Interactive Graph Repairs for Neighborhood Constraints

Paul Juillard*
EPFL
Lausanne, Switzerland
paul.juillard@epfl.ch

Angela Bonifati
Lyon 1 University & IUF
Lyon, France
angela.bonifati@univ-lyon1.fr

Andrea Mauri
Lyon 1 University
Lyon, France
andrea.mauri@univ-lyon1.fr

ABSTRACT

Graphs are unifying abstractions to encode, inspect and update interlinked data. They are becoming popular in several application domains with real users. In many cases, graph data is inconsistent and necessitates transformations to fix the underlying constraint violations. In this paper, we address the problem of interactive graph repairing, where users are key to the process of selecting and validating graph transformations after neighborhood constraint violations. We propose a theoretical approach to model the user involvement in the graph repair process, and we prove that minimizing the number of interactions is NP-hard. Then, we abstract away the solution space to Question-Answer-Repair (QAR) frameworks, composed of three key elements: selecting the question to be asked to the user, selecting the answer, and applying the answer as a graph repair operation. Then, we define and study the properties of termination, oracle optimality, and question difficulty to further characterize the solutions beyond the number of interactions. We evaluate our approach and the QAR frameworks on both synthetic and real-world datasets, considering different classes of users. We show that relatively high repair quality can be achieved with users that provide random answers, whereas more sophisticated users such as oracles achieve results equivalent to the ground truth.

1 INTRODUCTION

Graph data is ubiquitous in several application domains, such as life sciences, finance, security, logistics and planning, to name a few [36]. According to a recent survey with real users of graph processing systems [35], data cleaning is one of the computational tasks on which the participants routinely spend their time when working on graph applications. Several approaches exist in the literature to express graph constraints of different expressiveness to detect potential glitches and adopt automatic repairing strategies [8, 14, 38]. All these methods target the efficiency of the algorithms and repair optimizations while disregarding the role of the user in the repairing process.

However, there are situations in which a repair is preferred over another repair for several reasons, such as the freshness of the source or the most recent timestamp. A priority relation can be defined among all the repairs [39]. However, the underlying assumption in this previous work is that the priority relation is known in advance and characterizes the space of all repairs. In our work, we fall under the hypothesis of preferred repairs when the priority relation is not known in advance and the user is included in the repairing process. In particular, the appropriate repair is a decision that involves the users as they need to see the effect of the repair on the database (i.e. the repaired graph).

*Work done during a research internship at Lyon 1 University

Interactivity for improving the quality of knowledge graphs and knowledge bases, consisting of a set of facts, has been introduced in early work [5, 26]. The first approach uses existential rules to add new facts to the knowledge base and adopts an oracle-based approach, whereas the latter work relies on one-shot learning to reduce manual effort. The above works either propose only a marginal involvement of the users - e.g., just at the beginning - or oversimplify their characteristics - e.g., by considering humans as oracles. This ignores issues related to the quality of the responses with suboptimal users, which might become a hurdle in the process.

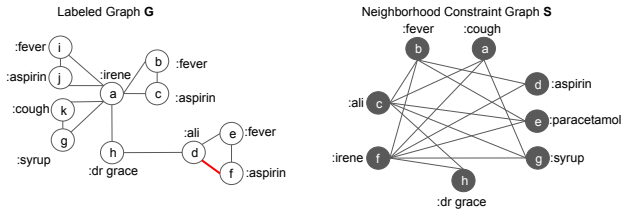
In this paper, for the first time we focus on the problem of *enabling and computing interactive graph repairs under graph constraints*, by involving the users in the process of choosing and validating the graph transformations needed for the repairs and thus incrementally constructing the ground truth. Our repair model considers both graph update [45] (e.g., vertex relabeling) and graph delete operations [11] - specifically deletion of edges. We adopt neighborhood constraints [38], that are inherently easy to understand and visually intuitive for the users. Indeed, such constraints check the consistency of pairs of edge labels in a graph. This type of constraint can be built using some external knowledge (e.g., as we show in the construction of the datasets *Restaurant* and *Sepsis* in our experimental study), obtained from a reliable source (e.g., in the generated dataset *Co-authors*, as shown later in the paper), or built using entity resolution methods on graph [38]. To deal with suboptimal users, we design cost models that account for repair efficiency in terms of user interactions as well as question difficulty.

We illustrate the problem with the following motivating example.

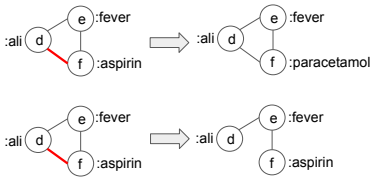
Assume a graph-based healthcare application about patients, their symptoms, and treatments. Figure 1(a) depicts a labeled graph G that exemplifies the data in this application. The graph includes two patients, *:ali* and *:irene*, who are under the care of their doctor, *:dr grace*. From the graph, we observe that *:irene* has experienced *:fever* twice and has been treated with *:aspirin*, in addition to receiving *:syrup* for a *:cough*. On the other hand, *:ali* has been treated with *:aspirin* once for a *:fever*.

