Big Graph Analyses: From Queries to Dependencies and Association Rules

Wenfei Fan · Chunming Hu*

Received: date / Accepted: date

Abstract This position paper provides an overview of our recent advances in the study of big graphs, from theory to systems to applications. We introduce a theory of bounded evaluability, to query big graphs by accessing a bounded amount of the data. Based on this, we propose a framework to query big graphs with constrained resources. Beyond queries, we propose functional dependencies for graphs, to detect inconsistencies in knowledge bases and catch spams in social networks. As an example application of big graph analyses, we extend association rules from itemsets to graphs for social media marketing. We also identify open problems in connection with querying, cleaning and mining big graphs.

Keywords Big graphs · Bounded evaluabilility · Dependencies · Association rules · Social media marketing · Knowledge base enrichment

1 Introduction

The study of graphs has generated renewed interest in the past decade. Graphs make an important source of big data, and have found prevalent use in, *e.g.,* social media marketing, knowledge discovery, transportation networks, mobile network analysis, computer vision, the study of adolescent drug use [55], and intelligence analysis for identifying terrorist organizations [87]. In light of these, a large number of algorithms, optimization techniques, graph partition strategies and parallel systems have been developed for graph computations.

Are we done with the study of graphs?

Not yet! Real-life graphs introduce new challenges to query evaluation, data cleaning and data mining, among other things. They demand a departure from traditional theory to systems and applications, and call for new techniques to query big graphs, improve data quality and identify associations among entities.

(1) Querying big graphs. Consider a class $\mathcal{Q}$ of graph queries, such as graph traversal (*e.g.,* depth-first search DFS and breadth-first search BFS), graph connectivity (*e.g.,* strongly connected components), graph pattern matching (via *e.g.,* graph simulation or subgraph isomorphism), and keyword search. Given a query $Q \in \mathcal{Q}$ and a big graph G, the problem of querying big graphs is to compute the answers $Q(G)$ to Q in G.

When G is "big", it is often costly to compute $Q(G)$. Indeed, DFS takes $O(|G|)$ time, not to mention graph pattern matching via subgraph isomorphism, for which it is NP-complete to decide whether $Q(G)$ is empty, *i.e.,* whether there exists a match of pattern Q in G (cf. [88]). Worse yet, real-life graphs are often of large scale, *e.g.,* Facebook has billions of users and trillions of links, which amount to about 300PB of data [63].

One might be tempted to think that we could cope with big graphs by means of parallel computing. That is, when G grows big, we add more processors and parallelize the computation of $Q(G)$, to make the computation scale with G. Based on this assumption, several parallel graph query systems have been developed, *e.g.,*

Wenfei Fan
University of Edinburgh and Beihang University
Tel: +44 (0)131 651-3877
Fax: +44 (0)131 651 1426
E-mail: wenfei@inf.ed.ac.uk

Chunming Hu (contact author)
BDBC, Beihang University
Tel: 86-10-82339679
Fax: 86-10-82339679
E-mail: hucm@act.buaa.edu.cn

Pregel [84], GraphLab [81], GraphX [61], Giraph [59], Giraph++ [101], Blogel [104] and Trinity [2].

However, there exist graph computation problems that are are *not* parallel scalable. That is, for some query classes $\mathcal{Q}$, their parallel running time cannot be substantially reduced no matter how many processors are used. Consider, for example, graph simulation [68], a quadratic-time problem. It has been shown that no parallel algorithms for the problem can scale well with the increase of processors used [48]. This is actually not very surprising. The degree of parallelism is constrained by the *depth* of a computation, *i.e.,* the longest chain of dependencies among its operations [74]. As a consequence, some graph computation problems are "inherently sequential" [62]. Add to the complication that parallel algorithms nowadays are typically developed over a shared-nothing architecture [91]. For such algorithms, with the increase of processors also come higher communication costs, not to mention skewed graphs, skewed workload, start-up costs and interference when processors compete for *e.g.,* network bandwidth.

Moreover, even for queries that are parallel scalable, small businesses often have constrained resources such as limited budget and available processors, and cannot afford renting thousands of Amazon EC2 instances.

With these observations come the following questions. Is it possible to efficiently compute $Q(G)$ when G is big and Q is expensive, and when we have constrained resources? In other words, can we provide small businesses with the benefit of big graph analysis?

We tackle these questions in this paper. We propose a theory of bounded evaluability, which helps us answer queries in big graphs with constrained resources [19–21, 23,37,40]. Based on the theory, we introduce a resource-constrained framework to query big graphs.

(2) Catching inconsistencies. To make practical use of big data, we have to cope with not only its quantity (volume) but also its quality (velocity). Real-life data is dirty: "more than 25% of critical data in the world's top companies is flawed" [58]. Dirty data is costly. Indeed, "bad data or poor data quality costs US businesses $600 billion annually" [32], "poor data can cost businesses 20%-35% of their operating revenue" [102], and "poor data across businesses and the government costs the US economy $3.1 trillion a year" [102].

The quality of real-life graph data is no better.

Example 1: It is common to find inconsistencies in knowledge bases that are being widely used.

(a) DBPedia: Flight A123 has two entries with the same departure time 14:50 and arrival time 22:35, but one entry is from Paris to New York, while the other is from Paris to Singapore [93].

(b) DBPedia: John Brown is claimed to be both a child and a parent of the same person, Owen Brown.

(c) Yago: Soccer player David Beckham is labeled with two birth places Leytonstone and Old Trafford [31].

(d) MKNF marks that all birds can fly and penguins are birds [69], despite their evolved wing structures.

To build a knowledge base of high quality, effective methods have to be in place to catch inconsistencies in graph-structured data. Indeed, consistency checking is a major challenge to knowledge acquisition and knowledge base enrichment, among other things. □

This highlights the need for theory and techniques to improve data quality. To catch semantic inconsistencies, we need data quality rules, which are typically expressed as dependencies. For relational data, a variety of dependencies have been studied, such as conditional functional dependencies (CFDs) [38] and denial constraints [9]. Employing the dependencies, a host of techniques have been developed to detect errors in relational data and repair the data (see [36] for a survey).

When it comes to graphs, however, the study of dependencies is still in its infancy. Even primitive dependencies such as functional dependencies and keys are not yet well studied for graph-structured data. Such dependencies are particularly important for graphs since unlike relational databases, real-life graphs typically do not come with a schema. Dependencies provide us with one of few means to specify a fundamental part of the semantics of the data, and help us detect inconsistencies in knowledge bases and catch spams in social networks [53], among other things. However, as will be seen shortly, dependencies for graph-structured data are far more challenging than their relational counterparts.

We introduce a class of graph functional dependencies, referred to as GFDs [53]. GFDs capture both attribute-value dependencies and topological structures of entities, and subsume CFDs as a special case. We show that GFDs can be used as data quality rules and are capable of catching inconsistencies commonly found in knowledge bases, as violations of the GFDs. We study the classical problems for reasoning about GFDs, such as their satisfiability, implication and validation problems. We also show that there exist effective algorithms for catching violations of GFDs in large-scale graphs, which are parallel scalable under practical conditions.

(3) Identifying associations. Association rules have been well studied for discovering regularities between items in relational data, and have proven effective in marketing activities such as promotional pricing and product placements [5, 106]. They have a traditional form $X \Rightarrow Y$, where X and Y are disjoint itemsets. For example, ({dipper, milk} ⇒ {beer}) is an association rule indicating that if customers buy dipper and milk, then the chances are that they will also buy beer.

The need for studying associations between entities in graphs is also evident, in emerging applications such as social media marketing. Social media marketing is predicted to trump traditional marketing. Indeed, "90% of customers trust peer recommendations versus 14% who trust advertising" [1], "60% of users said Twitter plays an important role in their shopping" [96], and "the peer influence from one's friends causes more than 50% increases in odds of buying products" [12].

Example 2: Association rules for social graphs are defined on entities in a graph, not on itemsets. As examples, below are association rules taken from [51, 52].

(a) If x and x' are friends living in the same city c, there are at least 3 French restaurants in city c that x and x' both like, and if x' went to a newly opened French restaurant y in c, then x may also go to y.

(b) If person x is in a music club, and among the people whom x follows, at least 80% of them like an album y, then it is likely that x will also buy y.

(c) If all the people followed by x buy Nova Plus (a brand of mobile phones), and none of them gives Nova Plus a bad rating, then the chances are that x may also buy Nova Plus.

These rules help us identify potential customers. For example, consider a newly opened French restaurant y. If a person x satisfies the conditions specified in rule (a) above, then restaurant y may opt to send x a coupon, and the chances are that x will become a customer of y. Similarly for rules (b) and (c), which help music album vendors and mobile phone manufactures find potential customers and advertise their new products. □

As opposed to association rules for itemsets, association rules for graphs, referred to as GPARs, involve social groups with multiple entities. GPARs depart from association rules for itemsets, and introduce several challenges. (1) To identify social groups, the rules need to be defined in terms of graph pattern matching, possibly with counting quantifiers (see rules (b) and (c)). (2) As will be seen later, conventional support and confidence metrics no longer work for GPARs. (3) It is intractable to discover top-ranked diversified GPARs, and conventional mining algorithms for traditional rules and frequent graph patterns cannot be directly used to discover such rules. (4) A major application of such rules is to identify potential customers in social graphs. This is costly: graph pattern matching by subgraph isomorphism is intractable. Worse still, real-life social graphs are typically big, as remarked earlier.

We propose a class of GPARs defined in terms of graph patterns [51] and counting quantifiers [52]. These GPARs differ from conventional association rules for itemsets in both syntax and semantics. They are useful in social media marketing, community structure analysis, social recommendation, knowledge extraction and link prediction [82], among other things. We propose topological support and confidence measures for GPARs. We also study the problem of discovering top-k diversified GPARs, and the problem of identifying potential customers with GPARs, establishing their complexity bounds and providing algorithms that are parallel scalable under practical conditions.

Organization. This paper is a progress report of our recent work. The remainder of the paper is organized as follows. We start with basic notations in Section 2. We then present a theory of bounded evaluation and a resource-bounded framework for querying big graphs in Section 3. We propose GFDs in Section 4, from formulation to classical decision problems to their applications. We present association rules for graphs in Section 5, and show how the rules help us in social media marketing. Open problems are identified in Section 6.

The study of big graphs has raised as many questions as it has answered. We hope that the paper will incite interest in the study of big graphs, and we invite interested colleagues to join forces with us in the study.

2 Preliminaries

We first review basic notations of graphs and queries that will be used in the rest of the paper.

Graphs. We consider *w.l.o.g.* directed graphs $G = (V, E, L)$, where (1) V is a finite set of nodes; (2) $E \subseteq V \times V$ is a set of edges, in which (v, v') denotes an edge from node v to v'; (3) each node v in V carries a label $L(v)$ taken from an alphabet Σ of labels, indicating the content of the node, as found in social networks, knowledge bases and property graphs.

We denote the size of G as $|G| = |V| + |E|$.

We will use two notions of subgraphs. A graph $G' = (V', E', L')$ is called a *subgraph of* G if $V' \subseteq V$, $E' \subseteq E$, and for each node $v \in V'$, $L'(v) = L(v)$.

Subgraph G' is said to be *induced by* V' if E' consists of all the edges in G whose endpoints are both in V'.

Graph pattern matching. As an example of graph queries, we take graph pattern matching defined in terms of subgraph isomorphism, stated as follows.

A *graph pattern* Q is a graph (V_Q, E_Q, L_Q), in which (a) V_Q is a set of *query nodes*, (b) E_Q is a set of *query edges*, and (c) each node $u \in V_Q$ carries a label $L_Q(u)$.

A *match* of pattern Q in a graph G is a subgraph G_s of G that is isomorphic to Q, *i.e.,* there exists a *bijective function* h from V_Q to the set of nodes of G_s such that (a) for each node $u \in V_Q$, $L_Q(u) = L(h(u))$, and (b) (u, u') is an edge in Q if and only if $(h(u), h(u'))$ is an edge in G_s. The answer $Q(G)$ to Q in G is the set of all matches of Q in G. The problem is as follows.

