Repairing Vertex Labels under Neighborhood Constraints

Shaoxu Song[†] Hong Cheng[§] Jeffrey Xu Yu[§] Lei Chen[‡]

[†]KLiss, MoE; TNList; School of Software, Tsinghua University, China sxsong@tsinghua.edu.cn

[§]The Chinese University of Hong Kong, China {hcheng,yu}@se.cuhk.edu.hk

[‡]The Hong Kong University of Science and Technology, China leichen@cse.ust.hk

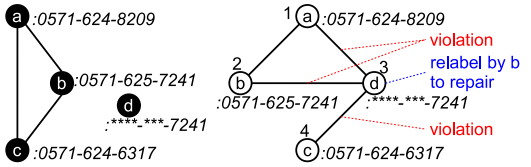
ABSTRACT

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. The vertex labels (data values) are often 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 vertexes in the graph, are employed to detect and repair erroneous vertex labels. In this paper, we study the problem of repairing vertex labels to make graphs satisfy neighborhood constraints. Unfortunately, the relabeling problem is proved to be NP-hard, which motivates us to devise approximation methods for repairing, and identify interesting special cases (star and clique constraints) that can be efficiently solved. We propose several approximate repairing algorithms including greedy heuristics, contraction method and a hybrid approach. The performances of algorithms are also analyzed for the special case. Our extensive experimental evaluation, on both synthetic and real data, demonstrates the effectiveness of eliminating frauds in several types of application networks. Remarkably, the hybrid method performs well in practice, i.e., guarantees termination, while achieving high effectiveness at the same time.

1. INTRODUCTION

This paper studies a problem of repairing vertex labels in a graph to make it satisfy certain neighborhood constraints (a.k.a. binary constraints [20]) on labels. Let $L = \{\ell_1, \ldots, \ell_{|L|}\}$ denote a set of labels. A *constraint graph* $\mathcal{S}(L, N)$ is an undirected graph, where N specifies the pair-wise neighborhood constraints of unique labels in L. For instance, Figure 1(a) illustrates a constraint graph that specifies the neighborhood among four labels $\{a, b, c, d\}$, where each node denotes a unique label. We consider a graph $\mathcal{G}(V, E)$, namely *instance graph*, which has labels from L for all the vertexes in V, given as a labeling function $\lambda : V \to L$, i.e., $\lambda(v) \in L, \forall v \in V$. For example, Figure 1(b) illustrates an instance graph with four vertexes $V = \{1, 2, 3, 4\}$, where each vertex is associated with a label from $L = \{a, b, c, d\}$. Since different vertexes may share the same labels, the sizes of instance graphs are usually much larger than constraint graphs (see statistics on real data in Table 3).

Proceedings of the VLDB Endowment, Vol. 7, No. 11



(a) constraint (b) instance

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) relation

Figure 1: Data repairing on similarity network

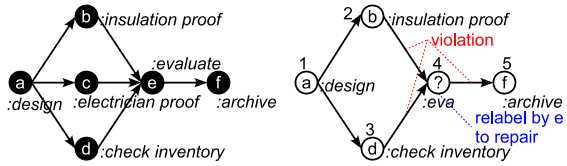
We say an instance graph $\mathcal{G}$ *satisfies* the constraint graph $\mathcal{S}$, if any two adjacent vertexes in $\mathcal{G}$ either share the same label or have adjacent labels in $\mathcal{S}$. A *violation* in $\mathcal{G}$ is an edge $(v, u) \in E$ such that $\lambda(v) \neq \lambda(u)$ and $(\lambda(v), \lambda(u)) \notin N$. For example, in Figure 1(b), edge $(1, 3)$ with labels a, d indicates a violation, as a, d are not adjacent according to the constraint in Figure 1(a). To eliminate violations w.r.t. integrity constraints, existing data repairing techniques [3, 19] modify values to make the data conform to integrity constraints. Following the same line, we should modify at least one value (label) of the vertexes v, u in order to address the violations in the instance graph, e.g., repairing vertex 3 with label b to eliminate the violations to vertexes $1, 2, 4$. We illustrate several typical scenarios below to motivate the vertex relabeling work.

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, captured by *differential dependencies* (DDs) [23], enable the tolerance of small variations, e.g., "Street" and its abbreviation "St.". Without such tolerance of variations, the number of tuples with (strict equality) relationships to a being repaired tuple t is quite limited, and thus the equality based repairing methods (such as FD based [3]) often fail to suggest possible repairs.

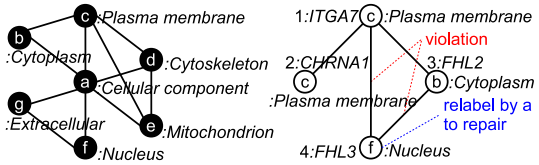
Example 1 (Similarity Networks)**.** Consider a relation in Figure 1(c) which collects customer information from various sources. A DD $(\mathsf{Address}, \mathsf{City} \rightarrow \mathsf{Tel}, \langle[0, 6], [0, 0], [0, 5]\rangle)$ states that if two tuples have similar Address values (i.e., with distance[1] in the range

[1]e.g., edit distance (see [22] for a survey of string similarity)



(a) workflow specification (constraint) (b) workflow execution (instance)

Figure 2: Example of workflow networks



(a) star constraint on GO terms (b) instance of protein network

Figure 3: Example of protein interaction networks

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 DDs rules can either be specified by domain experts or discovered from data [24].

We model the similarity relationships by graphs as follows. First, in Figure 1(a), 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 Figure 1(b). 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 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 vertexes 3 and 2 in Figure 1(b). However, their Tel numbers are not similar (with distance 7 not in $[0, 5]$), i.e., the labels of vertexes 3 and 2 denoted by d and b, respectively, are not an edge in Figure 1(a).

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 Section 3). 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 Figure 1(c). That is, no repair candidates can be suggested w.r.t. the FD. □

The execution of workflow or business process should follow certain specifications of business rules [10]. The notations of undirected graph in Figure 1 can be easily extended by adding directions to support workflow networks, where the constraint graph is analogous to workflow specification and the instance graph is workflow execution. A workflow execution $\mathcal{G}$ is invalid w.r.t. the workflow specification $\mathcal{S}$, if some edge (v, u) in execution $\mathcal{G}$ has labels $(\lambda(v), \lambda(u))$ not belonging to an edge in specification $\mathcal{S}$.