Then, an expert can express constraints on these relations, which can be translated into neighborhood constraints. Neighborhood constraints allow us to check which pairs of labels are allowed to be connected. They can be represented visually as a graph, which we will call *Neighbourhood Constraint Graph S* as shown in Figure 1(a).

In the given example, both patients, *:irene* and *:ali*, can have *:fever* and *:cough* since S has edges connecting each of them to the *:fever* and *:cough* nodes. Furthermore, they can both receive treatment with *:paracetamol*, while only *:irene* can be treated with *:aspirin*. This distinction arises from the fact that *:ali* is allergic, as indicated by the **absence** of an edge towards the *:aspirin* node. Consequently, the fact that *:ali* was treated with *:aspirin* - as depicted in G - represents a violation of the neighborhood constraints (highlighted in red).



(a) A graph with a violation (left) and and neighborhood constraint graph (right)



(b) Two possible repairs for G: relabeling (top) or deletion (bottom)

Figure 1: Inconsistent data detected by neighborhood constraints, and possible repairs.

This violation can be repaired in many possible ways, for example, as shown in Figure 1(b), either relabeling the node *:aspirin* into *:paracetamol* or deleting the edge between the two nodes.

A non-interactive algorithms, like in [38], or methods that prioritize repairs [39] cannot easily choose which repair to apply because they do not understand the semantic context as it lacks domain-specific knowledge (e.g., the fact that, in the example, the fever needs to be treated with medicine - the paracetamol).

Although configuration parameters such as weights in the algorithm, or other information about the data and its source [39] can tune the approach to either prefer delete or update repairs, this preference would be applied to every violation globally for the entire graph and not a case by case basis. Instead, a user would be able to evaluate each specific case independently (e.g., knowing that the proper solution for a given patient is to administer paracetamol instead of aspirin whereas for another patient the opposite prescription would be the correct fix). Also, while schema-level knowledge may help to decide which repair to perform (e.g., enforcing that *:ali* needs to be treated with a medicine), many real-world graphs do not exhibit a schema [35] and the proposals for schemas for graphs are still undergoing [4].

Contributions. In this work, we address the problem of interactive graph repair. Our contributions can be summarized as follows. First we provide a theoretical model of the user involvement in the graph repair process, and we show that minimizing the number of interactions under our assumptions is NP-hard. The solution space is represented using *Question-Answer-Repair (QAR) frameworks*, consisting of (1) the space of possible questions, (2) the space of admitted answers, and (3) the application of answers to repair the graph. Second, we define a set of desirable properties, namely termination, oracle optimality, and question difficulty to characterize our solution. In addition, we propose different variations of solutions built by defining an assortment of QAR frameworks. Finally, we evaluate our approach for the presented QAR frameworks with both synthetic and real-world datasets, considering different classes of users with behaviors ranging from answering randomly to oracles. We show that the QAR framework's performance heavily depends both on data characteristics and user behavior. All users are able to achieve relatively good results, with some reaching almost perfect repair quality with respect to ground truth.

The remainder of the paper is structured as follows: Section 2 summarizes the related works on graph dependencies, repairs, and humans-in-the-loop approaches, Section 3 introduces the main objects and theoretical concepts used through the paper, Section 4 provides the theoretical formulation of the interactive graph repair problem, while Section 5 describes in details our solution approach. Section 6 shows our experimental evaluation. Finally, Section 7 concludes and discusses future work.

2 RELATED WORK

Graph dependencies. As for relational databases, graph databases need to express dependencies over their data to ensure the quality of the data and to specify its desired structure. These dependencies have progressively developed and generalized to *Graph Entity Dependencies* (GEDs) [14]. These can express conditions on multi-labeled edges and vertices of Property Graphs, by matching subgraph patterns. Some theoretical results on classical problems exist for GEDs, such as satisfiability, implication, and validation[14]. In addition, the Chase procedure [34] is adapted to GEDs and known to be finite and respects the Church-Rosser property, namely that all terminating Chase procedures have the same result. A more complete overview of graph dependencies, before GEDs can be found in [8]. In this work, we focus on a subclass of GEDs called Neighborhood Constraints that can be expressed on vertex-labeled graphs. Neighborhood Constraints express which vertex labels are allowed to be connected. They are introduced in [38] and some theoretical results have been shown. We make use of the related results without studying them further.

Graph repairs. There are several methods to clean data which does not respect integrity constraints: the deletion model [11] consists in pruning the rotten data, while the update model [45] applies transformations to repair the data to make it satisfy the constraints. In this work, we work with a combination of both. Expressing constraints and repairing graph databases is studied over different dependency classes [8, 14, 25]. In [38], repairing is done with both deletion and update for the aforementioned Neighborhood Constraints. The authors define conflicts and different types of repairs, namely relabelling (update) and edge deletion. The problem of minimally repairing the graph in this scenario is shown to be NP-complete, but elaborate fixing strategies are shown to achieve constant factor approximations in the case of bounded maximum graph degree. Moreover, special tractable cases are studied and experiments show the effectiveness and application scenarios of the design. As in [38] we use Neighborhood Constraints to express integrity constraints and use the same repair model, namely vertex-relabeling and edge-deletion. However, we improve on the use of heuristics by leveraging user intelligence by developing a user-interaction model to involve humans in graph repair.

