2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00235

Consistent Subgraph Matching over Large Graphs

Ye Yuan[♯] Delong Ma[†] Aoqian Zhang[♯] Guoren Wang[♯]
[♯]*Beijing Institute of Technology* [†]*Northeastern University, China*
{yuan-ye, aoqian.zhang, wanggr}@bit.edu.cn, madelong@stumail.neu.edu.cn

***Abstract*—Subgraph matching over graphs has been extensively studied, due to its wide applications in knowledge bases, social networks, and among others. To catch the inconsistency and errors that commonly exist in these graphs, this paper studies consistent subgraph matching (CSM), i.e., finding the common matches in every consistent graph repair w.r.t a set of *conditional graph dependencies* (CGDs). We concentrate on subset, superset and symmetric difference graph repairs. We study fundamental problems for CGDs and CSM. We show that the satisfiability, implication, and validation problems of CGDs are coNP-complete, coNP-complete and NP-complete, respectively. We also show that the CSM problem (under any kind of repair) is NP-complete. We provide (parallel) algorithms to solve CSM, and guarantee to reduce running time when given more processors. Using real-life and synthetic graphs, we empirically verify the efficiency and effectiveness of our algorithms.**

I. INTRODUCTION

Given a query graph Q and a data graph G, subgraph matching is defined as finding all subgraph instances of G that are isomorphic to Q. Subgraph matching has been widely used in network traffic analysis, social networks and knowledge discovery, among others. These applications often contain inconsistent data, due to interoperability and distribution; e.g., in RDF and social/scientific networks [1]–[4]. As an example, inconsistency might arise while integrating several sources into a single RDF graph, or while performing statistical inference on a scientific or social network. Subgraph matching over these inconsistent graphs results in undesirable answers. A solution to this problem is to advocate the existing graph dependencies (e.g., [5]–[10]) which can detect inconsistencies and repair them in graphs. Subgraph matching is then performed over the repaired graphs. However, the following example shows that this solution may fail.

Example 1. *The knowledge base usually stores the entities and their relationships in a specific domain. Figure 1 shows a partial knowledge graph G representing the relationships of various entities in a college. There are some basic rules to follow in this knowledge base. For example, a graduate teacher can guide up to two students and at least one every year*[1]*. We call this rule* edge constraint*. Based on the edge constraint, the teacher t_1 in Figure 1 guides three students s_1, s_2 and s_3, exceeding the allowed maximum number (two). Such errors in G are called* edge errors*. Existing graph dependencies (e.g., [5]–[10]) only catch attributes' or numeric inconsistencies of nodes (*node errors*) and fail to catch the edge errors. If a query Q is issued over G, we may obtain incorrect query answers.*

To make G obey the edge constraint, we can delete edges with label "guide" from t_1, and there can be six repairs: deleting from G one or two of the three edges (t_1, s_1), (t_1, s_2) and (t_1, s_3). Thus, we can get a total of six repaired graphs. Shall we issue Q over the six repaired graphs to obtain the answers? Existing works of subgraph matching cannot solve this problem.

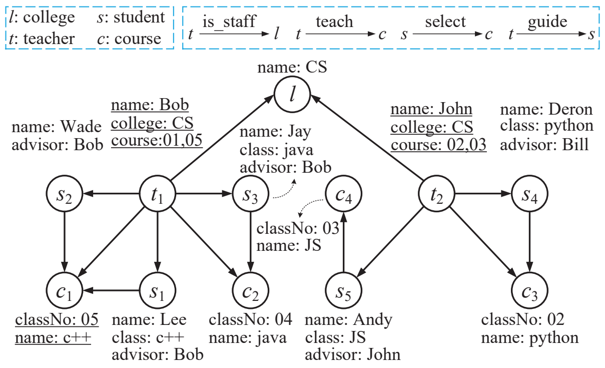


Fig. 1. Graph G representing a knowledge base of a college.

The example raises several questions. How should we extend graph dependencies to catch edge errors? Does the extension make it harder to reason about the dependencies? If the edge constraint is imposed on graphs, how to define subgraph matching over a large number of repaired graphs? Is the new problem for subgraph matching harder than the traditional one? Putting these together, above all, can we develop practical and efficient algorithms to process subgraph matching over such large inconsistent graphs?

Contributions & organization. This paper makes an effort to answer above questions, from foundation to practice.

(1) CGDs. We propose a class of conditional graph dependencies, referred to as CGDs (Section II). CGDs are a combination of (a) a pattern P to identify nodes by subgraph isomorphism, (b) a node constraint V_c, and (c) an edge constraint E_c conditional on V_c. They extend graph functional dependencies (GFDs [5], [7]) by supporting the edge constraint E_c. The edge constraint is conditional upon the related node constraints. We show that CGDs are able to catch edge inconsistencies commonly found in real-life graphs. Moreover, they subsume GFDs [5], [7] and graph quality rules (GQRs) [10] as special cases. Thus, they capture inconsistencies that can be detected

[1]For a clear presentation of the following examples, we assume the number constraint for guided students by a teacher.

by GFDs and GQRs, besides the edge errors that are beyond the capacity of GFDs and GQRs.

Consider a pattern P_1 in Fig. 2 and its related CGD $\phi_1 = P_1[x, y]([2,3]|\emptyset \rightarrow x.course = y.classNo)$. Here $V_c[x, y] = \emptyset \rightarrow x.course = y.classNo$ states that if a teacher x teaches a course y, then the value of attribute $x.course$ coincides with the value of attribute $y.classNo$. Also, $E_c[x, y] = [2, 3]$ means that a teacher can teach at most three courses and at least two. Consider the matches $m_1 = (t_1, c_1)$ and $m_2 = (t_1, c_2)$ of P_1 in G (Fig. 1). m_1 satisfies V_c (since $t_1.course = c_1.classNo$) while m_2 does not satisfy V_c (since $t_1.course \neq c_1.classNo$). Edges (t_1, c_1) and (t_1, c_2) cannot satisfy E_c, since (t_1, c_2) does not satisfy V_c that E_c should be conditional on.

(2) CSM. The above example raises the need for developing an inconsistency-tolerant semantics for subgraph matching in a simple yet general framework. In order to tackle this problem, we propose consistent subgraph matching (CSM) following the widespread approach of consistent query answering for relational databases [11] (Section II). CSM is based on the notion of *graph repair*, which represents a possible minimal way in which consistency over the graph could be restored. More formally, a graph repair is a graph that satisfies the constraints and "minimally differs" from the original graph. Based on CGDs, we define three kinds of graph repairs, i.e., the subset, superset and symmetric difference graph repairs. In general, a graph does not admit a unique repair. This leads to an inconsistency-intolerant semantics based on the consistent answers of a query, i.e., the answers that hold in every possible repair. The problem of computing consistent answers to subgraph matching is named as *consistent subgraph matching*.

(3) Fundamental problems. We study the following problems associated with CGDs and CSM (Section III): (a) validation to decide whether a given set Σ of CGDs has a model, i.e., a graph satisfying all the CGDs in Σ; (b) consistency to check whether the CGDs in Σ have conflicts with each other; (c) implication to determine a given set Σ of CGDs implies another CGD ϕ, i.e., $\Sigma \models \phi$; and (d) matching to find consistent answers of Q in G. We settle the complexity of these problems: coNP-complete, coNP-complete, NP-complete and NP-complete, respectively. Hence for CGDs and CSM, their fundamental problems are no harder than their counterparts for the existing graph dependencies and conventional subgraph matching.

(4) Practical algorithms. We provide an algorithmic framework that unifies conventional subgraph matching and inconsistency cleaning in a generic search process (Section IV). We also develop parallel algorithms for CSM (Section V). We identify a practical condition under which CSM is parallel scalable, i.e., guaranteeing provable reduction in sequential running time with the increase of processors. Under the condition, we develop graph partitioning algorithms, also parallel scalable, by exploring the inter-fragment parallelism.

(5) Experimental study. Using real-life and synthetic graphs, we experimentally verify the effectiveness of CSM and the scalability of our algorithms (Section VI). We find the following. (a) CSM has average above 90% query accuracy in the face of various of nodes' and edges' errors, whereas conventional subgraph matching obtains average below 25% query accuracy in the same scenarios. (b) CSM is feasible on large graphs. It takes 85 seconds on graphs of 30M nodes and 80M edges by using 12 processors. Our matching algorithm is parallel scalable: it is on average 3.2 and 3.7 times faster on real-life and synthetic graphs, respectively, when the number of processors increases from 4 to 12. (c) CGDs can capture consistent patterns in knowledge graphs that cannot be expressed with existing graph dependencies.

TABLE I
ABBREVIATION AND NOTATION.

Symbol	Description
G, $P[\overline{x}]$	the data graph, pattern
$h(\overline{x})$	the match of $P[\overline{x}]$ in G
$\phi \doteq P[\overline{x}](E_c\|V_c)$	the conditional graph dependency (CGD)
$V_c \doteq X \rightarrow Y$	the node constraint
$E_c \doteq NUM(x, l, y)$=$[min, max]$	the edge ($e = (x, l, y)$) constraint
l_v, l_e	the literals of V_c and E_c
Σ	the set of CGDs
$G \models \Sigma$	G satisfies every CGD in Σ
Q	the query pattern
$Q(G)$	the set of matches of Q in G
$\star$-Cons(G, Q, Σ)	the consistent answers of Q in G under Σ
Σ_v	a set of valid CGDs in Σ
EOD	the execution order
UP	the union of all the patterns in Σ
VP	the valid pattern of UP
Eq	the equivalence relation of fixes
ConsGraph	the algorithmic framework
ValCGD	the algorithm to compute Σ_v
ExOrd	the algorithm to compute EOD
SubMat	the algorithm to compute $\star$-Cons(G, Q, Σ)
PConsGraph	the algorithm to parallelize ConsGraph

II. PROBLEM DEFINITION

We summarize the abbreviations and notations of this paper in Table 1.

