The VLDB Journal (2017) 26:611–635
DOI 10.1007/s00778-017-0466-5

REGULAR PAPER

Graph repairing under neighborhood constraints

Shaoxu Song[1] · **Boge Liu**[1] · **Hong Cheng**[2] · **Jeffrey Xu Yu**[2] · **Lei Chen**[3]

Received: 20 January 2016 / Revised: 3 January 2017 / Accepted: 14 May 2017 / Published online: 23 May 2017

Abstract A broad class of data, ranging from similarity networks, workflow networks to protein networks, can be modeled as graphs with data values as vertex labels. Both vertex labels and neighbors could be dirty for various reasons such as typos or erroneous reporting of results in scientific experiments. *Neighborhood constraints*, specifying label pairs that are allowed to appear on adjacent vertices in the graph, are employed to detect and repair erroneous vertex labels and neighbors. In this paper, we study the problem of repairing vertex labels and neighbors to make graphs satisfy neighborhood constraints. Unfortunately, the problem is generally hard, which motivates us to devise approximation methods for repairing and identify interesting special cases (star and clique constraints) that can be efficiently solved. First, we propose several label repairing approximation algorithms including greedy heuristics, contraction method and an approach combining both. The performances of algorithms are also analyzed for the special case. Moreover, we devise a cubic-time constant-factor graph repairing algorithm with both label and neighbor repairs (given degree-bounded instance graphs). Our extensive experimental evaluation on real data demonstrates the effectiveness of eliminating frauds in several types of application networks.

Keywords Data repairing · Data quality · Graph data cleaning

✉ Shaoxu Song
sxsong@tsinghua.edu.cn

Hong Cheng
hcheng@se.cuhk.edu.hk

Jeffrey Xu Yu
yu@se.cuhk.edu.hk

Lei Chen
leichen@cse.ust.hk

[1] Key Laboratory for Information System Security, Ministry of Education, TNList, School of Software, Tsinghua University, Beijing, China

[2] Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Shatin, N. T., Hong Kong

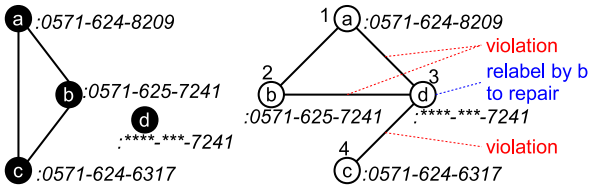
[3] Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong

1 Introduction

Graph-structured data are prevalent such as similarity networks, workflow networks or protein networks. Owing to the existence of errors, a graph may violate certain neighborhood constraints (also expressed as a graph, see motivation examples below). This paper studies a problem of repairing vertex labels and neighbors in a graph to make it satisfy certain neighborhood constraints on labels.

1.1 Motivation Examples

The general idea of data repairing is to suggest possible repairs of a tuple t by other tuples with (equality or similarity) relationships to t. Instead of the rigid equality in conventional *functional dependencies* (FDs), the similarity/distance relationships between tuples enable the tolerance of small variations, e.g., "Street" and its abbreviation "St.". Such relationships could be captured by *metric functional dependencies* [23] with metric introduced in the right-hand side of the dependency, *matching dependency* [12] with metric in the left-hand side, or *differential dependencies* (DDs) [27] in both sides. Without such tolerance of variations, the number of tuples with (strict equality) relationships to a being



ID	Address	City	Tel
1	No.721, West Lake St.	HZ	0571-624-8209
2	No.735, West Lake St.	HZ	0571-625-7241
3	No.735, West Lake Street	HZ	****-***-7241
4	640, West Lake Street	HZ	0571-624-6317

(c)

Fig. 1 Label repairing on similarity network. **a** Constraint, **b** instance, **c** relation.

repaired tuple t is quite limited, and thus the equality-based repairing methods (such as FD-based [3]) often fail to suggest possible repairs.

Example 1 (Similarity Networks) Consider a relation instance in Fig. 1c which collects customer information from various sources. A DD (Address, City $\rightarrow$ Tel, $\langle[0, 6], [0, 0], [0, 5]\rangle$) states that if two tuples have similar Address values (i.e., with distance[1] in the range of $[0, 6]$) and the same City values (with distance in $[0, 0]$), they should have similar Tel values as well (distance within $[0, 5]$) by sharing the same area code and another two digits for the same region. For instance, tuples 1 and 2 of customers in the same street (of the same city) have similar Tel numbers by sharing the same prefix "0571-62". Such DD rules can either be specified by domain experts or discovered from data [28].

We model the similarity relationships by graphs as follows. First, in Fig. 1a, a constraint graph is constructed to represent the similarity/distance requirements on Tel (right-hand-side attribute in the DD) values, where each node denotes a distinct value of Tel domain in the relation. We put an edge between two nodes if their distance is within $[0, 5]$ specified by the DD. Next, an instance graph is built according to the similarities on Address and City (left-hand-side attributes in the DD) of tuples as shown Fig. 1b. Each vertex corresponds to a tuple in the relation, with its Tel value as the label, while an edge indicates that the Address and City values of these two tuples have distance within $[0, 6]$ and $[0, 0]$, respectively.

To validate whether a relation satisfies the DD, it is equivalent to investigate whether the corresponding instance graph satisfies the constraint graph. That is, for each tuple pair with similar Address and same City (having an edge in the instance graph), their Tel values must be similar as well (the pair of labels belong to an edge in the constraint). Since the

[1] e.g., Edit distance (see [25] for a survey of string similarity).

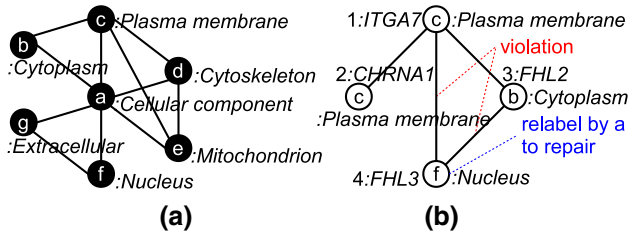


Fig. 2 Example of protein interaction networks. **a** Star constraint on GO terms, **b** instance of protein networks

data are collected from sources with various representation formats and dirty information, violations exist. For instance, suppose that several digits of Tel are lost/hidden in tuple 3. The distance on Address of tuples 3 and 2 (equal to 4 within $[0, 6]$) and their equal City values (distance 0 in $[0, 0]$) indicate an edge between vertices 3 and 2 in Fig. 1b. However, their Tel numbers are not similar (with distance 7 not in $[0, 5]$), i.e., the labels of vertices 3 and 2 denoted by d and b, respectively, are not an edge in Fig. 1a.

Consequently, the data repairing is to relabel vertex/tuple 3 by b:0571-625-7241, since it is the Tel value (label) most similar to the observed d that can satisfy the constraint (see more discussion on repairing cost in Sect. 3.2). Given an FD (Address, City $\rightarrow$ Tel), the existing equality-based repairing method [3] obviously cannot suggest such a repair, since tuple 3 does not have any other tuple with equality relationships on Address in Fig. 1c. That is, no repair candidates can be suggested w.r.t. the FD. □

Indeed, inaccurate values might not only appear in labels (Tel), but also in Address values, which leads to errors in neighborhood in the instance graph (see motivation example in Example 10). In addition to label repairing, we perform neighbor repairing to eliminate violations.

While the similarity network captured from DD is a possible scenario, the proposed graph repairing is applicable to other real-world networks, for instance, the protein interaction networks as illustrated in Fig. 2 in Example 2, or the coauthor networks in Fig. 15 in Sect. 8.1.2. In protein networks, each protein corresponds to a vertex. Edges denote the binary protein–protein interactions. Protein's gene ontology (GO) term is used as the vertex label. Repairing could be applied to correct the faulty GO annotations (see details in Example 2). Moreover, workflow networks could also be naturally represented as constraint graph (denoting workflow specification) and instance graph (i.e., workflow execution). It is to repair the errors occurring in workflow execution which violate the workflow specification (see details in Example 2 in [34]).

The constraint graphs can either be constructed by knowledge (such as the similarity network built by DD in Sect. 8.1.1), obtained from reliable sources (e.g., the coauthor network from DBLP in Sect. 8.1.2) or built by entity

resolution methods over graphs (see discussion in Sect. 8.1.2 as well).

1.2 Contributions

Our major contributions in this are summarized as:

1. We investigate the complexity of the vertex label repairing (relabeling) problem (in Sect. 3.3). The relabeling problem is proved to be NP-complete (Proposition 1). For the special case of star constraints, it can be approximated within a constant factor (Proposition 3), while the clique constraints case can be solved in PTIME.
2. We study greedy heuristics for relabeling (in Sect. 4). The greedy method may fail, since relabeling a vertex to eliminate some violations could introduce new violations to other vertices. Nevertheless, we illustrate that the greedy method always terminates in the special case of star constraints (Proposition 4).
3. We devise a contraction method which guarantees termination (in Sect. 5). The contraction operation requires the contracted vertices to have the same label in order to stop violation spread. We prove that the total number of contraction operations is bounded (Proposition 5), while the contraction results may be arbitrarily bad in terms of relabeling cost due to enforcing the same labels. For the special case of star constraint, the contraction method appears to be a factor-2 approximation (Proposition 6).
4. We present an approach (AlterGC) combining the advantages of greedy and contraction techniques (in Sect. 6). Referring to non-termination of greedy method and possibly bad results of contraction, the AlterGC approach conducts the high cost contraction only when no further greedy relabeling can be applied.
5. We introduce the graph repairing problem, with the consideration of both label and neighbor repairs (in Sect. 7). It is not surprising that the graph repairing problem is still NP-hard (Theorem 9). We thus devise an approximation algorithm that runs in cubic time and is a constant-factor approximation (Proposition 13) given a degree-bounded instance graph.
6. We report an extensive experimental evaluation for the proposed methods on real datasets (in Sect. 8). The experiments verify major theoretical results including termination, approximation bound, time performance and relabeling accuracy of frauds. In particular, the f-measure accuracy of repairing with DD in similarity network is significantly higher than that of existing methods with FD. The AlterGC approach performs very well in practice, with high effectiveness and efficiency. Finally, with both label and neighbor errors, we show that the proposed Grepair algorithm shows both higher repair accuracy and better time performance.

Table 1 Notations

Symbol	Description
$\mathcal{S}(L, N)$	Constraint graph $\mathcal{S}$ with label set L, neighborhood N
$\mathcal{G}(V, E, \lambda)$	Instance graph $\mathcal{G}$ with vertex set V, edge set E, vertex labeling λ
λ	Labeling of vertex
$\asymp$	Labels match constraint
$\models$	Graph satisfies constraint
δ_l	Cost of relabeling a vertex
Δ_l	Cost of relabeling a graph in Eq. 1
δ_n	Cost of neighbor repair in Eq. 11
Δ	Cost of graph repair in Eq. 12
θ	Weight of label and neighbor repairing costs in Eq. 12
$T(v, \ell)$	Set of vertices with violations to vertex v having $\lambda(v) = \ell$ in Eq. 2
R	A node of contracted vertices
$V(\mathsf{R})$	Set of all vertices in R in Eq. 5

Table 1 lists the frequently used notations. A preliminary version of this paper appears in [30]. Compared to the preliminary conference version, the major differences include: (1) We present the proofs of major theoretical results, including Proposition 1, Theorem 2, Proposition 3, Proposition 4 and Proposition 6, for the vertex label repairing (relabeling) problem. (2) We add a completely new section, i.e., Sect. 7, to consider the graph repairing with both vertex label and neighbor repairs. (3) We add a new Sect. 8.2, to report the new experiments on evaluating graph repairs, where both label and neighbor errors are considered.

2 Related Work

The constraint satisfaction problem (CSP) [24] finds an assignment of values to variables (analogous to labels and vertices, respectively) such that certain constraint is satisfied. The problem of assigning values in CSP is different from our studied problem of graph repairing. CSP with binary constraints finds a bijection (assignment) from instance graph to constraint graph. In our graph repairing problem, however, vertices in the instance graph have already been assigned with labels. We target on repairing some vertices (on labels or neighbors) to eliminate violations to the constraint graph. Therefore, techniques for CSP assignment are not directly applicable to graph repairing.

2.1 Constraints on Graphs

Besides the constraint graph studied in this work, pattern graphs can also be considered as more complex constraints,

e.g., by associating comparison operator on attributes of vertices [13] or extending edges to reachability constraints of paths [7]. Such patterns are useful in graph similarity matching [38,39]. Fan et al. [11] propose keys for graphs aiming to uniquely identify entities represented by vertices in a graph. Again, graph patterns are employed to recursively define the keys. The major difference between graph pattern matching and our graph repairing problem is about the satisfaction. Informally, a match of the pattern graph is a subgraph of the instance graph $\mathcal{G}$ such that each edge in the pattern exactly corresponds to an edge in the subgraph [14], i.e., it is not required that each edge in $\mathcal{G}$ could be mapped to the pattern graph. In contrast, referring to the definition in Sect. 3.1, the satisfaction in our study means that two vertices of each edge in the instance graph $\mathcal{G}$ should have the same label or two labels appearing as an edge in the constraint graph $\mathcal{S}$.

A document type definition (DTD) is often employed to define the legal structure of XML document by a list of legal elements and attributes. When illegal structures are observed, XML documents need to be modified in order to make them valid w.r.t. a given DTD [4,31]. It is to compute the minimal tree or graph editing between the XML document and the DTD. In contrast, the studied graph repairing is to minimize the distance between the repaired and original graphs, such that the repaired graph satisfies the constraint.

2.2 Graph-based data repairing

To eliminate data violations w.r.t. integrity constraints, there are a variety of repair models proposed in previous work. Among them, two typical models, i.e., deletion and modification, can be adapted to the graph data.

The deletion-based model [8] allows deleting elements in data, in order to eliminate violations to the constraints. In terms of graph notations, it is to delete vertices (as well as the incident edges). However, the data instance will lose information in this deletion model. In particular, the structural information, an important aspect of graph data, could be lost after deletion.

The modification-based model [35] performs value modification instead of deletion. In this paper, we also adopt this value modification model, i.e., vertex relabeling. An interesting variation [22] is studied by allowing a value modified to a variable that stands for a special value outside the current domain. This special value, which will not introduce violations to the existing data, plays a similar role as the center label in a special type of star constraint (Definition 2).

The insertion-based model [1] introduces value insertion and is used for constraints on existence, e.g., adding tuples to satisfy inclusion dependencies in relational databases. In our constraint graph of label neighborhood, since there is no effect to violation elimination by adding vertices, the insertion model does not help.

The idea of equivalence classes [3] is often used in repair algorithms, where tuples are grouped into classes each of which has a certain equal value. Unfortunately, such equivalence classes do not exist w.r.t. neighborhood constraints as pair-wise relationships in a general graph. Thereby, for the general constraint graph, existing techniques [3,8,22] developed on the equivalence of tuples cannot be applied to our relabeling problem.

Effective methods and algorithms are also proposed for repairing various data types, which may not be used in graphs. For example, Flesca et al. [16] study the repairing of numerical data. Zhang et al. [36] propose to repair time series data. Song et al. [26] consider the repairing of timestamps.

3 Graph Repair with Label Modification

In this section, we present the problem of graph repairing via label modification. It is worth noting that there always exists a relabeled graph $\mathcal{G}'$ of any given instance graph $\mathcal{G}$, which satisfies any constraint $\mathcal{S}$, by simply relabeling all vertices of the graph $\mathcal{G}$ to any single label mentioned in $\mathcal{S}$. Intuitively, this important semantics of labeling schemes motivates two aspects of graph relabeling: (1) instead of arbitrarily yet unnecessarily relabeling all the vertices, we should minimally modify the data, referring to the minimum change principle for repairing [3] as introduced in Sect. 3.2. (2) By enforcing the same labels for vertices in a local structure, it always eliminates violations and leads to the practical contraction repairing algorithm with termination guarantees, as presented in Sect. 5.1.1.

3.1 Preliminaries

Let $L = \{\ell_1, \ldots, \ell_{|L|}\}$ denote a set of labels.

A *constraint graph* $\mathcal{S}(L, N, \lambda)$ is an undirected graph, where N specifies the pair-wise neighborhood constraints of unique labels in L. Intuitively, it specifies whether two labels are allowed to appear as neighbors. For instance, Fig. 1a illustrates a constraint graph that specifies the neighborhood among four labels $\{a, b, c, d\}$, where each node denotes a unique label. The edge (a, b) indicates that labels a and b could appear as neighbors. We allow two neighboring nodes with the same label, referring to the application scenarios that two tuples may have the same values in similarity networks (Example 1), or two nearby locations may have the same point-of-interest label in check-in data.

Consider another graph $\mathcal{G}(V, E, \lambda)$, namely *instance graph*, which has labels from L for all the vertices in V, given as a labeling function $\lambda : V \to L$. Fig. 1b illustrates an instance graph with four vertices $V = \{1, 2, 3, 4\}$, where each vertex is associated with a label from $L = \{a, b, c, d\}$.