- Input: A graph G and a pattern Q.
- Output: The set $Q(G)$ of all matches of Q in G.

The graph matching problem is intractable: it is NP-complete to decide whether $Q(G)$ is empty (cf. [88]).

3 Querying Big Graphs

We start with querying big real-life graphs with constrained resources, in order to provide small businesses with the benefit of big graph analyses. We first present a theory of bounded evaluability in Section 3.1. We then propose a resource-constrained framework to cope with the sheer volume of big graphs, based on the theory and approximate query answering in Section 3.2. This section is based on results from [21, 37, 46, 48, 49].

3.1 Bounded Evaluability

Consider graph pattern queries Q defined in terms of subgraph isomorphism. As remarked earlier, such pattern queries are intractable and expensive.

Can we still efficiently compute exact answers $Q(G)$ to pattern queries when graphs G is big and when we have constrained resources such as a single processor?

Bounded evaluability. We approach this by *making big graphs small*. The idea is to make use of a set $\mathcal{A}$ of access constraints, which are a combination of indices and simple cardinality constraints defined on the labels of neighboring nodes of G. Given a query Q, we check whether Q is *boundedly evaluable* under $\mathcal{A}$, *i.e.,* whether for *all* graphs G that satisfy the access constraints of $\mathcal{A}$, there exists a subgraph $G_Q \subset G$ such that

(a) $Q(G_Q) = Q(G)$, and

(b) the size $|G_Q|$ of G_Q and the time for identifying G_Q are determined by $\mathcal{A}$ and Q only, *independent of* $|G|$.

If Q is boundedly evaluable, we generate a query plan that for all G satisfying $\mathcal{A}$, computes $Q(G)$ by accessing (visiting and fetching) a small G_Q in time *independent of* $|G|$, no matter how big G is. More specifically, we identify G_Q by reasoning about the cardinality constraints of $\mathcal{A}$, and fetch G_Q by using the indices in $\mathcal{A}$.

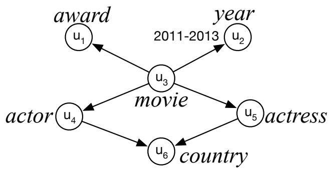

Fig. 1 Pattern query Q_0 on IMDb

A large number of real-life queries are actually boundedly evaluable under simple access constraints, as illustrated by the example below, taken from [21].

Example 3: Consider IMDb [71], a graph G_0 in which nodes represent movies, casts, countries, years and awards from 1880 to 2013, and edges denote various relationships between the nodes. An example query on IMDb is to *find pairs of first-billed actor and actress (main characters) from the same country who co-stared in a award-winning film released in 2011-2013.*

The query can be represented as a graph pattern Q_0 shown in Figure 1. It is to first find the set $Q_0(G_0)$ of matches, *i.e.,* subgraphs G' of G_0 that are isomorphic to Q_0; it then extracts and returns actor-actress pairs from each match G'. The challenge is that $Q_0(G_0)$ takes exponential time to compute on the IMDb graph, which has 5.1 million nodes and 19.5 million edges.

Not all is lost. Using simple aggregate queries one can readily find the following real-life cardinality constraints on the movie dataset from 1880–2013:

(a) in each year, every award is presented to no more than 4 movies (C1);

(b) each movie has at most 30 first-billed actors and actresses (C2), and each person has only one country of origin (C3); and

(c) there are no more than 135 years (C4, *i.e.,* 2013-1880), 24 major movie awards (C5) and 196 countries (C6) in IMDb in total [71].

An index can be built on the labels and nodes of G_0 for each of these cardinality constraints, yielding a set $\mathcal{A}_0$ of 8 access constraints. For instance, given a year and an award, the index for C1 returns at most 4 movies that received the award in that year.

Under $\mathcal{A}_0$, query Q_0 is *boundedly evaluable.* We can compute $Q_0(G_0)$ by accessing *at most* 17923 nodes and 35136 edges in G_0, *regardless of* the size of G_0, *no matter how big* G_0 is, by the following query plan:

(a) we first identify a set V_1 of 135 year nodes, 24 award nodes and 196 $\mathsf{country}$ nodes, by using the indices built for access constraints C4-C6;

(b) we then fetch a set V_2 of at most $24 \times 3 \times 4 = 288$ award-winning movies released between 2011–2013, with no more than $288 \times 2 = 576$ edges connecting movies to awards and years, by using those award and year nodes in V_1 and the index for C1;

(c) after these, we fetch a set V_3 of at most $(30 + 30) * 288 = 17280$ actors and $\mathsf{actress}$es with 17280 edges, by using the set V_2 and the index for C2; and

(d) we connect the actors and actresses in V_3 to $\mathsf{country}$ nodes in V_1, with at most 17280 edges by using the index for constraint C3. Finally, we output (actor, $\mathsf{actress}$) pairs connected to the same $\mathsf{country}$ in V_1.

The query plan visits at most 135 + 24 + 196 + 288 + 17280 = 17923 nodes, and 576 + 17280 + 17280 = 35136 edges, by using the cardinality constraints and indices in $\mathcal{A}_0$, as opposed to tens of millions of nodes and edges in IMDb. Moreover, the number of nodes and edges is decided by Q_0 and cardinality bounds in $\mathcal{A}_0$; it remains a constant no matter how big IMDb grows. □

Bounded evaluation. We next provide more insight into bounded evaluation of graph pattern queries. We invite the interested reader to consult [21] for details.

Access schema. An *access constraint* is of the form

$$S \to (l, N),$$

where $S \subseteq \Sigma$ is a (possibly empty) set of labels, l is a label in Σ, and N is a natural number. Recall that Σ is the alphabet of labels (see Section 2).

A graph $G(V, E, L)$ *satisfies* the access constraint if

- for any S-labeled set V_S of nodes in V, there exist at most N common neighbors of V_S with label l; and
- there exists an *index on S for l* that for any S-labeled set V_S in G, finds all common neighbors of V_S labeled with l in $O(N)$-time, independent of $|G|$.

Here V_S is a set in which each node is labeled with a distinct label in S. A node v is a *common neighbor* of V_S if for each node $v' \in V_S$, either (v, v') or (v', v) is an edge in G. In particular, when V_S is $\emptyset$, all nodes of G are common neighbors of V_S.

Intuitively, an access constraint is a combination of (a) *a cardinality constraint* and (b) *an index* on the labels of neighboring nodes. It tells us that for any S-node labeled set V_S, there exist a bounded number of common neighbors V_l labeled with l and moreover, V_l can be efficiently retrieved with the index.

Example 4: Constraints C1-C6 on IMDb given in Example 3 are access constraints φ_i (for $i \in [1, 8]$):

φ_1: $(\mathsf{year}, \mathsf{award}) \to (\mathsf{movie}, 4)$;
φ_2: $\mathsf{movie} \to (\mathsf{actors}, 30)$;
φ_3: $\mathsf{movie} \to (\mathsf{actress}, 30)$;
φ_4: $\mathsf{actor} \to (\mathsf{country}, 1)$;
φ_5: $\mathsf{actress} \to (\mathsf{country}, 1)$;
φ_6: $\emptyset \to (\mathsf{year}, 135)$;
φ_7: $\emptyset \to (\mathsf{award}, 24)$;
φ_8: $\emptyset \to (\mathsf{country}, 196)$.

Constraint φ_1 states that for any pair of year and award nodes, there are at most 4 movie nodes connected to both, *i.e.,* an award is given to at most 4 movies each year; similarly for φ_2–φ_5. Constraint φ_6 is simpler. It says that (between 1880 and 2013) there are at most 135 years in the entire graph; note that the set S (*i.e.,* the set V_S) for φ_6 is empty; similarly for φ_7 and φ_8. □

We denote a set $\mathcal{A}$ of access constraints as an *access schema*. We say that G *satisfies* $\mathcal{A}$, denoted by $G \models \mathcal{A}$, if G satisfies all the access constraints in $\mathcal{A}$.

Deciding bounded evaluability. To make practical use of bounded evaluation, we need to answer the following question, to decide whether a given query is boundedly evaluable under a set of available access constraints.

- *Input*: A pattern query Q, an access schema $\mathcal{A}$.
- *Question*: Is Q boundedly evaluable under $\mathcal{A}$?

The question is nontrivial for relational queries. It is decidable but EXPSPACE-hard for SPC queries and is undecidable for queries in the relational algebra [37].

The good news is that for graph pattern queries, the problem is in low polynomial time in the size of Q and $\mathcal{A}$, independent of data graphs G. Indeed, for pattern queries $Q = (V_Q, E_Q, L_Q)$, it is in $O(|\mathcal{A}||E_Q| + \|\mathcal{A}\||V_Q|^2)$ time to decide whether Q is boundedly evaluable under $\mathcal{A}$ [21], where $|E_Q|$ and $|V_Q|$ are the numbers of nodes and edges in Q, respectively; $\|\mathcal{A}\|$ is the number of constraints in $\mathcal{A}$, and $|\mathcal{A}|$ is the size of $\mathcal{A}$. In practice, Q and $\mathcal{A}$ are much smaller than data graphs G.

With this complexity bound, an algorithm for deciding the bounded evaluability of graph pattern queries is given in [21]. It is based on a characterization of bounded evaluability, *i.e.,* a sufficient and necessary condition for deciding whether a pattern query Q is boundedly evaluable under an access schema $\mathcal{A}$.

Generating bounded query plans. After a pattern query Q is found boundedly evaluable under an access schema $\mathcal{A}$, we need to generate a "good" query plan for Q that, given any (big) graph G, computes $Q(G)$ by fetching a small G_Q such that $Q(G) = Q(G_Q)$ and $|G_Q|$ is determined by Q and $\mathcal{A}$, independent of $|G|$.

In a nutshell, a *query plan* P for Q under $\mathcal{A}$ consists of three phases, presented as follows.

(1) Plan P tells us what nodes to retrieve from G. It starts with a sequence of *node fetching* operations of the form $\mathsf{fetch}(u, V_S, \varphi)$, where u is a l-labeled node in Q, V_S denotes a S-labeled set of Q, and φ is a constraint $\varphi = S \to (l, N)$ in $\mathcal{A}$. On a graph G, the operation is to retrieve a set $V(u)$ of *candidate matches* for u from G: given V_S that was retrieved from G earlier, it fetches common neighbors of V_S from G that are labeled with l. These nodes are fetched by using the index of φ and are stored in $V(u)$. In particular, when $S = \emptyset$, the operation fetches all l-labeled nodes in G as $V(u)$ for u.

The operations $\mathsf{fetch}_1, \mathsf{fetch}_2, \cdots, \mathsf{fetch}_n$ in P are executed one by one. In fetch_i, its V_S consists of nodes from V_j fetched earlier by fetch_j for $j < i$.

(2) From the data fetched by P, a subgraph $G_Q(V_\mathsf{P}, E_\mathsf{P})$ is built. It takes care to ensure that $Q(G_Q) = Q(G)$. More specifically, (a) V_P consists of candidates $V(u)$ fetched for each pattern node u in Q; and (b) E_P consists of edges (v, v') in $V_u \times V_{u'}$ if (u, u') is a pattern edge in Q; checking whether (v, v') is an edge in G is also confined to the nodes fetched via access constraints and thus can also be done with bounded data access.

(3) Finally, plan P simply computes $Q(G_Q)$ as $Q(G)$.

We say that P is a *bounded query plan for* Q if for all graphs $G \models \mathcal{A}$, it builds a subgraph G_Q of G such that (a) $Q(G_Q) = Q(G)$, and (b) it accesses G via fetch operations only, and each fetch is controlled by an access constraint φ in $\mathcal{A}$. Since P fetches data from G by using the indices in $\mathcal{A}$ only, the time for fetching data from G by all operations in P depends on $\mathcal{A}$ and Q only. That is, P fetches a bounded amount of data from G and builds a small G_Q from it. As a consequence, $|G_Q|$ is also *independent of* the size $|G|$ of G.