Following the works [5], [7], [8], we define graphs, patterns and pattern matching.

Assume three countably infinite sets Θ, Υ and U, denoting labels, attributes and constant values, respectively.

Graphs. We consider directed graphs $G = (V, E, L, F_A)$ with labeled nodes and edges, where (a) V is a finite set of nodes; each node $v \in V$ carries a label $L(v) \in \Theta$; (b) $E \subseteq V \times \Theta \times V$ is a finite set of edges, in which each $e = (v, l, v')$ denotes an edge from node v to v' labeled with l; we write e as (v, v') and $L(e) = l$ if it is clear in the context; (c) each node $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, ..., A_n = a_n)$ of attributes (properties), where $A_i \in \Upsilon$ and $a_i \in U$, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$. Note that our defined graphs are property graphs [12] without properties/values on edges for a clear presentation.

Patterns. A graph pattern is a graph $P[\overline{x}] = (V_p, E_p, L_p)$, where (1) V_p (resp. E_p) is a finite set of pattern nodes (resp. edges); (2) L_p is a function that assigns a label $L_p(u)$ (resp. $L_p(e)$) to each node $u \in V_p$ (resp. edge $e \in E_p$); and (3) $\overline{x}$

denotes the nodes in V_p as a list of distinct variables. Labels $L_p(u)$ and $L_p(e)$ are drawn from the alphabet Θ. Moreover, we allow wildcard '_' as a special label in P.

Pattern matching. We say that a label l matches l', denoted by $l \asymp l'$, if either (a) both l and l' are in Θ and $l = l'$, or (b) $l' \in \Theta$ and l is '_', i.e., wildcard matches any label.

A *match* of pattern $P[\overline{x}]$ in graph G is a subgraph isomorphism h from P to G such that for each node $u \in V_p$, $L_p(u) \asymp L(h(u))$; and for each edge $e = (u, l, u')$ in P, there exists an edge $e = (h(u), l', h(u'))$ in G such that $l \asymp l'$.

We denote the match as a vector $h(\overline{x})$ if it is clear from the context, where $h(\overline{x})$ consists of $h(x)$ for all variables $x \in \overline{x}$. Intuitively, $h(\overline{x})$ is a list of entities identified by pattern P.

For example, Figures 1 and 2 show a data graph G and five patterns $\{P_1, P_2, P_3, P_4, P_5\}$. The matches of P_2 in G are (t_1, l) and (t_2, l).

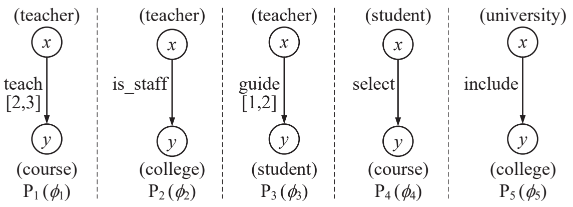


Fig. 2. Conditional graph dependencies (CGDs)

Conditional graph dependency. A conditional graph dependency (CGD) ϕ is defined as $P[\overline{x}](E_c|V_c)$, where $P[\overline{x}]$ is a graph pattern, and V_c and E_c are node and edge constraints.

(1) Node constraint V_c is $X \rightarrow Y$, where X and Y are (possibly empty) sets of literals of $\overline{x}$. For $x, y \in \overline{x}$, a literal of $\overline{x}$ is:

(a) $x.A = c$, where c is a constant, and A is an attribute in Υ; or

(b) $x.A = y.B$, where A and B are attributes in Υ.

(2) Edge constraint E_c: for each edge $e = (x, l, y)$, if the literals of $x, y \in \overline{x}$ are defined as node constraints, then $NUM(x, l, y) = [min, max]$, where min (resp. max) is a positive number representing the minimum (resp. maximum) number of allowed edges with label l.

Intuitively, ϕ combines a pattern P to identify subgraphs $G' = h(P)$ in a graph G, a node constraint V_c on nodes and a constraint E_c on edges of G'. But E_c is conditional on V_c, i.e., the edge constraint E_c depends on the constraints imposed on the nodes E_c incident to.

The following example shows five CGDs for patterns in Figure 2:

Example 2. *(1) $\phi_1 = P_1[x, y]([2, 3]|\emptyset \rightarrow x.course = y.classNo)$. It states that if a teacher x teaches a course y, then the value of attribute $x.course$ coincides with the value of attribute $y.classNo$. Also, a teacher can teach at most three courses and at least two.*

(2) $\phi_2 = P_2[x, y](\emptyset \rightarrow x.college = y.name)$. It says that if a teacher x is a staff of a college y, then the values of $x.college$ and $y.name$ must be the same.

(3) $\phi_3 = P_3[x, y]([1, 2]|\emptyset \rightarrow x.name = y.advisor)$. This CGD states that the attribute value of student y's advisor must be the same as his teacher x's name. Also, a teacher can guide up to two students and at least one.

(4) $\phi_4 = P_4[x, y](\emptyset \rightarrow x.class = y.name)$. It says that if a student x selects a course y, then the value of $x.class$ and $y.name$ must be the same.

(5) $\phi_5 = P_5[x, y](\emptyset \rightarrow x.postcode = y.postcode)$. This CGD shows that if a university x includes a college y, then the postcodes of college y and university x must be consistent. □

Semantics. Consider a conditional graph dependency $P[\overline{x}](E_c|V_c)$, a match $h(\overline{x})$ of P in graph G, and literals l_v of V_c and l_e of E_c. We say that $h(\overline{x})$ satisfies $l_e|l_v$, denoted by $h(\overline{x}) \models l_e|l_v$, if:

(1) when l_v is $x.A = c$, there exists an attribute A at node $v = h(x)$, and $v.A = c$;

(2) when l_v is $x.A = y.B$, nodes $v = h(x)$ and $v' = h(y)$ carry attributes A and B, respectively, and $v.A = v'.B$; and

(3) when l_e is $NUM(x, l, y) = [min, max]$, if nodes $h(x)$ and $h(y)$ satisfy the conditions in (1) and (2), the number of edges with label l from a node $h(x)$ to nodes $\{h(y)\}$ is within $[min, max]$.

Note that $h(x)$ is the same node involved in a set of matches $\{h(\overline{x})\}$, whereas $\{h(y)\}$ are a set of different nodes in the matches $\{h(\overline{x})\}$ and $|\{h(y)\}| = |\{h(\overline{x})\}|$.

We write $h(\overline{x}) \models X$ if $h(\overline{x})$ satisfies all literals in X. We write $h(\overline{x}) \models X \rightarrow Y$ (or $h(\overline{x}) \models V_c$) if $h(\overline{x}) \models X$ implies $h(\overline{x}) \models Y$. We write $h(\overline{x}) \models E_c|V_c$ if $h(\overline{x}) \models l_e|l_v$ for every pair (l_v, l_e) of literals of V_c and E_c.

Example 3. *Consider the CGD ϕ_3 in Example 2. If ϕ_3 is applied to G in Figure 1, we find the match (t_2, s_4) violates ϕ_3 (i.e., $h(t_2, s_4) \not\models V_c(\phi_3)$) because the value (Bill) of S_4's advisor is different from t_2's name (John). The matches (t_1, s_1), (t_1, s_2) and (t_1, s_3) all satisfy $V_c(\phi_3)$. They however violate $E_c|V_c(\phi_3)$, because that the number (three) of students guided by t_1 exceeds the allowable upper limit (two).*

Graph repair. A repair of a database D under a set of constraints Σ is a database D' that satisfies Σ but "minimally" differs from D. We formalize this idea for graphs and CGDs below, following closely its formalization in the relational context [11].

Our defined graph G consists of nodes, edges and node attributes. Therefore, a repair G' of G can be deletions and insertions of nodes, edges and node attributes. In this paper, we only consider those with edges and node attributes. For the minimal difference between G and G', we only consider the minimal difference between $E(G)$ and $E(G')$.

Based on these assumptions, we define two relations between two graphs $G = (V, E, L)$ and $G' = (V, E', L')$:

(1) We define $G \subseteq G'$ if $E \subseteq E'$ and $L(E) \subseteq L'(E')$.

(2) We define the symmetric difference ($\oplus$) between G and G' as $G \oplus G' := (V, (E \oplus E'))$, where $E \oplus E' := (E \setminus E') \cup (E' \setminus E)$. That is, edges of $G \oplus G'$ are those edges that appear in either G or G' but not in both. Notice that $G \oplus G'$ also has the node set V.

Let G and G' be two distinct graphs over Θ, and let Σ be a finite set of CGDs over Θ. Then:

(1) G' is a $\subseteq$-repair (i.e., subsect repair) of G under Σ, if (a) $G' \subseteq G$, (b) $G' \models \Sigma$, and (c) there is no graph G'' over Θ such that $G'' \models \Sigma$ and $G' \subsetneq G'' \subseteq G$.

(2) G' is a $\supseteq$-repair (i.e., superset repair) of G under Σ, if (a) $G \subseteq G'$, (b) $G' \models \Sigma$, and (c) there is no graph G'' over Θ such that $G'' \models \Sigma$ and $G \subseteq G'' \subsetneq G'$.

(3) G' is a $\oplus$-repair (i.e., symmetric difference repair) of G under Σ, if (a) $G' \models \Sigma$, and (b) there is no graph G'' over Θ such that $G'' \models \Sigma$ and $G \oplus G'' \subsetneq G \oplus G'$.

Consistent subgraph matching. We are now ready to define our most important notion, that of a *consistent answer* to a pattern matching. The consistent answers are matches that belong to the evaluation of the pattern matching over every single repair of the original graph, i.e., the result of intersection of match sets in each repair. For a graph G and a patten Q, denote by $Q(G)$ be the match set of Q in G.