Human-in-the loop methods. Human-in-the-loop methods integrate human insight or judgments into machine-driven processes. They are typically used when an algorithm alone cannot achieve good results because specific domain knowledge is needed [30, 37] or because the existing algorithms are not good enough [6, 29].

Involving humans is generally very expensive in terms of time and effort (and thus money). For this reason, in the last decade, a lot of effort was dedicated to optimizing the use of crowdsourcing [7, 22, 23] and developing sound methods to run experiments [33], leading to its successful application in different

domains, such as image classification [24], transcription [37, 42], natural language processing [19], health [44] and many others.

In the data management community, crowdsourcing has been used to answer queries that cannot be answered by computers alone, focusing on minimizing the cost of answering such queries [13, 15, 28, 41]. Specifically in the graph domain, Cong et al. [12] studied the Interactive Graph Search (IGS) problem - i.e., finding a target node using human intelligence - developing algorithms that minimize the number of questions asked to the users.

Interactive data repairing has also been used in our previous work [5], where we focused on knowledge bases that are evolving, i.e. augmented with new facts using existential rules. Repairing was studied in a setting in which denial constraints are applied to evolving knowledge bases, and thus interacting with existential rules. Knowledge bases are leveraging facts and flat relational data and are quite different from graph-shaped data, as handled in this paper. We focus on more expressive data models and models of user interaction using special constraints for graphs. The constraints we consider (neighboring constraints) are topological as they check the admitted adjacent labels on two edges in a path.

3 PRELIMINARIES

In this section, we introduce the basic ingredients of the interactive graph repair framework, namely the concepts of labeled graphs, neighborhood constraints, violations, and repairs. In this paper, we consider vertex-labeled undirected graphs. Note that edge-labeled graphs (both directed and undirected) can be easily converted into the above [17].

Definition 3.1 (Labelled graph)**.** *A labelled graph G is a triple $G = (V, E, \lambda)$, where V is the set of vertices, $E \in V^2$ is the set of edges and $\lambda : V \rightarrow L$ be a labelling function assigning to a vertex in V a single label l from finite set of labels in L.*

For the sake of simplicity, from now on, we will refer to labeled graphs as graphs, unless stated otherwise. Graphs will be used to represent data - called instance graph G. To express constraints over the graphs, we introduce the Neighborhood Constraint Graph.

Definition 3.2 (Neighborhood Constraint Graph [38])**.** *Let $L = \{l_1, ..., l_{|L|}\}$ denote a set of labels. A neighborhood constraint graph $S = (L, N, \lambda)$ is an undirected graph, where N is the set of pairs of labels in L allowed to be connected. Intuitively, it specifies whether two labels are allowed to appear as neighbors.*

Figure 1 shows example instance and constraint graphs. As previously stated, G states that *:ali* is prescribed *:aspirin*, which is dangerous. When the data does not satisfy the constraint graph, it is called a violation.

Definition 3.3 (Violation)**.** *For a graph G and a constraint graph S, an edge $e = (u, v) \in E$ satisfies the set of constraints in S iff for $l_1 = \lambda(u)$ and $l_2 = \lambda(v)$ there exists an edge in the constraint graph $n = (l_1, l_2) \in N$. Otherwise, e is a violation of the constraints S.*

By extension, G violates S if any of its edges violate S. Otherwise, G satisfies S. For instance, in Figure 1, the instance graph G violates S. The task of enumerating the violations of a graph is solved by procedure `Violations(G,S)`, which takes as input an instance graph and a constraint graph and outputs the list of all violations. This can be done by iterating over the edges. If the list is empty, the constraints are satisfied.

Given a graph with violations, repairing the graph consists of applying modifications to the instance graph to obtain a graph that satisfies the constraints. Before introducing the concepts of graph repair formally, we establish the allowed graph transformations. As in previous work [38], we consider relabeling and edge deletion.

Definition 3.4 (Graph transformations)**.** *Let $G = (V, E, \lambda)$ be a labelled graph. A relabeling of a vertex $v \in V$ to label $l \in L$ returns the transformed graph where v has label l. Formally, $relabel(G, v, l) = G' = (V, E, \lambda')$ where $\lambda'(v) = l$ and $\forall u \in V \setminus v : \lambda(u) = \lambda'(u)$*
An edge-deletion returns the graph without an edge $e \in E$. Formally, $edge\_delete(G, e) = G' = (V, E \setminus e, \lambda)$

With these two available transformations, the possible repairs for a violation are the following:

Definition 3.5 (Repair)**.** *Given a graph G, constraints S and a violation e. A relabeling of G yielding G' in which e respects S is a relabeling repair for e. An edge-deletion of e is a deletion repair for e.*

Note that in this scenario, a relabeling repair might solve the violation e whilst introducing others. The number of possible repairs for a violation $e = (u, v)$ is $|neighbors(S, u) \cup neighbors(S, v)| + 1 = O(d_S)$, with d_S the maximum degree of S.