An algorithm is developed in [21] that, given any boundedly evaluable pattern query Q under an access schema $\mathcal{A}$, finds a bounded query plan for Q in $O(|V_Q||E_Q||\mathcal{A}|)$ time. As remarked earlier, Q and $\mathcal{A}$ are much smaller than data graphs G.

Effectiveness. The approach has been verified effective using real-life graphs consisting of billions of nodes and edges [21]. We find the following. (1) Under a couple of hundreds of access constraints, more than 60% of pattern queries are boundedly evaluable. (2) Bounded query plans outperform conventional algorithms such as VF2 [26] by 4 orders of magnitude, and access G_Q such that $|G_Q| = 3.2 * 10^{-5} * |G|$ on average, reducing $|G|$ of PB size to 32 GB. (3) It takes at most 37ms to decide whether a pattern query Q is boundedly evaluable and to generate a bounded query plan for bounded Q.

Related work. As remarked earlier, the principle behind bounded evaluation is to make big graphs small. There are typically two ways to reduce search space. (1) Graph indexing uses precomputed global information of G to compute distance [25], shortest paths [64] or substructure matching [90]. (2) Graph compression computes a summary G_c of a big graph G and uses G_c to answer all queries posed on G [15, 45, 86].

In contrast to the prior work, (1) bounded evaluation is based on access schema, which extends traditional indices by incorporating cardinality constraints, such that we can reason about the cardinality constraints and decide whether a query can be answered by accessing a bounded amount of data in advance, before we access the underlying graphs. Moreover, the indices in an access schema are based on labels of neighboring nodes, which are quite different from prior indexing structures. (2) Instead of using one-size-fit-all compressed graphs G_c to answer *all queries* posed on G, we adopt a *dynamic data reduction scheme* that finds a subgraph G_Q of G for each query Q. Since G_Q consists of only the information needed for answering Q, it allows us to compute $Q(G)$ by using G_Q that is much smaller than G_c and hence, using much less resources. (3) When Q is boundedly evaluable, for *all* graphs G that satisfy $\mathcal{A}$ we can find G_Q of size *independent of* $|G|$; in contrast, $|G_c|$ may be proportional to $|G|$.

The theory of bounded evaluation was first studied for relational queries [19, 20, 23, 37, 40]. It has proven effective on a variety of real-life datasets. It is shown that on average 77% of SPC queries [23] and 67% of relational algebra queries [19] are boundedly evaluable under a few hundreds of access constraints. Bounded evaluation outperforms commercial query engines by 3 orders of magnitude, and in fact, the gap gets larger on bigger datasets. The evaluation results from our industry collaborators are even more encouraging. They find that more than 90% of their big-data queries are boundedly evaluable, improving the performance from 25 times to 5 orders of magnitude [22].

The theory is extended from relations to graphs in [21], showing that bounded evaluation is also effective for graph pattern queries defined in terms of subgraph isomorphism and graph simulation.

3.2 A Resource-Constrained Framework

As remarked earlier, we can answer about 60% of pattern queries in big graphs by accessing a bounded amount of data no matter how big the graphs grow. Then, what should we do about the queries that are not boundedly evaluable under an access schema? Can we still answer those queries with constrained resources?

To this end, we propose a resource-constrained framework to query big graphs, which can be readily built on top of (parallel) graph query engines.

A framework to query big graphs. The framework, referred to as RESOURCE (RESOURce Constrained Engine), aims to answer queries posed on big graphs when we have constrained resources such as limited available processors and time. To measure the constraints on resources, it takes a *resource ratio* $\alpha \in (0, 1]$ as a parameter, indicating that our available resources allow us to only access a α-fraction of a big graph. Employing an access schema $\mathcal{A}$, RESOURCE works as follows. Given a query Q and a graph G that satisfies $\mathcal{A}$,

(1) it first checks whether Q is *boundedly evaluable* under $\mathcal{A}$, *i.e.,* whether exact answers $Q(G)$ can be computed by accessing a fraction $G_Q \subseteq G$ such that its size $|G_Q|$ is *independent of* the size $|G|$ of G;

(2) if so, it computes $Q(G)$ by accessing a bounded fraction G_Q of G, by generating a bounded query plan under $\mathcal{A}$ as described in Section 3.1;

(3) otherwise, it answers Q in G by means of data-driven approximation [42,49], which accesses a small $G_Q \subseteq G$ in the entire process such that $|G_Q| \leq \alpha|G|$, possibly by also using access constraints in $\mathcal{A}$.

That is, under resource constraint specified by α, RESOURCE computes exact answers $Q(G)$ whenever bounded evaluation is possible by employing access schema $\mathcal{A}$; otherwise, it returns approximate answers $Q(G_Q)$ within the given budget $\alpha|G|$.

We next give more details about RESOURCE.

(1) Resource ratio α. The ratio is decided by available resources and the complexity of the class of queries to be processed. For example, for graph pattern queries (an essentially exponential-time process), one may pick an α smaller than the one for reachability queries (to decide whether there exists a path from one node to another, which is a linear-time problem). Intuitively, it indicates the "resolution" of the data we can afford: the larger α is, the more accurate the query answers are.

(2) Data-driven approximation. For each class $\mathcal{Q}$ of graph queries of users' choice, one can develop a data-driven approximation algorithm. Given a query $Q \in \mathcal{Q}$ posed on a (possibly big) graph G, the approximation algorithm identifies a fraction G_Q such that $|G_Q| \leq \alpha|G|$, and computes $Q(G_Q)$ as approximate answer to Q in G. A detailed presentation of the data-driven approximation scheme can be found in [42].

Such a data-driven approximation algorithm has been developed for graph pattern queries for personalized social search [49], as used by Graph Search of Facebook. Experimenting with real-life social graphs, we find that the algorithm easily scales with large-scale graphs: when graphs grows big, we simply decrease α and hence, access smaller amount of data. Better still, the algorithm is accurate: even when the resource ratio α is as small as $15*10^{-6}$, the algorithm returns matches with 100% accuracy. That is, when G consists of 1PB of data, $\alpha|G|$ is down to 15GB, *i.e.,* data-driven approximation makes big data small, without paying too high a price of sacrificing the accuracy of query answers.

(3) Algorithms underlying RESOURCE. RESOURCE can be built on top of existing graph query engines provided with the following algorithms:

1. *offline algorithms* for discovering access constraints from real-life graphs and for maintaining the constraints in response to changes to the graphs; and
2. *online algorithms* for deciding whether a query is boundedly evaluable under an access schema, generating a bounded query plan for a boundedly evaluable query, and for data-driven approximation. As remarked earlier, these algorithms are already available for graph pattern queries (Section 3.1 and [49]).

The framework can also incorporate other techniques for querying big graphs, by making big graphs small, including but not limited to the following.

(a) Query driven approximation. For an expensive query class $\mathcal{Q}$, we can approximate its queries by adopting a cheaper class Q' of queries. For instance, for social community detection, one may want to use bounded graph simulation [43, 83], which takes cubic-time, instead of subgraph isomorphism, for which the decision problem is NP-complete. Another example is to compute top-k diversified answers for queries of $\mathcal{Q}$, instead of computing the entire set $Q(G)$ of answers [47] (see [42] for details of query-drive approximation).

(b) Query preserving graph compression. We may compress a big graph G relative to a query class $\mathcal{Q}$ of users' choice [45]. More specifically, a *query preserving graph compression* for $\mathcal{Q}$ is a pair $\langle R, P\rangle$ of functions, where $R(\cdot)$ is a *compression function*, and $P(\cdot)$ is a *post-processing function.* For any graph G, $G_c = R(G)$ is the compressed graph computed from G by $R(\cdot)$, such that (i) $|G_c| \leq |G|$, and (ii) for all queries $Q \in \mathcal{Q}$, $Q(G) = P(Q(G_c))$. Here $P(Q(G_c))$ is the result of post-processing the answers $Q(G_c)$ to Q in G_c.

That is, we *preprocess* G by computing the compressed G_c of G offline. After this step, for any query

$Q \in \mathcal{Q}$, the answers $Q(G)$ to Q in the original big G can be computed by evaluating the same query Q on the smaller G_c online, *without decompressing* G_c. The compression schema may be lossy: we do not need to restore the original graph G from G_c. That is, G_c only needs to retain the information necessary for answering queries in $\mathcal{Q}$, and hence can achieve a better compression ratio than lossless compression schemes. The effectiveness of this approach has been verified in [45].

(c) Query answering using views. Given a query $Q \in \mathcal{Q}$ and a set $\mathcal{V}$ of view definitions, *query answering using views* is to reformulate Q into another query Q' such that (i) Q and Q' are equivalent, *i.e.,* for all graphs G, Q and Q' produce the same answers in G, and moreover, (ii) Q' refers only to $\mathcal{V}$ and its extensions (small cached views) $\mathcal{V}(G)$, *without accessing the underlying G*.

More specifically, given a big graph G, one may identify a set $\mathcal{V}$ of views (pattern queries) and materialize them with $\mathcal{V}(G)$ of matches for patterns of $\mathcal{V}$ in G, as a *preprocessing* step offline. We compute matches of input queries Q online by using $\mathcal{V}(G)$ only. In practice, $\mathcal{V}(G)$ is typically much smaller than G, and hence, this approach allows us to query big G by accessing small $\mathcal{V}(G)$. Better still, the views can be incrementally maintained offline in response to changes to G, and adaptively adjusted to cover various queries [50].

One can further extend the traditional notion of query answering using views, by incorporating bounded evaluation, as studied for relational queries [20].

(d) Parallel query processing. RESOURCE can be built on top of a parallel graph query engine, and hence combine parallel query processing with bounded evaluation and data-driven approximation. In particular, we promote GRAPE, a parallel GRAPh Engine [54]. It allows us to "plug in" existing sequential graph algorithms, and makes the computations parallel across multiple processors, without drastic degradation in performance or functionality of existing systems.

GRAPE has the following unique feature. The state-of-the-art parallel graph systems require users to recast existing graph algorithms into a new model. While graph computations have been studied for decades and a large number of sophisticated sequential graph algorithms are already in place, to use Pregel, for instance, one has to "think like a vertex" and recast the existing algorithms into Pregel; similarly when programming with other systems. The recasting is nontrivial for people who are not very familiar with the parallel models. This makes these systems a privilege for experienced users only, just like computers three decades ago that were accessible only to people who knew DOS or Unix.

In contrast, GRAPE supports a simple programming model. For a class $\mathcal{Q}$ of graph queries, users only need to plug in three existing sequential (incremental) algorithms for $\mathcal{Q}$, without the need for recasting the algorithms into a new model. GRAPE automatically *parallelizes* the computation across processors, and inherits all optimization strategies developed for sequential graph algorithms. This makes parallel graph computations accessible to users who know conventional graph algorithms covered in undergraduate textbooks.

Better still, GRAPE is based on a principled approach by combining partial evaluation and incremental computation, and can be modeled as fixpoint computation. As shown in [54], it guarantees its parallel processing to terminate with correct answers as long as the sequential algorithms plugged in are correct.

In addition, automated parallelization does not imply performance degradation. Indeed, GRAPE outperforms Giraph [59] (a open-source version of Pregel [84]), GraphLab [81] and Blogel [104] in both response time and communication costs, for a variety of computations such as graph traversal, pattern matching, connectivity and keyword search. We invite the interested reader to consult [54] for the details of GRAPE.

Related work. In addition to bounded evaluation, RESOURCE highlights data-driven approximation. Recall that traditional approximate query answering is often based on synopses such as sampling, sketching, histogram or wavelets (see [24, 27] for surveys). It is to compute a synopsis G' of a graph G, and use G' to answer all queries posed on G. As opposed to a one-size-fit-all G', data-driven approximation dynamically identifies G_Q for each input query Q, and hence achieves a higher accuracy of approximate query answers.