Example 2 (Workflow Networks)**.** Consider a workflow of part design in a train manufacturer (see Section 8 for more details about the real dataset in experiments). The constraint of workflow specification, in Figure 2(a), states that once a part is designed, insulation proof or electrician proof as well as check inventory steps can be processed. After that, the part design is evaluated and archived. Figure 2(b) illustrates an execution (instance) of the workflow. However, inconsistencies are often introduced due to erroneous recording of step names, e.g., for simplicity in practice, a curt abbreviation "eva" may sometimes be manually inputted standing for the step "evaluate". By using neighborhood constraints of steps such as check inventory→evaluate or evaluate→archive, we are able to repair the step names (labels) in the imprecise execution instance.

Note that multiple instances [26] often exist in workflow networks. For example, for a high sensitive evaluate step, it should be conducted/evaluated multiple times by (often different) staffs in a department, i.e., after staff A evaluating, another staff B will evaluate it again for sure. Such multiple instances of evaluate lead to a workflow instance of evaluate→evaluate. By allowing two adjacent nodes sharing the same label, our relabeling settings can naturally support such multiple instances. □

Without altering inaccurate information, it is unlike to conduct advanced analysis over workflow data [4]. Cleaning the workflow data is highly non-trivial and known as the first challenge in the Process Mining Manifesto by the IEEE Task Force on Process Mining [25]. To our best knowledge, this is the first attempt of the task.

1.2 Hardness and Special Cases

According to our analysis (Theorem 1) in Section 4, it is highly nontrivial (NP-hard) to conduct the relabeling w.r.t. neighborhood constraints. The difference between neighborhood constraints in graphs and integrity constraints in databases prevents applying existing data repairing techniques [3, 19]. Indeed, due to the spread of violations during the relabeling, the widely used greedy heuristics even fail to terminate as illustrated in Section 5.

Thereby, we investigate interesting special cases that may be efficiently solved or possibly approximated. The problem is found tractable in *clique constraints* case, where the transitivity of neighborhood on labels is observed. For instance, the constraint graph in Figure 1(a) consists of two cliques, {a,b,c} and {d}, where {d} is a clique with one single vertex.

Another class of *star constraints* is also interesting and worth investigating, where a center label connects to all the other labels in the constraint graph. This center is not only practically found important but also theoretically essential to improve the performance of finding solutions. Although the case of star constraints is still hard (by leveraging the connection to the vertex cover problem in Theorem 2), the approximation can be improved (Theorem 3).

Example 3 (Protein Networks)**.** Consider *protein interaction networks* constructed from statistically assessed pair-wise protein interaction affinities (see Section 8 of experiments for more details). Each vertex in the network denotes a protein, e.g., 1:ITGA7 in Figure 3(b). An edge is drawn between two vertexes (proteins) if their affinity level is above a pre-selected threshold [14]. Each protein is associated with a gene ontology[2] (GO) term description as vertex

[2]www.geneontology.org

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 [30]. Proteins with high affinity but irrelevant GO terms may contain faults. For example, 1:ITGA7 and 4:FHL3 have an edge (high affinity) in Figure 3(b), but their labels c:Plasma membrane and f:Nucleus are not adjacent (irrelevant) in Figure 3(a).

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 3(a) is a star constraint, where the GO term a:Cellular component can be regarded as a center term 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 special center label (e.g., replace f:Nucleus of 4:FHL3 by a:Cellular component), it guarantees to eliminate violations in the graph. □

1.3 Contributions

This paper presents the first study on repairing vertex labels under neighborhood constraints. We focus on efficient repairing methods in general cases and investigate the corresponding performance in special cases. Table 1 summarizes our major theoretical results.

(1) We investigate the complexity of the relabeling problem (in Section 4). The general relabeling problem is proved to be NP-complete. For the special case of star constraints, we find that it can be approximated within a constant factor, while the clique constraints case can be solved in PTIME.

(2) We study greedy heuristics for relabeling (in Section 5). The rationale behind greedy method's failure is that relabeling a vertex to eliminate some violations may introduce new violations to other vertexes. Nevertheless, we illustrate that the greedy method always terminates in the special case of star constraints.

(3) We develop a contraction method which guarantees termination (in Section 6). The contraction operation requires the contracted vertexes to have the same label in order to stop the violation spread. We prove that the total number of contraction operations is bounded, 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 shows surprisingly good performance as a factor-2 approximation.

(4) We present a hybrid approach by putting together the advantages of greedy and contraction techniques (in Section 7). Referring to non-termination of greedy method and possibly bad results of contraction, the hybrid approach conducts the high cost contraction operation when no further greedy relabeling can be applied.

(5) We report an extensive experimental evaluation for the proposed relabeling methods on both synthetic and real data sets (in Section 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 hybrid approach performs very well in practice, with high effectiveness and efficiency.

2. RELATED WORK

Constraints on Graphs. Label neighborhood constraint is a binary constraint that involves the labels of two vertexes. A constraint is considered n-ary if it involves n variables. The constraint satisfaction problem (CSP) [21], which finds an assignment of values to variables (analogous to labels and vertexes, respectively) such that certain constraint is satisfied, is known generally hard. Given a binary constraint graph, the problems are different between assignment and relabeling. CSP with binary constraints finds a bijection (assignment) from instance graph to constraint graph. In our relabeling problem, however, vertexes in the instance graph have already been assigned with labels. We target on relabeling some vertexes to eliminate violations to the constraint graph.

Besides the binary 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 vertexes [11] or extending edges to reachability constraints of paths [7]. The major difference between graph pattern matching and our relabeling 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 [12], i.e., it is not required that each edge in $\mathcal{G}$ could be mapped to the pattern graph. In contrast, the satisfaction in our study enforces two vertexes of each edge in $\mathcal{G}$ to have the same label or two labels as an edge in the constraint graph.

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 the data instance, in order to eliminate violations to the constraints. In terms of graph notations, it is to delete vertexes (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 [29] performs value modification instead of deletion. In this paper, we also adopt this value modification model, i.e., vertex relabeling. An interesting variation [19] 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).