By extension, we call G' a repair for a graph G and a set of constraints S if there exists a sequence of repairs $R = (r_1, ..., r_m)$ that applied in sequence to G yield G' and G' respects S. Note that with edge-deletions allowed, there always exists such a sequence. One can remove all the edges, in which case no constraint applies.

In Figure 1(b), the right repair is the result of deletion for the conflict between *:ali* and *:aspirin* in G. As there is no further conflict in G, it is a repair of G. Similarly, the left repair is the result of one of the possible relabeling of *:aspirin*, and does not induce more violations. It is thus another repair of G.

We are now ready to define the questions, which are the last ingredients of our interactive graph repair framework.

Definition 3.6 (Question)**.** *Given a graph G, constraints S, let I be the set of violations of the constraint graph S in G and R the set of graph transformations to repair the set of violations I. Also, let $i \in I$ be a violation and $R' \subseteq R$ the set of possible graph transformations applicable to a violation i. A question q asks which of the graph transformations $r \in R'$ is a fix for the violation i.*

Let's consider the example reported in Figure 1. A possible question asks to the user if the graph transformation *relabel(:aspirin,:paracetamol)*, among all the others, fix the violation *(:ali,:aspirin)*.

Notice that the above definition of a question is a general one, but a question can also be boolean and ask whether a repair r is a fix for a given violation. Indeed, the above definition of question can lead to an equivalent set of boolean questions, one question for each $r \in R$. For ease of exposition, in the next Section, we mainly focus on boolean questions to introduce the problem statement and its hardness.

4 THE INTERACTIVE GRAPH REPAIR PROBLEM

In this section, we formalize the problem of interactive graph repair, prove its NP-hardness and characterize its solution space.

4.1 Hardness analysis

One of the main issues related to interaction with users is the cost, both in terms of time and money, of asking for human

intervention from either domain experts or inexperienced users. We formalize the problem statement as the following.

Definition 4.1 (Interactive Graph Repair Problem)**.** *Let S be the Neighborhood Constraint Graph and G an instance graph that violates some constraints in S. An interaction is to ask a question q_i and receive an answer a_i. Assume w.l.o.g. that each interaction applies a single graph transformation (either relabelling or deletion).*

The Interactive Graph Repair (IGR) problem asks for a minimum length sequence of questions $Q = (q_1, ..., q_n)$ such that there exists an associated sequence of answers $A = (a_1, ..., a_n)$ that repairs the graph.

Note that in the definition we do not assume any property of the answers nor any particular user behavior. To simplify, we use a bounded version of the problem:

Definition 4.2 (C-Bounded Interactive Graph Repair Problem)**.** *Given the Neighborhood Constraint Graph S and an instance graph G, the C-bounded IGR problem asks for a sequence of questions of length $l < C$ that solves IGR.*

We first show some results for a special case of constraint set that will prove useful further.

Definition 4.3 (Centered Neighborhood Constraints)**.** *A centered neighborhood constraints $S(L, N)$ is a labeled graph such that for the labels $(l_1, ..., l_n)$ and a label l_0, $N = \{(l_0, l) : l \in L \setminus l_0\} \cup \{(l_0, l_0)\}$. Namely, all labels are connected to a center label - and the center label with itself - but no other.*

Lemma 4.4 (Equal length deletion-free repair for Centered Neighborhood constraints)**.** *For some graph G and a centered constraint set S, let $R = (r_1, ..., r_k)$ be a repair for G with at least a deletion repair. There exists an equal length repair R' for G without deletion repairs. Given R, finding some R' takes linear time.*

Proof. R' is constructed by replacing deletion repairs with relabelling to the center label of the centered constraint set. Let $r_i \in R$ be a deletion repair for a conflict (u, v). Let R' be $R - r_i + r'_i$ where r'_i is a relabelling of u with the center label. Because the constraints have a center, r'_i must be a repair. Assume towards contradiction that applying R', resulting in G', does not repair G. There cannot be any violation including u because it has the center label. As G' is not a repair of G, there must exist some $w \in V$ such that (v, w) is a conflict in G'. But R is a repair, so there is some $j \neq i : r_j \in R'$ that fixes (v, w), which is a contradiction. If R' still contains deletion repairs, repeat the process until R' is without deletion repairs. This process takes linear time in the length of R. □

Now we have all the elements to show that the problem is NP-Complete.