There has also been work on dynamic sampling for answering relational aggregate queries, *e.g.,* [4, 10]. Assuming certain information about a query load, *e.g.,* queries, the frequency of columns used in queries, or system logs, the prior work adaptively pre-computes samples offline and picks some samples for answering the "predictable queries" online. In contrast, we study graph queries, where sampling is much harder. This is because (a) the graph queries are rather "unpredictable" due to topological constraints embedded in graph queries, and (b) as opposed to homogeneous relational data, there is no "one-fit-for-all" schema available for data nodes in a graph. We also do not assume the existence of abundant query logs and workload for sampling strategy. Instead, we develop dynamic reduction techniques to identify and only access promising "areas" that lead to reasonable approximate answers.

Related to the data-driven approximation scheme are also anytime algorithms [107], which allow users either to specify a budget on resources (*e.g.,* running time, known as contract algorithms [108]), or to terminate the run of the algorithms at any time and return intermediate answers as approximate answers (known as interruptible algorithms [65]). Contract anytime algorithms have been explored for (a) budgeted search such as bounded-cost planning [98, 100, 107] under a user-specified budget; and (b) graph search via subgraph isomorphism, to find intermediate answers within the budget, either by assigning dynamically maintained budgets and costs to nodes during the traversal [17], or by deciding search orders based on the frequencies of certain features in queries and graphs [95].

In contrast, RESOURCE (a) computes exact answers whenever bounded evaluation is possible, instead of heuristics; (b) it aims to strike a balance between the cost of finding solutions and the quality of the answers, by dynamic data reduction; and (c) it takes a given (arbitrarily small) ratio α as a parameter, accesses promising nodes only and guarantees bounded search space, by leveraging access schema as much as possible.

4 Dependencies for Graphs

We now turn to the other side of big graphs, namely, the quality of graph-structured data. As remarked earlier, when the data is dirty, query answers computed in the data may not be correct and may even do more harm than good, no matter how efficient and scalable our systems and algorithms are for querying big graphs.

To catch inconsistencies in graphs, we propose a class of functional dependencies for graphs, referred to as GFDs, in Section 4.1. We settle the classical problems for reasoning about GFDs in Section 4.2. We make use of GFDs to catch errors in real-life graphs in Section 4.3.

The main results of this section come from [35, 53].

4.1 GFDs: Graph Functional Dependencies

We now present GFDs introduced in [53]. GFDs are defined with graph patterns. To simplify the discussion, we extend the notation of Section 2 and write a graph pattern as $Q[\bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where V_Q and E_Q are the same as before; L_Q is extended to also associate edges with labels; $\bar{x}$ is a list of distinct variables, one for each node in V_Q; and μ is a bijective mapping from $\bar{x}$ to V_Q, *i.e.,* it assigns a distinct variable to each node v in V_Q. For $x \in \bar{x}$, we use $\mu(x)$ and x interchangeably when it is clear in the context.

We also allow wildcard '_' as a special label in L_Q.

GFDs. A GFD φ is a pair $Q[\bar{x}](X \rightarrow Y)$, where

- $Q[\bar{x}]$ is a graph pattern, called the *pattern* of φ; and
- X and Y are two sets of literals of $\bar{x}$.

Here a *literal* of $\bar{x}$ has the form of either $x.A = c$ or $x.A = y.B$, where $x, y \in \bar{x}$, A and B denote attributes (not specified in Q), and c is a constant.

Intuitively, GFD φ specifies two constraints:

- a *topological constraint* imposed by pattern Q, and
- *attribute dependency* specified by $X \rightarrow Y$.

Recall that the "scope" of a relational functional dependency (FD) $R(X \rightarrow Y)$ is specified by a relation schema R: the FD is applied only to instances of R. Unlike relational databases, graphs do not have a schema. Here Q specifies the scope of the GFD, such that the dependency $X \rightarrow Y$ is imposed only on the attributes of the vertices in each subgraph identified by Q. Constant literals $x.A = c$ enforce bindings of semantically related constants, along the same lines as CFDs [38].

Example 5: To catch the inconsistencies in real-life knowledge bases described in Example 1, we use GFDs defined with patterns Q_1–Q_4 of Fig. 2 as follows.

(1) Flight: GFD $\varphi_1 = Q_1[x, x_1\text{-}x_5, y, y_1\text{-}y_5](X_1 \rightarrow Y_1)$, in which pattern Q_1 specifies two flight entities, where μ maps x to a flight, x_1–x_5 to its id, departure city, destination, departure time and arrival time, respectively; similarly for y and y_1–y_5; in addition, val is an attribute indicating the content of a node (not shown in Q_1). In φ_1, X_1 is $x_1.\mathsf{val} = y_1.\mathsf{val}$, and Y_1 consists of $x_2.\mathsf{val} = y_2.\mathsf{val}$ and $x_3.\mathsf{val} = y_3.\mathsf{val}$.

Intuitively, GFD φ_1 states that for all flight entities x and y, if they share the same flight id, then they must have the same departing city and destination.

(2) Parent-child: GFD $\varphi_2 = Q_2[x, y](\emptyset \rightarrow \mathsf{false})$, where Q_2 specifies a pair of persons connected by child and parent relationships. It states that there exists no person entity x who is both a child and a parent of another person entity y. Note that X in Q_2 is an empty set, *i.e.,* no precondition is imposed on the attributes of Q_2; and Y is Boolean constant false, a syntactic sugar that can be expressed as, *e.g.,* $y.A = c \wedge y.A = d$ for distinct constants c and d, for attribute A of y.

(3) Birth places: GFD $\varphi_3 = Q_3[x, y, z](\emptyset \rightarrow y.\mathsf{val} = z.\mathsf{val})$, where Q_3 depicts a person entity with two distinct cities as birth places. Intuitively, φ_3 is to ensure that for all person entities x, if x has two birth places y and z, then y and z share the same name.

(3) Generic is_a: GFD $\varphi_4 = Q_4[x, y](\emptyset \rightarrow x.A = y.A)$. It enforces a general property of is_a relationship: if en-

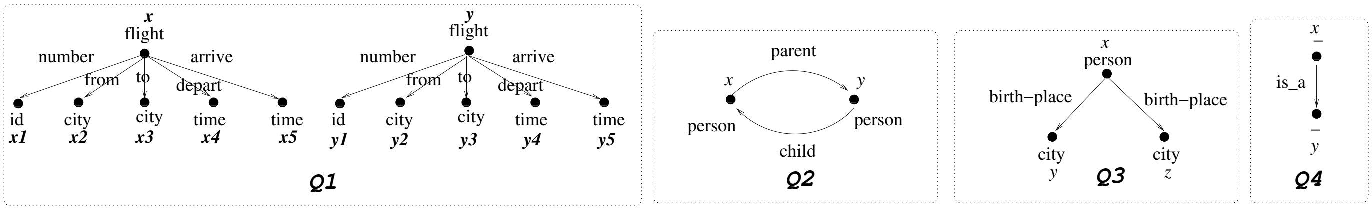

Fig. 2 Graph patterns

tity y is_a x, then for any property A of x (represented by attribute A), $x.A = y.A$. In particular, if x is labeled with bird, y with penguin, and A is can_fly, then φ_4 catches the inconsistency described in Example 1. Observe that x and y in Q_4 are labeled with wildcard '_', to match arbitrary generic entities. □

Semantics. To interpret GFD $\varphi = Q[\bar{x}](X \rightarrow Y)$, we use the following notations. We denote a match of pattern Q in a graph G as a vector $h(\bar{x})$, consisting of $h(x)$ (*i.e.,* $h(\mu(x))$) for all $x \in \bar{x}$, in the same order as $\bar{x}$.

Consider a match $h(\bar{x})$ of Q in G, and a literal $x.A = c$ of $\bar{x}$. We say that $h(\bar{x})$ *satisfies* the literal if *there exists* attribute A at the node $v = h(x)$ (*i.e.,* $v = h(\mu(x))$) and $v.A = c$; similarly for literal $x.A = y.B$. We denote by $h(\bar{x}) \models X$ if $h(\bar{x})$ satisfies *all* the literals in X; similarly for $h(\bar{x}) \models Y$. Here we write $h(\mu(x))$ as $h(x)$, where μ is the mapping in Q from $\bar{x}$ to nodes in Q. We write $h(\bar{x}) \models X \rightarrow Y$ if $h(\bar{x}) \models Y$ whenever $h(\bar{x}) \models X$.

We say that graph G *satisfies* GFD $\varphi = Q[\bar{x}](X \rightarrow Y)$, denoted by $G \models \varphi$, if *for all* matches $h(\bar{x})$ of Q in G, we have that $h(\bar{x}) \models X \rightarrow Y$.

To check whether $G \models \varphi$, we need to examine all matches of Q in G. In addition, observe the following.

(1) for a literal $x.A = c$ in X, if $h(x)$ has *no* attribute A, then $h(\bar{x})$ trivially satisfies $X \rightarrow Y$. That is, node $h(x)$ is *not* required to have attribute A; similarly for literals $x.A = y.B$. This allows us to accommodate the semi-structured nature of graphs.

(2) In contrast, if $x.A = c$ is in Y and $h(\bar{x}) \models Y$, then $h(x)$ must have attribute A by the definition of satisfaction above; similarly for $x.A = y.B$.

(3) When X is $\emptyset$, $h(\bar{x}) \models X$ for any match $h(\bar{x})$ of Q in G. That is, empty X indicates Boolean constant true.

(4) When $Y = \emptyset$, it indicates that Y is constantly true, and φ becomes trivial. When Y is false and $X = \emptyset$, $G \not\models \varphi$ if there exists a match of Q; *i.e.,* φ states that Q is an "illegal" pattern that should not find any matches.

Intuitively, if a match $h(\bar{x})$ of pattern Q in G *violates* the attribute dependency $X \rightarrow Y$, *i.e.,* $h(\bar{x}) \models X$ but $h(\bar{x}) \not\models Y$, then the subgraph induced by $h(\bar{x})$ is inconsistent, *i.e.,* its entities have inconsistencies.

Example 6: Recall the inconsistencies about Flight A123 in DBPedia from Example 1, and GFD φ_1 from Example 5. Then there exists a match $h(\bar{x})$ of the pattern Q_1 of φ_1 in the graph depicting DBPedia, such that $h(x)$ and $h(y)$ have the same id, *i.e.,* $h(\bar{x}) \models X_1$; however, $h(\bar{x}) \not\models h(Y_1)$, a violation of φ_1. That is, φ_1 catches the inconsistencies of the flight in DBPedia.

Similarly, we can apply φ_2–φ_4 of Example 5 as data quality rules to knowledge bases, and catch the other inconsistencies described in Example 1. □

We say that a graph G *satisfies* a set Σ of GFDs if for all $\varphi \in \Sigma$, $G \models \varphi$, *i.e.,* G satisfies every GFD in Σ.

Special cases. GFDs have the following special cases.

(1) As shown in [53], relational FDs and CFDs can be expressed as GFDs when tuples in a relation are represented as nodes in a graph. In fact, GFDs are able to express equality-generating dependencies (EGDs) [3].

(2) GFDs can specify certain type information. For an entity x of type τ, GFD $Q[x](\emptyset \rightarrow x.A = x.A)$ enforces that x must have an A attribute, where Q consists of a single vertex labeled τ and denoted by variable x. However, GFDs *cannot* enforce that an attribute A of x has a finite domain, *e.g.,* Boolean. In relational databases, finite domains are specified by a relational schema, which are typically not in place for real-life graphs.

(3) As shown by φ_2 of Example 5, we can express "forbidding" GFDs of the form $Q[\bar{x}](X \rightarrow \text{false})$, where X is satisfiable. A forbidding GFD states that there exists no nonempty graph G such that Q can find a match $h(\bar{x})$ in G and $h(\bar{x}) \models X$. That is, Q and X put together specify an inconsistent combination.

(4) As indicated by φ_4 of Example 5, GFDs can express generic is_a relationship. Along the same lines, GFDs can enforce inheritance relationship subclass as well.

Related work. There has been work on extending relational FDs to graph-structured data, mostly focusing on RDF [6,18,28,66,67,77,105]. This line of work started from [77], by extending relational techniques to RDF. Based on triple patterns with variables, [6,28] define FDs with triple embedding, homomorphism and coincidence of variable valuations. Employing clustered values, [105] defines FDs with conjunctive path patterns; the work is extended to CFDs for RDF [66]. FDs are also defined by mapping relations to RDF [18], using tree patterns in which nodes represent relation attributes.