Besides deletion and modification, 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 vertexes, the insertion model does not help towards graph constraint satisfaction.

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 pairwise relationships in a general graph. Thereby, for the general constraint graph, existing techniques [3, 19, 8] developed on the equivalence of tuples cannot be applied to our relabeling problem.

3. PRELIMINARIES

In this section, we introduce syntax for representing constraints in graph. Table 2 lists the frequently used notations.

Syntax. 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 Figure 1(a), 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 $\mathcal{S}$.

Table 1: Major theoretical results

Constraints	Complexity	Hardness of approximation	Greedy method	Contraction method
General	NP-complete (Theorem 1)	–	no guarantee of termination (Example in Section 5)	guarantee to terminate (Proposition 5)
Star	NP-complete (Theorem 2)	NP-hard to within a factor $\alpha \approx 1.36$ (Theorem 3)	$(\ln n + 1)$-approximation (Proposition 4)	factor 2-approximation (Proposition 6)
Clique	PTIME	–	–	–

Table 2: Notations

Symbol	Description
$\mathcal{S}(L, N)$	constraint graph $\mathcal{S}$ with label set L, neighborhood N
$\mathcal{G}(V, E)$	instance graph $\mathcal{G}$ with vertex set V, edge set E
λ	labeling of vertex
$\asymp$	labels match constraint
$\vDash$	graph satisfies constraint
δ	cost of relabeling a vertex
Δ	cost of relabeling a graph
$T(v, \lambda(v))$	set of vertexes with violations to vertex v having $\lambda(v)$
R	a node of contracted vertexes

An instance graph $\mathcal{G}(V, E)$ *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 $(\lambda(v), \lambda(u)) \in N$.

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 Figure 1(b), the edge $(1, 3)$ indicates a violation to $\mathcal{S}$, as their labels $(a, d) \not\asymp \mathcal{S}$ are neither the same nor adjacent in Figure 1(a) of constraints.

Cost Function. The relabeling cost is evaluated by the difference between the original and repaired data. 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.

Therefore, 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,

$$\Delta(\mathcal{G}', \mathcal{G}) = \sum_{v \in V} \delta(\lambda(v), \lambda'(v)), \tag{1}$$

where $\lambda'(v)$ is the new label of $v \in V$ in the repaired $\mathcal{G}'$, and $\delta(\lambda(v), \lambda'(v))$ denotes the (distance) cost of relabeling vertex v from label $\lambda(v)$ to $\lambda'(v)$. The distance metric δ could be any string distance function or simply the count of modifications [19].

Problem Statement. We are now ready to formalize the 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 $\Delta(\mathcal{G}', \mathcal{G})$ is minimized.

4. PROBLEM ANALYSIS

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

Theorem 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 $\Delta(\mathcal{G}', \mathcal{G}) \leq c$ is* NP*-complete.*

Proof idea. To show the hardness, we give a reduction from the set cover problem, which is known to be NP-hard [2]. Due to the limitation of space, please refer to the full version technique report of this paper [13] for proof details. □

Tractable Special Case. Recognizing the hardness, we identify special cases of the relabeling problem that turn out to be tractable.

Definition 1. *We call $\mathcal{S}(L, N)$ a* clique constraint*, if transitivity on neighborhood is satisfied, i.e., for any $(\ell_1, \ell_2) \in N$ and $(\ell_2, \ell_3) \in N$, it implies $(\ell_1, \ell_3) \in N$.*

Transitivity of neighborhood on labels can be applied inside each clique. 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 Figure 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 (see the example of real dataset RSNT in Section 8).

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

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 GO terms (as mentioned in the introduction and observed in the real dataset HPRD in Section 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 $\delta(\ell_0, \ell_i) < \delta(\ell_j, \ell_i), i \neq j$, $i = 1, \ldots, |L| - 1; j = 1, \ldots, |L| - 1$.*

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

Unfortunately, the problem is still hard in this special case.

Theorem 2. *For a star constraint $\mathcal{S}$, the problem of relabeling a graph with the minimum cost is* NP*-hard.*

Proof idea. To prove the NP-hardness, we show reduction from the vertex cover problem, which is one of Karp's 21 NP-complete problems [18]. Please also refer to the full version technique report of this paper [13] for proof details. □

Although it is still hard in the special case, the problem with star constraint turns out to be different in an approximation-preserving way. 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 [19]. It is unlikely however to approximate within an arbitrary small factor.

Theorem 3. *For a star constraint $\mathcal{S}$, the problem of relabeling a graph with the minimum cost is* NP*-hard to approximate to within any factor less than* $10\sqrt{5} - 21 \approx 1.36$.

Proof idea. The proof follows by showing that the reduction in Theorem 2 is *gap-preserving*. Please also refer to the full version technique report of this paper [13] for proof details. □

Outline of Approximation Approaches. In the following, we focus on heuristics, which may empirically show good performance. We first study several greedy heuristics and show that their termination cannot be guaranteed. Then, we develop a contraction method which ensures termination but may have bad approximation results. Finally, we put things together as a hybrid approach which shows good performance in practice. Performance guarantees of the proposed algorithms are also analyzed for special cases.

5. GREEDY HEURISTICS

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

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 Theorem 1), it is natural to adopt the greedy method as follows.

We define the *violation set* of a vertex v with label $\lambda(v)$,

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

which denotes the set of neighbors u of the vertex v whose labels $\lambda(u)$ have violations to $\lambda(v)$. 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 by changing $\lambda(v)$ to $\lambda'(v)$,

$$\arg\max_{v \in V, \lambda'(v) \in L} |T(v, \lambda(v))| - |T(v, \lambda'(v))|. \quad (2)$$

