2018 IEEE 34th International Conference on Data Engineering

Rule-Based Graph Repairing: Semantic and Efficient Repairing Methods

Yurong Cheng #1, Lei Chen *2, Ye Yuan †3, Guoren Wang #†3

School of Computer Science and Technology, Beijing Institute of Technology, China
* Department of Computer Science and Engineering, Hong Kong University of Science and Technology
† School of Computer Science and Engineering, Northeastern University, China
1 yrcheng@bit.edu.cn 2 leichen@cse.ust.hk 3 {yuanye, wanggr}@mail.neu.edu.cn

***Abstract*—Real-life graph datasets extracted from Web are inevitably full of incompleteness, conflicts, and redundancies, so graph data cleaning shows its necessity. One of the main issues is to automatically repair the graph with some repairing rules. Although rules like data dependencies have been widely studied in relational data repairing, very few works exist to repair the graph data. In this paper, we introduce an automatic repairing semantic for graphs, called *Graph-Repairing Rules* (GRRs). This semantic can capture the incompleteness, conflicts, and redundancies in the graphs and indicate how to correct these errors. We study three fundamental problems associated with GRRs, implication, consistency and termination, which show whether a given set of GRRs make sense. Repairing the graph data using GRRs involves a problem of finding isomorphic subgraphs of the graph data for each GRR, which is NP-complete. To efficiently circumvent the complex calculation of subgraph isomorphism, we design a decomposition-and-join strategy to solve this problem. Extensive experiments on real datasets show that our GRR semantic and corresponding repairing algorithms can effectively and efficiently repair real-life graph data.**

I. Introduction

With the development of Web and network technologies, graphs increasingly show their importance in data structure modeling in many applications, such as knowledge bases [1][2], social networks [3], and bioinformatic networks [4]. These real-life graph datasets are usually extracted from Web or experiment results, which are inevitably full of incompleteness, conflicts, and redundancies.

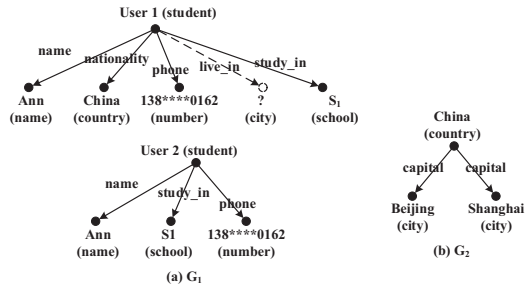


Fig. 1. Dirty Graphs

Example 1: Consider the following examples extracted from knowledge bases shown in Fig. 1. In a graph, semantics are presented by vertices and directed edges, together with labels on them. For example, in G_2 shown in Fig. 1(b), the vertex labelled by "China (country)", the directed edge labeled by "capital", and the vertex labelled by "Beijing (city)" present the meaning that "the capital of China is Beijing". Here, "China (country)" means the type of this vertex is "country", and its corresponding value is "China".

Incompleteness. For graph G_1 shown in Fig. 1(a), the information on the city where User 1 lives is missing. Such incompleteness would make it impossible to return results when queries are issued.

Conflicts. From graph G_2 shown in Fig. 1(b), it provides information that "China has two capital cities, Beijing and Shanghai". However, the fact is that a country has only one capital city. Thus, G_2 provides conflict information. Answers based on such a graph may cause errors in query results.

Redundancies. In G_1 of Fig. 1(a), User 1 and User 2 have the same name, study in the same school and have the same phone number. Thus we know that User 1 and User 2 are the same student. Redundancies in the graph data would increase the cost of storage. Moreover, when redundant data is used to answer aggregate queries, the results will be inaccurate.

Due to the above evidence of real-life graph data, it is necessary to develop efficient methods to clean graph data. Three factors should be considered by graph repairing: filling in the incomplete information, resolving the conflicts, and deleting redundancies. Extensive studies [5][6][7] focus on data repairing over relational data. However, very few studies exist to repair graph data. Different from relational databases recording only the attributes and values of entities, graph databases also present topological relationships between entities. This makes graph databases quite different from the traditional databases. A set of studies focus on entity resolution (determining whether two entities are the same) over graph data [8]. W. Fan et al. [9][10] have studied functional dependency for graphs and association rules in social network graphs respectively. However, none of them can cover all the three types of graph repairing requirements, which will be further explained together with the finding from experiments in Section V.

Therefore, **how can we design a proper graph-repairing semantic that can capture the incompleteness, conflicts, and redundancies in the graphs?** In this paper, we propose a comprehensive semantic called *Graph Repairing Rules* (GRRs). To our best knowledge, this is the first work that considers all the three repairing requirement (i.e., incompleteness, conflicts, and redundancies) using graph repairing rules.

Given a set of GRRs, **how can we know whether these**

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00075

GRRs make sense? In this paper, we study three fundamental problems associated with GRRs, implication, consistency and termination. We formally define these problems on GRR semantics and discuss some properties in these problems.

Taking advantages of the GRR semantics, **how to efficiently repair a given graph ?** As it will be seen that repairing the graph data using GRRs involves a problem of finding isomorphic subgraphs of the graph data for each GRR, which is NP-complete. Fortunately, since the size of each GRRs is usually very small [11], we design a decomposition-and-join strategy to efficiently circumvent the complex calculation of subgraph isomorphism. Specifically, it is conducted by decomposing the GRRs, finding matches for each decomposed part, and joining the matched results. However, different decomposition of GRRs influences the efficiency, and we prove that finding the optimal decomposition is NP-hard and cannot be approximated within any constant factor in polynomial time. Thus, we provide two heuristic algorithms for decomposition.

To summarize, our contributions in this work are as follows.

- We propose a GRR semantic for graph repairing, which can capture the incompleteness, conflicts, and redundancies in graphs.
- To determine whether a given set of GRRs make sense, we study three fundamental problems defined on GRR semantics: implication, consistency, and termination.
- We design a decomposition-and-join strategy. Since finding the optimal decomposition of GRRs in this strategy is proved to be NP-hard and cannot be approximated within any constant factor in polynomial time, we provide two heuristic algorithms for decomposition.
- Extensive experiments on real datasets show the effectiveness and efficiency of our solutions.

The rest of our paper is organized as follows. In Section II, we introduce some preliminaries and describe our GRR model. In Section III, we discuss the three fundamental problems associated to our GRR semantic. In Section IV, we illustrate how to efficiently repair the graph data using GRRs, which includes a baseline method and our improved decomposition-and-join strategy. Extensive experiments are conducted and analyzed in Section V. We summarize the related work and conclude our work in Section VI and Section VII respectively.

II. Graph Repairing Rules

Before defining our *Graph Repairing Rule* (GRR) semantics, we firstly introduce some basic preliminaries.

A. Preliminaries

Definition 2.1 (Graphs): The directed graph data is defined as $G = (V, E, L, F)$ with labelled vertices and edges. Here, V is a finite set of vertices, and $E \subseteq V \times V$ is a set of edges. Each vertex $v \in V$ (resp. edge $e \in E$) is attached with a label $L(v) \in L$(resp. $L(e) \in L$), which represents the attribute of this vertex (resp. edge). For each vertex v, there exists a function $f(L(v))$ that maps a constant value to the attribute of this vertex. □

For example, in Graph G_2 shown in Fig. 1(b), the attributes of each vertices are labelled in parentheses, such as "(country)". Its attribute value is "China". The labels/attributes of the two edges are both $L(e)$ ="capital".

Notice that we only study **simple** graphs in this paper, which means there are no: (1) multiple edges between any two vertices; (2) multiple labels at the one single vertex or edge; or (3) self loops on any vertex.

Definition 2.2 (Subgraphs [9]): Graph $G' = (V', E', L', F')$ is a subgraph of $G = (V, E, L, F)$, if $V' \subseteq V$, $E' \subseteq E$, and for each vertex $v' \in V'$, $L(v') = L(v)$, $f(L(v')) = f(L(v))$; for each edge $e' \in E'$, $L(e') = L(e)$. Denoted by $G' \subseteq G$. We say G' is a subgraph **induced** by V' if $G' \subseteq G$ and E' contains all the edges whose endpoints are both in V'. □

Definition 2.3 (Graph Patterns [9]): A graph pattern is defined as a directed graph $Q = (V_Q, E_Q, L_Q, g)$. V_Q is a set of vertices, called pattern vertices; $E_Q \subseteq V_Q \times V_Q$ is a set of edges, called pattern edges; for each vertex $v \in V_Q$ (resp. edge $e \in E_Q$), $L_Q(v)$ (resp. $L_Q(e)$) assigns a label to this pattern vertex (resp. edge) representing its attribute. g is mapping function that maps each pattern vertex with a label. Particularly, we allow a special label, wildcard "_", which means the label can be arbitrary. □

For example, Fig. 2 shows 9 graph patterns. In Q_2, 3 labels x, y, z are mapped to the 3 pattern vertices, which are specified as "country", "city", and "city" respectively. Their values are not specified.

Notice that a graph pattern is not necessarily to be connected, such as Q_1 in Fig. 2. Moreover, consistent with Definition 2.1, a graph pattern is also a **simple** graph.

Definition 2.4 (Isomorphism [9]): Two graphs, denoted as $G_1 = (V_1, E_1, L_1, F_1)$ and $G_2 = (V_2, E_2, L_2, F_2)$ are said isomorphic if there exists a bijective function h from V_1 to V_2 such that, (1) for each node $v \in V_1$, $L_1(v_1) = L_2(h(v_1))$; (2) $e(v, v')$ with $v, v' \in V$ is an edge of G_1 iff $e'(h(v), h(v'))$ is an edge of G_2 and $L_1(e) = L_2(e')$. In particular, $L_1(v_1) = L_2(h(v_1))$ always holds if $L_1(v_1)$ ="_". □

For example, the two connected components in Q_1 shown in Fig. 2(a) are isomorphic to each other.