We say two labels ℓ_1, ℓ_2 *match* a constraint graph $\mathcal{S}(L, N)$, denoted by $(\ell_1, \ell_2) \asymp \mathcal{S}$, if either $\ell_1 = \ell_2$ denotes the same label or $(\ell_1, \ell_2) \in N$ is an edge in $\mathcal{S}$. For example, in Fig. 4a, we have $(a, a) \asymp \mathcal{S}$, $(a, b) \asymp \mathcal{S}$, but $(a, d) \not\asymp \mathcal{S}$. It is worth noting that $(a, a) \asymp \mathcal{S}$ implies the self-loop relationship of labels in $\mathcal{S}$.

An instance graph $\mathcal{G}(V, E, \lambda)$ *satisfies* a constraint graph $\mathcal{S}(L, N)$, denoted by $\mathcal{G} \vDash \mathcal{S}$, if $(v, u) \in E$ implies $(\lambda(v), \lambda(u)) \asymp \mathcal{S}, \forall (v, u) \in E$. That is, for any edge $(v, u) \in E$, their labels $\lambda(v), \lambda(u)$ must match the constraint graph $\mathcal{S}$ with either $\lambda(v) = \lambda(u)$ or $(\lambda(v), \lambda(u)) \in N$. The verification of a given instance graph satisfying a given labeling schema can be easily done in polynomial time.

We call (v, u) a *violation* to the constraint graph $\mathcal{S}$, if $(v, u) \in E$ and $(\lambda(v), \lambda(u)) \not\asymp \mathcal{S}$. For example, in Fig. 4b, the edge $(5, 6)$ indicates a violation to $\mathcal{S}$, as their labels $(c, g) \not\asymp \mathcal{S}$ are neither the same nor adjacent in Fig. 4a of constraints.

3.2 Cost Function for Label Repair

The cost of vertex relabeling is evaluated by the difference between the original and repaired vertex labels. More precisely, the repairing target is to return a modified result that minimally differs from the original data [3]. This minimum change principle is widely adopted in improving data quality, under the rationale that people try to make as few mistakes as possible.

Following the same line of repairing in databases [3], we formalize the relabeling cost in graph by evaluating the modification on vertex labels. Let $\mathcal{G}'$ be a relabeled graph of $\mathcal{G}$. The relabeling cost is given by,

$$\Delta_l(\mathcal{G}', \mathcal{G}) = \sum_{v \in V} \delta_l \left(\lambda(v), \lambda'(v)\right), \tag{1}$$

where $\lambda'(v)$ is the new label of $v \in V$ in the repaired $\mathcal{G}'$, and $\delta_l(\lambda(v), \lambda'(v))$ denotes the (distance) cost of relabeling vertex v from label $\lambda(v)$ to $\lambda'(v)$. The metric δ_l could be any string distance function or simply the count of modifications [22]. The count cost is considered by default in the following study of decision problems,

$$\delta_l(\ell, \ell') = \begin{cases} 0 & \text{if } \ell = \ell' \\ 1 & \text{if } \ell \neq \ell' \end{cases}$$

on nonidentical arguments, where $\ell, \ell' \in L$.

String distance or counting-based metrics may not always capture the label semantics very well. A minor change could mean a completely different entity, for instance, a salary value of 10,000 is very different from 90,000, while their edit distance is small. Therefore, more advanced domain specific metrics could be employed as the relabeling cost, such as the absolute difference for numerical values.

3.3 Relabeling Problem Analysis

We now formalize the vertex label repairing (relabeling) problem: For a constraint graph $\mathcal{S}$ and an instance graph $\mathcal{G}$, it is to find a relabeled $\mathcal{G}'$ of $\mathcal{G}$ such that $\mathcal{G}' \vDash \mathcal{S}$ and the relabeling cost $\Delta_l(\mathcal{G}', \mathcal{G})$ is minimized.

Before discussing technical details, we analyze hardness of the relabeling problem and consequently identify special cases that may be addressed efficiently.

Proposition 1 *For a constraint graph $\mathcal{S}$, an instance graph $\mathcal{G}$ and a constant c, the problem of determining whether there exists a relabeled $\mathcal{G}'$ of $\mathcal{G}$ such that $\mathcal{G}' \vDash \mathcal{S}$ and the relabeling cost $\Delta_l(\mathcal{G}', \mathcal{G}) \leq c$ is* NP*-complete.*

Proof The problem is clearly in NP. Given an instance $\mathcal{G}'$, it can be verified in polynomial time whether $\mathcal{G}'$ satisfies the constraints in $\mathcal{S}$ and the cost of relabeling $\mathcal{G}$ to $\mathcal{G}'$ in Eq. 1 is no greater than c.

To show the hardness, we can use the same reduction from the vertex cover problem (which is one of Karp's 21 NP-complete problems [21]) in the proof of Theorem 2. This same reduction will also be used to show approximation bounds in Proposition 3. □

3.4 Tractable Special Case

Recognizing the hardness, we identify special cases of clique constraints where the relabeling problem turns out to be tractable. The clique constraints represent the transitivity of neighborhood on labels inside each clique, i.e., for any $(\ell_1, \ell_2) \in N$ and $(\ell_2, \ell_3) \in N$, it implies $(\ell_1, \ell_3) \in N$. Such clique constraints with transitivity feature are practical. For example, the Tel numbers in the same region are similar with each other by sharing the same area code and another two digits denoting the region, i.e., a clique in $\mathcal{S}$ in Fig. 1. Indeed, for a DD with equality constraints $[0, 0]$ on the right-hand-side attributes, the corresponding constraint graph consists of cliques with sizes 1. The DD in the application in Sect. 8.1.1 on the real Restaurant dataset corresponds to clique constraint graph exactly.

Definition 1 A *clique constraint* $\mathcal{S}(L, N)$ is a (disjoint) union of complete graphs (cliques).

It is worth noting that if there exists only one clique in $\mathcal{S}(L, N)$, then any graph is valid as long as it uses only the labels in L. This generalizes to the union of multiple cliques in $\mathcal{S}(L, N)$, which renders the repairing problem tractable.

Consequently, the relabeling process is to find connected components in the instance graph $\mathcal{G}(V, E, \lambda)$. To repair a

connected component by a clique, it is to substitute all the vertex labels not belonging to the clique by a label from the clique that minimally differs from the original label. For each connected component, we select a clique with the minimum total cost to relabel. The instance graph $\mathcal{G}$ can be relabeled efficiently in polynomial time.

3.5 Special Case of Star Constraints

Motivated by the center roles existing in some constraint graphs, such as the *cellular component* correlated to all the other gene ontology (GO) terms (as presented in Example 2 and observed in the real dataset HPRD in Sect. 8), we consider a special type of constraints with a star shape.

Definition 2 We call $\mathcal{S}(L, N)$ a *star constraint*, if there exists a center label $\ell_0 \in L$ that is adjacent to all the other labels $(\ell_0, \ell_i) \in N$, and $\delta_l(\ell_0, \ell_i) < \delta_l(\ell_j, \ell_i), i \neq j$, $i = 1, \ldots, |L| - 1$; $j = 1, \ldots, |L| - 1$.

The label ℓ_0 is adjacent to all the other labels in the constraint graph and has the relabeling cost $\delta_l(\ell_0, \ell_i)$ less than others. It serves as a center of the constraint graph; thus, we call such a structure star constraint.

Example 2 (Protein Networks) Consider *protein interaction networks* constructed from statistically assessed pair-wise protein interaction affinities (see Sect. 8 of experiments for more details). Each vertex in the network denotes a protein, e.g., 1:ITGA7 in Fig. 2b. An edge is drawn between two vertices (proteins) if their affinity level is above a preselected threshold [17]. Each protein is associated with a gene ontology[2] (GO) term description as vertex label, e.g., c:Plasma membrane for 1:ITGA7. Two proteins in a high affinity level probably belong to *the same or correlated* GO terms [37]. Proteins with high affinity but irrelevant GO terms may contain faults. For example, 1:ITGA7 and 4:FHL3 have an edge (high affinity) in Fig. 2b, but their labels c:Plasma membrane and f:Nucleus are not adjacent (irrelevant) in Fig. 2a.

Such faulty GO terms are prevalent during the automatic functional annotation. The commonly reported accuracy of GO annotation can only reach about 65–70% in practice [9]. Our proposed relabeling techniques can be applied to locate and suggest repairs of those wrongly annotated GO terms.

Figure 2a is a star constraint, where the GO term a:Cellular component can be regarded as a center that correlates to all the other gene products (describing the parts of a cell or its extracellular environment). By changing any label in violation to the center label (e.g., replace f:Nucleus of 4:FHL3 by a:Cellular component), it guarantees to eliminate violations in the graph. □

[2] www.geneontology.org.

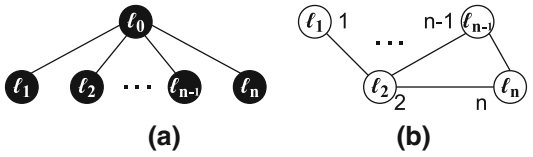


Fig. 3 Reduction from the vertex cover problem. **a** Constraint, **b** instance

Unfortunately, the problem is still hard in this case.

Theorem 2 *For a star constraint $\mathcal{S}$, an instance graph $\mathcal{G}$ and a constant c, the problem of determining whether there exists a relabeled $\mathcal{G}'$ of $\mathcal{G}$ such that $\mathcal{G}' \vDash \mathcal{S}$ and the relabeling cost $\Delta_l(\mathcal{G}', \mathcal{G}) \leq c$ is* NP*-complete.*

Proof The problem is clearly in NP. Given an instance $\mathcal{G}'$, it can be verified in polynomial time whether $\mathcal{G}'$ satisfies the constraints in $\mathcal{S}$ and the cost of relabeling $\mathcal{G}$ to $\mathcal{G}'$ in Eq. 1 is no greater than c.

To prove the NP-hardness, we show reduction from the vertex cover problem, which is one of Karp's 21 NP-complete problems [21]. Given a graph $\mathcal{G}(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, a vertex cover is a subset $C \subseteq V$ of vertices such that for each edge $(v_i, v_j) \in E$, C contains at least one of v_i or v_j.

Let $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ be the set of distinct labels. Consider graph $\mathcal{G}$ in the vertex cover problem as the instance graph, e.g., as the example illustrated in Fig. 3b. We assign $\lambda(v_i) = \ell_i$ for all the vertices $v_i \in V, i = 1, \ldots, n$, having $\lambda(v_i) \neq \lambda(v_j), i \neq j$. The constraint graph $\mathcal{S}(L, N)$ consists of neighborhoods $N = \{(\ell_0, \ell_i) \mid i = 1, \ldots, n\}$. That is, as presented in Fig. 3a, $\ell_0 \in L$ serves as the center node in the constraint graph. It is notable that all the edges in the instance graph, e.g., in Fig. 3b, are violations to the constraint. Suppose that $\delta_l(\ell_0, \ell_i) = 1$ for all $i \in [1, n]$. For the remaining label pairs between $v_i, v_j \in V$, we have $\big(\lambda(v_i), \lambda(v_j)\big) \notin N$ in the label neighborhood constraints. The corresponding cost is $\delta_l(\ell_i, \ell_j) = d$, where $d > 1$. The transformation completes and can be done in polynomial time.

As illustrated below, the graph $\mathcal{G}$ has a vertex cover C of size $|C| \leq c$ if and only if there is a label repair $\mathcal{G}'$ such that $\mathcal{G}' \vDash \mathcal{S}$ and $\Delta_l(\mathcal{G}', \mathcal{G}) \leq c$.

First, let C be a vertex cover with size c. For each edge $(v_i, v_j) \in E$, recall that $(\lambda(v_i), \lambda(v_j)) \notin N$ violates the constraint in $\mathcal{S}$. Let $v_i \in C$ be the vertex in the edge covered by C. We assign a new label $\lambda'(v_i) = \ell_0$ in the relabeled graph $\mathcal{G}'$, with the relabeling cost $\delta_l(\ell_0, \lambda(v_i)) = 1$. Since we have $(\ell_0, \lambda(v_j)) \in N$ in the constraint graph $\mathcal{S}$, the violation with respect to the edge (v_i, v_j) is removed. Consequently, by considering all the vertices in C, we have $\mathcal{G}' \vDash \mathcal{S}$ and $\Delta_l(\mathcal{G}', \mathcal{G}) = c$.

Conversely, suppose that there exists a $\mathcal{G}'$ with cost $\Delta_l(\mathcal{G}', \mathcal{G}) = c < |C^*|$, where C^* is a minimum vertex

cover. Let C be the set of vertices whose labels are relabeled in $\mathcal{G}'$ compared with the original $\mathcal{G}$. Referring to the cost $\delta_l(\ell_i, \ell_j) = d > 1, i \neq j, i \neq 0, j \neq 0$, the labels can only be re-assigned by ℓ_0. Since $\delta_l(\ell_0, \lambda(v_i)) = 1$ for all $v_i \in V$, we have $|C| = c$. Obviously, C is not a vertex cover since $|C| = c < |C^*|$, and there should be at least $|C^*| - |C|$ edges connecting between different vertices in $C^* \setminus C$ and $V \setminus C^*$. For each edge $(v_i, v_j) \in E$, $v_i \in C^* \setminus C$, $v_j \in V \setminus C^*$, we need to re-assign the label of at least one vertex in order to eliminate the corresponding violation $\big(\lambda(v_i), \lambda(v_j)\big) \notin N$. Thereby, we have $\Delta_l(\mathcal{G}', \mathcal{G}) \geq c + (|C^*| - |C|) = |C^*|$, which is a contradiction. □

Although it is still hard in the special case, as illustrated below, there exists constant-factor approximation for the special case of star constraint. The rationale is that repairing with the center label will never introduce new violations and thus stop the violation spread. This center label plays a similar role as the special value outside the domain in database repairing, which stops evoking violations to other functional dependencies [22]. It is unlikely, however, to approximate within an arbitrary small factor.

Proposition 3 *For a star constraint $\mathcal{S}$, the minimum label repair cannot be approximated within a factor of $10\sqrt{5} - 21 \approx 1.36$ unless P=NP.*

Proof We show that the reduction from the vertex cover problem in the proof of Theorem 2 is a *gap-preserving* reduction. As stated, for a star constraint $\mathcal{S}$, the graph $\mathcal{G}$ has a vertex cover C of size $|C| \leq c$ if and only if there is a $\mathcal{G}'$ such that $\mathcal{G}' \vDash \mathcal{S}$ and $\Delta_l(\mathcal{G}', \mathcal{G}) \leq c$. Let C^* denote a minimum vertex cover, and $\mathcal{G}^*$ be a relabeled graph to $\mathcal{G}$ with the minimum cost. If any approximation computes an approximate $\mathcal{G}'$ with $\Delta_l(\mathcal{G}', \mathcal{G}) \leq \alpha \cdot \Delta_l(\mathcal{G}^*, \mathcal{G})$ for some constant $\alpha > 1$, it will produce a vertex cover C such that $|C| \leq \alpha \cdot |C^*|$. More precisely, referring to the definition of gap-preserving reduction in [33], the reduction is gap-preserving since we have:

- if $|C^*| \leq k$, then $\Delta_l(\mathcal{G}^*, \mathcal{G}) \leq k$,
- if $|C^*| > \alpha k$, then $\Delta_l(\mathcal{G}^*, \mathcal{G}) > \alpha k$.

Since the minimum vertex cover cannot be approximated within a factor of $\alpha = 10\sqrt{5} - 21 \approx 1.36$ unless P = NP [10], the minimum label repair of a graph for a star constraint is also NP-hard to approximate to within any factor less than 1.36. □

4 Greedy Heuristics for Relabeling

In this section, we investigate greedy methods for vertex label repair. Typical examples are employed to explain pros and cons of the proposed techniques. By default, we use Fig. 4a as the constraint graph for all the examples in the following sections.

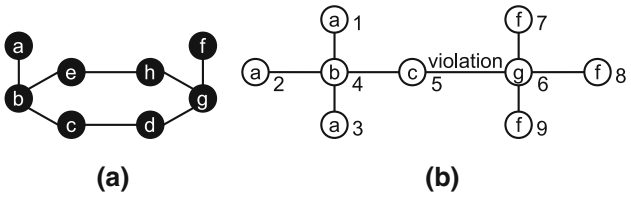


Fig. 4 Counter example of termination in greedy method. **a** Constraint, **b** instance

4.1 How Greedy Methods Fail

Intuitively, in each step of relabeling, it is desirable to eliminate more violations without introducing new ones by paying less cost. Motivated by the connection to classical combinatorial problems (in Proposition 1), it is natural to adopt the greedy method as follows.

We define the *violation set* of a vertex v with label ℓ,

$$T(v, \ell) = \{u \mid (v, u) \in E, (\ell, \lambda(u)) \not\asymp \mathcal{S}\}, \tag{2}$$

which denotes the set of neighbors u of the vertex v whose labels $\lambda(u)$ have violations to $\lambda(v) = \ell$. It is to greedily select a vertex with the maximum violation set to relabel in each iteration, i.e.,

$$\arg\max_{v \in V} |T(v, \lambda(v))|.$$

This *straightforward* greedy function is analogous to selecting a set with the largest number of uncovered elements in the set cover approximation. Once a vertex v is selected, we find a new label $\lambda'(v)$ for the vertex to eliminate violations.