The class of GFDs differs from the prior work as follows. (a) GFDs are defined for general property graphs, not limited to RDF. (b) GFDs support topological constraints by incorporating (possibly cyclic) graph patterns with variables, as opposed to [18,66,105]. In contrast to [6, 28, 67, 77, 105] that take a value-based approach to defining FDs, GFDs are enforced on graph-structured entities identified by graph patterns via subgraph isomorphism. (c) GFDs support bindings of semantically related constants like CFDs [38], as well as forbidding GFDs with false. These allow us to specify data quality rules for consistency checking, but cannot be expressed as the FDs of [6,18,28,66,105]. (d) The validation and implication problems for GFDs have been settled [53], while matching complexity bounds for the FDs previously proposed are yet to be developed.

Related to GFDs is a class of keys defined for RDF [35]. Keys are defined as a graph pattern $Q[x]$, with a designated variable x denoting an entity. Intuitively, it indicates that for any two matches h_1 and h_2 of Q in a graph G, $h_1(x)$ and $h_2(x)$ refer to the same entity and should be identified. Keys are recursively defined, *i.e.,* Q may include entities other than x to be identified (perhaps with other keys), in order to match entities with a graph structure. Such keys aim to detect deduplicate entities and to fuse information from different sources that refers to the same entity, in knowledge fusion and knowledge base expansion; they also find applications in social network reconciliation, to reconcile user accounts across multiple social networks. We invite the interested reader to consult [35] for details.

4.2 Reasoning about GFDs

There are two classical problems associated with any class of dependencies, namely, the satisfiability and implication problems, which are stated as follows.

Satisfiability. A set Σ of GFDs is *satisfiable* if Σ has a *model*; that is, there exists a graph G such that (a) $G \models \Sigma$, and (b) for each GFD $Q[\bar{x}](X \rightarrow Y)$ in Σ, there exists a match of Q in G. Intuitively, it is to check whether the GFDs are "dirty" themselves when used as data quality rules. A model G of Σ requires all patterns in the GFDs of Σ to find a match in G, to ensure that the GFDs in Σ do not conflict with each other.

The *satisfiability problem* for GFDs is to determine, given a set Σ of GFDs, whether Σ is satisfiable.

Over relational data, any set Σ of FDs is satisfiable, *i.e.,* there always exists a nonempty relation that satisfies Σ [36]. However, a set Σ of conditional functional dependencies (CFDs) may not be satisfiable, *i.e.,* there exists no nonempty relation that satisfies Σ [38]. As GFDs subsume CFDs, it is not surprising that a set of GFDs may not be satisfiable, as shown in [53].

Implication. A set Σ of GFDs *implies* another GFD φ, denoted by $\Sigma \models \varphi$, if for all graphs G, if $G \models \Sigma$ then $G \models \varphi$, *i.e.,* φ is a logical consequence of Σ. In practice, the implication analysis helps us eliminate redundant data quality rules defined as GFDs, and hence, optimize our error detection process by minimizing rules.

The *implication problem* for GFDs is to decide, given a set Σ of GFDs and another GFD φ, whether $\Sigma \models \varphi$.

Complexity. These problems have been well studied for relational dependencies. For FDs, the satisfiability problem is in $O(1)$ time (since all FDs are satisfiable) and the implication problem is in linear time (cf. [3]). For CFDs, the satisfiability problem is NP-complete and the implication problem is coNP-complete in the presence of finite-domain attributes, but are in PTIME when all attributes involved have an infinite domain [38].

These problems have also been settled for GFDs [53]:

- the satisfiability problem is coNP-complete; and
- the implication problem is NP-complete.

The complexity bounds are rather robust, *e.g.,* the problems remain intractable for GFDs defined with graph patterns that are acyclic directed graphs (DAGs).

As shown in [53], the intractability of the satisfiability and implication problems arises from subgraph isomorphism embedded in these problems, which is NP-complete (cf. [88]). The complexity is not inherited from CFDs although GFDs subsume CFDs as a special case. Indeed, the satisfiability analysis of CFDs is NP-hard only under a relational schema that enforces attributes to have a *finite domain* [38], *e.g.,* Boolean, *i.e.,* the problem is intractable when CFDs and finite domains are put together. In contrast, graphs do not come with a schema; while GFDs subsume CFDs, they cannot specify finite domains. That is, the satisfiability problem for GFDs is already coNP-hard in the absence of a schema; similarly for the implication analysis.

Several tractable special cases of the satisfiability and implication problems for GFDs are identified in [53].

Putting these together, our main conclusion is that while GFDs are a combination of a topological constraint and an attribute dependency and are more complicated than CFDs, reasoning about GFDs is no harder than their relational counterparts such as CFDs.

4.3 Putting GFDs in Actions

One of the applications of GFDs is to detect inconsistencies in graph-structured data. That is, we use GFDs as data quality rules along the same lines as CFDs, and catch violations of the rules by means of the validation analysis of GFDs, which is stated as follows.

Validation analysis. Given a GFD $\varphi = Q[\bar{x}](X \rightarrow Y)$ and a graph G, we say that a match $h(\bar{x})$ of Q in G is a *violation* of φ if $G_h \not\models \varphi$, where G_h is the subgraph induced by $h(\bar{x})$. For a set Σ of GFDs, we denote by $\mathsf{Vio}(\Sigma, G)$ the set of all violations of GFDs in G, *i.e.,* $h(\bar{x}) \in \mathsf{Vio}(\Sigma, G)$ if and only if there exists a GFD φ in Σ such that $h(\bar{x})$ is a violation of φ in G. That is, $\mathsf{Vio}(\Sigma, G)$ collects all entities of G that are inconsistent when the set Σ of GFDs is used as data quality rules.

The *error detection problem* is stated as follows:

- *Input*: A set Σ of GFDs and a graph G.
- *Output*: The set $\mathsf{Vio}(\Sigma, G)$ of violations.

Recall that the error detection problem is in PTIME for relational FDs and CFDs. In fact, when FDs and CFDs are used as data quality rules, errors in relations can be detected by two SQL queries that can be automatically generated from the FDs and CFDs [38].

In contrast, error detection is more challenging in graphs. Indeed, consider the decision version of the problem, referred to as *the validation problem* for GFDs. It is to decide whether $G \models \Sigma$, *i.e.,* whether $\mathsf{Vio}(\Sigma, G)$ is empty. This problem is coNP-complete [53].

Parallel scalable algorithms. The error detection problem is intractable. As remarked earlier, real-life graphs are often of large scale. Then, is error detection feasible in real-life graphs? The answer is affirmative, by using parallel algorithms to compute $\mathsf{Vio}(\Sigma, G)$.

As shown in [53], there exist *parallel scalable* algorithms for detecting errors in graphs by using GFDs, with the following property. Denote by

- $t(|\Sigma|, |G|)$ the running time of a "best" *sequential algorithm* to compute $\mathsf{Vio}(\Sigma, G)$, *i.e.,* the least worst-case complexity among all such algorithms; and
- $T(|\Sigma|, |G|, p)$ the time taken by a parallel algorithm to compute $\mathsf{Vio}(\Sigma, G)$ by using p processors.

Then there exist parallel algorithms $\mathcal{T}_p$ such that

$$T(|\Sigma|, |G|, p) = \frac{c * t(|\Sigma|, |G|)}{p}$$

under certain practical conditions. Intuitively, $\mathcal{T}_p$ guarantees to reduce its running time when p gets larger. That is, the more processors are used, the less time it takes to compute $\mathsf{Vio}(\Sigma, G)$. In other words, it can scale with large-scale graphs despite the complexity, by increasing resources employed when graphs get larger.

5 Association Rules for Graphs

Besides the quantity and quality of big graphs, we next consider how to make practical use of big graph analyses in social media marketing, an emerging application.

We first introduce a class of primitive graph-pattern association rules, referred to as GPARs, in Section 5.1. We then explore possible extensions of GPARs, by adding counting quantifiers in Section 5.2. To apply GPARs in social media marketing, we finally address how to discover GPARs and how to identify potential customers by using GPARs, in Section 5.3.

The results of the section are taken from [51, 52].

5.1 GPARs: Graph Pattern Association Rules

We start with the GPARs introduced in [51].

GPARs. A *graph-pattern association rule* (GPAR) $R(x, y)$ is defined as $Q(x, y) \Rightarrow q(x, y)$, where $Q(x, y)$ is a graph pattern in which x and y are two designated nodes in Q, and $q(x, y)$ is an edge labeled q from x to y, *i.e.,* a relationship between x and y. We refer to Q and q as the *antecedent* and *consequent* of R, respectively.

The rule states that *for all nodes* v_x and v_y in a (social) graph G, if *there exists* a match $h \in Q(G)$ such that $h(x) = v_x$ and $h(y) = v_y$, *i.e.,* v_x and v_y match the designated nodes x and y in Q, respectively, then the consequent $q(v_x, v_y)$ will likely hold.

Intuitively, $q(v_x, v_y)$ may indicate that v_x is a potential customer of v_y. Denote by $Q(x, G)$ the set of $h(x)$ for all matches h in $\mathcal{Q}(G)$, *i.e.,* the matches of x in G via Q. Then in a social graph G, $Q(x, y)$ identifies potential customers by computing matches $Q(x, G)$.

We model $R(x, y)$ as *a graph pattern* P_R, by extending Q with a (dotted) edge $q(x, y)$. We refer to pattern P_R simply as R when it is clear from the context.

Example 7: Recall association rule (a) described in Example 2. It can be expressed as a GPAR $R_1(x, y)$: $Q_5(x, y) \Rightarrow \mathsf{visit}(x, y)$, as depicted in Fig. 3. Its antecedent is the pattern Q_5 (excluding the dotted edge) and its consequent is $\mathsf{visit}(x, y)$. As opposed to conventional association rules, the GPAR is specified with a graph pattern Q_5 that enforces topological conditions on various entities: associations between customers (the friend relation), customers and restaurants (like, visit), city and restaurants (in), and city and customers (in).

This GPAR helps us identify potential customers for restaurant y. In a social graph G, we find matches of

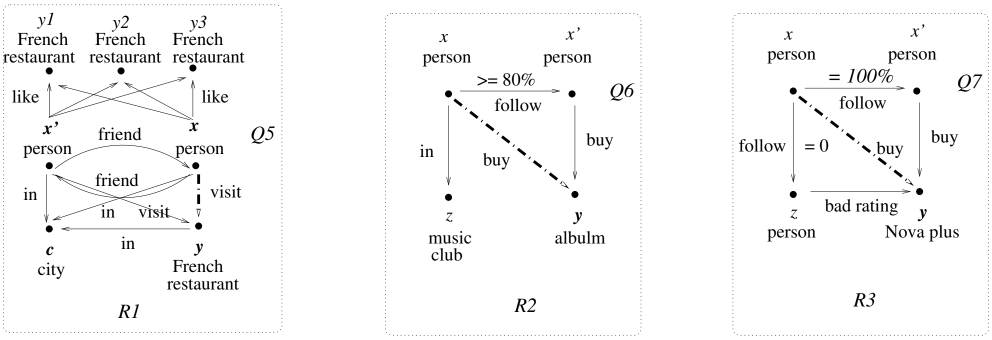

Fig. 3 Graph pattern association rules

pattern Q_5 via subgraph isomorphism; for x and y in each of the matches (subgraphs of G), *i.e.,* for x and y satisfying the antecedent of Q_1, the chances are that x likes y, and hence we can recommend y to x. □

To simplify the discussion, we define the consequent of GPAR in terms of a single predicate $q(x, y)$ following [5]. However, a consequent can be readily extended to *multiple* predicates and even to a *graph pattern.* We consider nontrivial GPARs by requiring that (a) P_R is connected; (b) Q is *nonempty, i.e.,* it has at least one edge; and (c) $q(x, y)$ does not appear in Q.

