controlled by: the number $|V_p|$ (resp. $|E_p|$) of pattern nodes (resp. edges), and node label f_v from an alphabet Σ of labels drawn from corresponding real-life graphs. We denote $(|V_p|, |E_p|)$ as the size of pattern queries, and generated a set of 30 pattern queries with size $(|V_p|, |E_p|)$ ranging from $(3, 2)$ to $(8, 16)$, for each data graph. We also produced a set of 100 synthetic pattern queries to constitute query workload, using label set Σ, that is used for synthetic graph generation.

(4) Views. We generated views for *Amazon* following [22], designed views to search for papers and authors in computer science for *Citation*, and generated views for *Youtube* following [15]. For each of the real-life graphs, a set $\mathcal{V}$ of 50 view definitions with different sizes *e.g.,* $(2, 1)$, $(3, 2)$, $(4, 3)$, $(4, 4)$ and structures are generated. For synthetic graphs, we randomly generated a set of 50 views whose node labels are drawn from a set Σ of 10 labels and with sizes of $(2, 1)$, $(3, 2)$, $(4, 3)$ and $(4, 4)$. For each view set $\mathcal{V}$, we force that $\mathsf{Q} \not\sqsubseteq \mathcal{V}$, for each Q used for testing.

(5) Access schema. We investigated real-life graphs, and extracted an access schema $\mathcal{C}$ with a set of access constraints φ for each of them. A set of typical access constraints are shown in Fig. 6. For each access constraint $\varphi = \langle Q_\varphi(Q_L \to Q_R), N\rangle$, we computed an index $\mathcal{I}_\varphi$ for it. Since N is no more than 100 for each chosen φ, hence the space cost of an index $\mathcal{I}_\varphi$ ranges from a few megabytes to dozens of megabytes, for each φ. In addition, the index $\mathcal{I}_\varphi$ is built upon a hashtable, with a distinct match G_L of Q_L as the key, and a set of matches G_s of Q_φ as values. Thus, the fetch time on $\mathcal{I}_\varphi$ for any G_L is very fast, and can be viewed as a constant. On synthetic graphs, we manually generated a set of access constraints and computed indexes for these access constraints, along the same line as performed on real-life graphs. As index construction is a one-off task and can be performed off-line, we do not report its computational time.

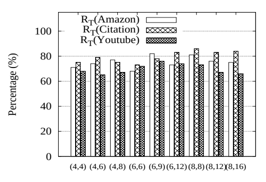
(5) Implementation. We implemented the following algorithms, all in Java: (1) JCont for determination of joint containment; (2) MCG [26] for finding maximally containing rewritings Q_g; (3) VF2 [11] for performing matching evaluation on G, BMatch for matching with $\mathcal{V}(G)$ and G_Q; and (4) Minimum for identifying a pair $\langle \mathcal{V}', \mathcal{C}'\rangle$ from $\mathcal{V}$ and $\mathcal{C}$, and $\mathsf{BMatch}_{\mathsf{min}}$ which revises BMatch by using $\langle \mathcal{V}', \mathcal{C}'\rangle$ identified by Minimum.

All the tests were run on a machine with an Intel Core(TM)2 Duo 3.00GHz CPU and 4GB memory, using Ubuntu. Each experiment was run 10 times and the average is reported.

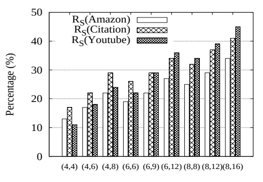
Experimental results. We next present our findings.

Exp-1: Joint containment checking. We first evaluate the performance of JCont vs. Minimum.