We aim to eliminate more violations by each relabeling. Instead of greedily selecting a vertex with the maximum violations, we can choose a vertex relabeling that can eliminate the most violations. Thereby, the greedy function is *revised* to evaluate the number of violations that are eliminated, i.e., subtracting the remaining violations after repairing $|T(v, (\lambda'(v))|$ from the original violations $|T(v, \lambda(v))|$,

$$\arg\max_{v \in V, \lambda'(v) \in L} |T(v, \lambda(v))| - |T(v, \lambda'(v))|. \tag{3}$$

Moreover, when the relabeling cost between labels is considered, we may further *normalize* the violation elimination gain by the relabeling cost $\delta_l\big(\lambda(v), \lambda'(v)\big)$,

$$\arg\max_{v \in V, \lambda'(v) \in L} \frac{|T(v, \lambda(v))|}{\delta_l\big(\lambda(v), \lambda'(v)\big)} - |T(v, \lambda'(v))|. \tag{4}$$

That is, a number of violations $|T(v, \lambda(v))|$ is eliminated by paying the cost $\delta_l(\lambda(v), \lambda'(v))$ with the least new violations $|T(v, \lambda'(v))|$ introduced. It is notable that simply normalizing formula (3) by the cost, i.e., $\frac{|T(v,\lambda(v))|-|T(v,\lambda'(v))|}{\delta_l(\lambda(v),\lambda'(v))}$, is *not* a valid attempt. A relabeling operation may introduce more violations than it eliminates, i.e., having $|T(v, \lambda(v))| < |T(v, \lambda'(v))|$. In such cases, the greedy function irrationally favors a relabeling with a higher cost $\delta_l\big(\lambda(v), \lambda'(v)\big)$, as the violation reduction value is negative.

Unfortunately, the greedy approach is not guaranteed to terminate (which is observed in the following experiments as well). Indeed, for any vertex relabeling selected with the maximum greedy value, if it has $|T(v, \lambda(v))| \leq |T(v, \lambda'(v))|$, the number of violations has no reduction after a greedy step.

Example 3 Consider the example in Fig. 4 with relabeling cost $\delta_l(c, d) = \delta_l(g, d) = 1$. According to the greedy function in formula (4), the best choice is to repair vertex 5 with label d. While the violation between vertices 5 and 6 is eliminated, a new violation between vertices 5 and 4 is introduced after relabeling vertex 5 to d. Based on the greedy function, the best choice next is to repair vertex 5 back to c. The relabeling steps repeat and cannot terminate. □

4.2 Special Case of Star Constraints

In Sect. 3.5 (Example 2), we have illustrated the importance of star constraint with a center role. Surprisingly, when given a star constraint, the greedy method is not bad, which can terminate and return a result with certain guarantee.

Proposition 4 *For a star constraint $\mathcal{S}$, the greedy method terminates and outputs a $\mathcal{G}'$ having $\frac{\Delta_l(\mathcal{G}',\mathcal{G})}{\Delta_l(\mathcal{G}^*,\mathcal{G})} \leq \ln n + 1$, where $\mathcal{G}^*$ is the relabeled graph with the minimum cost and $n = |E|$.*

Proof First, we illustrate that a vertex can only be relabeled to ℓ_0 in the optimal $\mathcal{G}^*$. Assume that there exists a vertex v which is relabeled from $\lambda(v)$ to $\lambda'(v) \neq \ell_0$ in $\mathcal{G}^*$. According to the definition of star constraints, we have $\delta_l(\lambda(v), \ell_0) < \delta_l(\lambda(v), \lambda'(v))$. Since ℓ_0 will not introduce violations to any $\ell_i \in L$, we can relabel the vertex v to ℓ_0 to generate a new relabeled graph, say $\mathcal{G}'$, having $\Delta_l(\mathcal{G}', \mathcal{G}) < \Delta_l(\mathcal{G}^*, \mathcal{G})$. Obviously, it contradicts with the assumption of $\mathcal{G}^*$ with the minimum cost.

Second, we show in the following that each greedy step will select a vertex with the max $\frac{|T(v,\lambda(v))|}{\delta_l(\lambda(v),\ell_0)}$ and relabel it to ℓ_0. Given any vertex v, the greedy function always selects a relabeling $\lambda'(v)$ with the minimum $|T(v, \lambda'(v))|$ and $\delta_l(\lambda(v), \lambda'(v))$. Since ℓ_0 will not introduce violations to any $\lambda(v)$, we have $|T(v, \ell_0)| = 0$ which is already the minimum. Moreover, the relabeling cost of ℓ_0 is smaller than any other $\ell_i \in L$, i.e., $\delta_l(\lambda(v), \ell_0) < \delta_l(\lambda(v), \ell_i)$. Consequently, the greedy function turns out to select a vertex v with the maximum $\frac{|T(v,\lambda(v))|}{\delta_l(\lambda(v),\ell_0)}$.

We consider the sequence of graphs $\mathcal{G}, \ldots, \mathcal{G}_i, \ldots, \mathcal{G}'$, where $\mathcal{G}_i$ is generated in the ith greedy step. Let

$$A(\mathcal{G}, \mathcal{S}) = \{(u, v) \in E \mid (\lambda(v), \lambda(u)) \not\asymp \mathcal{S}\}$$

denote all the violations in graph $\mathcal{G}$ w.r.t. constraint $\mathcal{S}$.

For the graph $\mathcal{G}_i$ in the ith greedy step, we have

$$\frac{|A(\mathcal{G}_i, \mathcal{S})|}{\Delta_l(\mathcal{G}_i^*, \mathcal{G}_i)} \leq \max_{v \in V(\mathcal{G}_i)} \frac{|T(v, \lambda(v))|}{\delta_l(\lambda(v), \ell_0)},$$

where $\mathcal{G}_i^*$ denotes the optimal solution of $\mathcal{G}_i$. It is worth noting that $\frac{|A(\mathcal{G}_i,\mathcal{S})|}{\Delta_l(\mathcal{G}_i^*,\mathcal{G}_i)}$ denotes the average number of violations that a unit of relabeling cost can eliminate by a vertex in the optimal $\mathcal{G}_i^*$ of $\mathcal{G}_i$, which must be smaller than the maximum number of violations that a cost unit can eliminate by a vertex $v \in V(\mathcal{G}_i)$, i.e., $\max_{v\in V(\mathcal{G}_i)} \frac{|T(v,\lambda(v))|}{\delta_l(\lambda(v),\ell_0)}$.

Let $v_i = \arg\max_{v \in V(\mathcal{G}_i)} \frac{|T(v,\lambda(v))|}{\delta_l(\lambda(v),\ell_0)}$ be the vertex selected in each $\mathcal{G}_i$. Since no new violations will be introduced in each step by relabeling v_i to ℓ_0, i.e., $A(\mathcal{G}_i, \mathcal{S}) \subseteq A(\mathcal{G}, \mathcal{S})$, the corresponding minimum relabeling cost should be smaller as well, having $\Delta_l(\mathcal{G}_i^*, \mathcal{G}_i) \leq \Delta_l(\mathcal{G}^*, \mathcal{G})$. Consequently, we have

$$\frac{|A(\mathcal{G}_i, \mathcal{S})|}{\Delta_l(\mathcal{G}^*, \mathcal{G})} \leq \frac{|T(v_i, \lambda(v_i))|}{\delta_l(\lambda(v_i), \ell_0)}.$$

Moreover, we observe that the violations reduce in the greedy steps, $|A(\mathcal{G}, \mathcal{S})|, \ldots, |A(\mathcal{G}_i, \mathcal{S})|, \ldots, |A(\mathcal{G}', \mathcal{S})|$, where $|A(\mathcal{G}_{i+1}, \mathcal{S})| = |A(\mathcal{G}_i, \mathcal{S})| - |T(v_i, \lambda(v_i))|$ and finally $|A(\mathcal{G}', \mathcal{S})| = 0$. Therefore, the greedy method must terminate in at most $|A(\mathcal{G}, \mathcal{S})|$ greedy steps.

To sum up, the relabeling cost can be computed by

$$\begin{aligned}
\Delta_l(\mathcal{G}', \mathcal{G}) &= \sum_i \delta_l(\lambda(v_i), \ell_0) \\
&\leq \sum_i \Delta_l(\mathcal{G}^*, \mathcal{G}) \frac{|T(v_i, \lambda(v_i))|}{|A(\mathcal{G}_i, \mathcal{S})|} \\
&\leq \Delta_l(\mathcal{G}^*, \mathcal{G}) \sum_i \sum_{j=1}^{|T(v_i,\lambda(v_i))|} \frac{1}{|A(\mathcal{G}_i, \mathcal{S})| - j + 1} \\
&= \Delta_l(\mathcal{G}^*, \mathcal{G}) \sum_{k=1}^{|A(\mathcal{G},\mathcal{S})|} \frac{1}{k} \\
&\leq (\ln |A(\mathcal{G}, \mathcal{S})| + 1)\Delta_l(\mathcal{G}^*, \mathcal{G}).
\end{aligned}$$

As the number of violations is no greater than that of edges, $|A(\mathcal{G}, \mathcal{S})| \leq |E|$, the conclusion is proved. □

Experimental results on real (HPRD) data with star constraints in Sect. 8 verify that the greedy method always terminates and shows high repairing accuracy.

5 Contraction Relabeling

In this section, we present a contraction method for vertex label repair, which is guaranteed to terminate.

5.1 The Idea of Contraction

5.1.1 Intuition

Recall that greedy relabeling fails as new violations may be generated between vertices which are eliminated in the previous steps. In order to avoid eliminating and generating violations on the same vertex pairs multiple times, we consider a group of vertices as a whole, called a *super node* or simply a *node*, which *always* ensures no new violations generated inside. To ensure no new violations, we require that the vertices in a node can only be repaired together to the same label.

To eliminate violations among vertices from two nodes, a node contraction operation is employed, i.e., merging all the contents (vertices and edges) of a node R_1 into the other R_2. All the vertices in the contracted node R_1 are enforced to assign the same label, which can avoid introducing violations in the new node R_2. Different from the application of contraction heuristic in tree decomposition or spanning tree, the contraction in this study aims to eliminate all violations (with possibly minimal vertex label modification) instead of generating a spanning tree with minimal edges.

Example 4 Suppose that three nodes, $\mathsf{R}_1, \mathsf{R}_2, \mathsf{R}_3$, are formed in the previous steps, in Fig. 5a. A vertex (say $\mathsf{t}+2$ for instance) can *only* be relabeled again together with all the other vertices in node R_1 that has been contracted, in order to stop violation spread inside the node. To eliminate the violation between R_1 and R_2, node R_1 is contracted to R_2 by enforcing all the vertices in R_1 to the same label f (more technique details for the contraction operation will be explained soon). The contraction terminates as there is no further violation between the remaining nodes R_2 and R_3. All the violations are eliminated in Fig. 5b, by ensuring that no violations exist inside a node. □

It is easy to see the termination by eventually contracting all the vertices into one node. We simply enforce a same label on all the vertices to eliminate violations. Such a straightforward solution will unnecessarily relabel the vertices without any violation. We propose to eliminate violations by paying less relabeling cost.

5.1.2 Framework

To perform the contraction, we first introduce several notations as follows. Let R denote a *node* in contraction, consisting of a group of vertices contracted to R. Each node

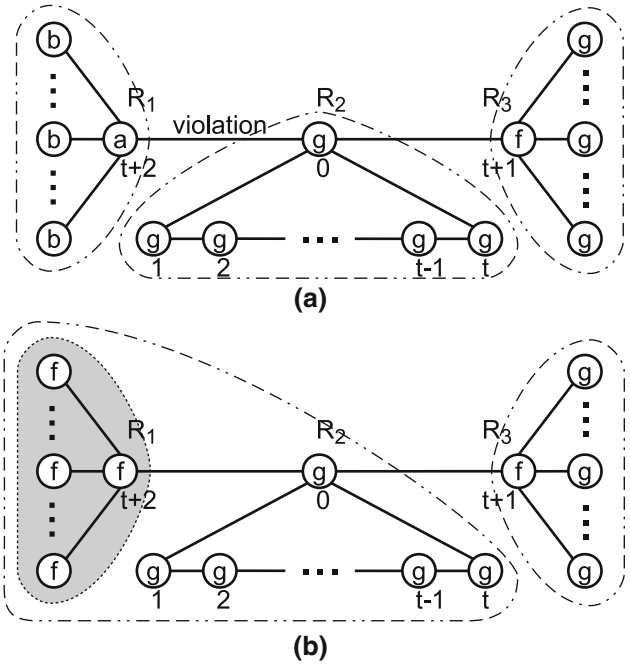


Fig. 5 Idea of contraction. **a** Before, **b** after

R has a unique *host* $h(\mathsf{R})$ which is a vertex in the original graph. The remaining vertices in R are grouped into a set of nested nodes R_i, namely *guests* denoted by set $U(\mathsf{R})$. Each guest $\mathsf{R}_i \in U(\mathsf{R})$ is previously contracted to R via a contraction operation. Let $V(\mathsf{R})$ be the set of all vertices in the original graph belonging to R, including both the host and vertices in all guests

$$V(\mathsf{R}) = \bigcup_{\mathsf{R}_i \in U(\mathsf{R})} V(\mathsf{R}_i) \cup \{h(\mathsf{R})\}. \tag{5}$$

The host and guests do not need to share the same label but should have no violations. As we will see soon, the host is essential to ensure that there always exists a valid candidate label to assign in the contraction. All the vertices in a guest $\mathsf{R}_i \in U(\mathsf{R})$ must have the same label, which is assigned by the contraction operation.

Example 5 In Fig. 6b, we illustrate an example with three nodes, in dash-dot circles. Each node has a unique host (e.g., vertex $\mathsf{5}$ in R_5) and none or multiple guests (in shaded area). All the vertices in a guest should share the same label, e.g., vertices $(\mathsf{6}, \mathsf{7}, \mathsf{8})$ with label b, while the vertices among different guest nodes/host ensure no violations but do not have to share the same label. □

Algorithm 1 presents an overview of the contraction procedure. Initially, each vertex v in the graph forms a single node R with host $h(\mathsf{R}) = v$ and an empty set of guest nodes $U(\mathsf{R}) = \emptyset$. The contraction is then conducted between two

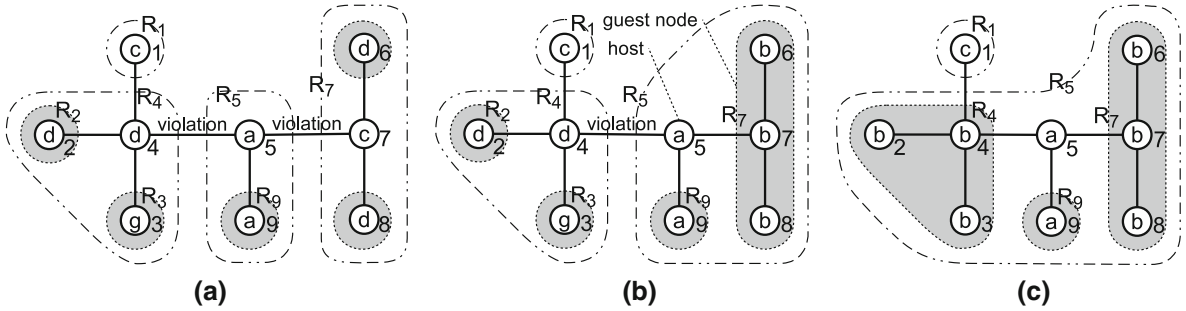


Fig. 6 Example of contraction operation

nodes, say R_1 and R_2 for example, whose vertices have violations. Suppose that R_2 is contracted to R_1 with the new label ℓ_2 (the decision of R_1, R_2, candidate label ℓ_2 and $\mathsf{cost}(\mathsf{R}_2)$ will be discussed soon). Then, all the vertices in the original graph belonging to R_2 will be relabeled to ℓ_2, i.e., $\lambda(v) = \ell_2$, $v \in V(\mathsf{R}_2)$. The node R_2 is added to R_1 as a guest node by $U(\mathsf{R}_1) = U(\mathsf{R}_1) \cup \{\mathsf{R}_2\}$. The host of R_1 after contraction leaves unchanged, i.e., still $h(\mathsf{R}_1)$. We have $V(\mathsf{R}_1) = V(\mathsf{R}_1) \cup V(\mathsf{R}_2)$ after the contraction. The contraction relabeling terminates, when there is no violation to contract.