Related work. Introduced in [5], association rules are traditionally defined on relations. Prior work on association rules for social networks [94] and RDF resorts to mining conventional rules and Horn rules (as conjunctive binary predicates) [56] on tuples with extracted attributes from graphs, instead of exploiting graph patterns. While [11] studies time-dependent rules via graph patterns, it focuses on evolving graphs and adopts different semantics for support and confidence.

GPARs extend association rules from relations to graphs. (a) It demands topological support and confidence metrics. (b) GPARs are interpreted with isomorphic functions and hence, cannot be expressed as conjunctive queries, which do not support negation or inequality needed for functions. (c) Applying GPARs becomes an intractable problem of multi-pattern-query processing in big graphs. (d) Mining (diversified) GPARs is beyond traditional rule mining from itemsets [106].

It should be remarked that conventional association rules [5] and a range of predication and classification rules [92] can be considered as a special case of GPARs, since their antecedents can be readily modeled as a graph pattern in which nodes represent items.

5.2 Adding Counting Quantifiers

In applications such as social media marketing, knowledge discovery and cyber security, more expressive patterns are needed, notably ones with counting quantifiers. In light of this, we extend GPARs with quantified graph patterns, by supporting counting quantifiers [52].

Quantified graph patterns. A *quantified graph pattern* $Q(x_o)$ is defined as (V_Q, E_Q, L_Q, f), where (a) V_Q, E_Q and L_Q are the same as in patterns defined in Section 2, (b) x_o is a designated node in V_Q, referred to as the *query focus* of Q, and (c) f is a function such that for each edge $e \in E_Q$, $f(e)$ is a predicate of

- a *positive form* $\sigma(e) \odot p\%$ for a real number $p \in (0, 100]$, or $\sigma(e) \odot p$ for a positive integer p, or
- $\sigma(e) = 0$, where e is referred to as a *negated edge.*

Here $\odot$ is either $=$ or $\geq$, and $\sigma(e)$ indicates the number of matches of edge e (via subgraph isomorphism with Q; see [52] for detailed semantics of $\sigma(e)$). We refer to $f(e)$ as the *counting quantifier* of e, and $p\%$ and p as *ratio* and *numeric aggregate*, respectively.

We leave out $f(e)$ from $Q(x_o)$ if it is $\sigma(e) \geq 1$.

We extend GPARs with quantified graph patterns.

Example 8: Association rules (b) and (c) described in Example 2 are defined with quantified graph patterns. They are depicted in Fig. 3 and illustrated as follows.

For (b), the GPAR is $R_2(x, y)$: $Q_6(x, y) \Rightarrow \mathsf{buy}(x, y)$. Its antecedent is a quantified pattern Q_6 (excluding the dotted edge) and its consequent is $\mathsf{buy}(x, y)$. Its query focus is x, indicating potential customers. Observe that edge $\mathsf{follow}(x, x')$ carries a *counting quantifier* "$\geq 80\%$". In a social graph G, a node v_x matches x if (i) there exists an isomorphism h from Q_6 to a subgraph G' of G such that $h(x) = v_x$, *i.e.,* G' satisfies the topological constraints of Q_5, and (ii) among all the people whom v_x follows, at least 80% of them account for matches of x' in $Q_6(G)$, satisfying the counting quantifier.

For (c), the GPAR is $R_3(x, y)$: $Q_7(x, y) \Rightarrow \mathsf{buy}(x, y)$, where the antecedent is again a quantified pattern Q_7 (excluding the dotted edge), and its query focus is x. Note that Q_7 carries both a *universal* quantification (= 100%) and a *negation* (= 0). More specifically, a node

v_x in G matches x in Q_7 only if (i) for all people x' followed by x, x' buys a Nova Plus, *i.e.,* counting quantifier "=100%" enforces a universal quantification; and (ii) there exists *no* node v_w in G such that $\mathsf{follow}(v_x, v_w)$ is an edge in G and there exists an edge from v_w to Nova Plus labeled "bad rating"; that is, counting quantifier "$= 0$" on edge $\mathsf{follow}(x_o, z_2)$ enforces negation. □

As demonstrated by Example 8, counting quantifiers express first-order logic (FO) quantifiers as follows:

- *negation* when $f(e)$ is $\sigma(e) = 0$ (*e.g.,* Q_7);
- *existential quantification* if $f(e)$ is $\sigma(e) \geq 1$; and
- *universal quantifier* if $f(e)$ is $\sigma(e) = 100\%$ (Q_7).

A conventional graph pattern Q is a special case of quantified patterns when $f(e)$ is $\sigma(e) \geq 1$ for all edges e in Q, *i.e.,* it carries existential quantification only.

We call a quantified pattern Q *positive* if it contains no negated edges, and *negative* otherwise. For example, in the quantified patterns shown in Fig. 3, Q_5 and Q_6 are positive, while Q_7 is negative.

Restrictions. To strike a balance between the expressive power and complexity, we assume a predefined *constant* l such that on any simple path (*i.e.,* a path that contains no cycle) in $Q(x_o)$, (a) there exist at most l quantifiers that are not existential, and (b) there exist no more than one negated edge, *i.e.,* we exclude "double negation" from quantified patterns.

The reason for imposing the restriction is twofold. (1) Without the restriction, quantified patterns can express first-order logic (FO) on graphs. Such patterns inherit the complexity of FO, in addition to #P complication. Then even the problem for deciding whether there exists a graph that matches such a pattern is beyond reach in practice. As will be seen shortly, the restriction makes discovery and applications of quantified patterns feasible in large-scale graphs. (2) Moreover, we find that quantified patterns with the restriction suffice to express graph patterns commonly needed in real-life applications, with *small* l. Indeed, empirical study suggests that l is at most 2, and "double negation" is rare, since "99% of real-world queries are star-like" [57].

One can extend $f(e)$ in $Q(x_o)$ to support other built-in predicates $>$, $\neq$ and $\leq$ as $\odot$, and conjunctions of predicates. To simplify the discussion, we focus on the simple form of quantified patterns given above.

Quantified pattern matching. We revise the statement of the graph pattern matching problem given in Section 2 for quantified patterns as follows.

- *Input*: A quantified pattern $Q(x_o)$ and a graph G.
- *Output*: The set $Q(x_o, G)$ of $h(x_o)$ for all h in $Q(G)$, *i.e.,* all matches of query focus x_o of Q in G.

Its decision problem, referred to as *the quantified matching problem*, is stated as follows.

- *Input*: A quantified graph pattern $Q(x_o)$, a graph G and a node v in G.
- *Question*: Is $v \in Q(x_o, G)$?

When $Q(x_o)$ is a conventional graph pattern, the problem is NP-complete. When it comes to quantified patterns, however, ratio aggregates $\sigma \odot p\%$ and negation $\sigma = 0$ increase the expressive power, and make the analysis more intriguing. It has been shown [52] that the increased expressive power does come with a price; however, the complexity bound of the quantified matching problem does not get much higher. More specifically, the quantified matching problem is

- DP-complete for general quantified patterns, and
- NP-complete for positive quantified patterns.

Here DP is a complexity class above NP (unless P = NP), denoting the class of languages recognized by oracle machines that make a call to an NP oracle and a call to a coNP oracle. That is, a language L is in DP if there exist languages $L_1 \in$ NP and $L_2 \in$ coNP such that $L = L_1 \cap L_2$ (see [88] for details about DP).

Relate work. Over relational data, quantified association rules [97] and ratio rules [76] impose value ranges or ratios (*e.g.,* the aggregated ratio of two attribute values) as constraints on *attribute values*. Similarly, mining quantitative correlated pattern [75] has been studied, with value ranges imposed on correlated attribute values, rather than on matches. GPARs with quantified patterns extend quantified and ratio association rules from relations to graph-structured data.

The need for counting in graph queries has long been recognized. To this end, SPARQLog [78] extends SPARQL with FO rules, including existential and universal quantification over node variables. Rules for social recommendation are studied in [79], using support count as constraints. QGRAPH [13] annotates nodes and edges with a counting range (count 0 as negated edge) to specify the number of matches that must exist in a database. Set regular path queries (SRPQ) [80] extends regular path queries with quantification for group selection, to restrict the nodes in one set connected to the nodes of another. For social networks, SocialScope [8] and SNQL [85] define algebraic languages with numeric aggregates on node and edge sets.

We define quantified patterns to strike a balance between their expressive power and complexity. It differs from the prior work in the following. (1) Using a uniform form of counting quantifiers, quantified patterns support numeric and ratio aggregates (*e.g.,* at least p friends and 80% of friends), and universal (100%) and existential quantification (≥ 1). In contrast, previous proposals do not allow at least one of these. (2) We focus on graph pattern queries, beyond set regular expressions [80] and rules of [79]. (3) We show that quantified matching is DP-complete at worst, slightly higher than conventional matching (NP-complete) in the polynomial hierarchy [88]. In contrast, SPARQL and SPARQLog are PSPACE-hard [78], and SRPQ takes EXPTIME [80]; while the complexity bounds for QGRAPH [13], SocialScope [8] and SNQL [85] are unknown, they are either more expensive than quantified patterns (*e.g.,* QGRAPH is a fragment of FO(count)), or cannot express numeric and ratio quantifiers [8, 85].

5.3 Discovering and Applying GPARs

To make practical use of GPARs, we next consider two problems, namely, GPAR discovery and application of GPARs for identifying potential customers. Below we focus on GPARs studied in [51] (Section 5.1) in the absence of counting quantifiers, unless stated otherwise.

Discovering GPARs. To discover nontrivial and interesting GPARs, we first present their topological support and confidence, which are a departure from their conventional counterparts over relations.

Support. The support of a pattern Q in a graph G, denoted by $\mathsf{supp}(Q, G)$, indicates how often Q is applicable. As for association rules over itemsets, the support measure should be *anti-monotonic*, *i.e.,* for patterns Q and Q', if $Q' \sqsubseteq Q$ (in terms of containment), then in any graph G, $\mathsf{supp}(Q', G) \geq \mathsf{supp}(Q, G)$.

One may want to define $\mathsf{supp}(Q, G)$ as *the number* $|Q(G)|$ of matches of Q in $Q(G)$, following its counterpart for itemsets [106]. However, as observed in [16, 33, 72], this conventional notion is *not* anti-monotonic. For example, consider pattern Q' with a single node labeled person, and Q with a single edge child(person, person). When posed on a real-life graph G, one may find that $\mathsf{supp}(Q', G) < \mathsf{supp}(Q, G)$ although $Q' \sqsubseteq Q$, as a person may have multiple children.

We define *the support* of pattern $Q(x_o)$ in G as $\mathsf{supp}(Q, G) = |Q(x_o, G)|$, *i.e.,* the number of distinct matches of the designated node x_o in $Q(G)$. One can verify that this support measure is *anti-monotonic.*

For GPAR $R(x, y)$: $Q(x, y) \Rightarrow q(x, y)$, we define $\mathsf{supp}(R, G) = Q(x, G) \cap q(x, G)$, using the designated node x in $Q(x, y)$, by treating R as a graph pattern.

Confidence. To find how likely $q(x, y)$ holds when x and y satisfy the constraints of $Q(x, y)$, we study the *confidence* of $R(x, y)$ in graph G, denoted as $\mathsf{conf}(R, G)$. We follow the local close world assumption (LCWA) [30], assuming that G is locally complete, *i.e.,* either G includes the complete neighbors of a node for any edge type, or it has no information about these neighbors.

We define $\mathsf{conf}(R, G) = \frac{|R(x,G)|}{|Q(x,G) \cap X_o|}$, where X_o is the set of candidates of x that are associated with an edge labeled q. Intuitively, X_o retains "true" negative examples under LCWA, *i.e.,* those that have required q relationship of x but are not a match of x.

These support and confidence measures apply to GPARs with or without counting (see [51, 52]).

The diversified mining problem. We want to find GPARs for a particular event $q(x, y)$. However, this often generates an excessive number of rules, which often pertain to the same or similar people [7, 103]. This suggests that we study a diversified mining problem, to discover GPARs that are both interesting and diverse.