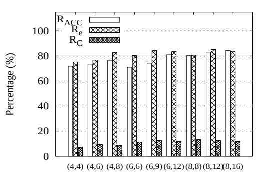
Performance of JCont *vs.* Minimum. We evaluate the efficiency of JCont vs. Minimum. Fixing $\mathcal{V}$ and $\mathcal{C}$ for real-life graphs, we varied the pattern size from $(4, 4)$ to $(8, 16)$, where each size corresponds to a set of pattern queries with different structures and node labels. We find the following. (1) JCont and Minimum both are efficient, *e.g.,* it takes JCont on average 145.5 ms to decide whether a pattern with size $(8, 16)$ is jointly contained in $\mathcal{V}$ and $\mathcal{C}$. (2) Both two algorithms spend more time over larger patterns, which are consistent with their computational complexities. Due



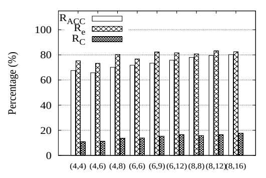
(a) Vary |Q| (real-life graphs)



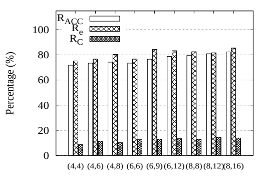
(b) Vary |Q| (real-life graphs)



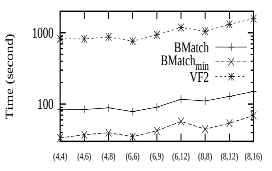
(c) Vary |Q| (*Amazon*)



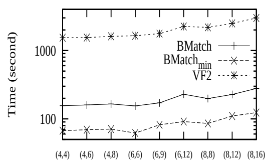
(d) Vary |Q| (*Citation*)



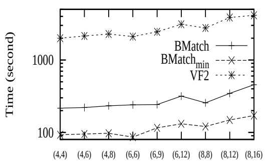
(e) Vary |Q| (*Youtube*)



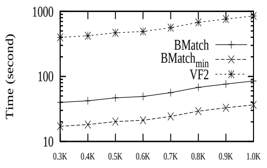
(f) Vary |Q| (*Amazon*)



(g) Vary |Q| (*Citation*)



(h) Vary |Q| (*Youtube*)



(i) Vary $|G|$ (synthetic)

Fig. 7. Performance evaluation

to space constraint, we do not report detailed figures here. (3) We compute a ratio $R_T = \frac{T_{\mathsf{JCont}}}{T_{\mathsf{Minimum}}}$, where T_{JCont} and T_{Minimum} are the time used by JCont and Minimum, respectively, to evaluate performance gap between JCont and Minimum. As shown in Fig. 7(a), JCont accounts for about 75.7% of the time of Minimum, on average, since it takes Minimum more time to pick a V_i from $\mathcal{V}$ (resp. φ_j from $\mathcal{C}$).

To investigate the effectiveness of Minimum, we defined a ratio $R_S = \frac{S_a}{S_b}$, where $S_a = |\mathcal{V}'(G)| + \Sigma_{\varphi_j \in \mathcal{C}'} N_j \cdot |Q_{\varphi_j}|$, $S_b = |\mathcal{V}(G)| + \Sigma_{\varphi_j \in \mathcal{C}} N_j \cdot |Q_{\varphi_j}|$, as the ratio of the total size of view extensions $\mathcal{V}'(G)$ and G_Q identified by Minimum to the total size of whole set of $\mathcal{V}(G)$ and indexes in $\mathcal{C}$. Fixing $\mathcal{V}$ and $\mathcal{C}$ over real-life graphs, we varied pattern size $(|V_p|, |E_p|)$ from $(4, 4)$ to $(8, 16)$ and evaluated the ratio R_S. As shown in Fig. 7(b), Minimum is effective, it finds a pair $\langle \mathcal{V}', \mathcal{C}' \rangle$ with total size S_a *substantially smaller* than S_b, *i.e.,* taking only about 25.2% of S_b for all real-life graphs, on average. As will be shown, using $\langle \mathcal{V}', \mathcal{C}' \rangle$ can substantially improve efficiency of matching computation.

Exp-2: Bounded pattern matching using views. We study the effectiveness, efficiency and scalability of BMatch, $\mathsf{BMatch}_{\mathsf{min}}$, compared to VF2 [11] using real-life and synthetic graphs.