Definition 2.5 (Graph Pattern Match [9]): A match of a pattern $Q = (V_Q, E_Q, L_Q, g)$ in a graph $G = (V, E, L, F)$ is a subgraph $G' = (V', E', L', F') \subseteq G$ isomorphic to Q. □

For example, in Fig. 1(a), a match of Pattern Q_3 in Fig. 2(e) is $x \mapsto$ "User 1", $y_1 \mapsto$ "China", and $y_2 \mapsto$ "138****0162".

B. Graph Repairing Rule Semantics

In this subsection, we introduce our Graph Repairing Rule (GRR) semantics in detail.

Definition 2.6 (Graph Repairing Rule): A Graph Repairing Rule (GRR) is defined by $\varphi = (Q, X) \rightarrow (Q', Y)$, where (1) Q (resp. Q') is a graph pattern before (resp. after) repairing, and (2) X (resp. Y) is the (possible empty) set of literals describing Q (resp. Q').

Compared to $Q(V, E, L, F)$, 7 operations may be needed to repair Q to $Q'(V', E', L', F')$: (1) adding new vertices to Q; (2) adding new edges between vertices of Q; (3) deleting existing vertices from Q; (4) deleting existing edges from Q; (5)

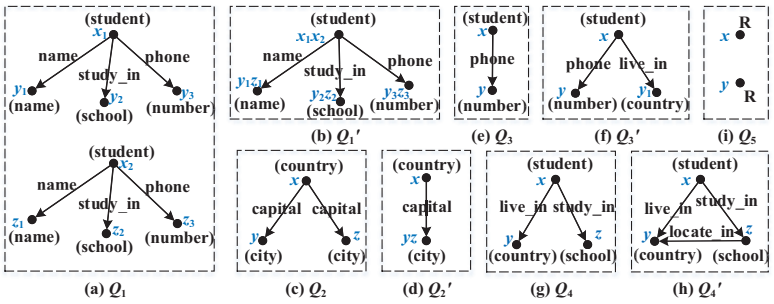


Fig. 2. Graph Repairing Rules

merging the vertices (resp. edges) with the same labels into one vertex (i.e., if in Q, $v_1, v_2, v_x \in V$, $e(v_x, v_1), e(v_x, v_2) \in E$ (or $e(v_1, v_x), e(v_2, v_x) \in E$) $L(v_1) = L(v_2) \in L$, $L(e(v_x, v_1)) = L(e(v_x, v_2))$ (or $L(e(v_1, v_x)) = L(e(v_2, v_x))$), then in Q', $v_1 = v_2$, $e(v_x, v_1)$=$e(v_x, v_2)$ (or $e(v_1, v_x)$=$e(v_2, v_x)$)); (6) combination of (1)∼(5); and (7) Q' keeps the same structure with Q, i.e., $Q' = Q$.

Literals in X (resp. Y) present the value requirements of Q (resp. Q'), which has 4 forms: (1) requiring the value of the attribute of a vertex to be a constant, i.e., $f(L(v_x)) = \{c\}$ (for simplicity, denoting $f(L(v_x))$ as $x.\mathsf{val}$ in the rest of this paper); (2) requiring the value of the attribute of a vertex not to be a constant, i.e., $x.\mathsf{val} \neq \{c\}$; (3) requiring the values of two attributes of two vertices to be same, i.e., $x.\mathsf{val} = y.\mathsf{val}$, and (4) combination of (1)∼(3). □

Example 2 (Graph Repairing Rule): Consider the following 5 GRRs.

(1) $\varphi_1 = (Q_1, X_1) \rightarrow (Q_1', Y_1)$, where Q_1 and Q_1' are shown in Fig. 2(a) and (b), $X_1 = \{y_1.\mathsf{val} = z_1.\mathsf{val}, y_2.\mathsf{val} = z_2.\mathsf{val}, y_3.\mathsf{val} = z_3.\mathsf{val}\}$, and $Y_1 = \phi$. This GRR states that for each match of Q_1, if two records of students have the same name, school, and phone number, these two students should be the same person. Here, Operation (5) is needed to repair Q_1 to Q_1'. Specifically, vertices x_1 and x_2 in Q_1 are merged into vertex x_1x_2 in Q_1'. Similarly, vertices y_1-y_3 and vertices z_1-z_3 in Q_1 are merged into y_1z_1-y_3z_3 in Q_1' respectively. Literals in X_1 are of Operation (3), and literals in Y_1 are handled Operation (1).

(2) $\varphi_2 = (Q_2, X_2) \rightarrow (Q_2', Y_2)$, where Q_2 and Q_2' are shown in Fig. 2(c) and (d), $X_2 = \{x.\mathsf{val} = \text{“China”}\}$ and $Y_2 = \{yz.\mathsf{val} = \text{“Beijing”}\}$. This GRR states that if a country has two records of capital, then the two records should be the same. Here, Operation (5) is needed to repair Q_2 to Q_2'. Specifically, vertices y and z in Q_2 are merged into vertex yz in Q_2', and edges $e(x, y)$ and $e(x, z)$ in Q_2 are merged into edge $e(x, yz)$ in Q_2'. Literals in Y_2 are handled Operation (3).

(3) $\varphi_3 = (Q_3, X_3) \rightarrow (Q_3', Y_3)$, where Q_3 and Q_3' are shown in Fig. 2(e) and (f), $X_3 = \phi$ and $Y_3 = \phi$. The structure changed from Q_3 to Q_3' states that if we know a student's phone number, we can infer the country that this student lives in. Here, Operation (1) and (2) are needed to repair Q_3 to Q_3'.

(4) $\varphi_4 = (Q_4, X_4) \rightarrow (Q_4', Y_4)$, where Q_4 and Q_4' are shown in Fig. 2(g) and (h), $X_4 = \phi$ and $Y_4 = \phi$. The structure changed from Q_4 to Q_4' states that if we know the country y that a student x lives in and the school z that x studies in, then we can infer that the school z is located in the country y. Here, Operation (1) is needed to repair Q_4 to Q_4'.

(5) $\varphi_4 = (Q_2', X_4) \rightarrow (Q_2', Y_4)$, where Q_2' is shown in Fig. 2(d), $X = \{x.\mathsf{val} = \text{“China”}, yz.\mathsf{val} \neq \text{“Beijing”}\}$, and $Y = \{yz.\mathsf{val} = \text{“Beijing”}\}$. This GRR states that if a country's value is “China”, and its capital's record is not “Beijing”, then we correct the value of its capital into “Beijing”. Here, Operation (7) is needed to repair Q_2' (keeping the same structure after being repaired), and literals in X_4 are handled Operation (1) and (2). □

The following property illustrates what types of dirty graphs can be repaired by the GRR semantic.

Property 1 (Repairing Types): The GRR semantic can handle 5 types of repairing.

(1) Attribute Supplement. GRRs for attribute supplement are such rules that can infer new attributes of an entity. Consider φ_3 in Example 2(3). After repairing, we will add a new attribute (i.e., a new edge labelled by) “live_in” and a new entity (i.e. vertex) labelled by “country” in the original graph to supply its incomplete information.

(2) Relationship Supplement. GRRs for relationship supplement are such rules that can infer new relationships between two existing entities. Consider φ_4 in Example 2(4). After repairing, we will add a new edge labelled by “locate_in” between the vertices matched by y and z in the original graph to supply its incomplete information.

(3) Relationship Resolution. GRRs for relationship resolution are such rules that can determine two relationships of an entity are the same, and combine them into one relationship. Consider φ_2 in Example 2(2). After repairing, the redundant relationship,“(country) capital (city)”, is removed.

(4) Attribute Value Correction. GRRs for attribute value correction are such rules that can detect the value of an attribute is wrong, and correct this value. Consider φ_5 in Example 2(5). After repairing, for matches of Q_2' in a given graph data, the wrong attribute values are changed into the right attribute value “Beijing”. This can correct conflicts of graphs.

(5) Entity Resolution. GRRs for entity resolution are such rules that can determine two entities in a graph are the same and merge them into one entity. Consider φ_1 in Example 2(5). After repairing, the redundant matches of the structure formed by $\{x_2, z_1, z_2, z_3\}$ are removed.

In summary, GRR semantic can repair the incompleteness (Types (1) and (2)), conflicts (Types (3) and (4)), and redundancies (Type (5)) in the graph data. □

Problem Definition: Based on our GRR semantic, we give our problem definition as follows.

Definition 2.7 (Graph Repairing): Given a set of GRRs Σ, for each GRR defined by $\varphi = (Q, X) \rightarrow (Q', Y)$, if G_Q is a match (Definition 2.5) of Q in graph G, and its values of attributes satisfy the constraints listed by X, we say that the match G_Q **satisfies** the GRR φ. We say the graph G is **repaired** by φ if all matches that satisfies φ are found and transformed into matches of Q' with attribute values satisfying Y. Recursively applying GRRs in Σ, we say G is repaired by Σ if it is repaired by all φs in Σ. □

Notice that data repairing is not from whimsicality. We cannot effectively repair any graphs if rules provide wrong evidences. The quality of the given GRRs will directly influence the quality of graph data repairing. When the set of GRRs Σ is provided, is it complete (which means the repairing result keeps the same if the graph is repaired based on $\Sigma \cup \{\varphi_{new}\}$, where φ_{new} is any other rule that is not contained by Σ) and consistent (which means the repairing evidences provided by Σ are consistent)? Can the repairing based on Σ be terminated? Similar to the repairing over relational databases [7][5], these three fundamental problems inferring whether the given set of GRRs Σ has a good quality for graph databases, which will be furthered studied respectively in the next section.