Definition 1. *(Consistent answers.) Assume that $\star \in \{\subseteq, \supseteq, \oplus\}$. Let G be a graph, Σ a set of CGDs and Q a query pattern, all of them over Θ. We define the set $\star$-Cons(G, Q, Σ) of $\star$-consistent answers of Q over G under Σ as:*

$$\star\text{-Cons}(G, Q, \Sigma) = \bigcap \{Q(G') | G' \text{ is a } \star\text{-repair of } G \text{ under } \Sigma\}, \quad (1)$$

where $\star$ is one kind of $\{\subseteq, \supseteq, \oplus\}$.

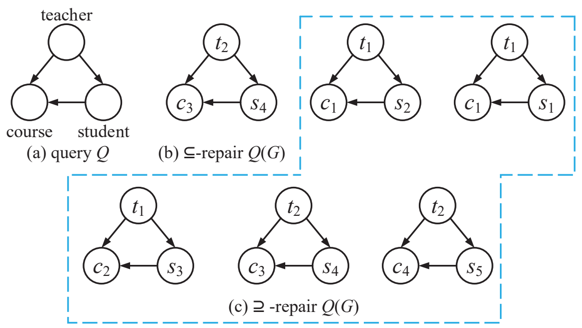


Fig. 3. Query Q and consistent answers of Q in G in Figure 1.

Example 4. *Given a query Q (Figure 3(a)), a graph G (Figure 1) and a set of CGDs (Figure 2), the consistent answers of $\subseteq$-repair and $\supseteq$-repair are shown in Figures 3(b) and 3(c), respectively. Due to the limited space, we omit the consistent answers for $\oplus$-repair.*

Taking $\subseteq$-repair as an example, we obtain its consistent answers as follows. Consider the CGD $\phi_3 = P_3[x, y]([1, 2]|\emptyset \to x.name = y.advisor)$ in Example 2. The number (3) of students guided by t_1 exceeds the allowable upper limit (2). There can be six graph $\subseteq$-repairs: deleting from G one or two of the three edges (t_1, s_1), (t_1, s_2) and (t_1, s_3). We issue Q over each graph repair.

For the repair R_1 of deleting the edge (t_1, s_1), we obtain a match set $Q(R_1) = \{(t_1, c_1, s_2), (t_1, c_2, s_3), (t_2, c_3, s_4)\}$. Similarly, we can also obtain the match sets of other five repairs. By intersecting these match sets, we obtain the consistent answer (t_2, c_3, s_4) of $\subseteq$-repair shown in Figure 3(b), i.e., only (t_2, c_3, s_4) appears in all the six match sets.

III. Fundamental Problems

A. Reasoning about Conditional Graph Dependencies

We next study three fundamental problems for CGDs. The validation analysis identifies inconsistencies under CGDs to be repaired. The satisfiability analysis helps us decide whether a repair exists under CGDs. The implication analysis reduces redundant constraints that can be already implied.

Validation. Given a set of CGDs Σ and a graph G, the *validation problem* for CGDs is to decide whether $G \models \Sigma$.

Given a CGD $\phi = P[\overline{x}](E_c|V_c)$ and a graph G, we say that a match $h(\overline{x})$ of P in G is a *violation* of ϕ if $G_h \not\models \phi$, where G_h is the subgraph induced by $h(\overline{x})$. For a set Σ of CGDs, we denote by Vio(Σ, G) the set of all violations of CGDs in G, if and only if there exists a CGD ϕ in Σ such that $h(\overline{x})$ is a violation of ϕ in G. That is, Vio(Σ, G) collects all nodes and edges of G that are inconsistent when the set Σ of CGDs is used as data quality rules.

The *validation problem* is stated as follows:

- Input: A set Σ of CGDs and a graph G.
- Output: The set Vio(Σ,G) of violations.

Its decision problem is to decide whether $G \models \Sigma$, i.e., whether Vio(Σ,G) is empty. The problem is nontrivial.

Theorem 1. Validation *of CGDs is coNP-complete.*

Proof sketch: We show that it is NP-hard to check, given G and Σ, whether $G \not\models \Sigma$, by reduction from subgraph isomorphism which is NP-complete [13]. For the upper bound, we give an algorithm that returns "yes" if $G \not\models \Sigma$: (a) guess a CGD $P[\overline{x}](E_c|V_c)$ from Σ and a mapping h from P to a subgraph of G; (b) check whether h is isomorphic; (c) if so, check whether $h(\overline{x}) \models X$ but $h(\overline{x}) \not\models Y$ or $h(\overline{x}) \models V_c$ but $h(\overline{x}) \not\models E_c|V_c$; if so, return "yes". This is in NP. $\square$

Satisfiability. A set Σ of CGDs is satisfiable if Σ has a model; that is, a graph G such that (a) $G \models \Sigma$, and (b) for each CGD $P[\overline{x}](E_c|V_c)$ in Σ, there exists a match of P in G.

Intuitively, it is to check whether the CGDs are "dirty" themselves. A model G of Σ requires all patterns in the CGDs of Σ to find a match in G, to ensure that the CGDs do not conflict with each other.

The satisfiability analysis has to check subgraph isomorphism among the patterns of the CGDs, which is NP-complete (cf. [13]). In light of this, we have the following.

Theorem 2. *The satisfiability problem is coNP-complete for CGDs.*

Implication. We say that a set Σ of CGDs implies another CGD ϕ, denoted by $\Sigma \models \phi$, if for all graphs G such that $G \models \Sigma$, we have that $G \models \phi$, i.e., ϕ is a logical consequence of Σ.

We assume w.l.o.g. the following: Σ is satisfiable, since otherwise it makes no sense to consider $\Sigma \models \phi$.

The *implication problem* for CGDs is to determine, given a set Σ of CGDs and another CGD ϕ, whether $\Sigma \models \phi$.

In practice, the implication analysis helps us eliminate redundant CGDs, and hence, optimize our matching algorithms by minimizing CGDs. By computing mappings from the pattern of each CGD in Σ to the pattern P of ϕ and checking whether the mappings are isomorphic to subgraphs of P, we can prove the following conclusion for the implication problem.

Theorem 3. *The implication problem is NP-complete for constant CGDs alone, even when all the CGDs are defined with DAG patterns.*

Finally, we show that CGDs has the *Church-Rosser property* [5].

Theorem 4. *Chasing with CGDs has the Church-Rosser property, i.e., for all Σ and G, all terminal chasing sequences of G by Σ have the same result, regardless of in what order the CGDs are applied.*

B. Complexity of Consistent Query Answering

The decision problem of CSM is stated as follow:

Input: graph G, CGDs Σ and query Q

Output: $\star$-Cons$(G, Q, \Sigma) \neq \Phi$

Theorem 5. *It is NP-complete to process a CSM query under any $\star$-repair ($\star \in \{\subseteq, \supseteq, \oplus\}$).*

Proof sketch: It is known that the subgraph isomorphism problem is already NP-hard [13]. Since the CSM problem subsumes subgraph isomorphism, the NP lower bound for subgraph isomorphism carries over to CSM. We show the upper bound by presenting an NP algorithm in two steps:

When Σ is fixed, we can compute chase(G, Σ) in polynomial time in $|G|$ as follows: We show that in any chasing sequence ρ of G and Σ, the graph repair G_r in any chase step has size at most $4|G||\Sigma|$. Based on the bound, one can readily verify that the length of ρ is at most $8|G||\Sigma|$. Clearly this step is in PTIME.

Given a subgraph isomorphism h from Q to G, we check whether h is a match of Q in G; if so, return true. This step is in PTIME. Hence the problem is in NP. $\square$

IV. A SEQUENTIAL ALGORITHM

A. Algorithmic Framework

Given a graph G, a set of CGDs Σ and a query Q, a naive query processing is to repair G based on every CGD in Σ and then computes the consistent answers according to Equation 1 by performing subgraph matching Q against every graph repair. The naive solution has two shortcomings:

```
Algorithm ConsGraph
Input: graph $G$, CGDs $\Sigma$ and query $Q$.
Output: $\star$-Cons$(G, Q, \Sigma)$.
1.  ValCGD computes a set $\Sigma_v$ of valid CGDs from $\Sigma$ and $Q$;
2.  ExOrd computes an execution order $EOD$ of CGDs in $\Sigma_v$;
3.  SubMat computes subgraph matching for every CGD in $\Sigma_v$
    and $Q$ based on $EOD$;
```

Fig. 4. Framework of consistent subgraph matching.

First, the number of graph repairs is very large and can be exponential in the worst case. Second, the set Σ is usually large, e.g., more than 200 CGDs are involved for a graph in our experiments. The consistent answers of Q in G only relate to a limited number of CGDs in Σ as shown in the experimental results. Therefore, it is not necessary to apply all CGDs in Σ to answer Q.

To conquer the two shortcomings, we propose a general algorithmic framework ConsGraph to answer the CSM query efficiently.

As show in the framework, ConsGraph consists of three procedures: (1) ConsGraph proposes an algorithm ValCGD to compute a set Σ_v of valid CGDs from Σ and Q. (2) ConsGraph proposes an algorithm ExOrd to compute an execution order EOD for CGDs in Σ_v. (3) ConsGraph proposes an algorithm SubMat to compute subgraph matching for every CGD in Σ_v and Q based on EOD. Finally, the set of computed subgraph matches is $\star$-Cons(G, Q, Σ). Note that the framework can process the property graphs without any modification.

Intuitively, ValCGD can obtain CGDs from Σ related to the final answers and avoid enumerating all the CGDs in Σ. In other words, the patterns of some CGDs may not overlap with query Q. In this case, these CGDs do not impact on the query answers. Using the optimal order of valid patterns returned by ExOrd, we can efficiently compute the consistent answers by performing subgraph matching (i.e., SubMat) of the valid patterns over G.

The following three subsections will introduce the algorithms ValCGD, ExOrd and SubMat, respectively.