Lemma 4.5. *C-bounded IGR is in NP.*

Proof. Given G, S and a sequence of questions $Q = \{q_1, ..., q_c\}$, we need to verify in polynomial time that Q is a valid solution. This means verifying that there exists an answer sequence $A = \{a_1, ..., a_c\}$ that yields some repaired G'. The encoding of the input being a sequence of boolean questions is done by using sequences of bits, each of which corresponds to a boolean question (being true or false) on a graph transformation as a possible repair to a violation. Therefore, since each question q correspond to a violation and a proposed graph transformation, it is fair to assume that the size of the questions - and as consequence the size of the answers - is linear w.r.t to the size of G and S. Next, note that the full space of possible answers is smaller than the set of allowed transformations, i.e. any relabelling or deletion, which has size in $O(|V|(1 + |L|)) = O(|V||L|)$. Thus, the space of possible sequence of answers to questions Q is in $O(|V|^c|L|^c)$, and can be enumerated in polynomial time. To verify if an answer sequence A yields a repaired graph takes polynomial time: applying a transformation is in linear time, and verifying the constraints on the resulting graph is in polynomial time [38]. Applying a polynomial-time procedure to a polynomial amount of elements takes polynomial time. Then, given a sequence of questions Q, one can generate and verify all possible answer sequences. If at least one of the yielded graphs respects S, then Q is a correct certificate to the C-bounded IGR problem. The verification is done in polynomial time, the problem is in NP. □

Lemma 4.6. *C-bounded IGR is NP-hard.*

Proof. We reduce the vertex cover problem to C-bounded IGR. Given an unlabeled graph $G = (V, E)$, the vertex cover problem asks for a set B of at most C vertices such that they cover all edges, i.e. for all $e = (u, v) \in E$, at least one of u or v is in B. This problem is known to be NP-hard.

The reduction is as follows: Let $G' = (V, E, \lambda)$ be a labeled graph with vertices uniquely labeled from 1 to $|V|$. Let $S(L, N)$ be a *centered* neighborhood constraints with $L = \{0, 1, ..., |V|\}$ with $N = \{(0, i) : i \in 1, ..., |V|\}$. That is, vertices are only allowed to be connected to label 0 or themselves. This takes polynomial time. We show that G has a vertex cover B of size $c \leq C$ if and only if G' has a question sequence Q with size $c' \leq C$ that solves C-bounded IGR

We first show that a sequence of questions that solves the C-bounded IGS problem implies a solution for the vertex cover problem. Let Q be a set of questions that solves the C-bounded IGR for G' and S, with some associated deletion-free repair R. Given Q, such R can be found in polynomial time by seeking an answer sequence that repairs the graph and obtaining a deletion-free version (lemma 4.4 and 4.5). Let B be the set of vertices relabeled by R. We argue that B is a vertex cover of G, showing that all edges have a vertex in B. Assume towards contradiction that there is some $(u, v) \in E$ such that $u, v \notin B$. By construction, $\lambda(u) \neq \lambda(v)$ in G'. After the application of the repairs in R, this constraint is still violated because none of u or v show in B. This is a contradiction with the fact that R yields a repaired graph.

Second, if there exists a vertex cover B of size less than C in G, we show there exists a question sequence Q that solves C-bounded IGR. Note that relabelling all vertices of B as 0 yields a repaired graph. For some ordering of B, let q_i be any question such that relabeling b_i is an answer. If the vertex cover is not minimal, such conflict may not exist. This is a better solution to the problem. This yields Q of length less than C for which the sequence of answers $\lambda(b_i) = 0$ is a repair.

This is a polynomial time reduction from the vertex cover problem to the C-bounded IGR problem. The C-bounded IGR must be NP-Hard. □

We can then prove the NP-completeness of the Bounded IGR problem in the following.

Theorem 4.7. *C-Bounded IGR is NP-Complete.*

Proof. Lemma 4.5 shows that the problem is in NP. Lemma 4.6 shows that the problem is NP-hard. □

4.2 The solution space

The problem's hardness implies that finding the optimal solution is intractable. In the following sections, we will introduce, investigate and compare solutions that use heuristics towards solving the IGR problem. Here, we characterize them with a common abstraction.

To formulate solutions to the interactive graph repair problem, we introduce the following user interaction model composed of three phases (QAR - Question, Answer, Repair). First, given the instance graph G and the set of neighborhood constraints S, we need to generate the question q (phase Q). Second, we need to ask question q to the user and define the acceptable answers (phase A). Finally, we need to apply the user's answer a to the graph G to repair the conflict (phase R). These three phases (QAR) are repeated until the graph is repaired, i.e. G does not violate any constraints in S.
This leaves us with the following design dimensions:

- the definition of the question space $\mathcal{Q} : G, S \rightarrow Q$. It defines the elements that will form the question posed to the user. For example, the question space Q could be the set of conflicts in G w.r.t to S asking the user to provide the desired repair; or it could allow the user to choose the repair from a panoply of possible graph transformations.
- the answer space definition $\mathcal{A} : G, S, Q \rightarrow A$. Given a question q, the function $\mathcal{A}(G, S, q)$ returns the allowed answers that a user might provide. Examples of possible answers are: all possible graph transformations (delete + relabeling) for a conflict c, or only the vertex relabeling edits.
- the question selection procedure, `SelectQuestion`. This function takes in input the graph G, the constraint graph S and the question space Q to select the next question to be asked to the users. Notice that this function can be arbitrarily complex and take into account various information, such as the topology of the input graph. However, in this work, we exemplify it as a greedy function (defined in Algorithm 1) that selects at each step the question that would fix the majority of the violations.
- the procedure to apply the answer to G, `Apply`. This function takes the answer and applies the fix to the graph G. Since in our work, we assume a question is answered by a single user, this function will simply apply the fix to the graph G.