III. Fundamental Problems

In this section, we define the three fundamental problems implication, consistency, and termination problems in GRR semantic, and provide some properties under these three problems respectively.

A. Implication

Definition 3.1 (Implication Problem): Given a set of GRRs Σ, and another GRR $\varphi \notin \Sigma$, we say that Σ **implies** φ (denoted as $\Sigma \models \varphi$), if for each graph G that is repaired by Σ, we have G is equivalently repaired by $\Sigma \cup \{\varphi\}$. The implication problem is to determine whether $\Sigma \models \varphi$. □

The implication problem can be used to detect whether Σ is complete, which means that the repairing result would keep the same if graph G is repaired by Σ and $\Sigma \cup \{\varphi\}$, where φ is any other rule that is not contained by Σ.

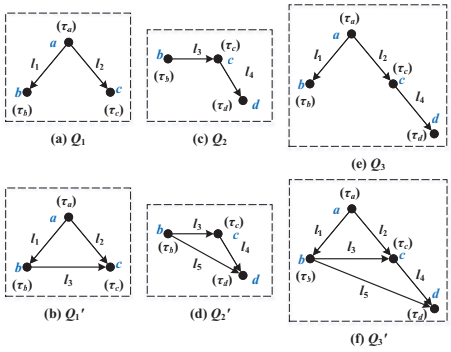


Fig. 3. GRR Implication

For example, Fig. 3 shows three GRRs, $\varphi_1 = (Q_1, X_1) \rightarrow (Q_1', Y_1)$, $\varphi_2 = (Q_2, X_2) \rightarrow (Q_2', Y_2)$, and $\varphi_3 = (Q_3, X_3) \rightarrow (Q_3', Y_3)$, where $X_1 = Y_1 = X_2 = Y_2 = X_3 = Y_3 = \phi$. φ_1 states that if a graph satisfies Q_1, we can add an edge labelled by l_3 between the vertex matched by b and the vertex matched by c. Similar are the meanings of φ_2 and φ_3. It is not hard to find that $\{\varphi_1\varphi_2\} \models \varphi_3$ and $\{\varphi_3\} \models \varphi_1$ holds, but $\{\varphi_3\}$ does not imply φ_2.

Definition 3.2 (Union Graph): A graph $G_u(V_u, E_u)$ is the union graph of a set of graphs $\{G_1(V_1, E_1) \ldots G_n(V_n, E_n)\}$ iff $V_u = \bigcup_{i=1}^{n} V_i$ and $E_u = \bigcup_{i=1}^{n} E_i$, which is denoted as $G_u = \bigcup_{i=1}^{n} G_i$. □

For example, Q_3' is the union graph of $\{Q_1', Q_2'\}$.

Lemma 1: Given a set of GRRs Σ with each $\varphi_i = (Q_i, X_i) \rightarrow (Q_i', Y_i) \in \Sigma$ and another $\varphi = (Q_\varphi, X_\varphi) \rightarrow (Q_\varphi', Y_\varphi) \in \Sigma$, $\Sigma \models \varphi$ holds if it satisfies: (1) Q_φ is subgraph isomorphic (referring to Definition 2.5) to $G_u' = \bigcup_{i=1}^{|\Sigma|} Q_i$; (2) Q_φ' is subgraph isomorphic to $G_u' = \bigcup_{i=1}^{|\Sigma|} Q_i'$; (3) $X_\varphi \subseteq \bigcup_{i=1}^{|\Sigma|} X_i$; and (4) $Y_\varphi \subseteq \bigcup_{i=1}^{|\Sigma|} Y_i$. Here, $|\Sigma|$ is the number of GRRs in Σ.

Proof: Conditions (1) and (3) ensure that the matches to be repaired by Q_φ can be covered by the union graph G_u, which means the matches of Q_φ can be equivalently found by Q_is in Σ. Conditions (2) and (4) ensure the repaired results of φ can be covered by G_u', which means the found matches can be equivalently repaired by Σ. According to Definition 3.1, Lemma 1 holds. ■

Corollary 1: If $\Sigma_s \subseteq \Sigma$ and $\Sigma_s \models \varphi$, then $\Sigma \models \varphi$.

Theorem 1: The implication problem for GRR is NP-hard (lower bound of complexity).

Proof: (Sketch). According to Corollary 1, determining whether $\Sigma \models \varphi$ involves the problem of subgraph isomorphism. Since the subgraph isomorphism problem has been proved to be NP-complete [12], the implication problem is NP-hard. ■

B. Consistency

Definition 3.3 (Consistency Problem): Given a set of GRRs Σ, the consistency problem is to decide whether there exists a non-empty graph that satisfies all the GRRs in Σ. □

The consistency problem can be used to detect whether the evidences provided by Σ are consistent.

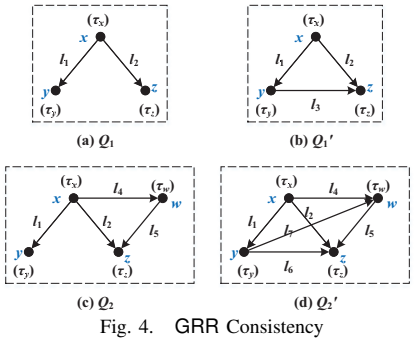


Fig. 4. GRR Consistency

For example, Fig. 4 shows two GRRs, $\varphi_1 = (Q_1, X_1) \rightarrow (Q_1', Y_1)$, $\varphi_2 = (Q_2, X_2) \rightarrow (Q_2', Y_2)$, where $X_1 = Y_1 = X_2 = Y_2 = \phi$. φ_1 states that if a graph satisfies Q_1, it is repaired by Q_1' through adding the edge labelled by l_3. φ_2 states that if a graph satisfies Q_2, it is repaired by Q_2' through

adding the edges labelled by l_6 and l_7. If $l_3 \neq l_6$, the repairing would not be consistent since using different GRRs would cause different labels on the same edge.

Definition 3.4 (Common Isomorphic Subgraph): Given two graphs G_1 and G_2, a graph G_c is called a common isomorphic subgraph if G_c is a subgraph of G_1 and subgraph isomorphic to G_2. A common isomorphic subgraph G_c is **maximal** if adding any more vertices or edges of G_1 to G_c, it would not be subgraph isomorphic to G_2. □

From the above example, we can see that if two GRRs ($\varphi_1 = (Q_1, X_1) \rightarrow (Q'_1, Y_1)$ and $\varphi_2 = (Q_2, X_2) \rightarrow (Q'_2, Y_2)$) are consistent, the repairing to the same vertices and edges or even their labels and values should be the same. In other words, if two GRRs are consistent, the repairing method of their maximal common isomorphic subgraph should be consistent. Thus, the following lemma holds.

Lemma 2: Given two GRRs, $\varphi_1 = (Q_1, X_1) \rightarrow (Q'_1, Y_1)$ and $\varphi_2 = (Q_2, X_2) \rightarrow (Q'_2, Y_2)$, a graph Q_c is the maximal common isomorphic subgraph of Q_1 and Q_2. After being repaired according to φ_1 (resp. φ_2), Q_c is changed into Q_{c_1} (resp. Q_{c_2}). Let $X_{c_1} \subseteq X_1$, $Y_{c_1} \subseteq Y_1$, $X_{c_2} \subseteq X_2$, and $Y_{c_2} \subseteq Y_2$ contains all literals requirements of Q_c in φ_1 and φ_2 respectively. If φ_1 and φ_2 are consistent, they should satisfy: (1) Q_{c_1} and Q_{c_2} are isomorphic; (2) $X_{c_1} = X_{c_2}$; and (3) $Y_{c_1} = Y_{c_2}$.

Corollary 2: A set of GRRs Σ is said to be consistent iff any pair of GRRs in Σ are consistent.

Theorem 2: The consistency problem for GRR is NP-hard (lower bound complexity).

Similar to Theorem 1, this theorem also holds since it also involves the problem of subgraph isomorphism according to Lemma 2.

Notice that the consistency of GRRs is a necessary pre-condition for graph repairing. According to Definition 3.3, no graphs can satisfy inconsistent GRRs. Thus, in the rest of our paper, we assume that all given GRRs in Σ are consistent. If a pair of inconsistent GRRs are found in Σ, we can just arbitrarily delete one from this pair of GRR, or consult experts to modify them into consistent ones.

C. Termination

As illustrated in the last paragraph, when discussing the problem of termination, we assume that the given set of textsfGRRs are consistent.

Definition 3.5 (Termination Problem): Given a set of GRRs Σ, Ord is a arbitrary order of GRRs. The graph data is repaired using each GRR in Ord sequentially. We say the graph is repaired once if all GRRs are applied in the order of Ord. After that, the original graph data G is repaired into G_1. Then using each GRR in Ord sequentially to repair G_1, we obtain G_2 ...Recursively repairing the graph using the above approach N times, until $G_N = G_{N-1}$. The termination problem is to determine whether the number of recursion times N is finite. □

We observe that the repairing order results in different repairing results. To conveniently analyze the influence of repairing order and the termination problem, we introduce the definition of *dependent GRRs*.