B. Computing Valid Conditional Graph Dependencies

The procedure ValCGD computes a set Σ_v of valid CGDs related to final answers from Σ and Q. It includes the following three steps:

(1) ValCGD merges Q and all the patterns P of Σ into a union pattern UP, i.e., $UP = \cup_{P \in \Sigma} P \cup Q$.

(2) ValCGD determines a valid subpattern VP of UP from Q.

(3) ValCGD computes a set Σ_v of valid CGDs (from Σ) involving in VP.

We first introduce the second and third steps and then the first step.

To define a valid pattern, we need the following definitions.

For a CGD $\phi = P[\overline{x}](E_c|V_c)$, a node u of P is a *conditional node* if u contains literals involved in V_c. Node u is a *conditional predecessor* of node v if $(u, v) \in E(P)$ and the literals of u *imply* the literals of v. A conditional path $P_c = v_1, ..., v_m$ is a path in UP and v_i is a conditional predecessor of v_{i+1} for $i \in [1, m-1]$.

After ValCGD computes UP in the first step (see below), we can obtain the common nodes of Q and Ps in Σ. Thus Q may contain conditional nodes in UP. Denote by V_{cq} the set of the conditional nodes in Q, then we have:

Definition 2. *(Valid pattern.) A valid pattern VP is a* connected *subgraph of UP and contains Q as a subgraph. For any node u in $VP \setminus Q$, there is a conditional path from u to a node in V_{cq}. A VP is maximum if no super-graph of VP in UP satisfies these conditions.*

To compute the maximum VP, ValCGD runs breadth-first searches (BFS) from every node in V_{cq}, until every BFS finds all its root nodes. During the search, a BFS visits a node u from v if u is a conditional predecessor of v. Once the maximum VP is obtained, ValCGD can select the set Σ_v of valid CGDs from Σ that have nodes in the maximum VP.

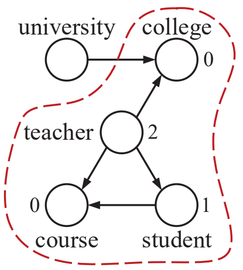


Fig. 5. Union and valid patterns of Q in Figure 3 and five patterns in Figure 2.

Example 5. *Figure 5 shows the union pattern UP of Q in Figure 3(a) and the five patterns (P_1, P_2, P_3, P_4 and P_5) in Figure 2. The valid pattern VP of UP is the subgraph in the red dashed circle, because any node of Q has no conditional predecessor in P_5 according to the CGD ϕ_5 in Example 2. Then P_1, P_2, P_3 and P_4 are valid CGDs, since they have nodes in the VP.*

Next, we introduce the first step. Assume that $\bigcup_{P \in \Sigma} = TP$, then we have $UP = TP \cup Q$.

ValCGD calculates $\bigcup_{P \in \Sigma} = TP$ offline. When Q is issued, ValCGD computes $UP = TP \cup Q$ online. ValCGD calculates TP by merging all the graphs in Σ one by one.

Either the offline phase or the online phase needs to merge two graphs. Assume the current resulted graph is A that will merge a graph B ($B \in \Sigma$ or B is Q). The merging of A and B needs to detect the maximum common subgraph (MCS) between A and B. The size of A is possibly large, if Σ has a large number of graphs. Therefore, the main challenge of calculating UP is how to efficiently compute the MCS between two large graphs.

In the following, we introduce a novel procedure MCS to deal with the challenge.

1) Detecting Common Subgraphs:

Given two graphs A and B, MCS gradually constructs and proves an optimal solution using a depth-first search. During the search, the algorithm maintains two variables: *curSol*, the solution under construction; and *maxSol*, the best solution found so far. In addition, every node v of A is associated with a subset $\alpha(v)$ of nodes of B that can be matched with v. At the beginning, *curSol* and *maxSol* are initialized to $\emptyset$, and $\alpha(v)$ is initialized to be the set of all nodes of B having same label of v. The first call MCS$(A, B, \alpha, \emptyset, \emptyset)$ returns a maximum common subgraph of A and B.

At each branching point, the algorithm first computes an upper bound UB of the cardinality of the best solution that can be found from this branching point, by calling the overestimate(A, B, α) function. It then compares UB with $maxSol$. If $UB \leqslant |maxSol|$, a solution better than $maxSol$ cannot be found from this branching point, and the algorithm prunes the current branch and backtracks. Otherwise, it selects a not-yet matched node v from A and tries to match it with every node w in $\alpha(v)$ in turn. As a consequence of matching v with w, (v, w) is added into *curSol*, and for each not-yet matched v' of A, $\alpha(v')$ is updated as follows: If v' is adjacent to v, remove all nodes non-adjacent to w from $\alpha(v')$; otherwise, remove all nodes adjacent to w from $\alpha(v')$.

Note that after updating $\alpha(v')$, $\forall w' \in \alpha(v')$, v is adjacent to v' in A iff w is adjacent to w' in B, so that the match (v', w') can be further added into $curSol$ to yield a feasible solution. If a solution better than $maxSol$ is found in a leaf of the search tree where further match is impossible, the algorithm updates $maxSol$ to be $curSol$ before backtracking to find an even better solution.

Figure 5 shows the union pattern of query Q and the five patterns mentioned before by running MCS.

C. Query Execution Order

Procedure ExOrd computes an execution order EOD of CGDs in Σ_v output by ValCGD. It includes two steps: (1) it determines an order nd of nodes in VP, and (2) it deduces an execution order EOD of CGDs in Σ_v based on nd.

ExOrd determines the order nd by constructing a directed dependency graph VP_d; it includes all conditional nodes of VP, and there exists an edge (x, y) in VP_d if and only if x is a conditional predecessor of y in VP. The rank $r(x)$ of each node x in VP_d is defined as $max_{(s_x,s_y)}E_{VP_d}\{r(y)+1\}$, in which E_{VP_d} denotes the edge set of a DAG VP'_d that collapses each strongly connected component of VP_d into a single node, and s_x (resp. s_y) refers to the corresponding node of x (resp. y) in VP'_d. When s_x has no outgoing edges in VP_d, $r(x) = 0$.

Assume that, based on nd, the nodes of VP is ordered as $V(VP) = \{v_1, v_2, ..., v_n\}$. Note that every CGD ϕ in Σ_v contains a set of nodes in $V(VP)$. Assume that ϕ_i and ϕ_{i+1} are two adjacent CGDs in EOD.

ExOrd sorts the nodes in each $\phi \in \Sigma_v$ based on nd. Assume that the sorted nodes in ϕ_i and ϕ_{i+1} are $V(\phi_i) = \{x_1, ..., x_p\}$ and $V(\phi_{i+1}) = \{y_1, ..., y_{p'}\}$ ($p \neq p'$), respectively. The EOD of CGDs in Σ_v has the following rules:

(1) Assume that the front rank of y_1 in nd is y_0. Then y_0 must be in $V(\phi_i)$.

(2) Assume that $x_1 = y_1$,...,$x_j = y_j$. Then $x_{j+1} < y_{j+1}$ in nd.

From the two rules, ExOrd can easily deduce the execution order EOD.

Example 6. *As shown in Figure 5, because nodes course and college have no outgoing edges, their ranks are both 0. Then the order nd in the VP is $\{course: 0, college: 0, student: 1, teacher: 2\}$, and the node orders of each pattern in nd are: $P_1 = \{course: 0, teacher: 2\}$, $P_2 = \{college: 0, teacher: 2\}$, $P_3 = \{student: 1, teacher: 2\}$ and $P_4 = \{course: 0, student: 1\}$. Based on the above two rules, the execution order EQD of CGDs in Σ_v is $\{P_4, P_1, P_2, P_3\}$.*

D. Subgraph Matching Search

In this step, procedure SubMat performs subgraph matching for every CGD in Σ_v and Q based on EOD.

1) Idea of SubMat: Based on the edge constraint E_c, we can have a lot of graph repairs. Based on the node constraints V_c, we have a fixed graph repair.

SubMat first repairs G based on V_c to obtain a graph repair G_r. SubMat then issues Q over G_r based on E_c to obtain the final answers, i.e., $\star$-Cons(G, Q, Σ). In other words, the first phase of SubMat is to repair node errors, and the second phase is to handle the edge errors.

To do the first phase for SubMat, we follow the state-of-the-art works [5], [8], [10] that repair nodes's errors.

Consider a graph $G = (V, E, L, F_A)$ which uses x and y to range over nodes. To repair G, we advocate an equivalence relation, denoted by Eq. It includes an equivalence class $[x.A]_{Eq}$ for attributes $x.A$ in $F_A(x)$. More specifically, $[x.A]_{Eq}$ is a set of attributes $y.B$ and constants c, including $x.A$ for all $x.A$ in $F_A(x)$.

We then can clean G's nodes by applying Eq as follows.

(1) For each $[x.A]_{\mathsf{Eq}}$, if constant $c \in [x.A]_{\mathsf{Eq}}$, we generate new attribute $x.A$ if $x.A$ does not yet exist, and repair $x.A$ with correct value c by letting $x_{Eq}.A = c$, no matter whether $x.A$ has a value or not.

(2) For each $[x.A]_{\mathsf{Eq}}$ and $y.B \in [x.A]_{\mathsf{Eq}}$, we generate new attributes when necessary. Moreover, we equalize $x.A$ and $y.B$ by letting $x_{\mathsf{Eq}}.A = y_{\mathsf{Eq}}.B = c$ if there exists $c \in [x.A]_{\mathsf{Eq}}$; otherwise we let $x_{\mathsf{Eq}}.A = y_{\mathsf{Eq}}.B = \#$, denoting value to be assigned to x_{Eq} and y_{Eq} like labeled nulls.

The process proceeds until all equivalence classes $[x.A]_{\mathsf{Eq}}$ of Eq are enforced on G. It yields a graph, referred to as *the repair of* G *by* Eq, denoted by G_{Eq}.