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

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

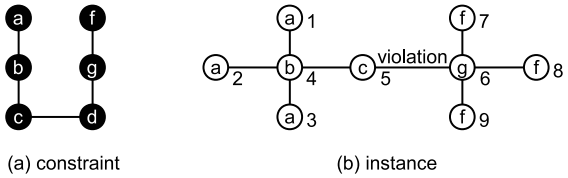


Figure 4: Counter example of termination in greedy method

That is, a number of violations $|T(v, \lambda(v))|$ is eliminated by paying the cost $\delta(\lambda(v), \lambda'(v))$ with the least new violations $|T(v, \lambda'(v))|$ introduced. It is notable that simply normalizing formula (2) by the cost, i.e., $\frac{|T(v,\lambda(v))| - |T(v,\lambda'(v))|}{\delta(\lambda(v),\lambda'(v))}$, 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 $\delta(\lambda(v), \lambda'(v))$, 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 4. Consider the example in Figure 4 with relabeling cost $\delta(c, d) = \delta(g, d) = 1$. According to the greedy function in formula (3), the best choice is to repair vertex 5 with label d. While the violation between vertexes 5 and 6 is eliminated, a new violation between vertexes 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. Similar counter examples are observed when adopting the straightforward greedy function, i.e., repeatedly relabeling vertex 6 between d and g. □

Special Case of Star Constraints. In the introduction, 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 $\frac{\Delta(\mathcal{G}', \mathcal{G})}{\Delta(\mathcal{G}^*, \mathcal{G})} \leq \ln n + 1$, where $\mathcal{G}^*$ is the relabeled graph with the minimum cost and $n = |E|$.*

Due to the limitation of space in this paper, we leave the proof details in the full version technique report [13]. Nevertheless, the experimental results on both synthetic and real (HPRD) data with star constraints in Section 8 verify that the greedy method can always terminate and show relatively high repairing accuracy.

6. CONTRACTION RELABELING

In this section, we present a contraction based method which is guaranteed to terminate by enforcing certain vertexes to the same labels to stop violation spread.

6.1 The Idea of Contraction

Intuition. Recall that greedy relabeling fails as vertexes could be repaired back and forth with possible new violations generated in the follow-up steps. Intuitively, we consider a group of vertexes as a whole, called a *super node* or simply a *node*, which *always* ensures no violations inside. The vertexes in a node can only be repaired together to the same label. To eliminate violations among vertexes from two nodes, a node contraction operation is employed, i.e., merging all the contents (vertexes and edges) of a node R_1 into the

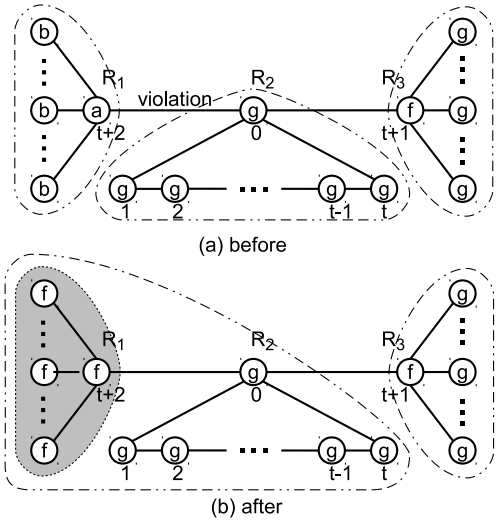


Figure 5: The idea of contraction

other R_2. All the vertexes in the contracted node R_1 are enforced to assign the same label, which can avoid introducing violations in the new node R_2.

Example 5. Suppose that there are three nodes, $\mathsf{R}_1, \mathsf{R}_2, \mathsf{R}_3$, formed in the previous steps, as shown in Figure 5(a). A vertex (say $\mathsf{t}+2$ for instance) can *only* be relabeled again together with all the other vertexes in node R_1 that has been contracted, in order to stop the 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 vertexes 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. Consequently, all the violations are eliminated in Figure 5(b), by ensuring that no violations exist inside a node. □

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

Framework. To perform the contraction, we first introduce several notations as follows. Let R denote a *node* in contraction, consisting of a group of vertexes contracted to R. Each node R has a unique *host* $h(\mathsf{R})$ which is a vertex in the original graph. The remaining vertexes in R are grouped into a set of nested nodes R_i, namely *guests* denoted by the 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 vertexes in the original graph belonging to R, which includes both the host and the vertexes in all guests

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

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 vertexes in a guest $\mathsf{R}_i \in U(\mathsf{R})$ must have the same label, which is assigned by the contraction operation.

Example 6. In Figure 6(b), 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 vertexes in a guest should share the same label, e.g., vertexes $(\mathsf{6}, \mathsf{7}, \mathsf{8})$ with label b, while the vertexes 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 nodes, say R_1 and R_2 for example, whose vertexes 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 vertexes 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 G do
2:     create a new node R
3:     h(R) := v
4:     U(R) := ∅
5: while G not satisfying S do
6:     R1, R2 := the nodes with most violations
7:     ℓ1, ℓ2 := the candidate labels for R1, R2, respectively
8:     if cost(R2) > cost(R1) then
9:         swap R1, R2                          {To contract R2 to R1}
10:    for each v ∈ V(R2) do
11:        λ(v) := ℓ2
12:    V(R1) := V(R1) ∪ V(R2)
13:    U(R1) := U(R1) ∪ {R2}
14: return 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 vertexes in the original graph that are relabeled to the same ℓ_i, i.e., relabeling results.

Correctness. Once two nodes with vertexes 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 each contraction operation, we reduce one node in the graph. The total number of contraction operations is bounded by the maximum number of nodes, i.e., the size of vertexes $|V|$ in $\mathcal{G}$. Therefore, the contraction relabeling repeats at most $|V|$ times. □

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 $O(|V| \cdot |E|)$.

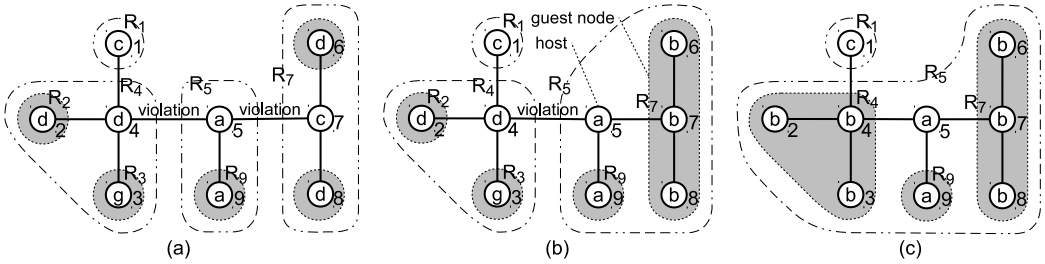


Figure 6: Example of contraction operation

6.2 Technique Details

Consider the contraction of any 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).