Definition 3.6 (Dependent GRRs): Two GRRs, $\varphi_1 = (Q_1, X_1) \rightarrow (Q'_1, Y_1)$ and $\varphi_2 = (Q_2, X_2) \rightarrow (Q'_2, Y_2)$, are said to be dependent if the repairing result using an order $\{\varphi_1, \varphi_2\}$ is different from that using an order $\{\varphi_2, \varphi_1\}$. □

For example, consider utilizing two GRRs in Property 1, $\varphi_1 = (Q_3, X_3) \rightarrow (Q'_3, Y)$ and $\varphi_2 = (Q_4, X) \rightarrow (Q'_4, Y)$ to repair the graph G_1 shown in Figure 1(a). If we apply φ_1 before φ_2, since the edge "student x live_in city y" does not exist in G_1, nothing can be repaired. However, if we apply GRR_2 before GRR_1, the edge "student x live_in city y" will be added into G_1 by Q'_3, and G_1 can be further repaired by Q_4 and Q'_4.

For two dependent GRRs, $\varphi_1 = (Q_1, X_1) \rightarrow (Q'_1, Y_1)$ and $\varphi_2 = (Q_2, X_2) \rightarrow (Q'_2, Y_2)$, if (Q_2, X_2) of φ_2 needs to take advantage of the repaired results obtained from $(Q'_1, Y_1$ of φ_1, we say φ_2 depends on φ_1, which is denoted as $\varphi_1 \prec \varphi_2$.

Definition 3.7 (Circular GRR): If $\varphi_1 = (Q_1, X_1) \rightarrow (Q'_1, Y_1)$, $\varphi_2 = (Q'_1, Y_1) \rightarrow (Q_1, X_1)$, then φ_1 and φ_2 are called circular GRRs. □

Theorem 3: If circular GRRs φ_1 and φ_2 both exist in Σ, then Σ is not consistent.

This theorem obviously holds according to Definition 3.3.

Theorem 4: Given a set of GRRs with size $|\Sigma|$, if all GRRs in Σ are consistent (Definition 3.3), the repairing must terminate within $|\Sigma|$ times of recursion no matter what the repairing order is.

Proof: Since all GRRs in Σ are consistent, no circular GRRs exist in Σ. It is proved that graph rewriting without circular cases must terminate [13]. Thus this theorem holds. The worst case of termination problem is that, all the $|\Sigma|$ GRRs in Σ are dependent, and $\varphi_1 \prec \varphi_2 \prec \ldots \prec \varphi_{|\Sigma|}$, but the repairing order $Ord = \{\varphi_{|\Sigma|}, \varphi_{|\Sigma|-1}, \ldots, \varphi_1\}$. In this case, the number of recursion $N = |\Sigma|$. Thus, the repairing must terminate within $|\Sigma|$ times of recursion. ■

From Theorem 4, we can see that whenever we know the Σ is consistent, we can determine the repairing based on Σ must terminate. Thus, the complexity of termination problem equals to that of the consistency problem.

Definition 3.6 infers that different order of applied repairing rules may cause different repairing results. In the following, we discuss what repairing is the best.

Definition 3.8 (Best Repair): Given a graph data G, a set of GRR Σ, when it terminates using a random order Ord of GRR to repair G, we obtain the repaired graph G'. For all possible Ords, if the edit distance [14] between G' and G is maximal (using order Ord_{best}), we say that graph G is best repaired. □

According to Definition 3.6, if rules $\varphi_1, \varphi_2 \in \Sigma$, and $\varphi_1 \prec \varphi_2$, we know that the repairing using φ_2 will take advantage of the repairing result of φ_1. If φ_2 is before φ_1 in Ord, some parts of the graph cannot be fully repaired by φ_2, if these parts need also be repaired by φ_1. In this case, the edit distance of G' and G cannot be maximized. Obviously, if φ_1 and φ_2 are not dependent rules, the order does not affect G'. Thus, the following theorem holds.

Theorem 5: For all dependent rules in Σ, say $\varphi_1 \prec \varphi_2 \prec \ldots \prec \varphi_n$, if Ord satisfies $\{\ldots\varphi_1\ldots\varphi_2\ldots\varphi_n\ldots\}$, then

$Ord = Ord_{best}$. The repaired graph obtained by Ord is best repaired.

IV. Repairing Using GRRs

Now let us consider how to repair the graph data using GRRs. In the rest of our paper, we assume the GRRs in the given set Σ are consistent. When repairing the graph, since the process of each recursion before termination (Definition 3.5) is similar, we only discuss the repairing using GRRs in **one recursion**, both in this section and in the experiments testing the efficiency in Section V-B. Moreover, for simplicity, in the rest of our paper, we let the labels of a vertex be the same with its name, for example, in Fig. 5, vertex a's label $\tau_a = a$. We just use a instead of τ_a to label it.

A. Baseline Method

A naive method is that for each GRR $\varphi = (Q, X) \rightarrow (Q', Y)$ in Σ, we find all matches of Q whose values of labels satisfy literal X, and then repair them according to Q' and Y.

Shortcomings of the baseline method: Since repairing the matches according to Q' and Y is only to add/delete some edges/vertices or change some values of the variables, repairing the given matches is very efficient. However, finding all matches of Q for each GRR is very costly, since it is invovles an isomorphic subgraph enumeration problem, which is NP-complete [15]. We need to conduct algorithms for subgraph isomorphism [16][12] $|\Sigma|$ times, which is a multiple subgraph isomorphism problem. This is the most costly part during the repairing process. However, existing studies mainly focus on subgraph isomorphism for a single query, instead of multiple subgraph isomorphism, which is fully discussed in Section VI. Thus, here, we study how to improve the efficiency of finding all matches of Qs in the given GRR set Σ.

B. Improved Method

Since the size of each GRR is usually very small, we can use a *decomposition-and-join* strategy to circumvent the complex subgraph isomorphic calculation. We can decompose the Qs in GRR into edges, find the matches of each edge in the graph data, and join the matched results to get the final answer. Let $M(e)$ be the average number of matches of all the decomposed edges, $|E_\Sigma|$ be the number of decomposed edges in Σ, according to [17], if Q are all tree structures, the complexity of *decomposition-and-join* strategy is $O(|E_\Sigma|M(e))$; if Q are all triangle-based structures (such as chordal graphs [18]), the complexity is $O(|\Sigma|(M(e)^{|Q|/2}))$. Motivated by the complexity analysis, we find that decompose Qs into trees is more efficient than into edges, due to the reason of fast tree structure joins. Moreover, determining isomorphic trees is logarithmic solvable [19]. Thus, it is advisable to decompose Qs in GRR into distinct tree covers, which is defined as follows.

Definition 4.1 (Disjoint Tree Cover): A disjoint tree cover of $G(V, E)$ is a set of trees (ignoring edge directions) $\mathcal{T} = \{T_1(V_1, E_1), \ T_2(V_2, E_2), \ \ldots, T_i(V_i, E_i), \ \ldots, T_q(V_q, E_q)\}$ such that $\bigcup_{i=1}^{q} E_i = E$ and $\bigcap_{i=1}^{q} E_i = \phi$. □

For example, the set $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$ shown in Fig. 6 is a disjoint tree cover of the graph Q_1 shown in Fig. 5.

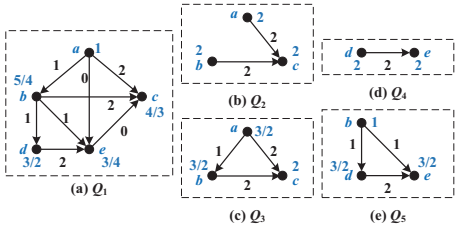


Fig. 5. A set of Patterns before Repairing

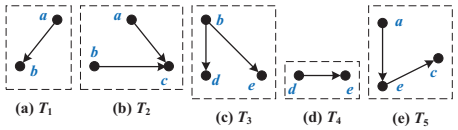


Fig. 6. Example Decomposition of Qs in Figure 5

Our decomposition-and-join strategy is summarized in Algorithm 1. The algorithm uses a set $\mathcal{T}$ to store the disjoint tree cover of the Qs, and use a set Σ_M to store the matches of Qs (will be detailed in Sections IV-B1 and IV-B2). Initially, $\mathcal{T} = \phi$ (Line 1). Firstly, for each Q, we find its disjoint tree cover $\{T_1, T_2, \ldots, T_q\}$ and let $\mathcal{T} := \mathcal{T} \cup \{T_1, T_2, \ldots, T_q\}$ (Lines 2-4). Notice that $\mathcal{T}$ is updated by $\mathcal{T} := \mathcal{T} \cup \{T_1, T_2, \ldots, T_q\}$, which means if a new generated tree has already in $\mathcal{T}$, it will never be put into $\mathcal{T}$ again. This process involves a tree isomorphism problem [19], which means to determine whether the new generated tree T is isomorphic to any tree in $\mathcal{T}$. [19] provides a logarithmic time algorithm to solve it. Then we find matches for each tree T in $\mathcal{T}$ (Lines 5-6). Finally, we join the matched results of the disjoint tree cover of each Q using methods in [20][21] (Lines 7-9), and get the final results (Line 10).