Algorithm 1 Contract($\mathcal{G}, \mathcal{S}$)

Input: An instance graph $\mathcal{G}$ and a constraint graph $\mathcal{S}$
Output: A relabeled $\mathcal{G}$ satisfying $\mathcal{S}$
1: **for** each vertex v in the graph $\mathcal{G}$ **do**
2: create a new node R
3: $h(\mathsf{R}) := v$
4: $U(\mathsf{R}) := \emptyset$
5: **while** $\mathcal{G}$ not satisfying $\mathcal{S}$ **do**
6: $\mathsf{R}_1, \mathsf{R}_2 :=$ the nodes with most violations
7: $\ell_1, \ell_2 :=$ the candidate labels for $\mathsf{R}_1, \mathsf{R}_2$, respectively
8: **if** $\mathsf{cost}(\mathsf{R}_2) > \mathsf{cost}(\mathsf{R}_1)$ **then**
9: swap $\mathsf{R}_1, \mathsf{R}_2$ {To contract R_2 to R_1}
10: **for** each $v \in V(\mathsf{R}_2)$ **do**
11: $\lambda(v) := \ell_2$
12: $V(\mathsf{R}_1) := V(\mathsf{R}_1) \cup V(\mathsf{R}_2)$
13: $U(\mathsf{R}_1) := U(\mathsf{R}_1) \cup \{\mathsf{R}_2\}$
14: **return** $\mathcal{G}$

The contraction on R_1 and R_2 will eliminate all the violations on the edges across these two nodes. Following the same intuition of violation elimination in greedy heuristics, as shown in Line 6 of Algorithm 1, each step would like to select a pair of nodes with the most violations between them, i.e.,

$$\arg\max_{(\mathsf{R}_1,\mathsf{R}_2)} |\{(u, v) \in E \mid u \in V(\mathsf{R}_1), v \in V(\mathsf{R}_2), (\lambda(u), \lambda(v)) \not\asymp \mathcal{S}\}|.$$

Finally, each guest node R_i records all the vertices in the original graph that are relabeled to the same ℓ_i, i.e., relabeling results.

5.1.3 Correctness

Once two nodes with vertices in violation are contracted, they will always satisfy constraints by enforcing the contracted node to a same label and stop violation spread inside the node. Although new violations may be introduced outside the node after a contraction operation, the contraction relabeling is guaranteed to terminate referring to the reducing of violation upper bound.

Proposition 5 *The contraction relabeling always terminates.*

Proof in the preliminary version of this paper [30]. □

Each contraction operation eliminates all the violations on edges between nodes R_1 and R_2, i.e., related to $|E|$. Both Line 6 for choosing nodes with maximum violations and Line 7 for deciding candidate labels (see details below) in Algorithm 1 have to consider all possible violations in edges between two nodes, with $O(|E|)$ cost. According to Proposition 5, the iteration terminates in at most $|V|$ contraction operations. Thereby, the computational complexity of Algorithm 1 is $O(|V| \cdot |E|)$.

5.2 Technique Details

Consider the contraction of an edge with violations in the current graph. It is expected to eliminate violations while keeping the relabeling change as small as possible. Following this discipline, the contraction operation mainly needs to address two issues: (i) what are the candidate labels for relabeling the nodes involved in the edge (analogous to Line 7 in Algorithm 1) and (ii) which side of the edge should be contracted (Line 8).

5.2.1 Deciding the Labels

We first study the selection of candidate labels ℓ_1, ℓ_2 for nodes $\mathsf{R}_1, \mathsf{R}_2$, respectively. Let's take R_2 for example. The new label ℓ_2 for R_2 should not introduce any violations to the current vertices in R_1.

It is notable that a new label for R_2 in the contraction always exists. Recall that host $h(\mathsf{R}_1)$ should have label with no violations to any guest node in R_1. Thereby, the straightforward method is to assign the host's label as the new label of R_2.

Consider all the possible candidate labels for R_2, denoted by

$$L(\mathsf{R}_2) = \{\ell' \mid (\ell', \lambda(h(\mathsf{R}_1))) \asymp \mathcal{S}, u \in V(\mathsf{R}_1), v \in V(\mathsf{R}_2) \\ [(u, v) \in E \Rightarrow (\lambda(u), \ell') \asymp \mathcal{S}]\}.$$

That is, the new label ℓ', assigned to the vertices v in R_2, must match the label of the host $h(\mathsf{R}_1)$ and should not introduce violations to the existing vertices u in R_1. It is worth noting that matching with host $(\ell', \lambda(h(\mathsf{R}_1))) \asymp \mathcal{S}$ is necessary, which ensures the aforesaid existence of a valid candidate label for the following contraction operations. For any candidate ℓ', let

$$T(\mathsf{R}_2, \ell') = \{u \mid \exists v \in V(\mathsf{R}_2)[(u, v) \in E \wedge (\lambda(u), \ell') \not\asymp \mathcal{S}]\}$$

be all the vertices u in the graph that are adjacent to some vertex v in R_2 and have violations to the new label ℓ' assigned to v.

Following the same principle of eliminating violations as greedy methods, we select label ℓ_2 for R_2 as

$$\arg\max_{\ell' \in L(\mathsf{R}_2)} |T(\mathsf{R}_2)| - |T(\mathsf{R}_2, \ell')|, \tag{6}$$

where $T(\mathsf{R}_2) = \cup_{v \in V(\mathsf{R}_2)} T(v, \lambda(v))$ denotes the previous violations to the vertices in R_2 before contraction.

Example 6 (Example 5 continued) In Fig. 6b, suppose that we want to decide the candidate label of R_4 for the contraction to R_5. To avoid introducing violations to R_5, we have $L(\mathsf{R}_4) = \{a, b\}$. However, the label a for R_4 will introduce a new violation to vertex $\mathsf{1}$ with $|T(\mathsf{R}_4, a)| = 1$, while relabeling with b eliminates all violations, i.e., $|T(\mathsf{R}_4, b)| = 0$. According to formula (6), the candidate label for R_4 is b. □

5.2.2 Deciding the Contraction

We now have two candidates for contraction, either relabeling R_1 to ℓ_1 or relabeling R_2 to ℓ_2. The corresponding costs raised by different relabeling are various.

We define the contraction cost as follows, say relabeling R to ℓ',

$$\mathsf{cost}(\mathsf{R}) = \sum_{v \in V(\mathsf{R})} \delta_l(\lambda(v), \ell') - \sum_{\mathsf{R}_i \in U(\mathsf{R})} \mathsf{cost}(\mathsf{R}_i). \tag{7}$$

The first part $\sum_{v \in V(\mathsf{R})} \delta_l(\lambda(v), \ell')$ denotes the cost of enforcing all vertices v in R to the new label ℓ'. Intuitively, by paying the cost of relabeling all these vertices, we eliminate the violations not only for the current contraction of R but also the former contractions of R_i that happened inside R. However, the violations w.r.t. guest nodes R_i have already been eliminated in the previous contraction, i.e., all the vertices in R_i already have the same label when R_i was contracted as a guest node to R. Thereby, the previously paid contraction costs, $\sum_{\mathsf{R}_i \in U(\mathsf{R})} \mathsf{cost}(\mathsf{R}_i)$, would not be counted again in the current contraction of R and deserve to be "paid back".

Example 7 (Example 5 continued) In Fig. 6b, we consider the candidate label d for contracting R_5–R_4. By relabeling all five vertices $\{5, 6, 7, 8, 9\}$ to the same d, it eliminates violations not only w.r.t. R_5 but also the previously contracted R_7. However, the cost of eliminating violations w.r.t. R_7 has already been counted in the former contraction of R_7–R_5, in Fig. 6a, and should be deducted from the current cost for eliminating violations of R_5.

Consequently, we can select the one with smaller cost to contract. For instance, in Fig. 6b, suppose that we have $\mathsf{cost}(\mathsf{R}_2) = \mathsf{cost}(\mathsf{R}_3) = \mathsf{cost}(\mathsf{R}_7) = \mathsf{cost}(\mathsf{R}_9) = 1$ in the previous steps. The candidate labels for $\mathsf{R}_4, \mathsf{R}_5$ are b, d, respectively, with $\delta_l(b, d) = \delta_l(a, d) = \delta_l(b, g) = 1$. According to formula (7), it follows

$$\begin{aligned} \mathsf{cost}(\mathsf{R}_4) &= 2\delta_l(b, d) + \delta_l(b, g) - \mathsf{cost}(\mathsf{R}_2) - \mathsf{cost}(\mathsf{R}_3) \\ &= 3 - 1 - 1 = 1, \\ \mathsf{cost}(\mathsf{R}_5) &= 2\delta_l(a, d) + 3\delta_l(b, d) - \mathsf{cost}(\mathsf{R}_7) - \mathsf{cost}(\mathsf{R}_9) \\ &= 5 - 1 - 1 = 3. \end{aligned}$$

Thereby, R_4 is contracted to R_5 as illustrated in Fig. 6c, according to $\mathsf{cost}(\mathsf{R}_4) < \mathsf{cost}(\mathsf{R}_5)$. □

5.3 Performance Analysis

Unfortunately, the contraction result could be arbitrarily bad in terms of relabeling cost in general cases.

Example 8 In Fig. 7, a contraction will be conducted for the violation in $(1, 2)$. To eliminate the violation, the candidate label of contracting R_1 with host $\mathsf{1}$ could be g, and the candidate label of R_2 could be b. Suppose that the relabeling costs are $\delta_l(a, g) = 2, \delta_l(b, f) = 1$. That is, we have $\mathsf{cost}(\mathsf{R}_1) = \delta_l(a, g) = 2$ which is greater than $\mathsf{cost}(\mathsf{R}_2) = \delta_l(b, f) = 1$. The node R_2 will be contracted to

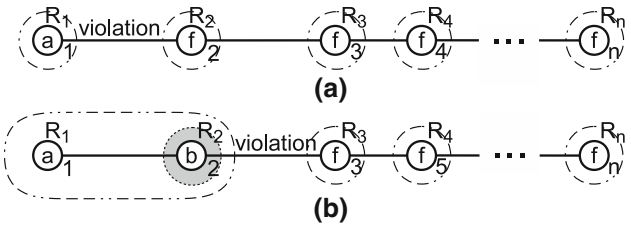


Fig. 7 Counter example of contraction effectiveness. **a** Before, **b** after

R_1 as presented in Fig. 7b. Following the same principle, the node R_3 with host 3 will be contracted to R_1 as well. By keeping on contracting nodes R_i with host i to R_1, $\mathsf{i} = 2, \ldots, \mathsf{n}$, and assigning a new label b, the total relabeling cost is $n - 1$. However, it is easy to see that the relabeling with the minimum cost is to relabel vertex 1 to g with cost 2. □

Special Case of Star Constraints

Although the contraction results could be bad in general case, the performance of contraction method is surprisingly good under star constraints.

Proposition 6 *For a star constraint $\mathcal{S}$, the contraction method always terminates and outputs a $\mathcal{G}'$ having $\Delta_l(\mathcal{G}', \mathcal{G}) \leq 2 \cdot \Delta_l(\mathcal{G}^*, \mathcal{G})$, where $\mathcal{G}^*$ is the relabeled graph with the minimum cost.*

Proof First, we show that in deciding the candidate labels for contraction, only the center ℓ_0 in the star constraint will be returned. Formula (6) always chooses the label which introduces the minimum violation, i.e., min $|T(\mathsf{R}_2, \ell')|$. Referring to Definition 2 of star constraints, the center ℓ_0 is a label having no violations to any others. Thereby, ℓ_0 is always selected as the candidate label, which can eliminate violations most, as $T(\mathsf{R}_2, \ell_0) = \emptyset$.

Then, we study the cost function for deciding the contraction. Since all the contractions will only use ℓ_0 to relabel, all the guest nodes R_i in any node R will share the same label, i.e., ℓ_0. When the node R is chosen to be contracted, the vertices in R_i have already been assigned the label ℓ_0, having $\delta_l(\lambda(v), \ell_0) = 0, v \in V(R_i)$. In other words, only the host $h(\mathsf{R})$ needs to be relabeled to ℓ_0. The contraction cost in formula (7) can be rewritten by

$$\mathsf{cost}(\mathsf{R}) = \delta_l(\lambda(h(\mathsf{R})), \ell_0) - \sum_{\mathsf{R}_i \in U(\mathsf{R})} \mathsf{cost}(\mathsf{R}_i). \tag{8}$$

We can observe that contraction will only occur for the violations between the hosts of nodes, since guest nodes are relabeled to ℓ_0. For any contraction of R–R', with hosts $h(\mathsf{R}) = v$ and $h(\mathsf{R}') = u$, respectively, we can uniquely represent the contraction cost paid to eliminate the violations in edge (u, v) by $\mathsf{cost}(u, v) = \mathsf{cost}(\mathsf{R})$. Those edges with violations eliminated without any contraction are denoted by $\mathsf{cost}(u, v) = 0$. Consequently, all the edges with violations in the original graph $\mathcal{G}$ are assigned a cost. Each contraction eliminates one or several edges having violations without introducing new violations.

The output $\mathcal{G}'$ consists of vertices v either being the hosts of nodes with the original label $\lambda(v)$ in $\mathcal{G}$ or belonging to contracted guest nodes with the new relabel ℓ_0. First, for the vertices v in contracted guest nodes, all the violations to v are eliminated by relabeling it to ℓ_0. According to the aforesaid cost formula (8), we have

$$\delta_l(\lambda(v), \ell_0) = \sum_{u \in T(v, \lambda(v))} \mathsf{cost}(u, v). \tag{9}$$

Second, recall that we always choose the node with less cost to contract. For any node R with host v contracted to the node R' with host u, it follows $\mathsf{cost}(\mathsf{R}') > \mathsf{cost}(\mathsf{R}) = \mathsf{cost}(u, v)$. Thereby, for the vertices u as the hosts of nodes in $\mathcal{G}'$, we have

$$\delta_l(\lambda(u), \ell_0) > \sum_{v \in T(u, \lambda(u))} \mathsf{cost}(u, v). \tag{10}$$

Let D be the set of all vertices in all contracted guest nodes, i.e., the vertices that are relabeled in $\mathcal{G}'$ from $\mathcal{G}$. We have

$$\begin{aligned}
\Delta_l(\mathcal{G}', \mathcal{G}) &= \sum_{v \in \mathsf{D}} \delta_l(\lambda(v), \ell_0) \\
&= \sum_{v \in \mathsf{D}} \sum_{u \in T(v, \lambda(v))} \mathsf{cost}(u, v) \quad \text{by Eq. 9} \\
&\leq \sum_{v \in V(\mathcal{G})} \sum_{u \in T(v, \lambda(v))} \mathsf{cost}(u, v) \\
&= \sum_{(v,u) \in E(\mathcal{G}), (\lambda(v), \lambda(u)) \not\approx \mathcal{S}} 2\, \mathsf{cost}(u, v)
\end{aligned}$$

Similarly, it is easy to verify that the vertices in the optimal solution $\mathcal{G}^*$ can only be relabeled to ℓ_0 as well. Let D^* be the set of all vertices conducted relabeling from $\mathcal{G}$ to $\mathcal{G}^*$. We have

$$\begin{aligned}
\Delta_l(\mathcal{G}^*, \mathcal{G}) &= \sum_{v \in \mathsf{D}^*} \delta_l(\lambda(v), \ell_0) \\
&\geq \sum_{v \in \mathsf{D}^*} \sum_{u \in T(v, \lambda(v))} \mathsf{cost}(u, v) \quad \text{by Eq. 9 and 10} \\
&\geq \sum_{(v,u) \in E(\mathcal{G}), (\lambda(v), \lambda(u)) \not\approx \mathcal{S}} \mathsf{cost}(u, v)
\end{aligned}$$

Since the optimal solution has to address all the edges with violations, some of which may have both vertices relabeled in $\mathcal{G}^*$, the last derivation step is explained.

Combining two derivations, the conclusion is proved. □

Experimental results on real datasets in Sect. 8 show that the contraction method always terminates.

6 Combining Greedy and Contraction

While the greedy method might not terminate, the contraction results could be bad in terms of relabeling costs. On the other side, as we will also see in the experiment, the contraction method always terminates, whereas the greedy heuristics could achieve good results in practice once the program terminates. It motivates us to find a method to combine the advantages of these two approaches for repairing vertex labels.

6.1 The Idea of Combination

The advantage of greedy relabeling is that it is effective in repairing the right results in most cases (as also observed in the experiments). Unfortunately, the greedy method may fail to eliminate violations in certain cases and thus lead to non-termination (no results). The advantage of contraction method is that it always reduce violations in each step and thus guarantees termination. However, the contraction relabeling may seriously modify the labels with extremely high relabeling cost.

To incorporate the advantages of both greedy and contraction methods, we can eliminate violations first by the greedy method and apply contraction only when necessary. That is, when no violations could be further reduced by greedy relabeling, the contraction operation is applied. These two operations are conducted alternatively, where the greedy operation has a higher priority.

6.2 AlterGC Algorithm

Algorithm 2 illustrates the pseudo-code of AlterGC, which alternatively performs greedy and contract operations. First, Lines 2–4 are the violation elimination operation from the previous greedy method. In particular, Line 3 specifies an additional condition that the relabeling should at least eliminate some violations, i.e., $|T(v, \lambda(v))| > |T(v, \lambda'(v))|$. When no such greedy elimination could be applied, i.e., no further violations can be reduced as aforesaid by the currently best relabeling $\lambda'(v)$, the contraction is conducted in Line 6.