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 vertexes 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 straight-forward 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 vertexes v in R_2, must match the label of the host $h(\mathsf{R}_1)$ and should not introduce violations to the existing vertexes 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 vertexes 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 the label ℓ_2 for R_2 as

$$\arg\max_{\ell' \in L(\mathsf{R}_2)} |T(\mathsf{R}_2)| - |T(\mathsf{R}_2, \ell')|, \tag{4}$$

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

Example 7 (Example 6 continued)**.** In Figure 6(b), 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 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 (4), the candidate label for R_4 is b. □

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',

$$\mathsf{cost}(\mathsf{R}) = \sum_{v \in V(\mathsf{R})} \delta(\lambda(v), \ell') - \sum_{\mathsf{R}_i \in U(\mathsf{R})} \mathsf{cost}(\mathsf{R}_i). \tag{5}$$

The first part $\sum_{v \in V(\mathsf{R})} \delta(\lambda(v), \ell')$ denotes the cost of enforcing all vertexes v in R to the new label ℓ'. Intuitively, by paying the cost of relabeling all these vertexes, 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 vertexes 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 8 (Example 6 continued)**.** In Figure 6(b), we consider the candidate label d for contracting R_5 to R_4. By relabeling all five vertexes $\{5, 6, 7, 8, 9\}$ 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 to R_5, in Figure 6(a), 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 Figure 6(b), 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 $\delta(b,d) = \delta(a,d) = \delta(b,g) = 1$. According to formula (5), it follows

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

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

6.3 Performance Analysis

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

Example 9. In Figure 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 1 could be g, and the candidate label of R_2 could be b. Suppose that the relabeling costs are $\delta(a,g) = 2, \delta(b,f) = 1$. That is, we have $\mathsf{cost}(\mathsf{R}_1) = \delta(a,g) = 2$ which is greater than $\mathsf{cost}(\mathsf{R}_2) = \delta(b,f) = 1$. The node R_2 will be contracted to R_1 as presented in Figure 7(b). 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. □

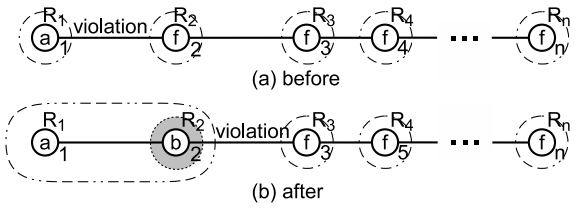


Figure 7: Counter example of contraction effectiveness

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 terminates, and outputs a $\mathcal{G}'$ having $\Delta(\mathcal{G}', \mathcal{G}) \leq 2 \cdot \Delta(\mathcal{G}^*, \mathcal{G})$, where $\mathcal{G}^*$ is the relabeled graph with the minimum cost.*

Again, we leave the proof details in the full version technique report [13], due to the limitation of space in this paper. The experimental results on all the four real datasets in Section 8 show that the contraction method can always terminate. In particular, the experiments on synthetic data verify the bounded approximation performance compared with the optimal solution.

7. PUTTING THINGS TOGETHER

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.

The Idea of Hybrid Approach. Intuitively, the greedy method succeeds with good performance in some case (termination case) as it always selects a relabeling that can eliminate most violations. Unfortunately, even this maximum violation elimination could be negative, which leads to the non-termination case. In such a scenario, the contraction operation is instrumental in carrying on the relabeling process. However, the contraction enforces all the vertexes in the contracted guest nodes to have the same label, which may hurt the results in terms of the relabeling cost. We should avoid contraction when unnecessary.

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

Hybrid Algorithm. Algorithm 2 illustrates the pseudo-code of the hybrid relabeling. 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. *The hybrid relabeling always terminates.*

Proof. Since there are at most $|V|$ vertexes considered for contraction, according to Proposition 5, the number of contraction operations in the **while** loop is bounded by $|V|$. Between two contraction operations is a series of greedy relabeling operations. $|T(v, \lambda(v))| > |T(v, \lambda'(v))|$ ensures at least one violation on an edge is eliminated in each greedy operation. As the number of violations is bounded by $|E|$, the number of greedy steps between two consecutive contraction operations is in $O(|E|)$. To sum up, the **while** loop executes at most $|V| \cdot |E|$ iterations, i.e., always terminates. □

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

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** $\mathcal{G}$

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

Example 10 (Example 9 continued)**.** We consider the example in Figure 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. □

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 hybrid algorithm is also guaranteed, the same as for the greedy algorithm.

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

Proof. Since the center label ℓ_0 in star constraints can eliminate all violations, i.e., $|T(v, \lambda(v))| > |T(v, \lambda'(v))| = 0$ is always true, the contraction operation in Line 6 of Algorithm 2 will never be executed. In other words, the hybrid Algorithm 2 is equivalent to the pure greedy algorithm. The property of termination and approximation bound are directly inherited from Proposition 4. □

Finally, our experimental evaluation below shows that the hybrid approach can always terminate as the contract one while it keeps the accuracy performance as high as the greedy method.

8. EXPERIMENTS

This section reports the experiments of proposed relabeling methods, on both synthetic and real data sets. 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 Approximation Ratio

In order to verify the performance of approximation methods proposed in Sections 5, 6 and 7, the first experiment is conducted on synthetic data to observe the difference between the optimal relabeling and the returned approximate answers.