Algorithm 1: Decomposition-and-Join Strategy

```
   Input: Σ, G
   Output: Σ_M
1  T = φ ;
2  for each Q in Σ do
3      Find a disjoint tree over {T_1, T_2, ..., T_q} of Q ;
4      T := T ∪ {T_1, T_2, ..., T_q} ;
5  for each tree T_i ∈ T do
6      Find matches of T_i in G ;
7  for each Q in Σ do
8      Join the matched results of {T_1, T_2, ..., T_q} ;
9      Put the joint results in Σ_M ;
10 Return (Σ_M) ;
```

We illustrate Algorithm 1 with Qs in Fig. 5 and disjoint tree covers in Fig. 6. For the five Qs in Fig. 5, Q_1's disjoint tree cover is $\mathcal{T}_{Q_1} = \{T_1, T_2, T_3, T_4, T_5\}$, where T_1, T_2, T_3, T_4 and T_5 are shown in Figure 6. Similarly, Q_2's disjoint tree cover is $\mathcal{T}_{Q_2} = \{T_2\}$; Q_3's disjoint tree cover is $\mathcal{T}_{Q_3} = \{T_1, T_2\}$; Q_4's disjoint tree cover is $\mathcal{T}_{Q_4} = \{T_4\}$, Q_5's disjoint tree cover is $\mathcal{T}_{Q_5} = \{T_3, T_4\}$. Then $\mathcal{T} = T_{Q_1} \cup T_{Q_2} \cup T_{Q_3} \cup T_{Q_4} \cup T_{Q_5} = \{T_1, T_2, T_3, T_4, T_5\}$. After we find matches for $\mathcal{T}$, we join the matched results of T_1, T_2, T_3, T_4 and T_5 (denoted as $M(T_1) \bowtie M(T_2) \bowtie M(T_3) \bowtie M(T_4) \bowtie M(T_5)$) as the result of Q_1. Similarly, $M(T_2)$ is the result of Q_2; $M(T_2) \bowtie M(T_3)$ is the result of Q_3; $M(T_4)$ is the result of Q_4; and $M(T_3) \bowtie M(T_4)$ is the result of Q_5.

Challenges in the DJ-Method: Compared to decomposing Qs into edges, decomposing Qs into trees may cause repeated scans of the matches of some edges. For example, if Qs in Fig. 5 is decomposed into trees in Fig. 7, the matches of edge $e(b, c)$ are scanned twice during the process of finding the matches of T_2 shown in Fig. 7(b) and T_5 shown in Fig. 7(e). To reduce the repeated scan, the overlap between decomposed trees should be as small as possible. Moreover, according to the complexity analysis in the first paragraph of this subsection, we know that the more the edges linked as tree structures, the more efficient the algorithm will be. This require each decomposed tree should be as large as possible. Thus, to obtain the most efficient algorithm, the decomposition of Qs should be optimized, which is defined as follows.

Definition 4.2 (Optimal Q Decomposition): Given a set of GRRs Σ, the Optimal Q Decomposition (OQD) Problem is to calculate a union set of disjoint tree covers $\mathcal{T} = \bigcup_{Q_i \text{ of } \varphi_i \in \Sigma} \mathcal{T}_{Q_i}$ (with $\mathcal{T}_{Q_i}$ is a disjoint tree cover of Q_i), such that (1) the size of each tree $T \in \mathcal{T}$ is as large as possible, and (2) the overlap among the trees $T \in \mathcal{T}$ is as small as possible. □

Theorem 6: The OQD problem is NP-hard, and it is NP-hard to approximate OQD problem within any constant factor in polynomial time.

The complete proof is detailed in our technique report [22]. In the following two subsections, we provide two heuristic algorithms to solve the OQD problem.

1) Minimum Spanning Tree Based Algorithm: When decomposing each Q, in order to know how many times an edge $e \in Q$ appears in other patterns, we label each edge e with a weight $w_Q(e)$, which is calculated as

$$w_Q(e) = n(e) - n_Q(e) \tag{1}$$

where $n(e)$ is how many times that an edge e appears in all patterns in Σ, and $n_Q(e)$ is how many times that e appears in the current pattern Q.

For example, in Fig. 5, the edge $e(a, c)$ appears 3 times in total, once in Q_1, once in Q_2, and once in Q_3. According to Equation (1), the weight of $e(a, c)$ in Q_1 is $w_{Q_1}(e(a, c)) = 3 - 1 = 2$. Similarly, $w_{Q_2}(e(a, c)) = w_{Q_3}(e(a, c)) = 2$.

Since we want to make the size of each decomposed tree as large as possible, and the overlap among tree as small as possible, we conduct our first heuristic algorithm by recursively calculating the minimum spanning tree (a spanning tree whose total weight on its edges is minimized)on each Q. This heuristic algorithm is named as *Minimum Spanning Tree Based Algorithm* (shorten by *MST-algorithm* in the rest).

The *MST-algorithm* is summarized in Algorithm 2. Firstly, we label each edge in Σ using the weight calculated by Equation 1 (Line 1). Then for each Q of φ in Σ, we find its minimum spanning tree T using method in [23] (Line 6), delete T from Q (Line 7), and let the result set $\mathcal{T} := \mathcal{T} \cup T$ (Line 8). Then conduct the minimum spanning tree algorithm over $Q_d = Q - T$ ...Recursively repeating the above process until Q_d is empty (Lines 5-7). The algorithm terminates until all Qs in Σ are decomposed (Lines 3-8).

For example, Fig. 5 shows 5 Qs and corresponding weights on each edge. For Q_1 in Fig. 5(a), its minimum spanning tree is T_1 shown in Fig. 7(a). After deleting T_1 from Q_1, the minimum spanning trees of $Q_1 - T_1$ is T_6 shown in Fig. 7(f). Thus, the disjoint tree cover of Q_1 is $\{T_1, T_6\}$. For Q_2 in Fig. 5(b), its minimum spanning tree is T_3 ...Similarly, through the MST-algorithm, we can get the disjoint tree covers of Q_3, Q_4 and Q_5 are $\{T_2, T_4\}$, $\{T_5\}$ and $\{T_5, T_7\}$. Thus, the decomposed tree set of Q_1, Q_2, Q_3, Q_4 and Q_5 is $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$.

Algorithm 2: MST-algorithm

Input: Σ
Output: $\mathcal{T}$
1 Label each edge in Σ a weight according to Equation (1) ;
2 $\mathcal{T} = \phi$;
3 **for** *each* Q *in* Σ **do**
4 $Q_d := Q$;
5 **while** $Q_d \neq \phi$ **do**
6 $T :=$ MST (Q_d) ;
7 $Q_d := Q - T$;
8 $\mathcal{T} := \mathcal{T} \cup T$;
9 Return $(\mathcal{T})$;

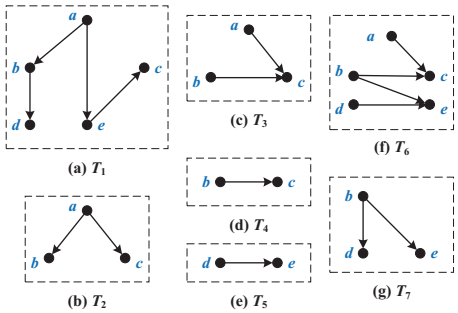


Fig. 7. MST-Algorithm Decomposition Result

2) Minimum Vertex Cover Based Algorithm: Since the size of Graph Repairing Rules (*GRR*s) is usually very small [11], we directly let the decomposed trees has only one layer. In other words, ignoring the direction of edges, the longest path between any pair of vertices in the decomposed trees should not be larger than 2. The small size of *GRR*s ensures such decomposition cannot be too fragmentary compared to the above MST-algorithm. On the other hand, finding matches for the trees with only one layer is very efficient. This lead to our second heuristic algorithm, named *Minimum Vertex Cover Based algorithm* (shorten as *MVC-algorithm* in the rest).

Here, we label each vertex in Q with a weight calculated as follows.

$$w(v) = \frac{\sum_{e_i \in Ne(v)} w(e_i)}{d(v)} \tag{2}$$

where $Ne(v)$ is a set of edges with v as their end vertex, $d(v)$ is the degree of v in Q, and $w(e)$ is calculated by Equation (1).

For example, in Fig. 5(a), the weight of vertex a is calculated as $w(a) = (1+0+2)/3 = 1$. The weight of each vertex is labelled in blue in Fig. 5.

Equation (2) can ensure that the larger degree a vertex has, and the fewer times its neighbor edges appear in other Qs, the smaller weight the vertex is labelled.

This lead to our second heuristic algorithm, named *Minimum Vertex Cover Based algorithm* (shorten as *MVC-algorithm* in the rest).

For each Q, we select a set of vertices such that these vertices (viewed as roots of trees) and their neighbor edges can cover Q and have a minimum total weight. Finding such set of vertices is to calculate the minimum vertex cover problem [24] for each Q. There exists a 2-approximate algorithm for the minimum vertex cover problem [24]. So we can use the algorithm proposed in [24] to approximately calculate the answer to our problem.

Algorithm 3: MVC-algorithm

Input: Σ
Output: $\mathcal{T}$
1 $\mathcal{T} = \phi$;
2 Label each vertex in Q a weight according to Equation (2) ;
3 **for** *each* Q *in* Σ **do**
4 $\quad \mathcal{MT}$:= MVC (Q_d) ;
5 $\quad \mathcal{T} := \mathcal{T} \cup \mathcal{MT}$;
6 Return $(\mathcal{T})$;

The *MVC-algorithm* is summarized in Algorithm 3. Firstly, we label each vertex in Σ using the weight calculated by Equation (2) (Line 2). Then, for each Q of φ in Σ, we decompose Q into trees with only one layer using algorithm solving the minimum vertex cover problem [24]. The decomposed trees are put into a set $\mathcal{MT}$ (Line 4). Let the result set $\mathcal{T} := \mathcal{T} \cup T$ (Line 5). After all Qs are decomposed, we return the results set $\mathcal{T}$ (Line 6).