The cleaning process can be implemented based on *chasing*. Formally, *a chase step of* G *by* Σ *at* (Eq, V_{Eq}) is

$$(\mathsf{Eq}, V_{Eq}) \Rightarrow_{(\phi,h)} (\mathsf{Eq}', V_{Eq'}).$$

Here $\phi = P[\bar{x}](X \to Y)$ is a V_c of CGD in Σ, V_{Eq} (resp. $V_{\mathsf{Eq'}}$) is a set of nodes in the repair G_{Eq} (resp. $G_{\mathsf{Eq'}}$) of G by Eq (resp. Eq'), and $h(\bar{x})$ is a match of P in G_{Eq} with $h(x) \in V_{\mathsf{Eq}}$ for a node $x \in \bar{x}$. Specifically, Eq' extends Eq by applying the above two rules in cleaning.

Let V_0 be the nodes with the ground trues. A chasing sequence ρ of G by Σ from V_0 is

$$(Eq_0, V_{Eq_0}), ..., (Eq_k, V_{Eq_k}).$$

Here $\mathrm{Eq}_0 = \Phi$, $V_{Eq_0} = \Phi$, and moreover, for all $i \in [1, k]$, there exists a CGD $\phi = P[\bar{x}](X \to Y)$ in Σ and a match h of P in the repair $G_{Eq_{i-1}}$ such that $(Eq_{i-1}, V_{Eq_{i-1}}) \Rightarrow_{(\phi,h)} (Eq_i, V_{Eq_i})$ is a valid chase step.

The sequence is *terminal* if there exists no CGD $\phi \in \Sigma$, match h of P in G_{Eq_k} and $(Eq_{k+1}, V_{Eq_{k+1}})$ such that $(Eq_k, V_{Eq_k}) \Rightarrow_{(\phi,h)} (Eq_{k+1}, V_{Eq_{k+1}})$ is valid.

Chasing sequence ρ terminates in one of the cases below.

(a) No CGDs in Σ can be applied to expand the chasing sequence. If so, we say that ρ is valid, and refer to $(\mathsf{Eq}_k, G_{\mathsf{Eq}_k}, V_{\mathsf{Eq}_k})$ as its result.

(b) Either Eq_0 is inconsistent or there exist $\phi, h, \mathsf{Eq}_{k+1}$ and $V_{\mathsf{Eq}_{k+1}}$ such that$(\mathsf{Eq}_k, V_{\mathsf{Eq}_k}) \Rightarrow_{(\phi,h)} (\mathsf{Eq}_{k+1}, V_{\mathsf{Eq}_{k+1}})$ but Eq_{k+1} is inconsistent. Such ρ is invalid, with result $\perp$(undefined).

After the chasing sequence, we can obtain a graph repair G_r that resolves all the node errors.

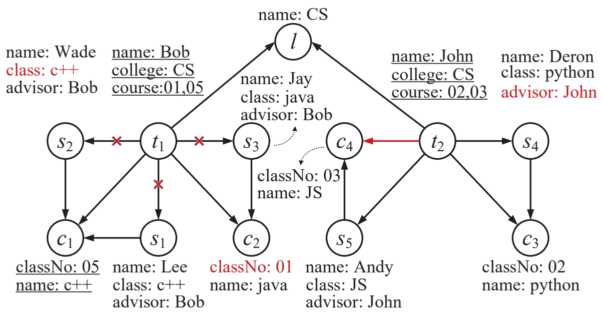


Fig. 6. Graph repair G_r.

Example 7. *Recall in Example 6, the executing order of valid graph patterns is P_4, P_1, P_2 and P_3. Then, the order of their related CGDs (in Example 2) is ϕ_4, ϕ_1, ϕ_2 and ϕ_3. We apply these CGDs to repair G (in Figure 1) based on the node constraints. Assume that the ground truth for nodes of G are underlined. The chasing includes four steps and starts from ϕ_4 with step (1) $(Eq_0,\emptyset) \Rightarrow_{(\phi_4,h_1)} (Eq_1,\{s_2\})$, where $h_1 : x \mapsto s_2$, $y \mapsto c_1$; and Eq_1 extends Eq_0 by letting $[s_2.class]_{Eq_1} = [c_1.name]_{Eq_1} = \{s_2.class, c_1.name, \text{“}c++\text{”}\}$. This first chase step detects the missing attribute class of s_2 in G, then generates this attribute by letting its value to be “c++”. Hence s_2 is included in V_{Eq_1}.*

Following the similar idea, the chasing can further (2) repairs the value 04 of $c_2.classNo$ to 01 in Eq_2 based on ϕ_1; (3) adds the attribute $l.name$ of node l into Eq_3 based on ϕ_2; (4) repairs the value Bill of $s_4.advisor$ to John in Eq_4 based on ϕ_3. After the four steps, there is no CGDs in

Σ_v which can be further applied. The chasing is terminated. We finally obtain the graph repair G_r as shown in Figure 6 (ignore the deleted and inserted edges).

2) Algorithmic Details: Based on the above idea, SubMat consists of two phases: (1) it does the chase for every CGD in Σ_v as the *EOD* to obtain a repair G_r; and (2) it then performs subgraph matching of Q over G_r. Note that chase of the first phase only fixed nodes' errors.

Note that SubMat can deal with property edges straightforwardly. Specifically, in the first phase SubMat also repairs edge properties errors (if they exist) and then does the same subgraph matching in the second phase.

Each chase step of the first phase starts from scratch: it (a) picks a CGD $\phi = P[\bar{x}](X \to Y)$ from Σ_v, (b) finds a mapping h from $P[\bar{x}]$ to G, and (c) checks whether X is entailed by Eq at $h(\bar{x})$. After computing h, the chase may find that X is not entailed by Eq and ϕ cannot be enforced; then it has to start steps (b) and (c) again.

Specifically, each chase step first finds *partial matches* that involve nodes in E_q, which were updated in the last iteration. For $P[\overline{x}](X \to Y)$, the partial matches compose of a set S_ϕ of subgraph isomorphisms $h(\overline{x}_s)$ from variables that appear in X to nodes in Eq. It ensures that each $h(\overline{x}_s)$ in S_ϕ satisfies X and contains at least one node from E_q.

The step then completes partial matches S_ϕ of P by the generic subgraph matching [14]. It verifies subgraph isomorphisms by checking edge connections. During the checking, it attaches an edge constraint l_e to the edges of the matches. Note that it does not repair edges based on l_e. It also deduces new fixes to be included in Eq. The procedure only *incrementally enumerates* matches.

The second phase of SubMat works as follows.

Given $e = (x, l, y)$ in $P[\bar{x}]$, let $u = h(x)$ and $v = h(y)$, $Deg(u, l)$ be the number of edges from u to v in G_r, and b the number of nodes $\{h(y)\}$ without edges from $h(x)$. For the constraint $NUM(x, l, y) = [min, max]$ of edge $e = (x, l, y)$, SubMat does the following for the three repairs, respectively.

Case 1: $\subseteq$-repair. For this repair, we have $Deg(u, l) > max$. SubMat removes all edges (u, l, v) from G_r. SubMat performs subgraph matching of Q over G_r to obtain the match set $Q(G_r)$. Then $\subseteq -$Cons(G, Q, Σ)= $Q(G_r)$.

Case 2: $\supseteq$-repair. For this repair, we have $Deg(u, l) < min$. Let $a = min - Deg(u, l)$. Two cases are considered. (a) If $a < b$, SubMat obtains $\supseteq -$Cons(G, Q, Σ)= $Q(G_r)$ after subgraph matching of Q in G_r. (b) If $a = b$, SubMat adds b edges $(h(x), l, h(y))$ to G_r. After subgraph matching of Q in G_r, $\supseteq -$Cons(G, Q, Σ)= $Q(G_r)$.

Case 3: $\oplus$-repair. If $Deg(u, l) < min$, the answers are the same as those of $\supseteq$-repair. If $Deg(u, l) > max$, the answers are the same as those of $\subseteq$-repair.

Example 8. *Following Example 7, SubMat performs Q in Figure 3(a) against the graph repair G_r in Figure 6. Note that the second phase of SubMat is to deal with the edge errors. Under the $\subseteq$-repair semantics, only ϕ_3 can detect edge errors in G_r: the three students guided by t_1 exceeded the allowable max number (i.e., $Deg(t_1, guide) = 3 > max = 2$). SubMat removes the three edges (the edges with red cross in Figure 6), and then performs subgraph matching. The consistent answer of $\subseteq$-repair is shown in Figure 3(b).*

Under the $\supseteq$-repair semantics, only ϕ_1 can detect edge errors in G_r: the teacher t_2 teaches only one course less than the required minimum number (two). Notice that there is no edge from t_2 to a course node c_4. Since $a = min - Deg(t_2, teach) = 2 - 1 = 1$ and $a = b = 1$, SubMat adds the edge (t_2, t_4) to G_r (red edge in Figure 6) and performs subgraph matching. The consistent answer of $\supseteq$-repair is given in Figure 3(c).

V. A PARALLEL ALGORITHM

Similar to conventional subgraph isomorphism, consistent subgraph matching may be cost-prohibitive over big graphs G. This suggests that we develop a parallel algorithm for consistent subgraph matching that guarantees to scale with big G. We develop such an algorithm (PConsGraph), which makes consistent subgraph matching feasible in real-life graphs, despite its high complexity.

A. Parallel Scalability

To characterize the effectiveness of parallelism, we advocate a notion of *parallel scalability* following [15]. Consider a problem I posed on a graph G. We denote by $t(|I|, |G|)$ the running time of the best *sequential algorithm* for solving I on G, *i.e.*, one with the least worst-case complexity among all algorithms for I. For a parallel algorithm, we denote by $T(|I|, |G|, n)$ the time it takes to solve I on G by using n processors, taking n as a parameter. Here we assume $n \ll |G|$, i.e., the number of processors does not exceed the size of G; this typically holds in practice as G often has trillions of nodes and edges, much larger than n.