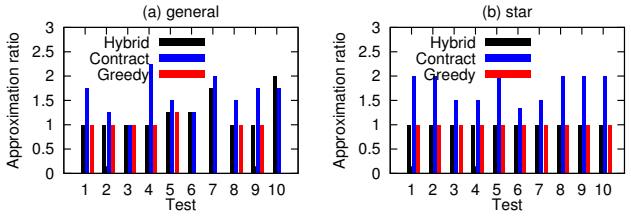


Figure 8: Approximation ratio on synthetic data

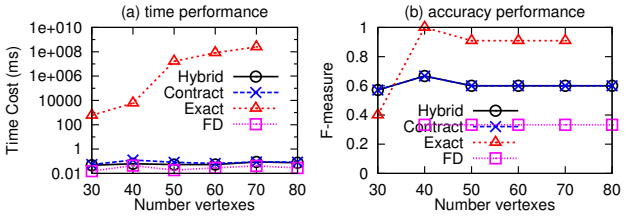


Figure 9: Restaurant with various data sizes

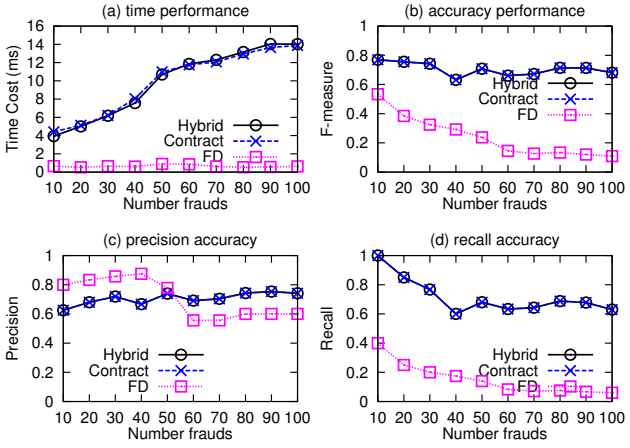


Figure 10: Restaurant with various inserted frauds

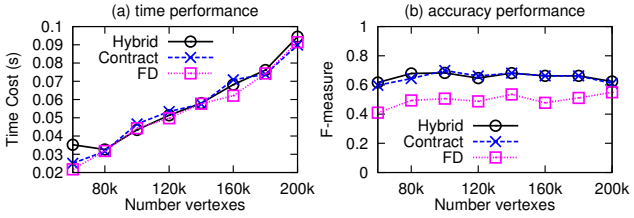


Figure 11: CiteSeer with various data sizes

Table 3: Data set statistics

Data set	Instance size		Constraint size		Type
	Vertex	Edge	Label	Neighborhood	
Restaurant	864	2288	12	0	Clique
Coauthor	1680	3089	648	2092	General
WFNT	4742	34221	174	172	General
HPRD	9460	34998	305	6255	Star
USCC	180946	84294	1919	1891	General
CiteSeer	200000	4537	128637	0	Clique

We employ the widely used graph generation tool [6, 17], GraphGen[3], to generate 10 testbeds with 9-13 vertexes in instance graphs and 6-11 vertexes in constraint graphs.

The evaluation focuses on the approximation ratio of approximate answers compared with the optimal one. First, we compute the optimal solution $\mathcal{G}^*$ with the minimum relabeling cost $\Delta(\mathcal{G}^*, \mathcal{G})$. For any approximate solution $\mathcal{G}'$, the approximation ratio of $\mathcal{G}'$ to $\mathcal{G}^*$ is reported by $\frac{\Delta(\mathcal{G}', \mathcal{G})}{\Delta(\mathcal{G}^*, \mathcal{G})}$.

Figure 8 illustrates the approximation ratio performance of the proposed methods, including Greedy[4], Contraction and Hybrid approaches. Both general constraints and the special case of star constraints are considered. As illustrated, the Greedy method cannot terminate and fails to return results in Tests 6, 7 and 10 in Figure 8(a), i.e., non-termination observed in 30% of randomly generated graphs. Meanwhile the Contraction approach can terminate in all the 10 tests. These results verify our conclusion of termination for the Contraction method in Proposition 5. Moreover, according to Proposition 6, the approximation ratio of the Contraction method should be no greater than 2 in star constraints, which is also observed in all the tests in Figure 8(b). Finally, the Hybrid method can achieve much better approximation ratio performance (equal to 1 in many tests) than Contraction while still guaranteeing termination, in both general and star constraints.

8.2 Experiments on Repairing Performance

The second group of experiments evaluates both the effectiveness and efficiency performance of the proposed methods in practice. We employ six real data sets in the experiments. Statistics of data sets are summarized in Table 3.

Since the original data are clean without violations (except Coauthor Networks), we randomly replace labels as fraud tagging. 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 [27], 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 time costs of approaches to study the efficiency performance.

8.2.1 Experiments on Similarity Networks

To evaluate our proposed relabeling techniques over similarity networks, we directly implement an existing repairing method based on FD [3] (does not rely on the proposed notations of graphs).

Restaurant Similarity Network, is a collection of 864 restaurant records[5] that contains 112 duplicates and is widely used for record matching [16]. We employ a DD discovered from the data, (Name,Address $\rightarrow$ Areacode, $\langle[0,0],[0,6],[0,0]\rangle$), and a corresponding FD (Name,Address $\rightarrow$ Areacode). 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 $[0,0]$,

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

[4] the advanced greedy function with normalization in formula (3)

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

i.e., equality. Therefore, the constructed constraint graph belongs to clique constraints where transitivity is applicable.

CiteSeer[6] is a dataset of 200000 citation records. The similarity network is constructed according to a DD (Title,Author $\rightarrow$ Subject, $\langle[0,0],[0,5],[0,0]\rangle$). To compare with the existing data repairing method [3], we employ a corresponding FD (Title,Author $\rightarrow$ Subject).

First, in Figure 9, 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 250000 seconds (about 70 hours) for a subset of 70 vertexes. Considering the NP-hardness of computing the optimal solution (Theorem 1), we didn't 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 5 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 failed to terminate (see more discussion below), we report the results by Hybrid, Contract, and the existing FD based repairing [3], in Figure 10. Since data set 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, Hybrid 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 equality 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 Figure 10(d), the recall accuracy of the proposed similarity network based Hybrid approach is significantly higher than that of FD.

Figure 11 evaluates the scalability of repairing methods over various sizes of similarity networks. As shown, the time costs of all the approaches increase similarly and slowly with the increase of the number of vertexes, while the corresponding accuracy performances are stable. Again, the accuracy of Hybrid approach is similar to Contract and higher than FD.