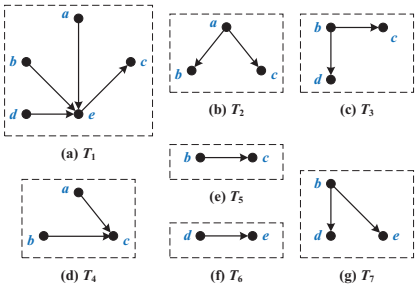


Fig. 8. MVC-Algorithm Decomposition Result

For example, Fig. 8 shows the decomposition result of the Qs in Fig. 5 using the MVC-algorithm. Q_1 in Fig. 5(a) is decomposed into T_1, T_2 and T_3 in Figs. 8(a), 8(b) and 8(c) respectively. Similarly, Q_2 is decomposed into $\{T_4\}$, Q_3 is decomposed into $\{T_2, T_5\}$, Q_4 is decomposed into $\{T_6\}$, Q_5 is decomposed into $\{T_6, T_7\}$. Thus, the decomposed result $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$.

V. Performance Evaluation

In this section, we report our experiment results. We use real-life graphs to test the efficiency and scalability of our algorithms.

A. Experiment Datasets and Environment

Datasets: We conduct our algorithms over 4 real-life graph data, DBpedia[1] (denoted as DP), USPatent[2] (denoted as UP), YAGO2[3] (denoted as YG), and WebGraph[4] (denoted as WG), which are detailed in Table I. In Table I, $|V|$ presents the number of entities (vertices) in the graph; $|E|$ presents the number of relationships between entities (edges) in the graph; $|L|$ presents the attributes (labels) of the vertices and edges.

TABLE I
Real Datasets

Datasets	$\|V\|$ (million)	$\|E\|$ (million)	$\|L\|$	$\|\Sigma\|$
DBPedia (DP)	28	33.4	360	100
USPatent (UP)	3.8	16.5	415	269
YAGO2 (YG)	3.5	7.35	49	1568
WebGraph (WG)	0.875	5.105	97	65

The efficiency of the algorithms is influenced by the size of the graph data, average pattern size of the GRRs, and the size of Σ. Thus, we will test the scalability of our algorithms w.r.t the graph size $|G|$, average pattern size of the GRRs in Σ denoted as $|Q|$, and the size of Σ. These parameters are listed in Tabel II, and the default values are shown in bold. In the scalability experiment, we also use the real-life datasets in Table I, but "cut out" them into different sizes. Namely, for different $|G|$s, we extract induced subgraphs from the real-life graphs with given numbers of vertices. The default value of $|G|$ is the size of each real-life graph data. The corresponding GRRs are also selected from the mined GRRs in each induced real-life graph.

TABLE II
Synthetic Datasets

Graph Size $\|G\|$	64K, 256K, 1M, 4M, 8M, 16M
Pattern Size $\|Q\|$	2, **4**, 6, 8
Rule Set Size $\|\Sigma\|$	**50**, 100, 150, 200

Error Generation and Ground Truth: Since we do not know what a "pure" real-life graph is like, we randomly inject noise into the graph, as suggested by [25]. The types of noise corresponding to the types of GRRs illustrated in Property 1 are as follows. (1) *Attribute Missing*, we delete a neighbor edge and corresponding attribute on this edge of a vertex; (2) *Relationship Missing*, we delete the edge between two entities; (3) *Attribute Resolution*, for an edge $e(v_1, v_2)$, we make a copy $e'(v_1, v_2')$ where e_1 and e_2 have the same attribute, and v_2' is a new inserted vertex containing the same types and values with v_2; (4) *Attribute Value Error*, we change the attribute value into another one; (5) *Entity Resolution*, we copy a vertex together with its corresponding types and values, and connect it with its original neighbor edges. We take these noised graphs as the one to repair, and the original graphs as the ground truth. Totally, we inject $20\% \times |V|$ errors in each graph, where $|V|$ is the number of vertices in a graph dataset. This is because according to Google's statistics, the Google Knowledge Base contains about 20% errors[5]. We take such a percentage to simulate the real. There are 5 types of errors, we uniformly inject each type of errors into the graph, which means each is about 20% of the total errors.

GRR Generation: We use the rule-mining method in [26] to help to obtain the GRRs. We run their method

[1]http://wiki.dbpedia.org/datasets
[2]http://www.uspto.gov/learning-and-resources/electronic-data-products/data
[3]http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/
[4]http://law.di.unimi.it/datasets.php
[5]https://searchengineland.com/googles-knowledge-graph-errors-126098

on the noised real-life graphs above, and use the original graph as the ground truth. The rule-mining method in [26] can check the functional dependencies between entities and attributes, and express them in a logic sentence form. Using the method introduced in [8], we can change the logic form into graph pattern form Q. Notice that [26] only provide an implication relationship, which has a different form from our GRR. If the mined logic rule implies to add new attribute to entities, we transform it into Type (1) of Property 1. If the mined logic rule implies to add new relations, we transform it into Type (2) of Property 1. If the mined logic rule implies entities' relationship conflicts, we transform it into Type (3) of Property 1. If the mined logic rule implies error values of attributes, we transform it into Type (4) of Property 1. The Type (5) of Property 1, "Entity Resolution", cannot be generated by [26]. To get this type of rules, we extract the randomly added vertices and their neighbor vertices and edges (in the process of making noise) as pattern Q.

Experiment Environment: Our experiments are conducted on a server with 64GB Memory, 8TB hard disk, and 4 6-core Intel(R) Xeon(R) CPU X5675 @3.07GHz, with Linux system. The development language is C++ and its standard template library (STL).

Algorithms: We compare 3 algorithms in our experiments. (1) The first one is the baseline algorithm (detailed in Section IV-A), which is denoted as *Base* in our experiments. (2) The second one is using the decomposition-and-join strategy with MST-algorithm decomposing the GRRs (detailed in Section IV-B1), which is denoted as *DJ-MST*. (3) The third one is using the decomposition-and-join strategy with MVC-algorithm decomposing the GRRs (detailed in Section IV-B2), which is denoted as *DJ-MVC*.

B. Experiment Results

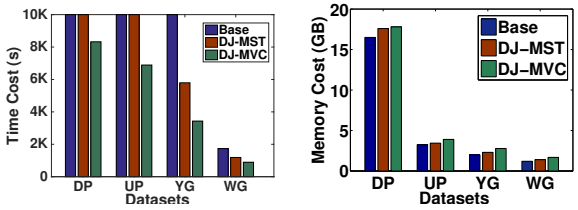
Exp-1: Effectiveness: In this experiment, we show the repairing effectiveness of our semantic. We conduct this experiment over the two largest datasets, DBPedia and UsPatent. The results of the other two datasets can be found in our technique report [22]. Denoting the set of created errors as Er, and the set of corrected errors as Er', we calculate the precision ($\frac{|Er \cap Er'|}{|Er'|}$) and recall ($\frac{|Er \cap Er'|}{|Er|}$) of GRRs.

Moreover, there are two related studies focusing rules in graphs. One is about the functional dependencies for graphs (GFDs) proposed by [9]. The other one is about the association rules with graph patterns (ARGPs) proposed by [10]. We compare the repairing quality using our GRR semantic with the GFD semantic and the ARGP semantic. The results are shown in Table III.

TABLE III
ACCURACY OF GRR AND GFD SEMANTIC

	DBPedia		UsPatent	
Semantic	Precision	Recall	Precision	Recall
GRR	100%	90.3%	100%	84.7%
GFD	100%	76.5%	100%	69.9%
ARGP	100%	10.2%	100%	14.1%

From Table III, we can see that the precision of all the semantic is 100%, this means that if the rules are right, our repairing would be right. The reason why the recall of our GRR semantic is not 100% is that the noise is randomly injected, and the selected vertices or edges happen to be not covered by any GRR in Σ. For example, if the GRRs in Σ do not contain such rules of correcting values of vertex v_1, and the injecting error happens to change the value of vertex v_1. No matter what value v_1 is changed, it would never be repaired.



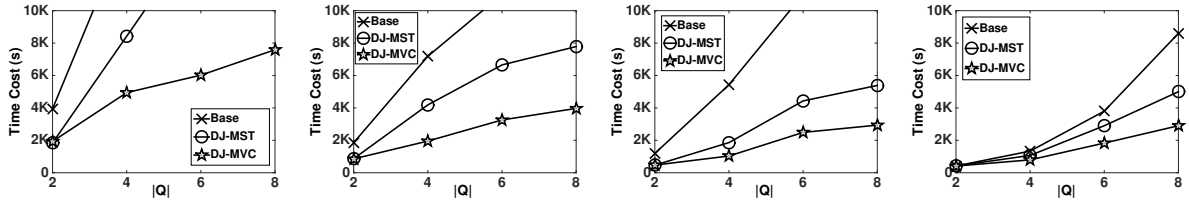
(a) Time Cost (b) Memory Cost

Fig. 9. Real-life Graph Repairing

Compared to the other two semantics, the recall of our GRR semantic is higher than the other two. This is because in GFD semantic, the rules are defined as $\varphi_1 = (Q[\overline{x}], X \rightarrow Y)$, where $Q[\overline{x}]$ is a graph pattern, and X and Y are two sets of literals of $\overline{x}$. Since the GFD semantic does not support the structure change, directly using GFD cannot repair the graph data in Type (1) and (2) of Property 1. In ARGP semantic, the rules are defined as $Q(x, y) \Rightarrow q(x, y)$. Here, $Q(x, y)$ is a graph pattern in which x and y are two designated nodes, and $q(x, y)$ is an edge labelled q from x to y. This semantic is designed specially for detecting more evidence in graphs, and cannot find conflicts and redundancies in the data. Using this semantic can only repair the data in Type (1) of Property 1. Furthermore, by comparing the difference between the corrected sets Er's, we find that all the errors corrected by GFD and ARGP semantic are also repaired by our GRR semantic.