Parallel scalability. An algorithm is parallel scalable if

$$T(|I|, |G|, n) = O(\tfrac{t(|I|, |G|)}{n}) + (n|I|)^{O(1)}.$$

That is, the parallel algorithm achieves a linear reduction in sequential running time, plus a "bookkeeping" cost $O((n|I|^l))$ that is independent of $|G|$, for a constant l.

A parallel scalable algorithm guarantees that the more processors are used, the less time it takes to solve I on G. Hence given a big graph G, it is feasible to efficiently process I over G by adding processors when needed.

B. Parallel Scalable Algorithm

As shown in the framework, ConsGraph consists of three procedures: ValCGD, ExOrd and SubMat. ValCGD and ExOrd mainly deal with the CGDs and query Q independent of graph G. SubMat performs subgraph matching over G.

To parallel ConsGraph, PConsGraph works with a coordinator S_c and n workers S_i. Clearly, PConsGraph can perform ValCGD and ExOrd at S_c without the parallelism. Therefore, PConsGraph needs to parallel SubMat over the

n workers. To achieve this, we propose a novel scheme to partition G across the n workers.

To maximize parallelism, a partition scheme should guarantee that for any graph G, (1) each of n computers manages a small fragment of approximate equal size, and (2) a query can be evaluated locally at each fragment without incurring inter-fragment communication. We call such a scheme *hop preserving partition*.

The scheme needs an integer d which is the radius of a graph pattern. For a node v in graph G and an integer d, the d-hop neighbor $N_d(v)$ of v is defined as the subgraph of G induced by the nodes within d hops of v.

Given a graph $G = (V, E, L)$, an integer d and a node set $V' \subseteq V$, a *d-hop preserving partition* $P_d(V')$ of V' distributes G to a set of n computers such that it is
(1) *balanced*: each computer S_i manages a fragment F_i, which contains the subgraph G_i of G induced by a set V_i of nodes, such that $\bigcup V_i = V'$ ($i \in [1, n]$) and the size of F_i is bounded by $c * \frac{|G|}{n}$, for a small constant c; and
(2) *covering*: each node $v \in V'$ is *covered* by $P_d(V')$, *i.e.*, there exists a fragment F_i such that $N_d(v)$ is in F_i.

We say that $P_d(V')$ is *complete* if $|V'| = |V|$.

We may want to find an optimal partition such that the number $|V'|$ of covered nodes is maximized. Unfortunately, it is NP-hard to create a balanced d-hop preserving partition. Indeed, conventional balanced graph partition is a special case when d=1, which is already NP-hard [13]. We provide an approximation algorithm for d-hop preserving partition with an approximation ratio.

Below we present such an algorithm, denoted by Part. Given a graph G stored at the coordinator S_c, it starts with a base partition of G, where each fragment F_i has a balanced size bounded by $c * \frac{|G|}{n}$. This can be done by using an existing balanced graph partition strategy (e.g., [16]). Part then extends each fragment F_i to a d-hop preserving counterpart.

(1) It first finds the "border nodes" $F_i.O$ of F_i that have d-hop neighbors not residing in F_i, by traversing F_i in parallel.

(2) Each worker S_i then computes and loads $N_d(v)$ for each $v \in F_i.O$, by "traversing" G via parallel breadth-first search (BFS) search [17]. Moreover, Part uses a balanced loading strategy [18] to load approximately equal amount of data to each worker in the search. The process repeats until no fragments can be expanded.

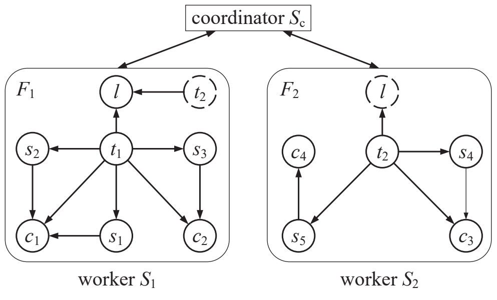


Fig. 7. Parallel matching

For example, assume that there are two workers and the radius of query is one. After Part partitions G (Figure 1) across the two workers, the border nodes are l in F_1 and t_2 in F_2 as shown in Figure 7. Then each worker computes and loads $N_d(v)$ which is not residing in F_i for each border node, i.e., the nodes t_2 and l with dashed circle.

For Part, we have the following lemma.

Lemma 1. *If $\Sigma_{v \in G} |N_d(v)| \leq c * \frac{|G|}{n}$, for any constant $\epsilon > 0$, there is a parallel scalable algorithm with approximation ratio $1 + \epsilon$ to compute a d-hop preserving partition.*

```
Algorithm 2 PConsGraph
Input: CGDs $\Sigma$, query $Q$, graph $G$, coordinator $S_c$, $n$ workers $S_1$,...,$S_n$
Output: the answer set $\star$-Cons$(G, Q, \Sigma)$
1.  Part($G$); /*Preprocessing*/
/*executed at coordinator $S_c$*/
2.  $\star$-Cons$(G, Q, \Sigma)$:=$\emptyset$; post $Q$ to each worker;
3.  if every worker $S_i$ returns answer $\star$-Cons$(F_i, Q, \Sigma)$ then
4.      $\star$-Cons$(G, Q, \Sigma)$:=$\bigcup\star$-Cons$(F_i, Q, \Sigma)$;
5.  return $\star$-Cons$(G, Q, \Sigma)$;
```

Fig. 8. **Algorithm** PConsGraph

Parallel algorithm. Using Part, we next develop algorithm PConsGraph. As shown in Algorithm 2, PConsGraph takes as input a set of CGDs Σ and query Q of radius at most d, and a graph G distributed across n workers by Part, where fragment F_i of G resides at worker S_i. It works as follows.

(1) The coordinator S_c runs ValCGD and ExOrd to obtain an execution order EOD of CGDs ϕ in Σ_v. S_c then posts every ϕ and Q to each worker S_i.

(2) Every worker S_i runs SubMat to obtain $\star$-Cons(F_i, Q, Σ) which is sent to S_c.

(3) Once all the workers have sent their partial matches to S_c, the coordinator computes $\star$-Cons(G, Q, Σ) as the union of all $\star$-Cons(F_i, Q, Σ).

Example 9. *Figure 7 shows that the graph G in Figure 1 is partitioned into two parts F_1 and F_2 across two workers S_1 and S_2. Take $\subseteq$-repair as an example to illustrate PConsGraph.*

After the coordinator S_c runs ValCGD and ExOrd to obtain an execution order EOD of CGDs, S_c posts every CGD and Q to each worker. In S_1, according to the CGD ϕ_3, PConsGraph removes three edges (t_1, s_1), (t_1, s_2) and (t_1, s_3) from F_1. Then the answer of $\subseteq$-repair in F_1 is $\emptyset$. Since F_2 satisfies ϕ_3, PConsGraph directly performs subgraph matching over F_2. After the partial matches $Q(F_2)$ are sent to S_c, we can obtain consistent answers of $\subseteq$-repair in Figure 3(b).

Finally, we have the following theorem for PConsGraph.

Theorem 6. *Given G distributed over n computers by a d-hop preserving partition p_d, (1) $\star$-Cons$(G, Q, \Sigma) = \bigcup_{i \in [1,n]}$ $\star$-Cons(F_i, Q, Σ), and (2) PConsGraph is parallel scalable for graphs G with $\sum_{v \in G} |N_d(v)| \leq c * \frac{|G|}{n}$, taking $O(\frac{t(Q,G)}{n} + n)$ time, where d is the largest radius of ϕs and Q, c is a* *predefined constant, and $t(\Sigma, Q, G)$ is the worst-case running time of sequential consistence subgraph matching algorithms.*

VI. Evaluation

Using real-life and synthetic graphs, we evaluated the accuracy, efficiency and (parallel) scalability of our algorithms.

A. Setup

Test-bed. All algorithms are deployed on a cluster with 13 machines connected with a high speed kilomega network, where one machine is selected as the coordinator and the remaining machines are workers. Each machine has 4-core Intel Core i7-880 3.06GHz CPU, 32 GB of memory, 1TB of HDD, and is running CentOS Linux 7.6.

Real-life graphs. We used two real-life graphs: (a) DBpedia [19], a knowledge graph with 31 million entities of 190 types and 37.6 million edges of 180 types; and (b) Yago2, an extended knowledge base of YAGO [20] with 2.5 million entities of 15 types and 6.6 million edges of 38 types.

Synthetic graphs. We designed a generator to produce synthetic graphs G, controlled by the number of nodes $|V|$ (from 10M to 50M) and the number of edges $|E|$ (up to 200M), with labels L and attributes F_A drawn from an alphabet of 100 symbols. Each node in G is assigned 5 attributes with values drawn from a domain of 200 distinct constants. We took G with 30M nodes and 80M edges as default.

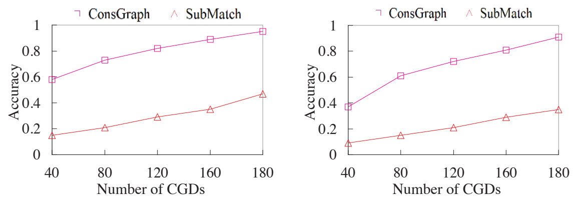
We introduced noises to the synthetic graphs. More specifically, we randomly (a) updated values of the node attributes that appear in CGDs Σ, controlled by attribute rate err%, the ratio of the number of updated attributes to the total number of such attributes; and (b) changed edges that appear in CGDs Σ, controlled by edge rate err%, the percentage of the number of inconsistency edges to the total number of such edges.

CGDs. We discovered a set Σ of CGDs for each graph G by extending algorithms of [21]. We mined 230 and 160 CGDs from DBpedia and Yago2, respectively, in which the graph patterns consists of at most 5 nodes and 10 edges, and the number of literals is at most 8. All CGDs in Σ were examined manually to guarantee correctness.