8.2.2 *Experiments on Coauthor Networks*

Coauthor Network is motivated by the entity resolution task [15]. 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), etc. It is non-trivial to identify which Lei Chen is involved in a specific citation record (e.g., from CiteSeer where different authors sharing the same name are not fully distinguished). Existing approach (such as CENTER [15]) employs clustering techniques by computing the similarity on coauthors. We employ the coauthor relationships from DBLP[7], where different Lei Chen(s) are distinguished, and model them as the constraint graph. As shown in Figure 12, 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,

[6]http://citeseerx.ist.psu.edu/

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

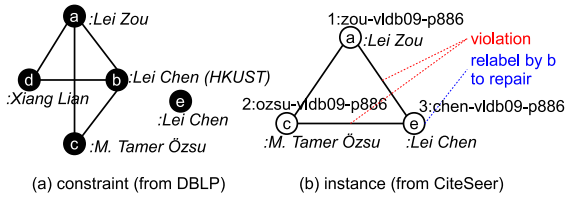


Figure 12: Example of coauthor networks

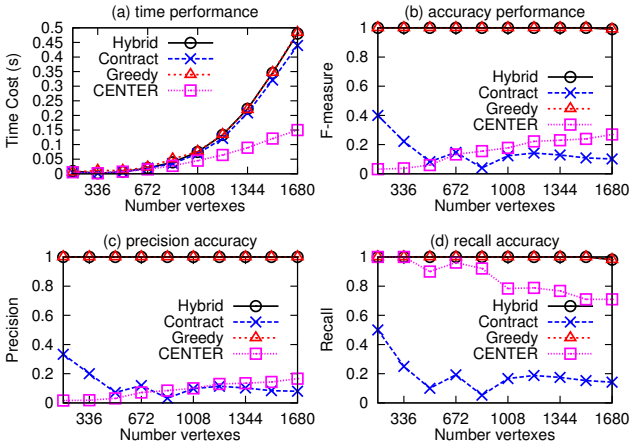


Figure 13: Coauthor networks with various sizes

e.g., Lei Chen (HKUST) or Lei Chen (Wisconsin), for the original imprecise one (simply Lei Chen without identification). For the existing CENTER approach, each author corresponds to the center node of a cluster, 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.

As shown in Figure 13, while the recall of CENTER method is somehow close to our proposed Hybrid/Contract approach, the precision of CENTER is significantly lower. F-measure is improved from 0.3 (of CENTER) to 0.98 by our proposed Hybrid/Contract 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.

8.2.3 *Experiments on Larger Data Sets*

WFNT is a set of 490 workflow networks collected from a train manufacturer [28], each of which corresponds to a workflow execution with 4–45 activities (vertexes). All these workflow executions compose the instance graph. Since the activities do not occur in random and should confirm to certain specifications on predecessor-successor relationships, we employ these specifications as the constraint graph. As mentioned in the introduction, any two activities can appear adjacently in a workflow execution network if their adjacency is specified in the workflow specification.

HPRD[8], 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 Section 1, edges denote the binary protein-protein interactions. Protein's GO term is used as the vertex label. The constraint graph repre-

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

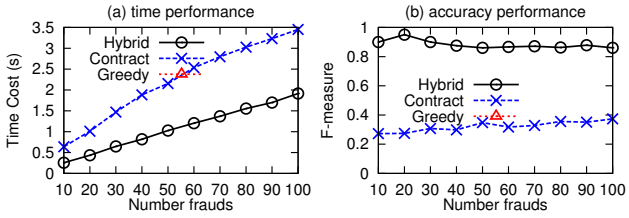


Figure 14: WFNT with various inserted frauds

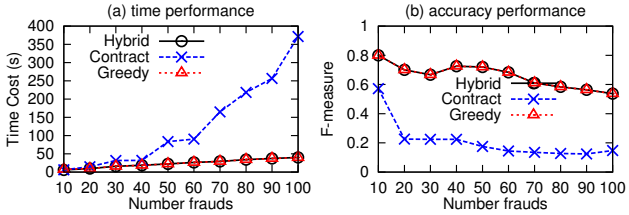


Figure 15: HPRD with various inserted frauds

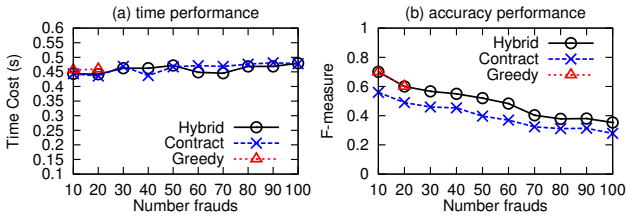


Figure 16: USCC with various inserted frauds

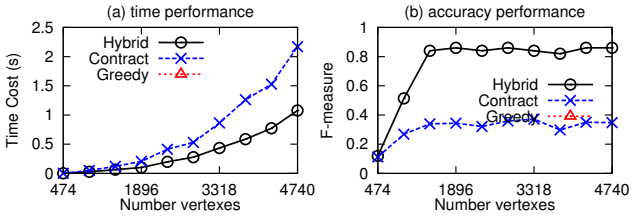


Figure 17: WFNT varying graph sizes

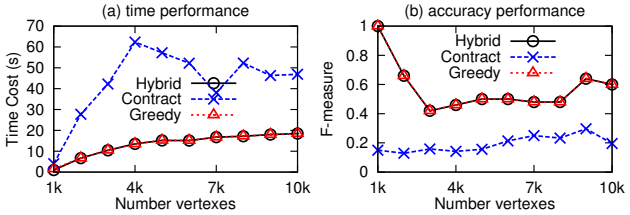


Figure 18: HPRD varying graph sizes

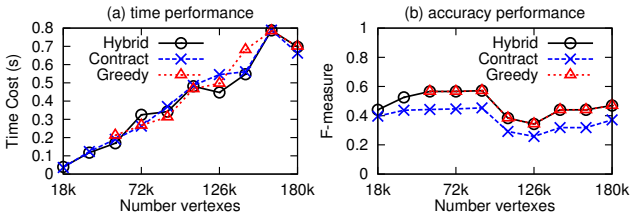


Figure 19: USCC varying graph sizes

sents GO term correlations. Since the GO term *cellular component* is correlated with all the others, the constraint is in star shape.

USCC[9], US City and County Web Data, provides city and county location data in US. We treat each city as a vertex with a county label. An edge is added between two cities in the instance graph if the corresponding geographic distance is less than 100km, i.e., nearby. An edge between two counties in the constraint graph indicates that there exists at least two nearby cities from these two counties, respectively. For any two nearby cities, they should be either in a same county or in two counties that are not far away.