Effectiveness. We define following three metrics and evaluate effectiveness of *bounded pattern matching using views* with real-life graphs.

(1) We defined a ratio R_{ACC} as accuracy, to measure the result quality when access schema is absent, by following F-measure [28]. Here, $R_{\mathsf{ACC}} = \frac{2 \cdot \mathsf{recall} \cdot \mathsf{precision}}{\mathsf{recall} + \mathsf{precision}}$, where recall = $\frac{|\mathsf{S}|}{|\mathsf{S_t}|}$, precision = $\frac{|\mathsf{S}|}{|\mathsf{S_m}|}$, $\mathsf{S_t}$ consists of matches in $\mathsf{Q}(G)$, $\mathsf{S_m}$ is the set of matches in $\mathsf{Q_g}(G)$, and S consists of "true" matches that can be identified from $\mathsf{Q_g}(G)$, where $\mathsf{Q_g}$ refers to the maximally containing rewriting of Q *w.r.t.* $\mathcal{V}$.

(2) We used $R_{\mathcal{C}} = \frac{\Sigma_{\varphi_i \in \mathcal{C}'} N_i \cdot |Q_{\varphi_i}|}{|\mathsf{Q}(G)|}$ to show the propotion of $|G_Q|$ in $|\mathsf{Q}(G)|$. Here, G_Q is constructed from $\mathcal{C}'$, which is a subset of $\mathcal{C}$ and takes a set of access constraints φ_i that are used by BMatch when matching evaluation, and $|\mathsf{Q}(G)|$ is the total size of matches of Q in G.

(3) We used $R_e = \frac{|\bar{E}_g|}{|E_p|}$ to show how large a pattern query needs to be "covered" by access constraints.

Figures 7(c)-7(e) report three ratios on real-life graphs, which tell us the following. The ratio R_{ACC} is, on average, 77.3%, 73.6%, and 76.8% on *Amazon*, *Citation* and *Youtube*, respectively. In the meanwhile, the average ratios of $R_{\mathcal{C}}$ and R_e reach 10.4% and 18.8%, 14% and 20.6%, 11.7% and 19.4% on *Amazon*, *Citation* and *Youtube*, respectively. These together show that access constraints often cover a small but critical part of pattern queries, *e.g.,* 16.11% of total edges of Q on average, and provide limited but key information, *e.g.,* 10.89% of the size of match result to improve accuracy by more than 20%, on average.

Efficiency. Figures 7(f), 7(g) and 7(h) show the efficiency on *Amazon*, *Citation* and *YouTube*, respectively. The x-axis represents pattern size $(|V_p|, |E_p|)$. The results tell us the following. (1) BMatch and $\mathsf{BMatch_{min}}$ substantially outperform VF2, taking only 9.5% and 3.8% of its running time on average over all real-life graphs. (2) All the algorithms spend more time on larger patterns. Nonetheless, BMatch and $\mathsf{BMatch_{min}}$ are less sensitive to the increase of |Q| than VF2, as they reuse earlier computation cached in view extensions and hence save computational cost. (3) $\mathsf{BMatch_{min}}$ is more efficient than BMatch, taking only 41.2% time on average over real-life graphs, as it uses smaller $\mathcal{V}(G)$ and G_Q.

Scalability. Using synthetic graphs, we evaluated the scalability of BMatch, $\mathsf{BMatch_{min}}$ and VF2. Fixing $|\mathsf{Q}| = (4, 6)$, we varied the node number $|V|$ of data graphs from $0.3M$ to $1M$, in $0.1M$ increments, and set $|E| = 2|V|$. As shown in Fig. 7(i), $\mathsf{BMatch_{min}}$ scales best with $|G|$ and is on average 1.4 and 22.1 times faster than BMatch and VF2, which is consistent with the complexity analysis, and the observations in Figures 7(f), 7(g) and 7(h).