Measurements The accuracy of the algorithms is evaluated by precision, recall, and F-measure defined as 2· (precision · recall)/(precision + recall). Here precision is the ratio of true answers to all answers derived; and recall is the percentage of returned answers in all true answers.

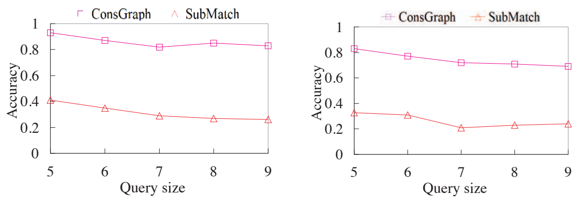
Query workload. We first generate star queries, and then extend the stars by adding nodes and edges to generate queries with complex structure, e.g., cliques, circles and multiple stars. We use Qi to denote the size of a query Q, where i is the number of nodes of Q. For each Qi, we generate a set of 20 queries to report the average performance. In the experiment, we set the query sets as $Q2$, $Q4$, $Q6$, $Q8$ and $Q10$, and $Q6$ is the default one.

Algorithms. To evaluate accuracies, we implement two algorithms: (1) ConsGraph, our proposed method (for $\subseteq, \supseteq, \oplus$-repair); and (2) SubMatch, which only fixed attributes' errors and then executed subgraph matching over fixed graphs without considering the edge errors. To evaluate efficiency, we implement three algorithms: (1) ConsGraph, our proposed sequential algorithm; (2) PConsGraph, our proposed parallel algorithm; and (3) RandMatch, which repairs graphs in a random order of all the CGDs in Σ and then applies the same procedure as ConsGraph.



(a) Yago2 ($\subseteq$-repair) (b) DBpedia ($\supseteq$-repair)

Fig. 9. Querying accuracy of real graphs w.r.t $|\Sigma|$.



(a) Yago2 ($\subseteq$-repair) (b) DBpedia ($\supseteq$-repair)

Fig. 10. Querying accuracy of real graphs w.r.t $|Q|$.

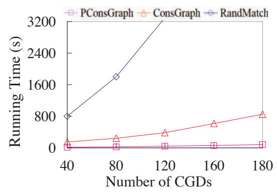
B. Experimental Results of Real Graphs

Exp-1: accuracy. We report the accuracies of ConsGraph and SubMatch over Yago2 ($\subseteq$-repair) and DBpedia ($\supseteq$-repair).

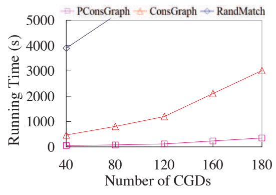
Varying $|\Sigma|$. We varied $|\Sigma|$, the number of CGDs, from 40 to 180 over both Yago2 and DBpedia. As shown in Figure 9, (1) the more CGDs are available, the higher accuracy gets by both methods, as expected; and (2) ConsGraph consistently outperforms SubMatch in quality by 53.2% on average, and the improvement becomes more substantial with the increase of $|\Sigma|$. The reason is that SubMatch neglects the edge errors which may have a large impact on the final accuracy.

Varying $|Q|$. Varying $|Q|$ (query size) from 5 to 9, we report results on Yago2 ($\subseteq$-repair) and DBpedia ($\supseteq$-repair) in Figures 10(a) and 10(b), respectively. As shown there, both accuracies of ConsGraph and SubMatch decrease very slowly when $|Q|$ becomes larger. We also observe that the accuracy on DBpedia is lower than that on Yago2. This is because given the same $|\Sigma|$ for DBpedia and Yago2, the cleaning algorithm will fix a given set of errors, but DBpedia is even larger than Yago2.

Exp-2: efficiency. We report the efficiency of PConsGraph, ConsGraph and RandMatch over Yago2 ($\subseteq$-repair) and DBpedia ($\supseteq$-repair).

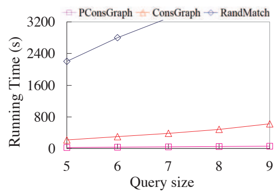


(a) Yago2 (⊆-repair)

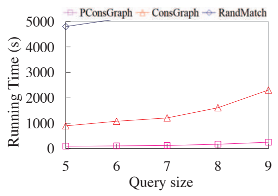


(b) DBpedia (⊇-repair)

Fig. 11. Querying time of real graphs w.r.t $|\Sigma|$.

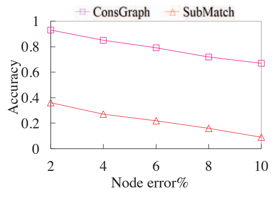


(a) Yago2 (⊆-repair)

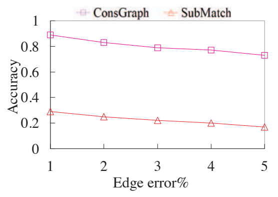


(b) DBpedia (⊇-repair)

Fig. 12. Querying time of real graphs w.r.t $|Q|$.



(a) $|V|$ =30M (⊆-repair)



(b) $|V|$ =30M (⊇-repair)

Fig. 13. Querying accuracy of synthetic graphs w.r.t node and edge errors.

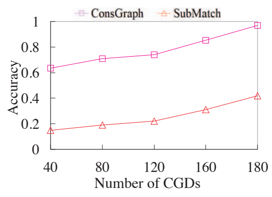
Varying $|\Sigma|$. Varying $|\Sigma|$ in the same way as in Exp-1, we evaluated the impact of CGDs on the efficiency. We find the following from Figures 11(a) and 11(b) on Yago2 and DBpedia, respectively. (1) All algorithms take longer to process more CGDs, as expected. (2) PConsGraph does the best in all cases, and is on average 9.8 times faster than the sequential ConsGraph. In contrast, RandMatch does not terminate in 100 minutes when applying 120 and 80 CGDs on Yago2 and DBpedia, since RandMatch performs subgraph matching for all the CGDs which is extremely expensive.

Varying $|Q|$. Fixing $|\Sigma| = 120$, we varied the average query size $|Q|$ from 5 to 9 over two real graphs. Results in Figure 12 show that all algorithms take longer to process queries with more nodes and edges. We also find that the running time of ⊇-repair is larger than that of ⊆-repair, since ⊇-repair incurs even more consistent answers than ⊆-repair.

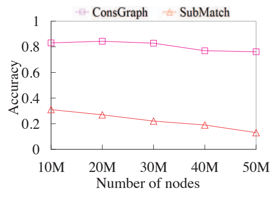
C. Experimental Results of Synthetic Graphs

Exp-3: accuracy. We report the accuracies of ConsGraph and SubMatch over the synthetic graphs.

Varying error%. Fixing the node error 6% and edge error 3%, we varied them from 2% to 10% and 1% to 5%, respectively. Their results are reported in Figures 13(a) (⊆-repair) and 13(b) (⊇-repair), respectively. We find the following. (a) ConsGraph consistently outperforms SubMatch; it is up to 61.5% and 57.2% more accurate for node errors and edge errors, respectively. (b) The accuracy of the two algorithms decreases when err% grows, as expected. However, they are less sensitive to the increase of edge errors. This verifies that node errors often have a larger impact on accuracy than edge errors.

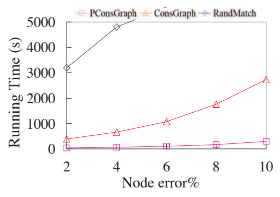


(a) $|V|$ =30M (⊕-repair)

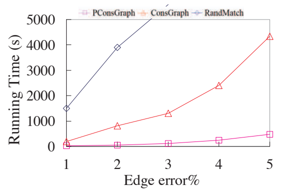


(b) $|V|$ =30M (⊕-repair)

Fig. 14. Querying accuracy of synthetic graphs w.r.t $|\Sigma|$ and $|G|$.



(a) $|V|$ =30M (⊕-repair)



(b) $|V|$ =30M (⊕-repair)

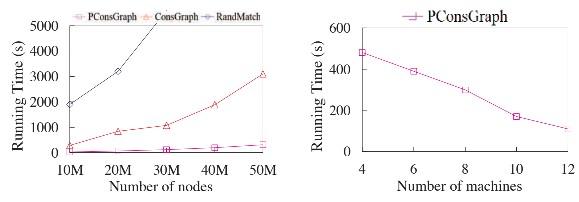
Fig. 15. Querying time of synthetic graphs w.r.t node and edge errors.

Varying $|\Sigma|$ and $|G|$. Fixing $|\Sigma| = 120$ and $|G| = 30M$, we varied them from 40 to 180 and from 10M to 50M, respectively. As shown in Figures 14(a) and 14(b) (both for ⊕-repair), (1) the more CGDs are available, the higher accuracy gets by ConsGraph and SubMatch; (2) the larger the graph size is, the lower accuracy gets for both algorithms due to that larger graphs are more difficult to repair; and (3) ConsGraph outperforms SubMatch in accuracies by 56.5% and 51.8% on average for $|\Sigma|$ and $|G|$, respectively.

Exp-4: efficiency. We report the efficiency of PConsGraph, ConsGraph and RandMatch over the synthetic graphs.

Varying error%. Varying node and edge errors in the same way as in Exp-3, we evaluated their impacts on the efficiency for ⊕-repair. We find the following from Figures 15(a) and 15(b). (1) All algorithms need longer time to repair more errors CGDs, as expected. (2) PConsGraph and ConsGraph are 115 times and 20 times faster than RandMatch, respectively. (3) Both PConsGraph and ConsGraph are well scalable w.r.t errors, but PConsGraph is more smooth as more errors are available.