Since there are no existing methods for repairing over these datasets, we focus on comparing the techniques proposed in this paper. The Greedy heuristics performs well, i.e., with high accuracy and low time cost as shown in Figure 15, if it can terminate. Unfortunately, as illustrated in Figures 14 and 16, the Greedy method fails to terminate in most tests of WFNT and USCC with general constraints, especially when the number of frauds is large. On the other hand, although the Contraction method can always terminate in all the tests, as presented in in Figures 14(b) and 15(d), the f-measure accuracy could be bad, which verifies our analysis of the Contraction approach in Section 6. Nevertheless, as illustrated in Figures 15(b) and 16(b), the f-measure of Hybrid approach is as good as that of the greedy method in almost all the tests (where the greedy algorithm can terminate). The Hybrid approach also guarantees termination as the Contraction method does, while the corresponding accuracy is as good as the Greedy heuristics one.

Figures 17, 18 and 19 report the performance of scalability by increasing the number of vertexes up to 180k in instance graphs (test beds with smaller sizes of n vertexes are prepared by using the first n vertexes listed in the data set). As shown in figures, the time performance scales well, which grows almost proportionally with data sizes. The results verify the complexity analysis of Contract and Hybrid algorithms in Sections 6 and 7, respectively. Note that WFNT data consist of 490 small workflow networks. When preparing tests with less vertexes, e.g., 474 in Figure 17, isolate activities (vertexes) may be selected such that neighborhood based repairing cannot perform. Nevertheless, the results turn to be higher and stable when test sizes are large (greater than 1896).

Specific tests may zigzag such as Figures 18 and 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 Figures 18(b) and 19(b).

In most figures, only two different curves are observed for two reasons. 1) The greedy algorithm cannot terminate and has no result, e.g., in Figures 14 and 17. 2) When the greedy algorithm terminates, the hybrid approach can show (almost) equally good results, e.g., in Figures 15 and 18. In the second case, there are indeed three curves in the results. The curves of Hybrid and Greedy methods overlap since they show similar performance. These results verify our analysis that 1) the greedy algorithm cannot terminate in a number of tests, and 2) the hybrid approach is comparable to the greedy algorithm if the greedy one can terminate.

To summarize, 1) experiments in Figures 9, 10 and 11 illustrate that hybrid and contraction methods show similar/better performance compared with existing techniques, while the greedy technique fails and has no much effect in the hybrid approach. 2) Figures 13, 15 and 18 demonstrate that hybrid and greedy approaches show similar and better results than that of the contraction method.

[9]http://explore.data.gov/d/veb9-7ksg

3) Most importantly, in Figures 14 and 17, the hybrid approach can still show good performance, when both greedy and contraction approaches fail (Greedy fails to terminate and Contract has low accuracy). In short, by taking advantages in both techniques, the hybrid approach always has the best performance in all the experiments.

9. CONCLUSIONS

This paper studies a novel problem of repairing vertex labels under the constraints of label neighborhood. Graph constraint satisfaction and relabeling 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. The 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 a hybrid approach by cooperating contraction and greedy relabeling. Experimental evaluation on both synthetic and real data verifies our major theoretical analysis, and demonstrates that the hybrid approach always takes the advantages of one of the proposed greedy/contraction techniques (if the other fails).

Acknowledgment. This work is supported in part by China NSFC under Grants 61202008 and 61370055, Hong Kong RGC GRF Project No. CUHK 411211, 411310 and 418512, National Grand Fundamental Research 973 Program of China under Grant 2012-CB316200, Hong Kong RGC/NSFC Project N_HKUST637/13, Microsoft Research Asia Gift Grant and Google Faculty Award 2013.

10. REFERENCES

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[2] M. Bellare, S. Goldwasser, C. Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In *STOC*, pages 294–304, 1993.

[3] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD Conference*, pages 143–154, 2005.

[4] A. Bonifati, F. Casati, U. Dayal, and M.-C. Shan. Warehousing workflow data: Challenges and opportunities. In *VLDB*, pages 649–652, 2001.

[5] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *SIGMOD Conference*, pages 447–458, 2010.

[6] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD Conference*, pages 857–872, 2007.

[7] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast graph pattern matching. In *ICDE*, pages 913–922, 2008.

[8] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[9] A. Conesa, S. Götz, J. M. García-Gómez, J. Terol, M. Talón, and M. Robles. Blast2go: a universal tool for annotation, visualization and analysis in functional genomics research. *Bioinformatics*, 21(18):3674–3676, 2005.

[10] D. Deutch and T. Milo. A quest for beauty and wealth (or, business processes for database researchers). In *PODS*, pages 1–12, 2011.

[11] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD Conference*, pages 925–936, 2011.

[12] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.

[13] Full Version. Repairing vertex labels under neighborhood constraints. *http://ise.thss.tsinghua.edu.cn/sxsong/doc/grelabel.pdf*.

[14] M. A. Gilchrist, L. A. Salter, and A. Wagner. A statistical framework for combining and interpreting proteomic datasets. *Bioinformatics*, 20(5):689–700, 2004.

[15] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[16] R. Isele and C. Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.

[17] C. Jin, S. S. Bhowmick, X. Xiao, J. Cheng, and B. Choi. Gblender: towards blending visual query formulation and query processing in graph databases. In *SIGMOD Conference*, pages 111–122, 2010.

[18] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[19] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[20] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[21] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. In *AAAI*, pages 17–24, 1990.

[22] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[23] S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.*, 36(3):16, 2011.

[24] S. Song, L. Chen, and H. Cheng. Parameter-free determination of distance thresholds for metric distance constraints. In *ICDE*, pages 846–857, 2012.

[25] W. M. P. van der Aalst and et al. Process mining manifesto. In *Business Process Management Workshops (1)*, pages 169–194, 2011.

[26] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[27] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[28] J. Wang, S. Song, X. Zhu, and X. Lin. Efficient recovery of missing events. *PVLDB*, 6(10):841–852, 2013.

[29] J. Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.

[30] B. Zhang, B.-H. Park, T. V. Karpinets, and N. F. Samatova. From pull-down data to protein interaction networks and complexes with biological relevance. *Bioinformatics*, 24(7):979–986, 2008.