To formalize the problem, we first define a function $\mathsf{diff}(,)$ to measure the difference of GPARs. Given two GPARs R_1 and R_2, $\mathsf{diff}(R_1, R_2)$ is defined as

$$\mathsf{diff}(R_1, R_2) = 1 - \frac{|R_1(x, G) \cap R_2(x, G)|}{|R_1(x, G) \cup R_2(x, G)|}.$$

It measures the difference between GPARs in terms of the Jaccard distance of their match sets, by treating R_1 and R_2 as graph patterns. Such diversification has been adopted by recommender systems to avoid over-concentration and reduce too "homogeneous" items [7].

Given a set L_k of k GPARs that pertain to the same predicate $q(x, y)$, where k is a given natural number, we define the objective function $F(L_k)$ by following the practice of recommender systems [29, 60]:

$$(1 - \lambda) \sum_{R_i \in S} \frac{\mathsf{conf}(R_i)}{N} + \frac{2\lambda}{k-1} \sum_{R_i, R_i \in S, i<j} \mathsf{diff}(R_i, R_j).$$

This is known as *max-sum diversification*, and aims to strike a balance between the interestingness and diversity of the rules with a parameter λ controlled by users.

Based on the objective function, the *diversified* GPAR *mining problem* is stated as follows.

- *Input*: A graph G, a predicate $q(x, y)$, a support bound σ and positive integers k and d.
- *Output*: A set L_k of k GPARs pertaining to $q(x, y)$ such that (a) $F(L_k)$ is maximized; and (b) for each GPAR $R \in L_k$, $\mathsf{supp}(R, G) \geq \sigma$ and $r(P_R, x) \leq d$.

Here $r(P_R, x)$ denotes *the radius* of P_R (*i.e.,* R) at x, *i.e.,* the longest distance from designated node x to all nodes in P_R when P_R is treated as an undirected graph.

This is a bi-criteria optimization problem. It aims to discover GPARs for a particular event $q(x, y)$ with high support, bounded radius, and a balanced confidence and diversity. In practice, users can freely specify $q(x, y)$ of interests. Proper parameters (*e.g.,* support, confidence, diversity) can be estimated from query logs or be recommended by domain experts.

The problem is nontrivial. It is not surprising that its decision problem is intractable, since max-sum diversification is intractable itself [60]. Nonetheless, a parallel algorithm is developed in [51] that is able to find a set L_k of top-k diversified GPARs such that L_k has approximation ratio 2, and moreover, it is parallel scalable with the increase of processors under practical conditions. That is, while the problem is intractable, it is feasible to find useful GPARs in real-life graphs by leveraging parallel computing, provided that we can employ more processors when the graphs grow big.

It remains open whether there exist parallel scalable algorithms for discovering diversified top-k GPARs defined with quantified patterns (Section 5.2).

Identifying potential customers. We want to use GPARs to identify entities of interests that match certain behavior patterns specified by (quantified) patterns. We formalize this problem as follows [51, 52].

Consider a set Σ of GPARs that pertain to the same predicate $q(x, y)$, *i.e.,* their consequents are the same event $q(x, y)$. We define *the set of entities* identified by Σ in a (social) graph G with confidence η as follows:

$$\Sigma(x, G, \eta) = \{v_x \mid v_x \in Q(x, G),\ Q(x, y) \Rightarrow q(x, y) \in \Sigma, \\ \mathsf{conf}(R, G) \geq \eta\}.$$

We study the *entity identification problem*:

- *Input*: A set Σ of GPARs pertaining to the same $q(x, y)$, a confidence bound $\eta > 0$, and a graph G.
- *Output*: The set $\Sigma(x, G, \eta)$ of entities.

Intuitively, it can be used to find potential customers x of y in a social graph G that are identified by at least one GPAR in Σ, with confidence of at least η.

The problem is also nontrivial. Its decision problem is to determine, given Σ, G and η, whether $\Sigma(x, G, \eta) \neq \emptyset$. It is NP-hard for GPARs without counting quantifiers [51], and DP-hard when counting quantifiers are present [52]. Nonetheless, parallel algorithms are already in place for entity identification, which are parallel scalable under practical conditions, no matter whether the GPARs carry counting quantifiers or not. That is, these algorithm guarantee reduction in parallel running time when more processors are employed. In other words, it is feasible to identify potential customers in real-life social graphs by employing GPARs.

6 Conclusion

We have reported an account of our recent work in connection with big graph analyses. The area of big graphs is, however, a rich source of questions and vitality. Much more work needs to be done, and many questions remain to be answered. Below we list some of the topics for future work, which deserve a full treatment.

Querying big graphs. We start with two questions associated with RESOURCE. We then address a general question about the effectiveness of parallel computing.

(1) Discovering access schema. As we have seen in Section 3.1, bounded evaluation allows us to answer a large number of real-life queries by accessing a bounded amount of data no matter how big graphs grow. The key idea is to decide the bounded evaluability of an input query Q by reasoning about access schema $\mathcal{A}$, and to access only the part of data needed for answering Q by employing the indices in $\mathcal{A}$. Now the question is how we can discover "effective" access schema $\mathcal{A}$ from real-life graphs for answering queries in a given application?

The discovery problem is a bi-criteria optimization problem. On one hand, we want to find an access schema $\mathcal{A}$ such that $\mathcal{A}$ "covers" as many queries of the applications as possible. On the other hand, we want to reduce the cost of $\mathcal{A}$ and make the indices in $\mathcal{A}$ as small as possible. It is to strike a balance between the effectiveness of $\mathcal{A}$ and its cost. It also depends on whether the query load is known in advance or not.

(2) Accuracy guarantee. As remarked in Section 3.2, to answer queries that are not boundedly evaluable under $\mathcal{A}$, RESOURCE employs data-driven approximation. This gives rise to another question. For a class $\mathcal{Q}$ of graph queries and a resource ratio α, do there exist a data-driven approximation algorithm $\mathcal{T}$ and an accuracy bound η, such that given any query $Q \in \mathcal{Q}$ and graph Q, the approximate answers $Q(G_Q)$ computed by $\mathcal{T}$ under α are guaranteed to have accuracy at least η? That is, up to η, (a) each approximate answer in $Q(G_Q)$ is close enough to an exact answer to $Q(G)$, *i.e.,* it is a sensible answer in users' interest; and conversely, (b) for each exact answer in $Q(G)$, there exists an approximate answer in $Q(G_Q)$ that is close enough, *i.e.,* $Q(G_Q)$ "covers" all exact answers in $Q(G)$. One naturally wants to find an approximation scheme $\mathcal{T}$ that maximizes accuracy ratio η subject to the resource budget given by α.

(3) Parallel scalability. As remarked in Section 1, not all parallel algorithms have the property that the more processors (resources) are used, the faster their computations get. Worse still, there are graph query classes for which there exist no parallel algorithm that has this

property. A natural question is then how to characterize the effectiveness of parallel algorithms? In other words, we want to assess a parallel algorithm by evaluating its scalability with the increase of resources used.

Several models have been proposed for this purpose, *e.g.,* [41, 53, 73, 89, 99]. However, the study of this issue is still in its infancy. A characterization remains to be developed for general shared-nothing systems beyond MapReduce, to be widely accepted in practice.

Cleaning big graphs. Querying big graphs is hard, and cleaning big graphs is even harder.

(1) Discovering GFDs. To use GFDs to detect inconsistencies in real-life graphs, effective algorithms have to be in place to discover nontrivial and interesting GFDs from real-life graphs. GFD discovery is much harder than discovery of relational FDs (*e.g.,* [70]) and CFDs (*e.g.,* [39]), since GFDs are a combination of topological constraints and attribute dependencies. Among other things, the validation analysis of GFDs discovered is NP-complete, compared to low PTIME for its FD and CFD counterparts (Section 4.3). It is also more challenging than graph pattern mining since it has to deal with disconnected patterns (see φ_1 of Example 5) and forbidding GFDs that do not expect to find matches in any consistent graphs (*e.g.,* φ_3), not to mention their intractable satisfiability and implication analyses.

(2) Repairing graph-structured data. After we detect errors in a graph, we need effective methods to fix the errors, known as data repairing [9]. Repairing big data is much harder than error detection, and introduces a variety of challenges (see [34] for a survey). Even when only relational FDs are involved, the data complexity of the data repairing problem is already intractable [14], *i.e.,* the problem is NP-hard even when we only use fixed FDs. It is even more challenging when certain fixes have to be computed [44], *i.e.,* fixes that are guaranteed 100% correct and accurate, to repair "critical data" such as a knowledge base for medical data.

Big graph mining. As we have seen in Section 5, GPARs are catching up in practice when social media marketing is predicted to trump traditional marketing. However, an immediate topic is to develop effective algorithms for discovering GPARs with quantified patterns (counting quantifiers). As remarked earlier, quantified pattern matching is DP-complete for patterns with possibly negated edges, and real-life graphs are often big. It is not yet known whether parallel scalability is within reach for discovering general GPARs, although the problem has been settled in positive for GPARs without counting quantifiers [51].

Another question concerns how to determine parameters in the diversified GPAR mining problem, namely, support bound σ and radius bound d (Section 5.3). To make practical use of GPARs in social media marketing, we need to identify the "right" thresholds that yield interesting GPARs. Similarly, we need to determine the "right" threshold for confidence bound η in the entity identification problem for real-life applications.

Acknowledgements The results presented in this paper are taken from joint work with Yang Cao, Yinghui Wu, Xin Wang and Jingbo Xu. Fan is supported in part by ERC 652976, 973 Program 2014CB340302, NSFC 61133002 and 61421003, EPSRC EP/J015377/1 and EP/M025268/1, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, the Foundation for Innovative Research Groups of NSFC, Beijing Advanced Innovation Center for Big Data and Brain Computing, Shenzhen Science and Technology Fund JCYJ20150529164656096, Guangdong Applied R&D Program 2015B010131006, and NSF III 1302212.

References

1. Nielsen global online consumer survey. *http://www.nielsen.com/content/dam/corporate/us/en/newswire/uploads/2009/07/pr_global-study_07709.pdf*
2. Trinity. *http://research.microsoft.com/en-us/projects/trinity/*
3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
4. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: BlinkDB: queries with bounded errors and bounded response times on very large data. In: EuroSys, pp. 29–42 (2013)
5. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Record **22**(2), 207–216 (1993)
6. Akhtar, W., Cortés-Calabuig, A., Paredaens, J.: Constraints in RDF. In: SDKB, pp. 23–39 (2010)
7. Amer-Yahia, S., Lakshmanan, L.V., Vassilvitskii, S., Yu, C.: Battling predictability and overconcentration in recommender systems. IEEE Data Eng. Bull. **32**(4) (2009)
8. Amer-Yahia, S., Lakshmanan, L.V.S., Yu, C.: Socialscope: Enabling information discovery on social content sites. In: CIDR (2009)
9. Arenas, M., Bertossi, L., Chomicki, J.: Consistent query answers in inconsistent databases. In: PODS, pp. 68–79 (1999)
10. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: SIGMOD (2003)
11. Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.): Machine Learning and Knowledge Discovery in Databases, Proceedings, Part I, *LNCS*, vol. 6321. Springer (2010)
12. Bapna, R., Umyarov, A.: Do your online friends make you pay? A randomized field experiment in an online music social network. NBER working paper (2012)
13. Blau, H., Immerman, N., Jensen, D.: A visual language for querying and updating graphs. University of Massachusetts Amherst Computer Science Technical Report **37**, 2002 (2002)