Proposition 7 *AlterGC relabeling always terminates.*

Proof in the preliminary version of this paper [30]. □

According to the proof, the number of contractions is bounded by $O(|V|)$, while each contraction takes $O(|E|)$ in Algorithm 1. Between two contractions is a series of greedy relabeling operations, where the number of greedy operations is bounded by the maximum number of violations $|E|$. Calculating the violations takes $O(|E|)$. Thereby, the complexity is $O(|V| \cdot |E|^2)$.

Algorithm 2 AlterGC($\mathcal{G}, \mathcal{S}$)

Input: An instance graph $\mathcal{G}$ and a constraint graph $\mathcal{S}$
Output: A relabeled $\mathcal{G}$ satisfying $\mathcal{S}$
1: **while** $\mathcal{G}$ not satisfying $\mathcal{S}$ **do**
2: $(v, \lambda'(v))$:= the vertex with maximum greedy value
3: **if** $|T(v, \lambda(v))| > |T(v, \lambda'(v))|$ **then**
4: update v with $\lambda'(v)$ in $\mathcal{G}$ {one greedy operation}
5: **else**
6: conduct a contraction in $\mathcal{G}$
7: **return** $\mathcal{G}$

Example 9 (Example 8 continued) We consider the example in Fig. 7 again. The greedy relabeling will always be applied first when appropriate. By relabeling vertex 1 from a to g, we have $|T(1, g)| = 0$ less than the original $|T(1, a)| = 1$. Thereby, this greedy relabeling is conducted with a total cost 2, instead of $n - 1$ by the pure contraction method.

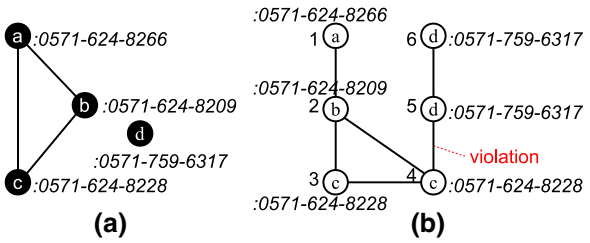
However, if the input instance graph for repairing is Fig. 7b, the AlterGC approach may goes bad. Specifically, by either repairing vertex 2 with $|T(2, g)| = 1$ or repairing vertex 3 with $|T(3, a)| = 1$, the number of violations does not reduce. Referring to Line 6 in Algorithm 2, the contraction will be conducted. Suppose that node R_3 is contracted to R_2. Similar to Example 8, it leads to a sequence of contractions on nodes R_i, $i = 3, \ldots, n$, with total relabeling cost $n - 2$. □

For the case of star constraints, since the center label will not introduce violation to any other labels, we can eliminate at least one violation in each greedy step (by using the center label). That is, each iteration in Algorithm 2 will always choose the greedy operation in Line 4. Since no contraction operation is conducted, the approximation performance of the AlterGC algorithm is also guaranteed, the same as for the greedy algorithm.

Proposition 8 *For a star constraint $\mathcal{S}$, the AlterGC algorithm terminates and outputs a $\mathcal{G}'$ having $\frac{\Delta_l(\mathcal{G}', \mathcal{G})}{\Delta_l(\mathcal{G}^*, \mathcal{G})} \leq \ln n + 1$, where $\mathcal{G}^*$ is the relabeled graph with the minimum cost and $n = |E|$.*

Proof in the preliminary version of this paper [30]. □

Experimental evaluation shows that the AlterGC approach always terminates as the contract one, while it keeps the accuracy as high as the greedy method.



ID	Address	City	Tel
1	722, West Lake St.	HZ	0571-624-8266
2	723, West Lake Str.	HZ	0571-624-8209
3	No.721, West Lake Street	HZ	0571-624-8228
4	***, West Lake Street	HZ	0571-624-8228
5	640, West Park Street	HZ	0571-759-6317
6	No.640, West Park Str.	HZ	0571-759-6317

(c)

Fig. 8 Neighbor repairing on similarity network. **a** Constraint, **b** instance, **c** relation

7 Graph Repair with Neighbor Modification

Besides inaccurate labels, errors on vertex neighbors (edges) may also incur violations to the constraint graph. In this section, we first show the motivation examples on how such errors occur and are possibly addressed by neighbor modification (Sect. 7.1). Section 7.2 presents the problem analysis on simultaneously repairing label and neighbor errors. A cubic-time constant-factor approximation algorithm (given degree-bounded instance graphs) is then devised in Sect. 7.3 to combine the label and neighbor repairs.

7.1 Motivation Example

To illustrate the motivation example on neighbor repair, we consider the similarity networks in Sect. 1.1.

Example 10 Consider again the DD (Address, City $\rightarrow$ Tel, $\langle[0, 6], [0, 0], [0, 5]\rangle$) in Example 1. For the relation instance in Fig. 8c, the corresponding instance graph w.r.t. the DD (on Address and City similarities) is presented in Fig. 8b. For instance, the edge between vertices 1 and 2 denotes that the similarities on Address and City of tuples 1 and 2, i.e., 3 and 0, are within the ranges [0, 6] and [0, 0], respectively. Figure 8a presents the constraint graph, referring to the similarity constraint [0, 5] over the Tel attribute.

Owing to the existence of inaccurate data, e.g., several digits are hidden (denoted by *) on Address in tuple 4, violations to the constraint appear. That is, tuples 4 and 5 have Address distance within [0,6] (having an edge between vertices 4 and 5), but their Tel values are dissimilar with distance >5 (their labels are not in neighborhood in the constraint graph).

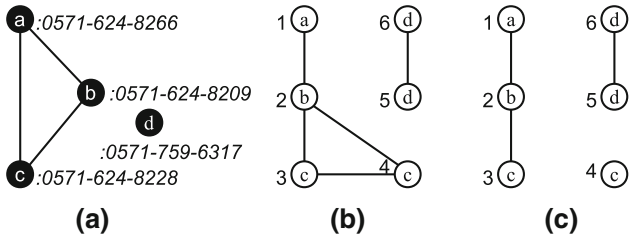


Fig. 9 Example of possible neighbor repairs. **a** Constraint, **b** repaired instance, **c** repaired instance

To eliminate the inconsistencies by simply using the previously proposed label repair, we need to relabel vertices 1, 2, 3 and 4 by label d: 0571-759-6317 or relabel vertices 5 and 6 by label c: 0571-624-8228. Obviously, such a repair fails to fix the inaccurate Address value (with hidden digits *) in tuple 4, but unnecessarily modifies a large number of Tel values.

Intuitively, to repair the inaccurate data in Fig. 8c, one may modify the Address value, instead of the Tel value. For instance, modify the Address value of tuple 4 by "No.721, West Lake Street". The neighborhoods (edges) in the corresponding instance graph change, i.e., replacing the neighbor set $\{2, 3, 5\}$ of vertex 4 in Fig. 8c by $\{2, 3\}$ in Fig. 9b. As shown in Fig. 9b, with the aforesaid repair on Address (neighbor repair), the violations are eliminated. □

7.2 Preliminaries

Motivated by the aforesaid neighbor repair example, we formally introduce the neighbor repair model, its repair cost function and the hardness with neighbor repairing.

7.2.1 Repair Model

Let $V(v) = \{u \mid (v, u) \in E\}$ denotes the set of vertices connected to vertex v, i.e., v's neighbors. While the graph relabel model considers label repair candidates from a domain of possible vertex labels, the graph neighbor repair model considers analogously the neighbor repair candidates that have been observed in other vertices, i.e., $V'(v) = V(u) \cup \{u\} \setminus \{v\}$ for some vertex u in the graph $\mathcal{G}$, or simply $V'(v) = \emptyset$ similar to the idea of allowing a modification outside the current domain [22].

Example 11 (Example 10 continued) Modifying the Address value of tuple 4 in the relation instance in Fig. 8c, from "***, West Lake Street" to "No.721, West Lake Street" (of tuple 3), it is equivalent to replace the neighbors of vertex 4 in the instance graph in Fig. 8b, from $V(4) = \{2, 3, 5\}$ to the neighbors of vertex 3, i.e., $\{2, 4\}$. Moreover, since the same Address value with distance 0 is also in the range of $[0, 6]$, vertex 3 (sharing the same Address value after

repairing) is also a neighbor of vertex 4. Consequently, as illustrated in Fig. 9b, the neighbors of vertex 4 after repairing is $V'(4) = V(3) \cup \{3\} \setminus \{4\} = \{2, 3\}$.

Moreover, referring to the idea of repairing to a fresh variable fv outside the currently known domain [22], i.e., dissimilar to the values of any existing tuples, no neighbor of the repaired vertex retains. For instance, another repair of tuple 4 is to assign the Address value by fv and consequently has $V'(4) = \emptyset$, as illustrated in Fig. 9c. □

7.2.2 Cost Function

The neighbor repair cost of changing vertex v's neighbors from $V(v)$ to $V'(v)$ is

$$\begin{aligned}\delta_n(V'(v), V(v)) &= |V(v) \cup V'(v)| - |V(v) \cap V'(v)| \\ &= |(V(v) \backslash V'(v)) \cup (V(v) \backslash V'(v))|, \end{aligned} \tag{11}$$

i.e., the difference of neighbor sets before and after repairing.

Considering the label repair cost $\delta_l\left(\lambda'(v), \lambda(v)\right)$, we define the total cost of graph repair by

$$\begin{aligned}\Delta(\mathcal{G}', \mathcal{G}) = (1-\theta) \sum_{v \in V} \delta_l\left(\lambda'(v), \lambda(v)\right) \\ + \theta \sum_{v \in V} \delta_n\left(V'(v), V(v)\right), \end{aligned} \tag{12}$$

where θ is a parameter on weighting the label and neighbor repair costs, $0 \leq \theta \leq 1$. (See Sect. 8.2.1 for a discussion on determining the θ weight by observing the repair costs in practice.)

Example 12 (Example 11 continued) As illustrate in Fig. 9, multiple neighbor repairs exist.

For the repair in Fig. 9b, the neighbor set of vertex 4 is changed from $V(4) = \{2, 3, 5\}$ to $V'(4) = \{2, 3\}$. Its neighbor repair cost is $\delta_n(V'(4), V(4)) = 1$ according to Eq. 11. It is notable that the neighbor set of vertex 5 is also changed in this repair, with cost $\delta_n(V'(5), V(5)) = 1$. Suppose that $\theta = 0.5$. Since no label repair is applied, the total repair cost is $\Delta(\mathcal{G}', \mathcal{G}) = 0.5 \times 2 = 1$ referring to Eq. 12.

For the repair in Fig. 9c, assigning fresh variable fv, the neighbor set of vertex 4 is changed from $V(4) = \{2, 3, 5\}$ to $V'(4) = \emptyset$. Its neighbor repair cost is $\delta_n(V'(4), V(4)) = 3$, which is higher than the aforesaid repair on vertex 4. Similarly, the costs $\delta_n(V'(2), V(2)) = 1$, $\delta_n(V'(3), V(3)) = 1$ and $\delta_n(V'(5), V(5)) = 1$ apply to vertices 2, 3 and 5, respectively. The total repair cost is $\Delta(\mathcal{G}', \mathcal{G}) = 0.5 \times 6 = 3$. □

7.2.3 Problem Analysis

The graph repairing problem is thus: For a constraint graph $\mathcal{S}$ and an instance graph $\mathcal{G}(V, E, \lambda)$, it is to find a repaired $\mathcal{G}'(V, E', \lambda')$ of $\mathcal{G}$ (with vertex label and neighbor repairs) such that $\mathcal{G}' \vDash \mathcal{S}$ and the total graph repair cost $\Delta(\mathcal{G}', \mathcal{G})$ is minimized. Graph repairing is different from the aforesaid vertex relabeling, which allows neighbor modification in addition to label changes. Therefore, the proof of Theorem 2 for relabeling does not directly apply to Theorem 9.

Theorem 9 *For a constraint graph $\mathcal{S}$, an instance graph $\mathcal{G}$ and a constant c, the problem of determining whether there exists a repaired $\mathcal{G}'$ of $\mathcal{G}$ (with vertex label and neighbor repairs) such that $\mathcal{G}' \vDash \mathcal{S}$ and the repairing cost $\Delta(\mathcal{G}', \mathcal{G}) \leq c$ is* NP*-complete.*

Proof The problem is clearly in NP. Given an instance $\mathcal{G}'(V, E', \lambda')$, it can be verified in polynomial time whether $\mathcal{G}'$ satisfies the constraint $\mathcal{S}$ and the cost of changing $\mathcal{G}(V, E, \lambda)$ to $\mathcal{G}'$ in Eq. 12 is no greater than c.

To prove the NP-hardness of the graph repairing problem, we show a reduction from the vertex cover problem, which is one of Karp's 21 NP-complete problems [21]. Given a graph $\mathcal{G}(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, a vertex cover is a subset $C \subseteq V$ of vertices such that for each edge $(v_i, v_j) \in E$, C contains at least one of v_i or v_j.

Similar to the proof of Theorem 2, we consider a constraint graph with $n + 1$ labels, $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$, as illustrated in Fig. 3. The constraint graph $\mathcal{S}(L, N)$ consists of n edges, $N = \{(\ell_0, \ell_1), \ldots, (\ell_0, \ell_n)\}$, where ℓ_0 serves a center node in this star constraint graph. We define $\delta_l(\ell_0, \ell_i) = 1$ for all $i \in [1, n]$. For the remaining label pairs between $\ell_i, \ell_j \in L, i \neq j$, we have $(\ell_i, \ell_j) \notin N$ in the label neighborhood constraints. The corresponding cost is $\delta_l(\ell_i, \ell_j) = d$, where $d > 1$.

$\mathcal{G}(V, E)$ in the vertex cover problem directly corresponds to the instance graph in the transformation, whose labeling is $\lambda(v_i) = \ell_i, i \in [1, n]$. It is notable that each edge is in violation, referring to the aforesaid $(\ell_i, \ell_j) \notin N, i \neq j$. Finally, we assign $\theta = 0.6$ in the repair cost function in Eq. 12. The transformation completes and can be done in polynomial time.

As illustrated below, the graph $\mathcal{G}$ has a vertex cover C of size $|C| \leq c$ if and only if there is a graph repair $\mathcal{G}'$ such that $\mathcal{G}' \vDash \mathcal{S}$ and $\Delta(\mathcal{G}', \mathcal{G}) \leq 0.4c$.

First, let C be a vertex cover with size c. For each edge $(v_i, v_j) \in E$, recall that $(\lambda(v_i), \lambda(v_j)) \notin N$ violates the constraint in $\mathcal{S}$. Let $v_i \in C$ be the vertex in the edge covered by C. We assign a new label $\lambda'(v_i) = \ell_0$ in the relabeled graph $\mathcal{G}'$, with the relabeling cost $\delta_l(\ell_0, \lambda(v_i)) = 1$. Since we have $(\ell_0, \ell_j) \in N$, $j \in [1, n]$ in the constraint graph $\mathcal{S}$, the violation with respect to the edge (v_i, v_j) is removed. Consequently, by considering all the vertices in C, we have $\mathcal{G}' \vDash \mathcal{S}$ and $\Delta(\mathcal{G}', \mathcal{G}) = 0.4c$, where no neighbor repair is performed.

Conversely, suppose that there exists a $\mathcal{G}'$ with cost $\Delta(\mathcal{G}', \mathcal{G}) = 0.4c$, having $c < |C^*|$, where C^* is a minimum

vertex cover. Let $V_n' \subseteq V$ be the set of vertices whose neighbors are changed in $\mathcal{G}'$, and $V_l' \subseteq V$ be the set of vertices whose labels are changed in $\mathcal{G}'$. We have

$$\Delta(\mathcal{G}', \mathcal{G}) = 0.4 \sum_{v \in V_l'} \delta_l\left(\lambda'(v), \lambda(v)\right) + 0.6 \sum_{v \in V_n'} \delta\left(V'(v), V(v)\right).$$

We build another repair $\mathcal{G}''$, where all the vertices in $V_l' \cup V_l'$ are relabeled to ℓ_0, and no neighbor repair is performed. Referring to $(\ell_0, \ell_j) \in N$, $j \in [1, n]$, repair $\mathcal{G}''$ also eliminates all the violations, having $\mathcal{G}'' \vDash \mathcal{S}$ and

$$\Delta(\mathcal{G}'', \mathcal{G}) = 0.4 \sum_{v \in V_l' \cup V_n'} \delta_l\left(\lambda'(v), \lambda(v)\right) = 0.4|V_l' \cup V_n'|.$$

Since any neighbor repair has cost $\delta\left(V'(v), V(v)\right) \geq 1$, it follows

$$0.4|V_l' \cup V_n'| = \Delta(\mathcal{G}'', \mathcal{G}) \leq \Delta(\mathcal{G}', \mathcal{G}) = 0.4c.$$

By relabeling all vertices in $V_l' \cup V_n'$ to ℓ_0, it eliminates all the violations in $\mathcal{G}$, i.e., $V_l' \cup V_n'$ is a vertex cover with size $|V_l' \cup V_n'| \leq c < |C^*|$, which is a contradiction. □

7.3 Approximation Algorithm

Intuitively, the approximation algorithm repairs one vertex each time with either label repair or neighbor repair. To ensure termination, it requires that all the violations to the vertex are eliminated after the repair operation.