We call *QAR framework* a quadruple of the aforementioned elements:

$$\mathcal{F} = (\mathcal{Q}, \mathcal{A}, \texttt{SelectQuestion}, \texttt{Apply})$$

The above abstraction defines the space of possible solutions to the IGR problem. Framework instantiations will precisely characterize a potential solution in the space. For instance, a framework could select a question randomly or select one that fixes most violations, it could ask boolean questions (e.g., "Is this a correct fix for this violation?"), or ask the users to propose a fix by themselves.

Figure 2 illustrates a possible interaction. Given the constraint graph S and the instance graph G with a violation (e.g., *:ali – :aspirin*), a framework instantiation will select a question q, generate the possible answers $\mathcal{A}(q)$ and ask the question to the user. Then, the provided answer (e.g., the relabeling of *:aspirin* into *:paracetamol*) is applied to the graph G, fixing the violation.

The next section will construct, study and compare some QAR framework instantiations and their properties.

5 INTERACTION DESIGN

The definition of the question and answer spaces can completely change how the interactions pan out. They can be very restricted to ensure users do not go off trail, or very open to allow more expert users to progress freely. A framework can ask the users for a simple yes/no answer, or a complete description of a bigger solution. We first investigate desirable properties and explore how different designs impact them. In particular, we study (i) the *quality* of the repair and (ii) the *efficiency* of a framework and (iii) the user perceived *difficulty*.

5.1 Properties

We consider the repaired graph G^* and the sequence of repairs R^* needed to achieve it to be optimal if G^* corresponds to the ground truth - independently of the number of repairs needed to achieve it.

It is desirable for a framework to allow a user to reach the desired G^*. We formalize this notion as the *oracle optimality* property, closely following [5]:

Definition 5.1 (Oracle)**.** *For a graph G and a set of constraints S, a user who knows the repaired graph G^* and associated repairs R^* is an oracle.*

Definition 5.2 (Oracle Optimality)**.** *A QAR framework $\mathcal{F}$ has the Oracle Optimality property iff a user with an optimal repair G^* with associated repairs R^* in mind can answer all questions within R^* such that G^* is returned. That is, for any selected question* q*, $R^* \wedge A_q \neq \emptyset$.*

A framework might not be oracle optimal, in which case the oracle either cannot reach the optimal repair because it obtains another repair first and the algorithm terminates or because they need to answer outside R^* (i.e. making detours) - because the specific QAR framework forces it.

To reach the optimal solution, more repairs may be necessary than simply fixing the inconsistency. For instance, in Figure 3, the optimal repair involves two transformations (e.g., relabeling *:fever* to *:caugh* and *:tea* to *:syrup*). However, deleting the edge between *:fever* and *:tea* would resolve the inconsistencies with just one transformation. Moreover, in this case, if a framework restricts the answer space to the transformations that are *repairs* it would be impossible to reach G^*, as none of $relabeling(G, b, :caugh)$ or $relabeling(G, c, :syrup)$ are repairs for the current violation.

This Oracle Optimality property expresses the framework's ability to obtain high *quality* repairs, as it states if a framework allows it to reach an oracle's solution.

As expressed in the IGR problem, to characterize the efficiency of a solution, we consider the number of questions asked to the user to repair the graph. For a lower bound on the number of questions, we show that for any k violations there exists a graph that can be repaired with one transformation, and thus a single question.

Proposition 5.3. *For any number of initial violations k, there exists a graph G and constraints S that can be repaired with one transformation.*

Proof. It is sufficient to construct an example. Let G be a star graph with k nodes - with $k > 2$ - , two labels, and where all edges are violations. Specifically, G has a center node v_0, and all $v_1, ..., v_k$

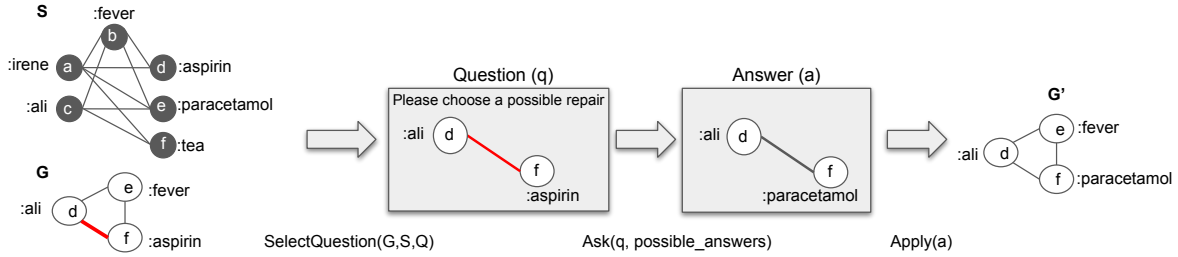


Figure 2: Possible iteration of a user repair. In this case, Q is the list o violations and *possible_answers* comprehends relabeling and deletions.