Varying $|G|$ and n. We report the running efficiency for ⊕-repair by varying $|G|$ from 10M to 50M and n from 4 to 12. Figure 16(a) tells us the following. Parallel algorithm PConsGraph scales well with $|G|$ and is feasible on large graphs despite the exponential theoretical cost. It takes 312 seconds when G has 50 million nodes and 100 million edges, as opposed to more than 3000 and 8000 seconds taken by ConsGraph and RandMatch. Figure 16(b) shows that



(a) $|V| = 10M$–$50M$ ($\oplus$-repair) (b) $|V|$ =30M ($\oplus$-repair)

Fig. 16. Querying time of synthetic graphs w.r.t $|G|$ and n.

PConsGraph scales well with n: the improvement is 3.7 times when n increases from 4 to 12; this validates the parallel scalability of PConsGraph.

VII. RELATED WORK

We categorize the related work as follows.

Graph pattern matching. There have been a large number of algorithms developed for graph pattern matching so far, which can generally be classified into three categories. In the first category, the algorithms follow the traditional backtracking search [14], [22]–[24] which inevitably incurred expensive costs. In the second category, the algorithms advocate encoding and indexing techniques [25]–[29]. For example, the labels within a radius of each node are encoded as signatures in [26], the neighbors or degrees of each node are encoded as indices in [27], and the shortest paths in k-neighborhoods are encoded in [28]. In the third category, the pattern will be decomposed into a series of tree-like subgraphs in [30] and a dense subgraph along with a forest in [31], respectively. Then they perform the subgraph matching and join the matches to obtain the query answers.

Graph dependencies. Integrity constraints and data dependencies such as functional dependencies (FDs) have been extended to detect inconsistencies in graphs [5]–[9]. These constraints incorporate subgraph isomorphism, regular path queries or machine learning-based rules to identify the fraction of graphs the value constraints that should hold. For example, a graph functional dependency (GFD) with subgraph pattern P enforces value constraints on node attributes identified by P via subgraph isomorphism [14]. Based on these graph dependencies, graph repairing has been studied to fix errors in graphs [10], [32], [33]. GRRs [33] computes graph repairs by enforcing changes that are explicitly encoded by two subgraph patterns. GQRs [10] assumes reliable ground truth, and deduces a certain fix of graphs. StarFDs [32] resolves erroneous attribute values of graphs under the constraint of star-structured regular path patterns. Dependencies have also been studied for RDF [2]–[4].

Recently, PG-Keys are proposed for property graphs [12]. PG-Keys include constraints which are EXCLUSIVE, MANDATORY and SINGLETON on nodes, edges and properties, respectively. Different from PG-Keys, CGDs extend GFDs with edge cardinality constraint. However, the semantics (i.e., null values, regular path queries and complex values) for PG-Keys inspire us to add similar semantics to CGDs and to study new consistent graph queries. [34] studied threshold queries in theory and in practice. When a threshold is set to a number on graphs, threshold queries can check cardinality constraints on graphs. However, [34] does not propose how to repair the graph if violations are verified. In this paper, CGDs study the consistent queries under the repair semantics for cardinality constraints.

Consistent query answering (CQA) CQA has been studied from the complexity over different data models, notions of repairs and classes of constraints. Under the relational model, for instance, this problem has been studied for set-based [11], [35], [36], cardinality-based [37], and attribute-based repairs [38]; for constraints expressed as traditional functional dependencies, inclusion dependencies and denial constraints [39], [40]; and for constraints expressed as tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs) that arise in the context of data integration and data exchange [39]. The work [41] studied the data complexity of CQA over graphs for regular path queries (RPQs) and regular path constraints (RPCs).

ConsGraph differs from all the prior works in the following. (1) Traditional graph pattern matching identifies answers from a graph, whereas ConsGraph finds answers from a set of graph repairs whose quantity may be exponential. (2) CGDs used in ConsGraph support constraints both on nodes and edges, extending GFDs, GEDs and GQRs that only imposing node constraints. Despite the increased expressive power of CGDs, the satisfiability, implication, and validation problems for CGDs are no harder than their counterparts for existing graph dependencies. (3) CQA for the relational databases only focuses on the complexity and does not provide practical algorithms. Though ConsGraph is harder than CQA for the relational model, we provide practical (parallel) algorithms to support a good scalability of ConsGraph over large graphs.

VIII. CONCLUSION

We have made a first attempt to study consistent subgraph matching (CSM) under the subset, superset and symmetric difference graph repairs. To catch the consistence, we have proposed conditional graph dependencies (CGDs) that subsume the existing graph dependencies: GEDs, GFDs and GQRs. We have established the complexity of their satisfiability, implication, validation and matching problems. We have also developed (parallel scalable) algorithms underlying CSM. Our experiments have verified that the proposed solutions are promising.

One topic for future work is to study the incremental CSM and parallelize the proposed algorithms. Another topic is to clean graphs under CGDs.

ACKNOWLEDGEMENT

Ye Yuan is supported by the NSFC (Grant No. 61932004). Aoqian Zhang is supported by the NSFC (Grant No. 62102023). Guoren Wang is supported by the NSFC (Grant No. 61732003, No. 61729201 and No. U2001211).

References

[1] Y. Yuan, G. Wang, L. Chen, and H. Wang, "Efficient subgraph similarity search on large probabilistic graph databases," in *Proc. of VLDB*, 2012, pp. 800–811.

[2] W. Akhtar, A. Cortés-Calabuig, and J. Paredaens, "Constraints in rdf," in *International Workshop on Semantics in Data and Knowledge Bases*. Springer, 2010, pp. 23–39.

[3] A. Cortés-Calabuig and J. Paredaens, "Semantics of constraints in rdfs." in *AMW*. Citeseer, 2012, pp. 75–90.

[4] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres, "Weaving the pedantic web," in *LDOW*, 2010.

[5] W. Fan and P. Lu, "Dependencies for graphs," *ACM Transactions on Database Systems (TODS)*, vol. 44, no. 2, pp. 1–40, 2019.

[6] H. Ma, M. A. Langouri, Y. Wu, F. Chiang, and J. Pi, "Ontology-based entity matching in attributed graphs," *Proc. VLDB Endow.*, vol. 12, no. 10, pp. 1195–1207, 2019.

[7] W. Fan, Y. Wu, and J. Xu, "Functional dependencies for graphs," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1843–1857.

[8] W. Fan, X. Liu, P. Lu, and C. Tian, "Catching numeric inconsistencies in graphs," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 381–393.

[9] W. Fan, Z. Fan, C. Tian, and X. L. Dong, "Keys for graphs," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1590–1601, 2015.

[10] W. Fan, P. Lu, C. Tian, and J. Zhou, "Deducing certain fixes to graphs," *Proceedings of the VLDB Endowment*, vol. 12, no. 7, pp. 752–765, 2019.

[11] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 1999, pp. 68–79.

[12] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K. W. Hare, J. Hidders, V. E. Lee, B. Li, L. Libkin, W. Martens *et al.*, "Pg-keys: Keys for property graphs," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2423–2436.

[13] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. W.H.Freeman, 1979.

[14] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, 2004.

[15] C. P. Kruskal, L. Rudolph, and M. Snir, *A complexity theory of efficient parallel algorithms*. Springer Berlin Heidelberg, 1988.

[16] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed computing*, vol. 48, no. 1, pp. 96–129, 1998.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2001.

[18] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *Proceedings of the WSDM*, 2013, pp. 507–516.

[19] "http://wiki.dbpedia.org."

[20] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 697–706.

[21] W. Fan, C. Hu, X. Liu, and P. Lu, "Discovering graph functional dependencies," *ACM Transactions on Database Systems (TODS)*, vol. 45, no. 3, pp. 1–42, 2020.

[22] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 337–348.

[23] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han, "Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1429–1446.

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

[25] B. Bhattarai, H. Liu, and H. H. Huang, "Ceci: Compact embedding cluster index for scalable subgraph matching," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1447–1462.

[26] H. He and A. K. Singh, "Graphs-at-a-time: query language and access methods for graph databases," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 405–418.

[27] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 364–375, 2008.

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

[29] Y. Wu, S. Yang, and X. Yan, "Ontology-based subgraph querying," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 697–708.

[30] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, 2012.

[31] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1199–1214.

[32] P. Lin, Q. Song, Y. Wu, and J. Pi, "Repairing entities using star constraints in multirelational graphs," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 229–240.

[33] Y. Cheng, L. Chen, Y. Yuan, and G. Wang, "Rule-based graph repairing: Semantic and efficient repairing methods," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 773–784.

[34] A. Bonifati, S. Dumbrava, G. Fletcher, J. Hidders, M. Hofer, W. Martens, F. Murlak, J. Shinavier, S. Staworko, and D. Tomaszuk, "Threshold queries in theory and in the wild," *arXiv preprint arXiv:2106.15703*, 2021.

[35] B. Ten Cate, G. Fontaine, and P. G. Kolaitis, "On the data complexity of consistent query answering," *Theory of Computing Systems*, vol. 57, no. 4, pp. 843–891, 2015.

[36] G. Fontaine, "Why is it hard to obtain a dichotomy for consistent query answering?" *ACM Transactions on Computational Logic (TOCL)*, vol. 16, no. 1, pp. 1–24, 2015.

[37] A. Lopatenko and L. Bertossi, "Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics," in *International Conference on Database Theory*. Springer, 2007, pp. 179–193.

[38] J. Wijsen, "Condensed representation of database repairs for consistent query answering," in *International Conference on Database Theory*. Springer, 2003, pp. 378–393.

[39] B. Ten Cate, G. Fontaine, and P. G. Kolaitis, "On the data complexity of consistent query answering," *Theory of Computing Systems*, vol. 57, no. 4, pp. 843–891, 2015.

[40] J. Wijsen, "Certain conjunctive query answering in first-order logic," *ACM Transactions on Database Systems (TODS)*, vol. 37, no. 2, pp. 1–35, 2012.

[41] P. Barceló and G. Fontaine, "On the data complexity of consistent query answering over graph databases," *Journal of Computer and System Sciences*, vol. 88, pp. 164–194, 2017.