Algorithm 3 presents the procedure of graph repair. Referring to weighting scheme in the total repair cost in Eq. 12, if $\theta\delta_n(V'(v), V(v)) > (1-\theta)\delta_l(\lambda'(v), \lambda(v))$, it chooses label repair; otherwise, neighbor repair. By always choosing the smaller one to perform the repair actions in Lines 6-9 in Algorithm 3, we can theoretically obtain an upper bound of repair cost $\Delta(\mathcal{G}', \mathcal{G})$ as proved in Lemma 11. In particular, we show in the proof of Lemma 11 that the repair cost in each step should be no greater than $2\theta r$, where θ is the weight of label and neighbor repairs in the graph repair cost defined in Eq. 12, and r is the maximum degree of $\mathcal{G}$. It leads to the constant-factor approximation bound, compared to the optimal repair, in Proposition 13 (given a degree-bounded graph).

Example 13 Consider the constraint and instance graphs in Fig. 10. Suppose that $\theta = 0.4$ and the cost of any label repair is 1.

According to Line 3 in Algorithm 3, vertex 2 involved with the most violations will be selected to repair first. To

Algorithm 3 Grepair($\mathcal{G}$, $\mathcal{S}$)

Input: An instance graph $\mathcal{G}$ and a constraint graph $\mathcal{S}$
Output: A repaired $\mathcal{G}$ satisfying $\mathcal{S}$
1: $\mathcal{G}_0 := \mathcal{G}$
2: **while** $\mathcal{G}$ not satisfying $\mathcal{S}$ **do**
3: $v :=$ vertex with maximum violations $|T(v, \lambda(v))|$
4: $V'(v) :=$ the neighbor repair eliminating all violations to v with the minimum $\delta_n(V'(v), V_0(v))$
5: $\lambda'(v) :=$ the label repair eliminating all violations to v (if exists) with the minimum $\delta_l(\lambda'(v), \lambda_0(v))$
6: **if** $\theta\delta_n(V'(v), V_0(v)) > (1-\theta)\delta_l(\lambda'(v), \lambda_0(v))$ **then**
7: update v with label repair $\lambda'(v)$ in $\mathcal{G}$
8: **else**
9: update v with neighbor repair $V'(v)$ in $\mathcal{G}$
10: **return** $\mathcal{G}$

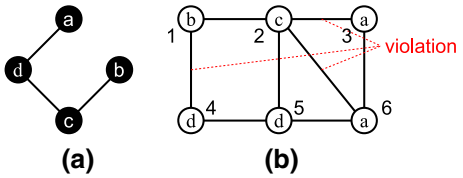


Fig. 10 Example for graph repairing. **a** Constraint, **b** instance

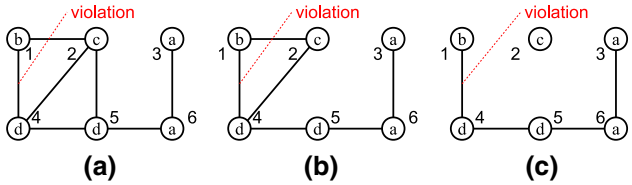


Fig. 11 Possible neighbor repairs on vertex 2

eliminate all the violations to vertex 2, there are three ways to repair vertex 2 using neighbor repairing: (1) replace its neighbors $V(2) = \{1, 3, 5, 6\}$ by vertex 4's neighbors (and vertex 4), having $V'(2) = \{1, 4, 5\}$ with cost 3 referring to Eq. 11, as illustrated Fig. 11a; (2) replace its neighbors by vertex 1's neighbors with cost 4, as shown in Fig. 11b; or (3) change its neighbor set to $\emptyset$ with cost 4, in Fig. 11c. Since there is no candidate label available for label repairing that eliminates all the violations to vertex 2, the neighbor repair with the minimum cost in Fig. 11a will be applied in Line 9.

After repairing vertex 2, as illustrated in Fig. 11a, vertex 4 is selected to repair. There are four possible neighbor repairs for vertex 4: (1) change its neighbors to vertex 5's (including vertex 5) in Fig. 11a, having $V'(4) = \{2, 5\}$ as shown in Fig. 12b, with cost 2 according to Equation 11 and the original $V(4) = \{1, 5\}$ in Fig. 10b; (2) change its neighbors to vertex 6's, having $V'(4) = \{3, 5, 6\}$ in Fig. 12c and cost 3; (3) change its neighbors to vertex 3's, having $V'(4) = \{3, 6\}$ in Fig. 12d and cost 4; or (4) change its neighbor set to $\emptyset$ with cost 2, in Fig. 12e. Moreover, in order to eliminate all the violations, the only way to repair vertex 4 by label repairing is to change its label from d to c with cost 1, as illustrated in Fig. 12a. Therefore, for vertex 4, the minimum label repair

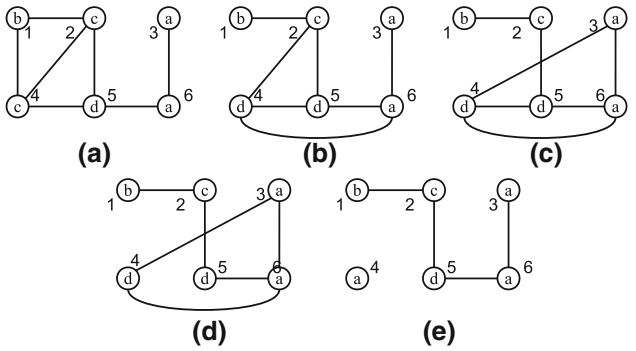


Fig. 12 Possible label (**a**) and neighbor (**b–e**) repair to vertex 4

cost is 1 and the minimum neighbor repair cost is 2. Referring to $0.4 \times 2 > 0.6 \times 1$ in Line 6, the algorithm will choose to relabel vertex 4 by c.

Finally, since all the violations are eliminated, the graph repairing algorithm will output Fig. 12a as the repaired graph of Fig. 10b. □

Proposition 10 *The* GREPAIR *algorithm always terminates, and runs in* $O(|V|^3)$ *time.*

Proof To illustrate the termination, we show that for each vertex v, there always exists a repair $V'(v) = \emptyset$ that eliminates all the violations to v. Therefore, by considering one vertex a time, the algorithm gradually eliminates all the violations and terminates.

To find the neighbor repair with the minimum cost for a vertex v, we need to compare its neighbors with the neighbors of other vertices u, with cost $O(|V|)$ for each comparison. Considering all the possible vertices, the neighbor repairing in each iteration needs $O(|V|^2)$. The minimum label repair can be found by simply comparing the labels of neighbors. Therefore, Algorithm 3 runs in $O(|V|^3)$ time. □

7.4 Performance Analysis

To evaluate the approximation performance, we study the upper (Lemma 11) and lower (Lemma 12) bounds of repair costs for possible repairs, from which the approximation ratio of Algorithm 3 is obtained (Proposition 13) w.r.t. a node degree-bounded graph.

Let $\mathcal{G}'$ be the approximate repair result returned by the GREPAIR algorithm.

Lemma 11 *An upper bound of repair cost of* $\mathcal{G}'$ *is*

$$\Delta(\mathcal{G}', \mathcal{G}) \leq 2\theta k r,$$

where k *is the number of vertices with violations in* $\mathcal{G}$*, and* r *is the maximum degree of* $\mathcal{G}$.

Proof Let $\mathcal{G}_i$ be the graph after the ith repairing step, $i = 0, 1, 2, \ldots$, where $\mathcal{G}_0 = \mathcal{G}$ denotes the original input. Let u be the vertex changed from $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$.

Case 1 According to Algorithm 3, when a label repair is conducted, we have $\theta\delta_n(V_i'(u), V_0(u)) > (1-\theta)\delta_l(\lambda_i(u), \lambda_0(u))$, where $V_i'(u)$ is the neighbor repair with the minimum cost. Since u is not previously repaired, it follows

$$\begin{aligned}\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0) &= (1-\theta)\delta_l(\lambda_i(u), \lambda_0(u)) \\ &< \theta\delta_n(V_i'(u), V_0(u)).\end{aligned}$$

For each neighbor repair, since there always exists a repair $V'(u) = \emptyset$ with repair cost no greater than r, we have $\delta_n(V_i'(u), V_0(u)) \leq r$ i.e.,

$$\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0) \leq \theta r.$$

Case 2 Otherwise, neighbor repair is performed, where only the neighbors of u and u's neighbors could be changed. We have

$$\begin{aligned}\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0) &= \theta \sum_{v \in V} \delta_n(V_i(v), V_0(v)) \\ &\quad - \theta \sum_{v \in V} \delta_n(V_{i-1}(v), V_0(v)) \\ &= \theta\Big(\sum_{v \in V \setminus \{u\}} (\delta_n(V_i(v), V_0(v)) \\ &\quad - \delta_n(V_{i-1}(v), V_0(v))) \\ &\quad + \delta_n(V_i(u), V_0(u)) \\ &\quad - \delta_n(V_{i-1}(u), V_0(u))\Big).\end{aligned}$$

According to the property of symmetric set difference and triangle inequality over the neighbor repair cost in Eq. 11, we have

$$\delta_n(V_i(v), V_0(v)) \leq \delta_n(V_{i-1}(v), V_0(v)) + \delta_n(V_i(v), V_{i-1}(v)).$$

It follows

$$\begin{aligned}\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0) &\leq \theta\Big(\sum_{v \in V \setminus \{u\}} \delta_n(V_i(v), V_{i-1}(v)) \\ &\quad + \delta_n(V_i(u), V_0(u)) - \delta_n(V_{i-1}(u), V_0(u))\Big).\end{aligned}$$

For $v \in V \setminus \{u\}$, according to the definition of neighbor repair on u from $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$, only the vertices in $V_i(u) \cup V_{i-1}(u) \setminus V_i(u) \cap V_{i-1}(u)$ will be affected, having $\delta_n(V_i(v), V_{i-1}(v)) = 1$. The total neighbor repair cost is bounded by $|V_i(u) \cup V_{i-1}(u) \setminus V_i(u) \cap V_{i-1}(u)|$, i.e.,

$$\begin{aligned}
&\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0) \\
&\quad \le \theta\Big(|V_i(u) \cup V_{i-1}(u) \backslash V_i(u) \cap V_{i-1}(u)| \\
&\qquad + \delta_n(V_i(u), V_0(u)) - \delta_n(V_{i-1}(u), V_0(u))\Big) \\
&\quad = \theta\Big(\delta_n(V_i(u), V_{i-1}(u)) \\
&\qquad + \delta_n(V_i(u), V_0(u)) - \delta_n(V_{i-1}(u), V_0(u))\Big).
\end{aligned}$$

According to the triangle inequality, i.e.,

$$\begin{aligned}
\delta_n(V_i(u), V_{i-1}(u)) \le\ & \delta_n(V_i(u), V_0(u)) \\
& + \delta_n(V_{i-1}(u), V_0(u)),
\end{aligned}$$

we have

$$\begin{aligned}
\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0) &\le 2\theta\delta_n(V_i(u), V_0(u)) \\
&\le 2\theta r,
\end{aligned}$$

referring again to $\delta_n(V_i(u), V_0(u)) \le r$.

Suppose that the graph repair algorithm performs q repair operations. Combining the aforesaid Cases 1 and 2, we have

$$\Delta(\mathcal{G}', \mathcal{G}) = \sum_{i=1}^{q} (\Delta(\mathcal{G}_i, \mathcal{G}_0) - \Delta(\mathcal{G}_{i-1}, \mathcal{G}_0)) \le q2\theta r.$$

Moreover, since each repair operation ensures eliminating the violations to the repaired vertex, the graph repair algorithm needs at most k operations, having $q \le k$. Finally, we have

$$\Delta(\mathcal{G}', \mathcal{G}) \le 2\theta k r.$$

The conclusion is proved. □

Let $\mathcal{G}^*$ be the optimal solution with the minimum repair cost.

Lemma 12 *A lower bound of repair cost of $\mathcal{G}^*$ is*

$$\Delta(\mathcal{G}^*, \mathcal{G}) \ge \frac{k}{r+1} \min\left\{(1-\theta)\delta_l^{\min}, \theta\right\},$$

where $\delta_l^{\min}$ is the minimum cost of label repair, k is the number of vertices with violations in $\mathcal{G}$, and r is the maximum degree of $\mathcal{G}$.

Proof Consider k vertices in $\mathcal{G}$ that are involved in violation. The degree of $\mathcal{G}$ is bounded by r. That is, by repairing any vertex, it eliminates the violations of at most $r+1$ vertices in k. To eliminate all the violations in k vertices, we need at least $\frac{k}{r+1}$ repair operations.

Let $\delta_l^{\min}$ and $\delta_n^{\min}$ denote the minimum cost of a label repair and a neighbor repair, respectively. Thereby, the minimum cost of a repair operation is thus

$$\min\left\{(1-\theta)\delta_l^{\min}, \theta\delta_n^{\min}\right\}$$

To sum up, for any repair of $\mathcal{G}$, including the optimal solution $\mathcal{G}^*$, we have

$$\Delta(\mathcal{G}^*, \mathcal{G}) \ge \frac{k}{r+1} \min\left\{(1-\theta)\delta_l^{\min}, \theta\delta_n^{\min}\right\}.$$

Moreover, each neighbor repair modifies at least one neighbor, i.e., $\delta_n^{\min} \ge 1$. The conclusion is proved. □

Combining the conclusions in Lemmas 11 and 12, it naturally leads to the following approximation bound.

Proposition 13 *The* GREPAIR *approximation algorithm outputs a $\mathcal{G}'$ having*

$$\frac{\Delta(\mathcal{G}', \mathcal{G})}{\Delta(\mathcal{G}^*, \mathcal{G})} \le 2r(r+1) \max\left\{\frac{\theta}{(1-\theta)\delta_l^{\min}}, 1\right\},$$

where r is maximum degree of instance graph and $\delta_l^{\min}$ is the minimum cost of label repair.

When the count function is used for label repair [22], as introduced in Sect. 3.2 having $\delta_l^{\min} \ge 1$, we have the approximation bound $2r(r+1) \max\left\{\frac{\theta}{1-\theta}, 1\right\}$.

Referring to Proposition 10, given a degree-bounded instance graph, Algorithm 3 is a constant-factor cubic-time approximation.

8 Experiments

This section reports the experiments of proposed methods on real datasets. All the algorithms are implemented in Java. The program runs on a server with four 2.67GHz CPUs and 128GB main memory.

8.1 Experiments on Label Repair

Since we do not have real errors with labeled truth in most datasets (except coauthor networks and GPS data below), following the same line of experimentally evaluating data repairing methods [15], artificial errors are introduced to the clean data. That is, we first randomly draw k vertices from the n vertices in the graph, where k is the number of label frauds to introduce. The labels of these k vertices are then replaced by randomly using the labels of other nodes in the graph. The relabeling methods are then applied

to repair graph labels. Instead of observing approximation ratio by computing the optimal solution, which is indeed not affordable for the large real data, we study the accuracy of relabeling results by comparing with the truth of fraud data previously replaced. In particular, let truth be the set of (vertex, label) pairs that are randomly replaced in $\mathcal{G}$, i.e., the original true data. Let found be the set of (vertex, label) pairs that are relabeled in $\mathcal{G}'$, i.e., relabeling results. To evaluate the accuracy, we use f-measure of precision and recall [32], given by $precision = \frac{|\text{truth} \cap \text{found}|}{|\text{found}|}$, $recall = \frac{|\text{truth} \cap \text{found}|}{|\text{truth}|}$, and $f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. It is natural that higher f-measure is preferred. Besides accuracy, we also observe the time costs of approaches.

8.1.1 Experiments on Similarity Networks

To evaluate our proposed relabeling techniques over similarity networks, we implement an existing repairing method based on FD [3] (does not rely on the proposed notations of graphs). The FD-based repairing [3] performs directly on the relational data (e.g., in Fig. 1c) rather than the derived similarity network (i.e., the corresponding Fig. 1b). Therefore, it cannot be applied to other graphs such as coauthor networks without relational settings.

Restaurant similarity network is a collection of 864 restaurant records[3] that contains 112 duplicates and is widely used for record matching [19]. We consider a DD, $(\text{Name}, \text{Address} \rightarrow \text{Areacode}, \langle [0,0], [0,6], [0,0] \rangle)$, and a corresponding FD $(\text{Name}, \text{Address} \rightarrow \text{Areacode})$. Existing approach [29] is employed for discovering the DD and determining the distance/similarity thresholds . Specifically, a support measure is used in DD discovery to evaluate the proportion of tuple pairs in the data whose distances satisfy the thresholds such as [0,6]. Among various candidate DDs with different thresholds, the DD, which has the largest support and is valid w.r.t. satisfaction over the data, will be returned in discovery and employed in our study.