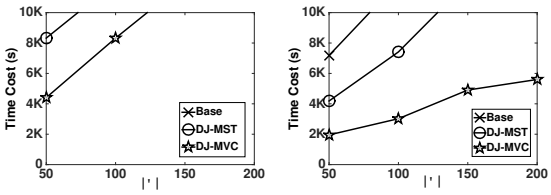
Exp-2: Real-life Graph Repairing: In this experiment, we use all the GRRs mined from each graph using method [26] to repair each real-life graph, and test the time cost and memory cost. The results are shown in Fig. 9.

From Fig. 9(a), we can see in the YAGO2 dataset (YG), the time cost of baseline algorithm is larger than 10,000s, while the DJ-MST algorithm takes 5795.7s and the DJ-MVC algorithm only takes 3432.4s. Notice that in the DBPedia dataset (DP) and UsPatent dataset (UP), the time cost of both the baseline algorithm and the DJ-MST algorithm is more than 10,000 seconds, and we do not show the exact value in the figure. Even though, there is no obstacle to show the efficiency of our DJ-MVC algorithm. We observe that the repairing of WebGraph (WG) and YG is much more efficient than that of DP and UP. The repairing of WG is efficient since its size is much smaller than the other three. Although the size of YG and UP is almost the same, the repairing of YG is more efficient since GPRs mined from YG are fewer than those from UP. The time cost of repairing DP is the longest since it has a extremely large size and its applied rules are much more than the other three. We will further discuss how such parameters ($|Q|$, $|G|$, and $|\Sigma|$) influence the efficiency of the repairing algorithms in our scalability experiments.



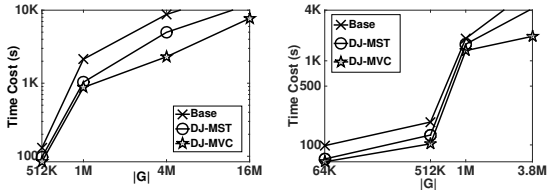
(a) DBPedia (b) UsPatent (c) YAGO2 (d) WebGraph

Fig. 10. Scalability w.r.t $|Q|$



(a) DBPedia (b) UsPatent

Fig. 11. Scalability w.r.t $|\Sigma|$



(a) DBPedia (b) UsPatent

Fig. 12. Scalability w.r.t $|G|$

From Fig. 9(b), we can see that the memory cost of the three algorithms are almost the same. The memory cost of our two decomposition-and-join strategy algorithms is a little larger than that of the baseline algorithm. This is reasonable since our decomposition-and-join strategy need more space to store the intermediate match results and conduct the join process. The storage of intermediate results takes about hundreds of MBs, which is much smaller compared to the size of graph data (it takes 1-10 GB to store). We observe that the changing trend of memory cost of the three algorithms are almost the same in all our experiment results, so in the rest, we do not show the memory cost comparison in the scalability experiments.

Exp-2: Scalability w.r.t $|Q|$*:* In this experiment, we test the time cost of the three graph repairing algorithms w.r.t the average pattern size $|Q|$ of each GRR. We conduct this experiment over 4 real-life graphs. We set the number of GRRs $|\Sigma| = 50$, and vary $|Q|$ from 2 to 8. The results are shown in Fig. 10.

We can see that in each dataset, the time cost of all the three algorithms increases with the increase of $|Q|$. However, the time cost of the baseline algorithm increases exponentially with the increase of $|Q|$, while the time cost of our two decomposition-and-join strategy algorithms increases nearly linearly. We also find that the increase of DJ-MST is faster than that of DJ-MVC. This is reasonable since finding matches for the tree components in the DJ-MST is much faster than finding matches for graph patterns in the baseline algorithm. Finding matches for the 2-hop trees in the DJ-MVC algorithm is even polynomial-time solvable. Thus, with the increase of $|Q|$, the increase of our decomposition-and-join strategy algorithms is nearly linear. Moreover, from the results in the 4 real-lifes graph, we can observe that when the graph size is large, even though the intermediate join results of DJ-MVC is larger than those of DJ-MST, the time cost of DJ-MVC is still smaller.

Exp-3: Scalability w.r.t $|\Sigma|$*:* In this experiment, we test the time cost of the three graph repairing algorithms w.r.t the number of GRR, $|\Sigma|$. We conduct this experiment over 2 real-life graphs, DBPedia (DP) and USPatent (UP), since the number of GRRs in these two graphs is large enough. We extract 200 GRRs with average size $|Q| = 4$ from all the mined rules in these graphs. We vary $|\Sigma|$ from 50 to 200, and record the repairing time of each graph. The results are shown in Fig. 11.

In Fig. 11(a), all the time cost of the baseline algorithm is larger than 10,000s, so it is not shown. From Fig. 11, our DJ-MVC algorithm is nearly 100 times faster than the baseline algorithm on DP, while on UP, our DJ-MVC algorithm is still more than 10 times faster. As well, we can see that the time cost of the three algorithms increases linearly with the increase of $|\Sigma|$. The increase of the baseline algorithm is faster than the two DJ-methods. It is reasonable since the baseline algorithm does not detect the repeated edges in Σ, while the scan of such repeated edges is largely reduced in our decomposition-and-join strategy. Furthermore, the increase of the time cost of DJ-MST algorithm is faster than that of the DJ-MVC algorithm. This is reasonable since the complexity of finding matches for each decomposed component in DJ-MTS algorithm is much higher than that in DJ-MVC algorithm.

Exp-4: Scalability w.r.t $|G|$*:* In this experiment, we test the time cost of the three graph repairing algorithms w.r.t the size of graph datasets $|G|$. We also conduct this experiment over DP and UP, the two largest graphs. We extract induced subgraphs from the original graph with number of vertices varies from 512K to 16M in DP, and 64K to 3.8M in UP respectively. The size of $|Q|$ and the number of GRRs $|\Sigma|$ are set as default values, $|Q| = 4$ and $|\Sigma| = 50$. The results are shown in Fig. 12.

From Fig. 12, we can see that the time cost of the three algorithms increases with the increase of $|G|$. The increase of the baseline algorithm is faster than the other two algorithms. It is reasonable since the larger the graph data is, the more time is spent by the repeated scan in the baseline algorithm. The larger the graph is, the increase will be more obviously. Moreover, in Fig. 12(b), when the graph size $|G| = 64$K, the time cost of DJ-MST (63s) is smaller than that of DJ-MVC (68s). This is because the number of intermediate results of DJ-MVC is

larger than those in DJ-MST. The time spent by join process is larger than that of fining matches. This illustrates that when the graph size is small, the time cost of finding matches is also small, and the join cost will play a more important role. However, when the graph size is large, finding matches is still the main costly process in the algorithms.

VI. Related Work

In this section, we summarize the existing work that is related to our paper, which are classified into 3 parts, relational and XML data repairing, graph data repairing, and subgraph isomorphic algorithms.

Relational and XML Data Repairing: Dependable data repairing has been widely studied in the field of relational databases and XML databases [27], and [28] is a good survey. The dependable rules can help detect conflicts and errors in the databases. Thus, serious studies propose semantics and algorithms to repair the relational databases based on dependable rules, such as FDs [29], CFDs[27], denial constraints [30], and fixing rules [7][31].

These semantic and algorithms cannot be applied into dependable graph data repairing since graph data has a complex topological structure. Although such topological structures can be equivalently transformed into relational databases, such that repairing semantics and methods can be applied to repair graph databases. However, such transformation involves in massive junction operations [32]. Its semantic expression is complex and the corresponding repairing is very inefficient. More detailed discussion can be found in our technique report [22].

Graph Data Repairing: In the field of graph repairing, [33] propose a vertex label repairing based on constraints on neighbour vertices. Different from this paper, our work can handle a more general rule-based graph repairing semantic besides vertex label repairing. Extensive studies focus on extending FDs and CFD into RDF environment [34][35]. They define the FDs and CFD based on RDF triples. Different from these existing works, our repairing semantic can be used in general graphs, not only in the RDF environments, and can support more general topological structures.

Few studies exist in general graph data repairing using dependable rules. One closer work is the association rules with graph patterns (ARGP) in [10]. In this semantic, rules are defined as $Q(x, y) \Rightarrow q(x, y)$. Here, $Q(x, y)$ is a graph pattern in which x and y are two designated nodes, and $q(x, y)$ is an edge labelled q from x to y. It has a physical meaning that if a graph satisfies pattern $Q(x, y)$, add an edge labelled q from x to y. This semantic is designed specially for detecting more evidence in graphs, and cannot find conflicts and redundancies in graph data. This conclusion is also verified in our Exp-1 in Section V. Another closer work is the functional dependency for graphs (GFD) semantic proposed by W. Fan et al. in [9]. In GFD semantic, rules are defined as $\varphi = (Q[\overline{x}], X \rightarrow Y)$, where $Q[\overline{x}]$ is a graph pattern, and X and Y are two sets of literals of $\overline{x}$. Their GFD semantic performs well in detecting the conflicts and redundancies in graphs. Their experiment results show that their GFD semantic performs better than those designed for RDF environments. However, since it is not designed for graph repairing, it failed to provide efficient automatic-repairing methods especially on attribute and value supplement. Since the GFD semantic does not support the structure change (like our GRR from Q to Q'), it cannot supply the incompleteness of the graph data. This conclusion is also verified in our Exp-1 in Section V. Moreover, after proposing the GFD semantic, the authors focus on distributed algorithms to detect conflicts in graph data using GFDs. They discuss the computational cost and communication cost, and provide algorithms on how to partition the graph data and to reduce the communication cost. However, in our work, we discuss how to repair the graphs using our GRRs. By analyzing the characters of GRRs, we find the most costly process in graph repairing is the repeated scans during the process of finding matches of repairing patterns. Consequently, we make effort on designing algorithms that can reduce such repeated scans. Thus, the main techniques between our work and [9] is also different.