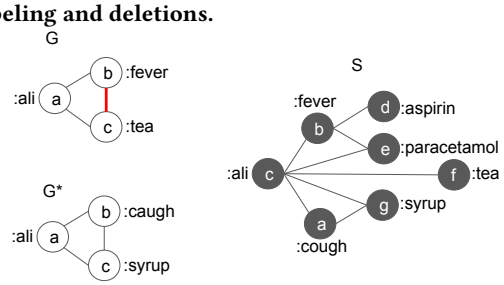


Figure 3: An instance graph G and its associated optimal repair G* (i.e., equal to the ground-truth) on the left. The corresponding constraint graph S is on the right.

have an edge to v_0. Let $\lambda(v_0) = l$ and $\forall 0 < i \leq k : \lambda(v_i) = l'$, and let neighboring l and l' be a violation of the constraints. There are k violations, and the unique transformation $relabel(G, v_0, l')$ will result in a valid graph with respect to S. □

This shows that there is no minimum number of repairs related to the number of violations in the instance graph. On the other hand, there exists infinite repair sequences even for small graphs.

Example 5.4. For instance, take G as in Figure 3. Assume some framework that allows relabeling a vertex multiple times. One can infinitely relabel vertex c, alternating between *:tea* and *:aspirin*, to which *:ali* is allergic, thus never reaching a repaired graph.

This leads to the definition of the Termination property:

Definition 5.5 (Termination). *A QAR framework $\mathcal{F}$ has the Termination property iff for any G of finite size, there exists no infinite questions and answers sequence that does not reach a repair of G.*

Note that the Termination property does not consider the possibility that a user may receive a question and never answer.

The ability of a user to answer a question varies a lot, and it depends on their knowledge, skills on the task's topic, and difficulty.

Based on the literature on crowdsourcing task difficulty [46], we model the question difficulty in relation to the intrinsic complexity of the task, that, in our case, corresponds to the number of question items (vertices) and the number of possible answers. With this, we define our difficulty measure for a question. Further modeling of user knowledge and behavior [16, 21], and investigation of the effect of the user interface [32] on the perceived difficulty of a graph repair task is out of the scope of this work and should be addressed in future works.

Definition 5.6 (Question difficulty). *Let q be a question selected by $\mathcal{F}$ and A_q be the associated possible answers. Further, let $|q|$ return the number of vertices needed to represent question q. Then, the difficulty of question q is defined as follows:*

$$difficulty(q) = \alpha|q| + \beta|A_q|$$

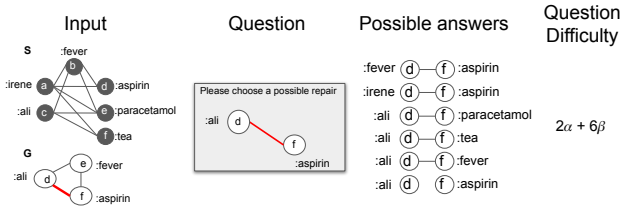


Figure 4: Question difficulty for $\mathcal{F}_{greedy}$ *with* $0 \leq \alpha, \beta$.

Where α and β are parameters that regulate the importance of the number of vertices and possible answers, respectively.

Example 5.7. Figure 4 shows how to question difficulty works with an example QAR framework using our running scenario. In the figure, the answer space is composed of the correct repairs. Thus

$$difficulty(\text{"repair d-f"}) = 2\alpha + (3 + 2 + 1)\beta$$

This is because the question needs 2 vertices to represent the violation, there are three possible relabeling for the node f, two possible relabeling for d, plus the deletion.

To characterize difficulty at the framework level, we are interested in the maximum question cost, as a function of the instance graph and constraints.

Definition 5.8 (Framework Difficulty). *For a QAR framework $\mathcal{F} = (\mathcal{Q}, \mathcal{A}, \texttt{SelectQuestion}, \texttt{Apply})$, Framework Difficulty is the maximum difficulty of a question q in $\mathcal{Q}$, as a function of G and S:*

$$\max_{q \in \mathcal{Q}(G,S)} difficulty(q)$$

Example 5.9. For instance, let's consider the situation from Figure 4 where the question asks to solve a violation by providing a repair. First, for any question, $|q| = 2$. Then, the maximum set of answers contains all possible relabeling for each node (u, v), plus one for the deletion. In the worst case, node u can be relabeled by all the neighboring labels of $\lambda(v)$ in S. There are at most d_S of them, the maximum degree of S. This results in a framework difficulty of $2\alpha + (2d_S + 1)\beta$.