Following the steps in the introduction, we construct similarity networks w.r.t. the DD. Note that the distance constraint on Areacode in the DD is $[0, 0]$, i.e., equality. Therefore, the constructed constraint graph belongs to clique constraints where transitivity is applicable. Given a DD, we need one pass through the tuple pairs in the data to construct both the instance and constraint graphs. The time cost of instance graph construction by comparing tuples on Name and Address attributes specified in the DD is 1473 ms. And similarly, the time cost for constraint graph on comparing Areacode attribute is 105 ms. The total cost (1578 ms) of constructing the instance and constraint graphs in preprocessing is much higher than the time cost of repairing as illustrated in Fig. 14. The result is not surprising, since repairing only needs to process the vertices involved in violations.

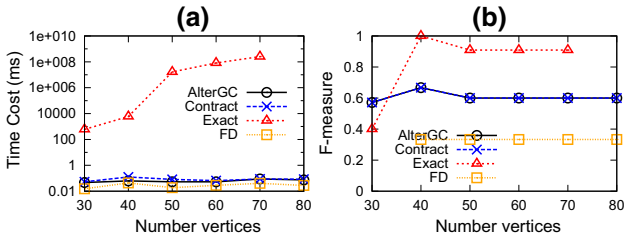


Fig. 13 Restaurant with various data sizes. **a** Time performance, **b** accuracy performance

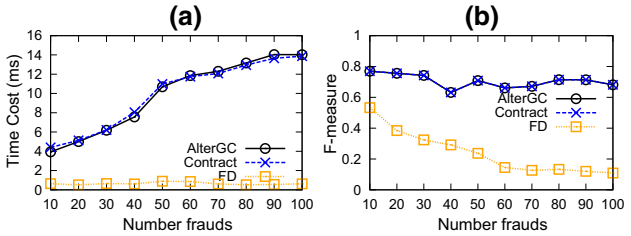


Fig. 14 Restaurant with various inserted frauds. **a** Time performance, **b** accuracy performance

First, in Fig. 13, we compare the approximation methods with the exact algorithm (implemented following the standard branching and bound strategy). Unfortunately, even for such a small Restaurant dataset, conducting the exact algorithm over the entire dataset is unlikely. As illustrated, it already takes more than 250,000 s (about 70 h) for a subset of 70 vertices. Considering the NP-hardness of computing the optimal solution (Proposition 1), we did not carry on the experiments on larger data sizes for evaluating the exact relabeling program. For such a small data size, we consider a number of five inserted frauds, and thus the results may not be stable in such a small sample. As shown, the exact algorithm could achieve better accuracy but with extremely higher time costs.

While the greedy method fails to terminate (see more discussion below), we report the results by AlterGC, contract and the existing FD-based repairing [3], in Fig. 14. Since dataset is small, we mainly observe performance variances by increasing the number of inserted frauds from 10 to 100. Generally, with the growth of frauds, the f-measure accuracy drops, while the relabeling time cost increases. Since frauds are randomly inserted, time costs and f-measure may not strictly grow or decrease with the increase of frauds. As illustrated, AlterGC and contract approaches show almost the same results, since the greedy technique fails to work.

Recall that the major superiority of our proposed graph relabeling approach is the ability of considering more similar neighbors in similarity networks than the FD repairing which only considers a limited number of tuples with equal-

[3] http://www.cs.utexas.edu/users/ml/riddle/data.html.

ity relationships. For the same reason analyzed in Example 1, owing to the rigid equality, the existing FD repairing fails to detect the relationships of tuples with small variations, which are successfully captured by similarity networks. Since more repair candidates can be suggested, as shown in Fig. 14b, the accuracy of the proposed similarity network-based AlterGC approach is significantly higher than that of FD.

It is true that the current method can repair only the fraction of the data that is within the scope of matching attributes of a dd, since each distinct DD corresponds to a different instance graph (as well as the constraint graph). For the application over large graphs (of similarity networks), multiple DDs need to be specified. Each DD may lead to distinct constraint and instance graphs. The graph repairing over multiple instance graphs under various constraint graphs is challenging. The repairing could not be performed independently over these instance graphs, since the same vertex in different instance graphs should have the same label. We leave this interesting yet challenging problem on repairing multiple correlated graphs as the future study. Nevertheless, for other types of networks, the evaluation over large graphs is performed in Sect. 8.1.4. As shown, the results over large HPRD network in Fig. 18 are generally similar to those in Fig. 14 on similarity network.

8.1.2 Experiments on Coauthor Networks

Coauthor Network is motivated by the entity resolution task [18]. We note that there are different authors sharing the same name, e.g., Lei Chen (HKUST), Lei Chen (Wisconsin), Lei Chen (Purdue), Lei Chen (RPI). In data sources such as CiteSeer, different authors sharing the same name (Lei Chen) are not fully distinguished. We employ the coauthor relationships from DBLP[4], where different Lei Chen(s) are distinguished, and model them as the constraint graph. As shown in Fig. 15, in the instance graph, each author in a citation record (from CiteSeer) denotes a vertex. An edge denotes that this author (in the record) coauthors with another in the same citation record. The relabeling problem is to find the "right" label, e.g., Lei Chen (HKUST) or Lei Chen (Wisconsin), for the original imprecise one (simply Lei Chen without identification).

While the CiteSeer data are used as the instance graph, we consider the DBLP data as the constraint graph (which are manually identified and maintained to a great extent). If such pre-maintained constraint graph is not available, we may employ the entity resolution in graph data [2] to construct the constraint graph. It is worth noting that Bhattacharya and Getoor [2] consider two types graphs, reference graph and entity graph, which are analogous to instance graph and constraint graph in our study, respectively. In particular, the

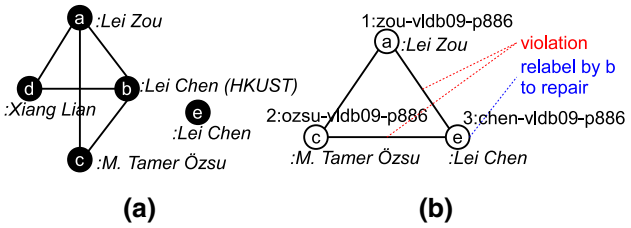


Fig. 15 Example of coauthor networks. **a** Constraint (from DBLP), **b** instance (from CiteSeer)

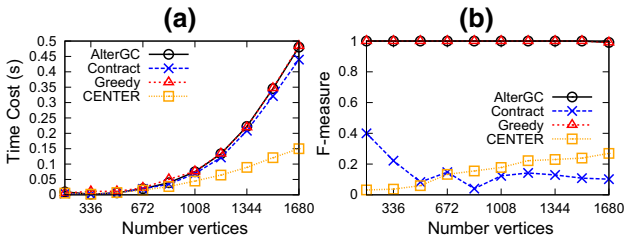


Fig. 16 Coauthor networks with various sizes. **a** Time performance, **b** accuracy performance

entity graph describes the relationships among the identified entities. The entity resolution in graph data is thus to build the entity graph (constraint graph) from the reference graph (instance graph). Once the constraint graph (entity graph) is obtained, we may similarly apply it to repair other instance graphs (from different sources), eliminating errors on either labels or coauthor relationships in Sect. 8.2.

To evaluate the label reparing, we adapt the existing duplicate detection methods. Since [2] considers complex hyper-graphs with multiple types of authors and papers having citation relationships, which are not available in our scenario, we employ a simpler CENTER approach [18] based on clustering with coauthor similarity. Each author corresponds to the center node of a cluster in CiteSeer, and the coauthor list of the author from DBLP is used as the features in classification. To evaluate the accuracy, the publication records in DBLP (with different authors sharing the same name fully distinguished) are used as the ground truth of the corresponding citation records in CiteSeer. Again, the existing CENTER method cannot be employed to repair other graphs such as HPRD without duplicates.

As shown in Fig. 16, f-measure is improved from 0.3 (of CENTER) to 0.98 by our proposed AlterGC method. With both the constraint (DBLP) and the instance (CiteSeer with imprecise labels embedded) from truly real datasets without any manual manipulation, this experiment demonstrates the superiority of our proposal in dealing with real imprecise values.

8.1.3 Experiments on Location Networks

The GPS dataset with real errors is collected by a person carrying a smartphone and walking around at campus. All

[4] http://dblp.uni-trier.de/db.

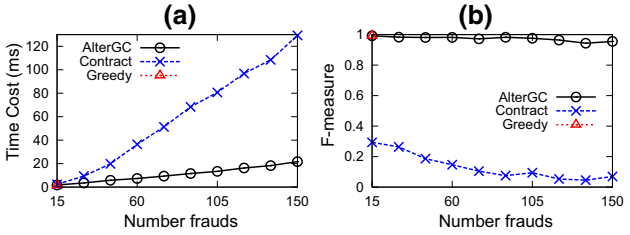


Fig. 17 GPS with various inserted frauds. **a** Time performance, **b** accuracy performance

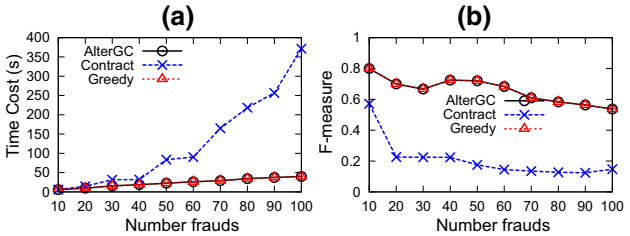


Fig. 18 HPRD with various inserted frauds. **a** Time performance, **b** accuracy performance

the GPS readings are discretized into 100 locations. Since we know exactly the path of walking, a number of 150 dirty points are manually identified (among total 2358 clear points in the trajectory). True locations of dirty points are also manually labeled, as ground truth. Referring to the walking speed, for any two points collected in 1 min, they should be in the same or adjacent locations. The instance graph thus consists of reading points as vertices and edges on two points with timestamp difference <1 min. The constraint graph uses locations as nodes (of labels) and neighborhoods denote adjacent locations. The repairing is thus to modify the location labels of reading points to make them satisfy the adjacent constraints.

Figure 17 presents the results on various numbers of frauds. The accuracy result is generally similar to Fig. 14b over Restaurant with artificial errors. That is, the greedy algorithm cannot return results in most tests, while the contract method always return repairs with low accuracy. Nevertheless, the AlterGC approach is effective with high f-measure. Moreover, the time cost in Fig. 17a is similar to Fig. 18a over HPRD data, where the AlterGC method can achieve significantly lower time costs by avoiding unnecessary contraction operations.

8.1.4 Experiments on Larger Datasets

HPRD[5], Human Protein Reference Database, consists of a human protein interaction network. It is often used as a massive network, e.g., for finding maximal clique [5]. As introduced in Sect. 1, edges denote binary protein–protein interactions. Protein's GO term is used as vertex label. The

[5] http://www.hprd.org/download

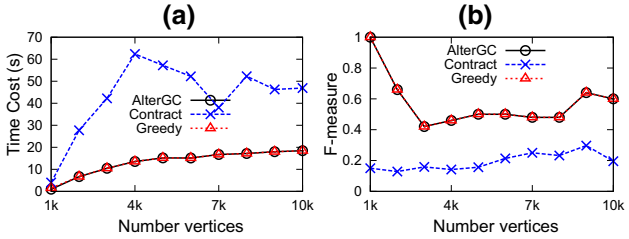


Fig. 19 HPRD varying graph sizes. **a** Time performance, **b** accuracy performance

constraint graph represents GO term correlations. Since the GO term *cellular component* is correlated with all the others, it is a star constraint.

Since there are no existing methods for repairing over these datasets, we focus on comparing the techniques proposed in this paper. Greedy heuristics performs well, i.e., with high accuracy and low time cost in Fig. 18, if it can terminate. Although the contraction method can always terminate in all the tests, as presented in Fig. 18b, the f-measure accuracy could be bad, which verifies our analysis of the contraction approach in Sect. 5. Nevertheless, as illustrated in Fig. 18b, the f-measure of AlterGC approach is as good as that of the greedy method in almost all the tests (where the greedy algorithm can terminate). The AlterGC approach also guarantees termination as the contraction method does, while the corresponding accuracy is as good as the greedy heuristics one.

Figure 19 reports the performance of scalability by increasing the number of vertices (test beds with smaller sizes of n vertices are prepared by using the first n vertices listed in the data). As shown, the time performance scales well, which grows almost proportionally with data sizes. The results verify the complexity analysis of contract and AlterGC algorithms in Sects. 5 and 6, respectively.

Specific tests may zigzag such as Fig. 19 for two reasons: (1) As mentioned, the randomly inserted frauds in different tests may affect performance results. (2) The heuristic relabeling steps could be easily influenced by a small difference among tests. Consequently, the f-measure accuracy may vary greatly among different data sizes due to the random insertion of frauds, for instance, as presented in Fig. 19b.

Summary: (1) Experiments in Figs. 13 and 14 illustrate that AlterGC and contraction methods show similar/better performance compared with existing techniques, while the greedy technique fails and has no much effect in the AlterGC approach. (2) Figures 16, 18 and 19 demonstrate that AlterGC and greedy approaches show similar and better results than that of the contraction method. In short, by taking advantages in both techniques, the AlterGC approach always has the best performance in all the experiments.

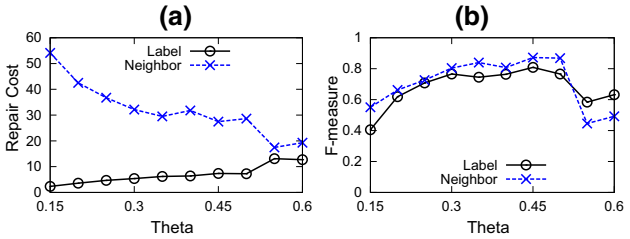


Fig. 20 Restaurant with various θ. **a** Repair cost, **b** accuracy performance

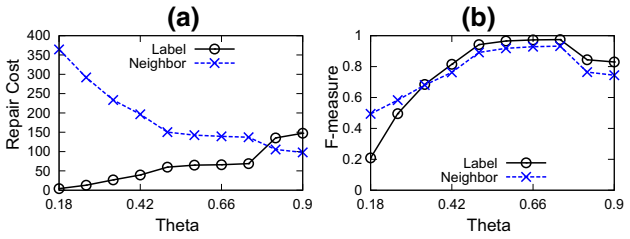


Fig. 21 Coauthor with various θ. **a** Repair cost **b** accuracy performance

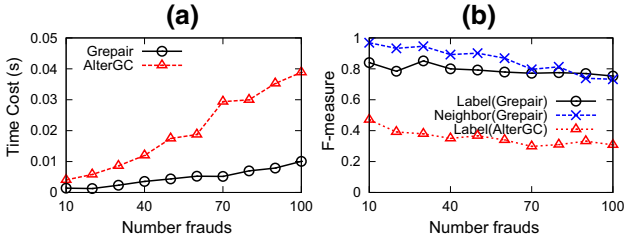


Fig. 22 Restaurant with various inserted frauds and $\theta = 0.45$. **a** Time performance, **b** accuracy performance

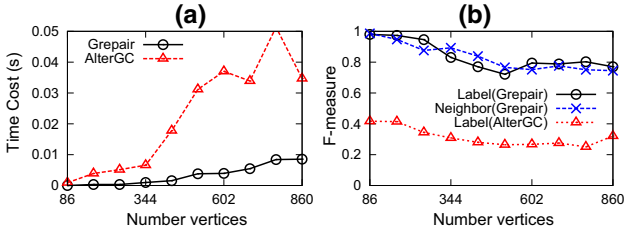


Fig. 23 Restaurant with various data sizes and $\theta = 0.45$. **a** Time performance, **b** accuracy performance

8.2 Experiments on Graph Repair

In this experiment, we evaluate graph repair with neighbor modification, proposed in Sect. 7. Besides label errors as introduced in the previous experiment in Sect. 8.1, we also consider neighbor errors. Again, we randomly draw k vertices from the n vertices in the graph, where k is the number of neighbor frauds to introduce. Neighbor errors on these k vertices are introduced by randomly replacing their neighbors with some other vertices' neighbors in the graph (referring to the motivation Example 10).

Let E_{original} denote the neighborhoods (edges) in the original instance graph, E_{injected} be the neighborhoods with injected neighbor errors, and E_{repaired} be the neighborhoods in the repaired graph. We define $\mathsf{truth} = (E_{\text{original}} \backslash E_{\text{injected}}) \cup (E_{\text{injected}} \backslash E_{\text{original}})$ the set of neighborhoods that are modified in neighbor error introducing, and $\mathsf{found} = (E_{\text{repaired}} \backslash E_{\text{injected}}) \cup (E_{\text{injected}} \backslash E_{\text{repaired}})$ the set of neighborhoods that are modified in neighbor repair. Consequently, $\mathsf{truth} \cap \mathsf{found}$ is the set of neighborhoods that are modified in neighbor error introducing and successfully repaired later. Again, the f-measure of precision and recall over truth and found is employed.

8.2.1 Evaluating Cost Weight Parameter θ

This experiment evaluates various θ values in weighting label and neighbor repair costs, in Eq. 12.