Summary. We obtain the following findings through our experimental evaluation. (1) It is efficient to determine whether a pattern query can be *boundedly evaluable with views*. For a pattern query Q with size $(8, 16)$, it only takes JCont 145.5 milliseconds to determine whether $\mathsf{Q} \sqsubseteq_J [\mathcal{V}, \mathcal{C}]$. (2) *Bounded pattern matching using views* is effective in querying large social graphs. For example, by using $\mathcal{V}(G)$ and G_Q, pattern matching via subgraph isomorphism takes only 9.3% of the time needed for computing matches directly in *YouTube*, and 10.6% on synthetic graphs. Moreover, our optimization technique is effective, Minimum can reduce size of $\mathcal{V}(G)$ and G_Q and improve BMatch by 143% on real-life graphs. Our matching algorithms also scale well with data size.

6 Conclusion

We have studied *bounded pattern matching using views*, for pattern queries defined in terms of subgraph isomorphism, from theory to algorithms. We have introduced *access schema* for graphs, proposed a notion of joint containment for characterizing bounded pattern matching using views, and provided an efficient algorithm for joint containment checking. Based on the characterization, we have developed an matching algorithm by using views and a size-bounded fraction, and moreover, we have also provided optimization strategy to improve efficiency of matching compuation. Our experimental results have verified the effectiveness, efficiency and scalability of our algorithms, using real-life and synthetic data.

The study of bounded pattern matching using views is still in its infancy. One issue is the selection problem for views and *access schema*. Another problem concerns scale-inpendence for GPM.

References

1. Citation. *http://www.arnetminer.org/citation/.*
2. Full version. *https://github.com/XinWang-PaperHub/Paper/raw/master/paper.pdf.*
3. Linkedin statistics. https://www.omnicoreagency.com/linkedin-statistics/.
4. Stanford large network dataset collection. *http://snap.stanford.edu/data/index.html.*

5. Youtube dataset. *http://netsg.cs.sfu.ca/youtubedata/*.
6. Facebook, 2013. http://newsroom.fb.com.
7. M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: success-tolerant query processing in the cloud. *PVLDB*, 5(3):181–192, 2011.
8. M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh. SCADS: scale-independent storage for social computing applications. In *CIDR*, 2009.
9. M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. Generalized scale independence through incremental precomputation. In *SIGMOD*, pages 625–636, 2013.
10. Y. Cao, W. Fan, F. Geerts, and P. Lu. Bounded query rewriting using views. *ACM Trans. Database Syst.*, 43(1):6:1–6:46, 2018.
11. L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. 2004.
12. W. Fan, F. Geerts, Y. Cao, T. Deng, and P. Lu. Querying big data by accessing small data. In *PODS*, pages 173–184, 2015.
13. W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 51–62, 2014.
14. W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractability to polynomial time. In *PVLDB*, 2010.
15. W. Fan, X. Wang, and Y. Wu. Answering pattern queries using views. *IEEE Trans. Knowl. Data Eng.*, 28(2):326–341, 2016.
16. M. Gerome and D. Suciu. Containment and equivalence for an xpath fragment. In *PODS*, 2002.
17. A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
18. A. Y. Halevy. Theory of answering queries using views. *SIGMOD Record*, 29(4), 2001.
19. M. R. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.
20. R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *KDD*, 2006.
21. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.
22. J. Leskovec, A. Singh, and J. Kleinberg. Patterns of influence in a recommendation network. In *Advances in Knowledge Discovery and Data Mining*. 2006.
23. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT*, 2003.
24. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
25. J. Wang, J. Li, and J. X. Yu. Answering tree pattern queries using views: a revisit. In *EDBT*, 2011.
26. X. Wang. Answering graph pattern matching using views: A revisit. In *DEXA*, 2017.
27. X. Wang and H. Zhan. Approximating diversified top-k graph pattern matching. In *DEXA*, pages 407–423, 2018.
28. Wikipedia. F-measure. *http://en.wikipedia.org/wiki/F-measure*.