14. Bohannon, P., Fan, W., Flaster, M., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: SIGMOD, pp. 143–154 (2005)
15. Boldi, P., Vigna, S.: The Webgraph framework I: compression techniques. In: WWW (2004)
16. Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: PAKDD (2008)
17. Bröcheler, M., Pugliese, A., Subrahmanian, V.S.: A budget-based algorithm for efficient subgraph matching on huge networks. In: ICDE Workshops (2011)
18. Calvanese, D., Fischl, W., Pichler, R., Sallinger, E., Šimkus, M.: Capturing relational schemas and functional dependencies in RDFS. In: AAAI (2014)
19. Cao, Y., Fan, W.: An effective syntax for bounded relational queries. In: SIGMOD (2016)
20. Cao, Y., Fan, W., Geerts, F., Lu, P.: Bounded query rewriting using views. In: PODS (2016)
21. Cao, Y., Fan, W., Huai, J., Huang, R.: Making pattern queries bounded in big graphs. In: ICDE (2015)
22. Cao, Y., Fan, W., Tian, C., Wang, Y.: Effective techniques for CDR queries: Technical report for Huawei Technologies (2015)
23. Cao, Y., Fan, W., Yu, W.: Bounded conjunctive queries. PVLDB pp. 1231 – 1242 (2014)
24. Catania, B., Guerrini, G.: Approximate queries with adaptive processing. In: Advanced Query Processing. Springer (2013)
25. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SICOMP **32**(5), 1338–1355 (2003)
26. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub) graph isomorphism algorithm for matching large graphs. TPAMI **26**(10), 1367–1372 (2004)
27. Cormode, G., Garofalakis, M., Haas, P.J., Jermaine, C.: Synopses for massive data: Samples, histograms, wavelets, sketches. FTDB **4**(1–3) (2012)
28. Cortés-Calabuig, A., Paredaens, J.: Semantics of constraints in RDFS. In: AMW, pp. 75–90 (2012)
29. Deng, T., Fan, W.: On the complexity of query result diversification. ACM Trans. Database Syst. **39**(2), 15 (2014)
30. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: KDD (2014)
31. Dylla, M., Sozio, M., Theobald, M.: Resolving temporal conflicts in inconsistent RDF knowledge bases. In: BTW (2011)
32. Eckerson, W.W.: Data quality and the bottom line: Achieving business success through a commitment to high quality data. Tech. rep., The Data Warehousing Institute (2002)
33. Elseidy, M., Abdelhamid, E., Skiadopoulos, S., Kalnis, P.: GRAMI: frequent subgraph and pattern mining in a single large graph. PVLDB **7**(7), 517–528 (2014)
34. Fan, W.: Data quality: From theory to practice. SIGMOD Record **44**(3), 7–18 (2015)
35. Fan, W., Fan, Z., Tian, C., Dong, X.L.: Keys for graphs. PVLDB **8**(12), 1590–1601 (2015)
36. Fan, W., Geerts, F.: Foundations of Data Quality Management. Morgan & Claypool Publishers (2012)
37. Fan, W., Geerts, F., Cao, Y., Deng, T., Lu, P.: Querying big data by accessing small data. In: PODS (2015)
38. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. ACM Trans. on Database Systems **33**(1) (2008)
39. Fan, W., Geerts, F., Li, J., Xiong, M.: Discovering conditional functional dependencies. IEEE Trans. on Data and Knowledge Engineering **23**(5), 683–698 (2011)
40. Fan, W., Geerts, F., Libkin, L.: On scale independence for querying big data. In: PODS, pp. 51–62 (2014)
41. Fan, W., Geerts, F., Neven, F.: Making queries tractable on big data with preprocessing. PVLDB **6**(8), 577–588 (2013)
42. Fan, W., Huai, J.: Querying big data: Bridging theory and practice. J. Comput. Sci. Technol. **29**(5), 849–869 (2014)
43. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.: Graph pattern matching: From intractability to polynomial time. PVLDB **3**(1), 1161–1172 (2010)
44. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. VLDB J. **21**(2), 213–238 (2012)
45. Fan, W., Li, J., Wang, X., Wu, Y.: Query preserving graph compression. In: SIGMOD, pp. 157–168 (2012)
46. Fan, W., Wang, X., Wu, Y.: Performance guarantees for distributed reachability queries. PVLDB **5**(11), 1304–1315 (2012)
47. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching. PVLDB **6**(13), 1510–1521 (2013)
48. Fan, W., Wang, X., Wu, Y.: Distributed graph simulation: Impossibility and possibility. PVLDB (2014)
49. Fan, W., Wang, X., Wu, Y.: Querying big graphs within bounded resources. In: SIGMOD, pp. 301–312 (2014)
50. Fan, W., Wang, X., Wu, Y.: Answering pattern queries using views. IEEE Trans. Knowl. Data Eng. **28**(2), 326–341 (2016)
51. Fan, W., Wang, X., Wu, Y., Xu, J.: Association rules with graph patterns. PVLDB **8**(12), 1502–1513 (2015)
52. Fan, W., Wu, Y., Xu, J.: Adding counting quantifiers to graph patterns. In: SIGMOD (2016)
53. Fan, W., Wu, Y., Xu, J.: Functional dependencies for graphs. In: SIGMOD (2016)
54. Fan, W., Xu, J., Wu, Y., Jiang, J., Cao, Y., Tian, C., Yu, W., Zhang, B., Zheng, Z.: Parallelizing sequential graph computations. In: SIGMOD (2017)
55. Faris, R.W., Ennett, S.T.: Adolescent aggression: The role of peer group status motives, peer aggression, and group characteristics. Social Networks **34**(4) (2013)
56. Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.: AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In: WWW (2013)
57. Gallego, M.A., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world SPARQL queries. In: USEWOD workshop (2011)
58. Gartner: 'Dirty data' is a business problem, not an IT problem (2007). *http://www.gartner.com/newsroom/id/501733*
59. Giraph: *http://giraph.apache.org/*
60. Gollapudi, S., Sharma, A.: An axiomatic approach for result diversification. In: WWW, pp. 381–390 (2009)
61. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: GraphX: Graph processing in a distributed dataflow framework. In: OSDI (2014)
62. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press (1995)
63. Grujic, I., Bogdanovic-Dinic, S., Stoimenov, L.: Collecting and analyzing data from e-government Facebook pages. In: ICT Innovations (2014)

64. Gubichev, A., Bedathur, S.J., Seufert, S., Weikum, G.: Fast and accurate estimation of shortest paths in large graphs. In: CIKM (2010)
65. Haenni, R., Lehmann, N.: Resource bounded and anytime approximation of belief function computations. IJAR **31** (2002)
66. He, B., Zou, L., Zhao, D.: Using conditional functional dependency to discover abnormal data in RDF graphs. In: SWIM, pp. 1–7 (2014)
67. Hellings, J., Gyssens, M., Paredaens, J., Wu, Y.: Implication and axiomatization of functional constraints on patterns with an application to the RDF data model. In: FoIKS (2014)
68. Henzinger, M.R., Henzinger, T., Kopke, P.: Computing simulations on finite and infinite graphs. In: FOCS (1995)
69. Huang, S., Li, Q., Hitzler, P.: Reasoning with inconsistencies in hybrid MKNF knowledge bases. Logic Journal of IGPL (2012)
70. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: An efficient algorithm for discovering functional and approximate dependencies. COMP. J. **42**(2), 100–111 (1999)
71. IMDb: *http://www.imdb.com/stats/search/*
72. Jiang, C., Coenen, F., Zito, M.: A survey of frequent subgraph mining algorithms. Knowledge Eng. Review **28**(01), 75–105 (2013)
73. Karloff, H.J., Suri, S., Vassilvitskii, S.: A model of computation for MapReduce. In: SODA, pp. 938–948 (2010)
74. Karp, R.M., Ramachandran, V.: A survey of parallel algorithms for shared-memory machines. Tech. Rep. UCB/CSD-88-408, EECS Department, University of California, Berkeley (1988). URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/1988/5865.html`
75. Ke, Y., Cheng, J., Ng, W.: Mining quantitative correlated patterns using an information-theoretic approach. In: SIGKDD (2006)
76. Korn, F., Labrinidis, A., Kotidis, Y., Faloutsos, C.: Ratio rules: A new paradigm for fast, quantifiable data mining. In: VLDB (1998)
77. Lausen, G., Meier, M., Schmidt, M.: SPARQLing constraints for RDF. In: EDBT, pp. 499–509. ACM (2008)
78. Ley, C., Linse, B., Poppe, O.: SPARQLog: SPARQL with rules and quantification. Semantic Web Information Management: A Model-Based Perspective p. 341 (2010)
79. Lin, W., Alvarez, S.A., Ruiz, C.: Collaborative recommendation via adaptive association rule mining. Data Mining and Knowledge Discovery **6**, 83–105 (2000)
80. Liptchinsky, V., Satzger, B., Zabolotnyi, R., Dustdar, S.: Expressive languages for selecting groups from graph-structured data. In: WWW (2013)
81. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., Hellerstein, J.M.: Distributed GraphLab: A framework for machine learning in the cloud. PVLDB **5**(8) (2012)
82. Lü, L., Zhou, T.: Link prediction in complex networks: A survey. Physica A: Statistical Mechanics and its Applications **390**(6), 1150–1170 (2011)
83. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Strong simulation: Capturing topology in graph pattern matching. ACM Trans. on Database Systems **39**(1) (2014)
84. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: SIGMOD (2010)
85. Martın, M., Gutierrez, C., Wood, P.: SNQL: A social networks query and transformation language. In: AMW (2011)
86. Navlakha, S., Rastogi, R., Shrivastava, N.: Graph summarization with bounded error. In: SIGMOD (2008)
87. Ogaard, K., Roy, H., Kase, S., Nagi, R., Sambhoos, K., Sudit, M.: Discovering patterns in social networks with graph matching algorithms. In: SBP (2013)
88. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
89. Qin, L., Yu, J.X., Chang, L., Cheng, H., Zhang, C., Lin, X.: Scalable big graph processing in MapReduce. In: SIGMOD (2014)
90. Qun, C., Lim, A., Ong, K.W.: D(k)-index: An adaptive structural summary for graph-structured data. In: SIGMOD (2003)
91. Ramakrishnan, R., Gehrke, J.: Database Management Systems. McGraw-Hill Higher Education (2000)
92. Romero, C., Ventura, S., De Bra, P.: Knowledge discovery with genetic programming for providing feedback to courseware authors. UMUAI **14**(5), 425–464 (2004)
93. Rula, A., Zaveri, A.: Methodology for assessment of linked data quality. In: LDQ@SEMANTiCS (2014)
94. Schmitz, C., Hotho, A., Jäschke, R., Stumme, G.: Mining association rules in folksonomies. In: Data Science and Classification, pp. 261–270 (2006)
95. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. PVLDB (2008)
96. Smith, C.: Twitter users say they use the site to influence their shopping decisions. Business Insider Intelligence (2013)
97. Srikant, R., Agrawal, R.: Mining quantitative association rules in large relational tables. In: SIGMOD (1996)
98. Stern, R.T., Puzis, R., Felner, A.: Potential search: A bounded-cost search algorithm. In: ICAPS (2011)
99. Tao, Y., Lin, W., Xiao, X.: Minimal MapReduce algorithms. In: SIGMOD (2013)
100. Thayer, J.T., Stern, R., Felner, A., Ruml, W.: Faster bounded-cost search using inadmissible estimates. In: ICAPS (2012)
101. Tian, Y., Balmin, A., Corsten, S.A., Shirish Tatikonda, J.M.: From "think like a vertex" to "think like a graph". PVLDB **7**(7), 193–204 (2013)
102. Wikibon: A comprehensive list of big data statistics (2012). *http://wikibon.org/blog/big-data-statistics/*
103. Xin, D., Cheng, H., Yan, X., Han, J.: Extracting redundancy-aware top-k patterns. In: KDD (2006)
104. Yan, D., Cheng, J., Lu, Y., Ng, W.: Blogel: A block-centric framework for distributed computation on real-world graphs. PVLDB **7**(14), 1981–1992 (2014)
105. Yu, Y., Heflin, J.: Extending functional dependency to detect abnormal data in RDF graphs. In: ISWC (2011)
106. Zhang, C., Zhang, S.: Association rule mining: models and algorithms. Springer (2002)
107. Zilberstein, S.: Using anytime algorithms in intelligent systems. AI magazine **17**(3) (1996)
108. Zilberstein, S., Charpillet, F., Chassaing, P., et al.: Real-time problem-solving with contract algorithms. In: IJCAI, pp. 1008–1015 (1999)