Figures 20 and 21 report both the label and neighbor repair accuracies, as well as the corresponding repair costs, over different datasets. Both the numbers of label and neighbor errors are 30 (see results on various errors sizes below).

First, when θ is small, according to the graph repair cost function in Eq. 12, neighbor repair is preferred. Therefore, a larger number of neighbor repair operations are performed with higher cost (compared to label repair), e.g., as shown in Fig. 20a.

With the increase of θ, more label repair will be performed and neighbor repair operation (cost) reduces, in Fig. 20a. However, with an excessively large θ, the label repair may be over-emphasized relative to neighbor repair. The neighbor repair operation (cost) significantly drops, e.g., after $\theta = 0.5$ in Fig. 20a.

Nevertheless, with a proper θ, we may obtain a repair with both high accuracies of neighbor and label repairing, as shown in Fig. 20b. To practically determine a proper θ, we can observe the corresponding neighbor and repair costs in Fig. 20a. After the significant decrease/increase of neighbor/label repair costs, the repair costs become stable in the range of $\theta \in [0.3, 0.5]$, before the significant decrease/increase appearing again. In such stable range, the neighbor and label repairs are balanced, and thus the corresponding repair accuracies are the highest. Similar results are also observed in other datasets in Fig. 21.

8.2.2 Evaluation on Scalability

Next, we observe the scalability over various fraud sizes and data sizes. Figures 22 and 25 report the repair accuracy and the corresponding time cost on various sizes of errors. Figures 23 and 26 present the results on various sizes of graphs.

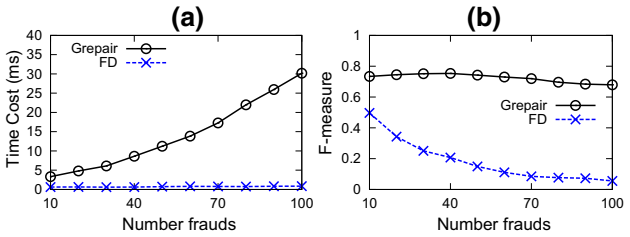


Fig. 24 Repairing on LHS and RHS over Restaurant. **a** Time performance, **b** accuracy performance

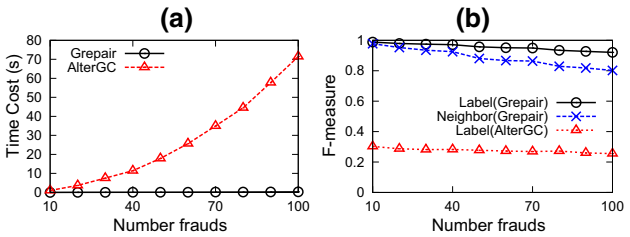


Fig. 25 Coauthor with various inserted frauds and $\theta = 0.7$. **a** Time performance, **b** accuracy performance

First, as shown in all the aforesaid figures, the label repair accuracy of Grepair (Algorithm 3) is significantly higher than that of AlterGC (Algorithm 2). The reason is that both neighbor and label frauds exist in the graphs. While Grepair could handle both errors separately, the AlterGC algorithm performs label repair only to eliminate the violations introduced by neighbor errors. Indeed, it is also the reason why the AlterGC method needs significantly higher time costs. Since neighbor errors exist, which might not be eliminated by the greedy label repair, the high cost contraction operations are performed.

For the Restaurant dataset captured from relational data, as illustrated in the motivation Example 11, the neighbor repair is indeed equivalent to repairing the left-hand-side (LHS) attribute values in a dependency (DD), while the label repair corresponds to repairing the right-hand-side (RHS) attribute values. Therefore, in addition to neighbor repair accuracy, in Fig. 24, we also report the accuracy of repairing both LHS and RHS attribute values over the relational data Restaurant. Again, the FD-based repairing [3] (on both LHS and RHS attributes) is compared. Similar to Fig. 14 with RHS errors only, the FD-based repairing with strict equality constraints may not be effective in addressing errors over the data with various information formats. By considering similarity relationships, the proposed Grepair achieves higher accuracy.

Since frauds are randomly introduced in the graphs, we repeat each test ten times in all the experiments. In addition to the average of f-measures, Figure 27 reports the variance of f-measures in the repeated tests, i.e., $\sigma_y^2 = \frac{1}{10}\sum_{i=1}^{10}(y_i - \bar{y})^2$, where y_i is the f-measure in each test i and $\bar{y}$ is the mean of f-measures in ten tests. It is not surprising that f-measure on neighbor repair has larger variance than that of label repair, since a fraud vertex may have a set of neighbors but only one label. Nevertheless, in Fig. 27, the variance is generally low in all the experiments. Similar results are also observed in all the other experiments (and thus omitted).

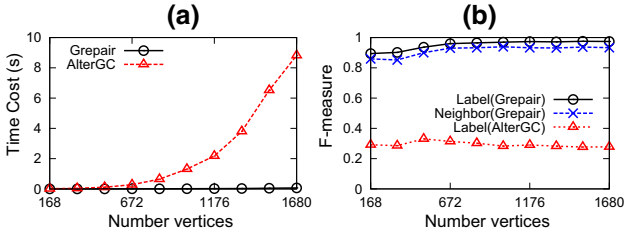


Fig. 26 Coauthor with various data sizes and $\theta = 0.7$. **a** Time performance, **b** accuracy performance

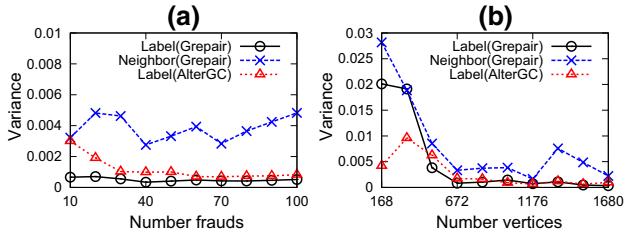


Fig. 27 Variance of f-measures in Figs. 25a and 26a

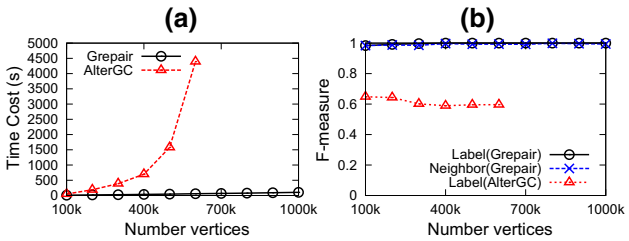


Fig. 28 Scalability over synthetic data. **a** Time performance, **b** accuracy performance

Finally, to evaluate the methods over even larger data sizes, we employ the widely used graph generation tool [6,20], GraphGen[6], to generate graphs with up to 1000 k vertices. Figure 28 illustrates the results, which are generally similar to those in Figs. 23 and 26 over real data sets. That is, without addressing neighbor errors, the accuracy of AlterGC method is low, while the corresponding time cost is high. (Owing the extremely high time cost and obviously lower accuracy, we omit the results of AlterGC over larger data sizes.) Nevertheless, the Grepair algorithm, considering both label and neighbor errors, has high accuracy on both label and neighbor repairs, in Fig. 28b.

Summary: (1) With a proper θ, determined by observing the corresponding repair costs, the neighbor and label repairs

[6] www.cse.ust.hk/graphgen

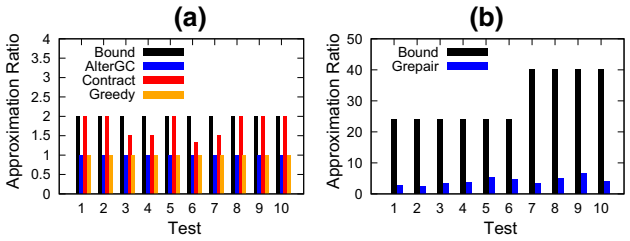


Fig. 29 Approximation ratio over synthetic data. **a** Label repair, **b** label and neighbor repair

could be balanced with the highest repair accuracy. (2) When neighbor errors exist, the Grepair algorithm with both neighbor and label repairing shows better accuracy and time performance as well, compared to the method with only label repairing.

8.3 Experiments on Approximation Performance

In order to evaluate the performance of the proposed approximation methods, we employ a synthetic dataset to observe the difference between the optimal repairs and the returned approximate answers. In particular, we verify the bounds of constant-factor approximation in Proposition 6 for label repairing, and Proposition 13 for graph repairing.

We employ again the graph generation tool GraphGen to generate ten test beds with 9–13 vertices in instance graphs and 6–11 vertices in constraint graphs. The evaluation focuses on the approximation ratio of approximate answers compared to the optimal one. We compute the optimal label repair $\mathcal{G}_l^*$ with the minimum relabeling cost $\Delta_l(\mathcal{G}_l^*, \mathcal{G})$, and the optimal graph repair $\mathcal{G}^*$ with the minimum graph repair cost $\Delta(\mathcal{G}^*, \mathcal{G})$. For any approximate label repair $\mathcal{G}_l'$ and approximate graph repair $\mathcal{G}'$, the approximation ratios are reported by $\frac{\Delta_l(\mathcal{G}_l', \mathcal{G})}{\Delta_l(\mathcal{G}_l^*, \mathcal{G})}$ and $\frac{\Delta(\mathcal{G}', \mathcal{G})}{\Delta(\mathcal{G}^*, \mathcal{G})}$.

Figure 29 illustrates the approximation performance of the proposed methods, as well as the corresponding theoretical bounds of approximation ratio presented in Propositions 6 and 13, respectively. Star constraints are considered in Fig. 29a in order to verify the results in Proposition 6. As shown, contraction approach terminate in all the ten tests, and the approximation ratio is no >2. These results verify our conclusion for the contraction method in Proposition 6. Remarkably, the AlterGC method can achieve much better approximation ratio performance (equal to 1 in many tests) than contraction while still guaranteeing termination. For graph repairing on both labels and neighbors in Fig. 29b, the approximation ratio of Grepair (Algorithm 3) is also lower than the theoretical bound. The result in Proposition 13 is verified as well.

9 Conclusions

This paper studies a novel problem of repairing vertex labels and neighbors under the constraints of label neighborhood. Graph constraint satisfaction and repairing have many application scenarios, ranging from similarity networks w.r.t. integrity constraints involving distance metrics, workflow networks of business processes, to protein interaction networks. First, the vertex label repairing (relabeling) problem is generally hard. Spreads of violations during relabeling prevent the approximation methods performing. We show that greedy heuristics cannot guarantee termination. Therefore, a contraction-based relabeling method is devised, which can always terminate, but may have bad results in terms of relabeling cost. For the special case of star constraints, however, both methods perform surprisingly good, where the greedy method theoretically guarantees termination and the contraction approach turns out to be factor-2 approximation. Nevertheless, to put together the beauty of violation elimination heuristics and termination, we present an approach AlterGC by cooperating contraction and greedy relabeling. Moreover, the graph repairing problem considers both vertex label and neighbor modifications. We analyze its NP-hardness and present a constant-factor cubic-time approximation algorithm (given degree-bounded instance graphs).

Experimental evaluation on real data demonstrates that the AlterGC approach always takes the advantages of either the proposed greedy or contraction techniques. Moreover, when both label and neighbor errors exist, the proposed Grepair algorithm shows both higher repair accuracy and better time performance.

In addition to the "allowed" semantics studied in this paper, further considering the "required" semantics in the neighborhood constraints is interesting. For instance, in RDF data, a vertex with Name value Albert Einstein is required to have a neighbor with Born value 1879. Repairing under such complicated constraints will obviously be more challenging. We leave this interesting yet more challenging work as future study.

Acknowledgements This work is supported in part by National Key Research Program of China under Grant 2016YFB1001101; China NSFC under Grants 61572272, 61325008, 61370055, 61672313 and 61202008; Tsinghua University Initiative Scientific Research Program.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In PODS, pp. 68–79 (1999)
2. Bhattacharya, I., Getoor, L.: Entity Resolution in Graph Data. University of Maryland technical report CS-TR-4758 (2005)
3. Bohannon, P., Flaster, M., Fan, W., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In SIGMOD Conference pp. 143–154 (2005)

4. Boobna, U., de Rougemont, M.: Correctors for XML data. In XSym, pp. 97–111 (2004)
5. Cheng, J., Ke, Y., Fu, A.W.-C., Yu, J.X., Zhu, L.: Finding maximal cliques in massive networks by h*-graph. In: SIGMOD Conference pp. 447–458 (2010)
6. Cheng, J., Ke, Y., Ng, W., Lu, A.: Fg-index: towards verification-free query processing on graph databases. In: SIGMOD Conference pp. 857–872, (2007)
7. Cheng, J., Yu, J.X., Ding, B., Yu, P.S., Wang, H.: Fast graph pattern matching. In ICDE, pp. 913–922 (2008)
8. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1–2), 90–121 (2005)
9. Conesa, A., Götz, S., García-Gómez, J.M., Terol, J., Talón, M., Robles, M.: Blast2go: a universal tool for annotation, visualization and analysis in functional genomics research. Bioinformatics **21**(18), 3674–3676 (2005)
10. Dinur, I., Safra, S.: The importance of being biased. In: STOC. pp. 33–42 (2002)
11. Fan, W., Fan, Z., Tian, C., Dong, X.L.: Keys for graphs. PVLDB **8**(12), 1590–1601 (2015)
12. Fan, W., Jia, X., Li, J., Ma, S.: Reasoning about record matching rules. PVLDB **2**(1), 407–418 (2009)
13. Fan, W., Li, J., Luo, J., Tan, Z., Wang, X., Wu, Y.: Incremental graph pattern matching. In: SIGMOD Conference pp. 925–936 (2011)
14. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.: Graph pattern matching: from intractable to polynomial time. PVLDB **3**(1), 264–275 (2010)
15. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. PVLDB **3**(1), 173–184 (2010)
16. Flesca, S., Furfaro, F., Parisi, F.: Querying and repairing inconsistent numerical databases. ACM Trans. Database Syst. **35**(2), 14 (2010)
17. Gilchrist, M.A., Salter, L.A., Wagner, A.: A statistical framework for combining and interpreting proteomic datasets. Bioinformatics **20**(5), 689–700 (2004)
18. Hassanzadeh, O., Chiang, F., Miller, R.J., Lee, H.C.: Framework for evaluating clustering algorithms in duplicate detection. PVLDB **2**(1), 1282–1293 (2009)
19. Isele, R., Bizer, C.: Learning expressive linkage rules using genetic programming. PVLDB **5**(11), 1638–1649 (2012)
20. Jin, C., Bhowmick, S.S., Xiao, X., Cheng, J., Choi, B.: Gblender: towards blending visual query formulation and query processing in graph databases. In: SIGMOD Conference pp. 111–122 (2010)
21. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, Berlin (1972)
22. Kolahi, S., Lakshmanan, L.V.S.: On approximating optimum repairs for functional dependency violations. In: ICDT, pp. 53–62 (2009)
23. Koudas, N., Saha, A., Srivastava, D., Venkatasubramanian, S.: Metric functional dependencies. In: ICDE, pp. 1275–1278 (2009)
24. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. In: AAAI, pp. 17–24 (1990)
25. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
26. Song, S., Cao, Y., Wang, J.: Cleaning timestamps with temporal constraints. PVLDB **9**(10), 708–719 (2016)
27. Song, S., Chen, L.: Differential dependencies: reasoning and discovery. ACM Trans. Database Syst. **36**(3), 16 (2011)
28. Song, S., Chen, L., Cheng, H.: Parameter-free determination of distance thresholds for metric distance constraints. In: ICDE, pp. 846–857 (2012)
29. Song, S., Chen, L., Cheng, H.: Efficient determination of distance thresholds for differential dependencies. IEEE Trans. Knowl. Data Eng. **26**(9), 2179–2192 (2014)
30. Song, S., Cheng, H., Yu, J.X., Chen, L.: Repairing vertex labels under neighborhood constraints. PVLDB **7**(11), 987–998 (2014)
31. Suzuki, N.: Finding an optimum edit script between an XML document and a DTD. In: SAC, pp. 647–653 (2005)
32. van Rijsbergen, C.J.: Information Retrieval. Butterworth, Oxford (1979)
33. Vazirani, V.V.: Approximation Algorithms. Springer, Berlin (2001)
34. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: a graph repair approach. In: ICDE, pp. 30–41 (2015)
35. Wijsen, J.: Database repairing using updates. ACM Trans. Database Syst. **30**(3), 722–768 (2005)
36. Zhang, A., Song, S., Wang, J.: Sequential data cleaning: a statistical approach. In: SIGMOD Conference, pp. 909–924 (2016)
37. Zhang, B., Park, B.-H., Karpinets, T.V., Samatova, N.F.: From pull-down data to protein interaction networks and complexes with biological relevance. Bioinformatics **24**(7), 979–986 (2008)
38. Zhu, X., Song, S., Lian, X., Wang, J., Zou, L.: Matching heterogeneous event data. In: SIGMOD Conference, pp. 1211–1222 (2014)
39. Zhu, X., Song, S., Wang, J., Yu, P.S., Sun, J.: Matching heterogeneous events with patterns. In: ICDE, pp. 376–387 (2014)