Subgraph Isomorphic Algorithms: Our repairing method related to the field of multiple subgraph isomorphic search over graph databases. As a fundamental query in graph databases, subgraph isomorphic query has been studies for years, and [36] is a good survey, which introduced most popular subgraph isomorphic algorithms and compared their efficiency using extensive experiments. Besides, focus on large graph environments, [37] and [21] propose improved algorithms that can efficiently propose a subgraph isomorphic query in a large graph. However, these studies all focus on a single subgraph isomorphic query over a single or multiple graphs (or graph transactions). They aim at how to improve the efficiency of a single query, while our work focuses more on how to reduce the repeated search among multiple queries. Besides, [38] studied the multiple query optimization problem, but it is designed for relational databases, and is hard to extend to graph databases due to the complex topological character of graphs. Furthermore, [39] propose methods for multiple query optimization for SPARQL. Our work has two differences with [39]. Firstly, [39] designed for RDF graphs while our work designed for general graph. Secondly, the optimization strategy is different. The optimization of [39] is based on grouping the queries based on common subgraphs of given queries, while our optimization is based on decomposing the queries such that the decomposed parts shares as little as possible. Grouping the queries based on common subgraphs in [39] is still NP-hard but the authors provided no theoretical analysis for approximation. While in our work, we prove that the optimization of multiple queries cannot be approximate in polynomial time, and our decomposition approximation methods are all polynomial algorithms.

VII. Conclusion

In this paper, we design an automatic repairing semantic for graphs (called GRRs), which can capture the incompleteness, conflicts, and redundancies in the graphs and indicate how to correct these errors. We study three fundamental problems associated with GRRs, implication, consistency, and termination. When repairing the graph data, to reduce the repeated scan during the process of finding matches of GRRs, we propose a decomposition-and-join strategy. It is conducted

through decomposing the GRRs, find matches for each decomposed part, and finally join matched results. Due to tradeoff between the repeated scan cost and the join cost, we focus on the problem of optimal GRRs decomposition. We provide two heuristic algorithms to solve this problem. Extensive experiments show that our GRR semantic and corresponding repairing algorithms can effectively and efficiently repair the real-life graph data.

ACKNOWLEDGMENT

Lei Chen is supported by the Hong Kong RGC GRF Project 16214716 , National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Webank Collaboration Research Project, and Microsoft Research Asia Collaborative Research Grant. Ye Yuan is supported by the NSFC (Grant No. 61572119 and 61622202) and the Fundamental Research Funds for the Central Universities (Grant No. N150402005). Guoren Wang is supported by the NSFC (Grant No. U1401256, 61732003 and 61729201). Guoren Wang is the corresponding author of this paper.

REFERENCES

[1] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum, "Naga: Searching and ranking knowledge," in *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008, pp. 953–962.

[2] S. Yang, Y. Wu, H. Sun, and X. Yan, "Schemaless and structureless graph querying," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 565–576, 2014.

[3] C. C. Aggarwal, H. Wang *et al.*, *Managing and mining graph data*. Springer, 2010, vol. 40.

[4] S. Maere, K. Heymans, and M. Kuiper, "Bingo: a cytoscape plugin to assess overrepresentation of gene ontology categories in biological networks," *Bioinformatics*, vol. 21, no. 16, pp. 3448–3449, 2005.

[5] L. Bertossi, S. Kolahi, and L. V. S. Lakshmanan, "Data cleaning and query answering with matching dependencies and matching functions," *Theory of Computing Systems*, vol. 52, no. 3, pp. 441–482, 2010.

[6] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification." in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, 2005, pp. 143–154.

[7] J. Wang and N. Tang, "Towards dependable data repairing with fixing rules," in *ACM SIGMOD International Conference on Management of Data*, 2014.

[8] D. V. Kalashnikov and S. Mehrotra, "Domain-independent data cleaning via analysis of entity-relationship graph," *Acm Transactions on Database Systems*, vol. 31, no. 2, pp. 716–767, 2010.

[9] W. Fan, Y. Wu, and J. Xu, "Functional dependencies for graphs," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, 2016, pp. 1843–1857.

[10] W. Fan, X. Wang, Y. Wu, and J. Xu, "Association rules with graph patterns," *Proceedings of the Vldb Endowment*, vol. 8, no. 12, pp. 1502–1513, 2015.

[11] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente, "An empirical study of real-world sparql queries," *arXiv preprint arXiv:1103.5043*, 2011.

[12] C. De La Higuera, J.-C. Janodet, É. Samuel, G. Damiand, and C. Solnon, "Polynomial algorithms for open plane graph and subgraph isomorphisms," *Theoretical Computer Science*, vol. 498, pp. 76–99, 2013.

[13] D. Plump, "Termination of graph rewriting is undecidable," *Fundamenta Informaticae*, vol. 33, no. 2, pp. 201–209, 1998.

[14] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and applications*, vol. 13, no. 1, pp. 113–129, 2010.

[15] I. Wegener, *Complexity theory: exploring the limits of efficient algorithms*. Springer Science & Business Media, 2005.

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

[17] H. Q. Ngo, C. Ré, and A. Rudra, "Skew strikes back: New developments in the theory of join algorithms," *ACM SIGMOD Record*, vol. 42, no. 4, pp. 5–16, 2014.

[18] A. Berry, M. C. Golumbic, and M. Lipshteyn, "Recognizing chordal probe graphs and cycle-bicolorable graphs," *SIAM Journal on Discrete Mathematics*, vol. 21, no. 3, pp. 573–591, 2007.

[19] S. R. Buss, "Alogtime algorithms for tree isomorphism, comparison, and canonization," in *Kurt Gödel Colloquium on Computational Logic and Proof Theory*. Springer, 1997, pp. 18–33.

[20] G. Zhu, X. Lin, K. Zhu, W. Zhang, and J. X. Yu, "Treespan: efficiently computing similarity all-matching," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 529–540.

[21] 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, pp. 788–799, 2012.

[22] [Online]. Available: http://lccpu3.cse.ust.hk/cyr/techReport.pdf

[23] R. L. Graham and P. Hell, "On the history of the minimum spanning tree problem," *Annals of the History of Computing*, vol. 7, no. 1, pp. 43–57, 1985.

[24] I. Dinur and S. Safra, "On the hardness of approximating minimum vertex cover," *Annals of mathematics*, pp. 439–485, 2005.

[25] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, and J. Lehmann, "User-driven quality evaluation of dbpedia," in *Proceedings of the 9th International Conference on Semantic Systems*. ACM, 2013, pp. 97–104.

[26] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek, "Amie: association rule mining under incomplete evidence in ontological knowledge bases," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 413–422.

[27] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 2, p. 6, 2008.

[28] W. Fan, "Dependencies revisited for improving data quality," in *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2008, pp. 159–170.

[29] G. Beskales, I. F. Ilyas, and L. Golab, "Sampling the repairs of functional dependency violations under hard constraints," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 197–207, 2010.

[30] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Put violations into context," in *In ICDE*. Citeseer, 2013.

[31] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *The VLDB Journal*, vol. 3, no. 2, pp. 173–184, 2012.

[32] K. Zhao and J. X. Yu, "Graph processing in rdbmss," 2017.

[33] S. Song, H. Cheng, J. X. Yu, and L. Chen, "Repairing vertex labels under neighborhood constraints," *Proceedings of the Vldb Endowment*, vol. 7, no. 11, pp. 987–998, 2014.

[34] D. Calvanese, W. Fischl, R. Pichler, E. Sallinger, and M. Simkus, "Capturing relational schemas and functional dependencies in rdfs." in *AAAI*, 2014, pp. 1003–1011.

[35] B. He, L. Zou, and D. Zhao, "Using conditional functional dependency to discover abnormal data in rdf graphs," in *Proceedings of Semantic Web Information Management on Semantic Web Information Management*. ACM, 2014, pp. 1–7.

[36] J. Lee, W. S. Han, R. Kasperovics, and J. H. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *Proceedings of the Vldb Endowment*, vol. 6, no. 2, pp. 133–144, 2012.

[37] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *Proceedings of the Vldb Endowment*, vol. 8, no. 5, pp. 617–628, 2015.

[38] T. K. Sellis, "Multiple-query optimization," *Acm Transactions on Database Systems*, vol. 13, no. 1, pp. 23–52, 1988.

[39] W. Le, A. Kementsietsidis, S. Duan, and F. Li, "Scalable multi-query optimization for sparql," vol. 41, no. 4, pp. 666–677, 2012.

[40] D. S. Hochba, "Approximation algorithms for np-hard problems," *ACM SIGACT News*, vol. 28, no. 2, pp. 40–52, 1997.

[41] H. He and A. K. Singh, "Query language and access methods for graph databases," in *Managing and mining graph data*. Springer, 2010, pp. 125–160.