We are interested in how the difficulty grows with input G and S. In this case:

$$difficulty(\text{q}) = O(d_S)$$

5.2 Proposed Frameworks

In this part, we progressively introduce novel QAR frameworks and study their properties in terms of oracle optimality, termination, and framework difficulty.

The first framework leverages a greedy heuristic for question selection, as done in the non-interactive repair [38], and allows

$\mathcal{F}$	Question	Answers	Oracle Optimality	Termination	Framework Difficulty
$\mathcal{F}_{greedy}$	violation	repairs	No	No	$O(d_S)$
$\mathcal{F}_{permissive}$	violation	any transformation	Yes	No	$O(\|L\|)$
$\mathcal{F}_{term}$	violation	decreasing repairs	No	Yes	$O(d_S)$
$\mathcal{F}_{bool}$	violation + repair	boolean	No	No	$O(1)$
$\mathcal{F}_{problem}$	$G + S$	G'	Yes	Yes	$O(\|G \times L\|)$

Table 1: Summary of defined QAR frameworks and their properties,with d_G and d_S the respective maximum degrees for G and S. All the QAR frameworks use the `SelectQGreedy` function to select the question, except for $\mathcal{F}_{bool}$ that selects the questions randomly, and $\mathcal{F}_{problem}$ that retrieves all the graph.

answers that correspond to repairs for the selected violation.

$$\mathcal{F}_{greedy} = (\mathcal{Q}_{conflict}, \mathcal{A}_{repair}, \texttt{SelectQGreedy}, \texttt{Apply})$$

Where $\mathcal{Q}_{conflict}$ is the set of conflicts in G and A_q be the set of possible repairs for q.

$$\mathcal{Q}_{conflict} = \{e = (u, v) : (\lambda(u), \lambda(v)) \notin N, e \in E\}$$

$$\mathcal{A}_{repair}(u, v) = \left\{ \begin{array}{l} relabel(G, u, l) : \ (\lambda(v), l) \in N, \\ relabel(G, v, l) : \ (\lambda(u), l) \in N, \\ \text{delete } (u, v) \end{array} \right\}$$

`SelectQGreedy` can be defined as the function that returns the question that contains the vertex u that acts in the most conflicts (u, v). Pseudocode for `SelectQGreedy` is in Algorithm 1. Then, the proposed fix is applied directly to the graph.

Note that with this framework, the user may choose a transformation that solves a violation but create a new one. For instance, in Figure 3 the answer could have been to relabel *:tea* to *:aspirin*, but *:ali* cannot be treated with *:aspirin*, and the repair would need another iteration. This is because this framework's specific design does not restrict the answer space, while another can forbid answers that would introduce new conflicts.

Algorithm 1 SelectQGreedy

```
Input: graph G=(V, E), constraints S, possible questions Q
Output: a question q
for v ∈ V do
    conflicts[v] ← 0
end for
for q=(u,v) ∈ Q do
    conflicts[u] ← conflicts[u] + 1
    conflicts[v] ← conflicts[v] + 1
end for
u ← argmax(conflicts[i], i ∈ V)
v ← argmax(conflicts[v], v ∈ V and q=(u,v) ∈ Q)
return q=(u,v)
```

To summarize, the rationale of this QAR framework is to greedily select questions with the most impact potential, and allow only repairs as answers. But we have seen this can have drawbacks. Firstly, this QAR framework is not Oracle Optimal, and, while it force repairs it does not take into account the side-effects, and might thus never terminate.

Proposition 5.10. *$\mathcal{F}_{greedy}$ has the following properties:*

(a) *$\mathcal{F}_{greedy}$ is not Oracle Optimal*
(b) *$\mathcal{F}_{greedy}$ may never terminate*
(c) *$difficulty(\mathcal{F}_{greedy}) = O(d_S)$*

Proof. (a) It suffices to provide a counterexample in which an oracle cannot answer from R^*. Take G and G^* as shown in Figure 3, such that $R^* = (relabel(G, b, : \ cough), relabel(G, c, : syrup))$. The greedily selected question `Ask((b,c))` yields the set of allowed answers $A_{repair(b,c)}$. But $A_{repair(b,c)} \cap R^* = \emptyset$.

(b) It suffices to find a graph with an infinite and deterministic question-answer sequence that never reaches a repair. Example 5.4 applies to $\mathcal{F}_{greedy}$ and does so.

(c) Example 5.9 applies. □

To address the lack of Oracle Optimality, we extend $\mathcal{F}_{greedy}$ by broadening the answer space to any transformation of the violation - i.e., relabeling and deletion:

$$\mathcal{A}_{trans}(u, v) = \left\{ \begin{array}{l} relabel(G, u, l) : l \in L, \\ relabel(G, v, l) : l \in L, \\ \text{delete } (u, v) \end{array} \right\}$$

Resulting in the following QAR framework:
$\mathcal{F}_{permissive}$ = $(\mathcal{Q}_{conflict}, \mathcal{A}_{trans}, \texttt{SelectQGreedy}, \texttt{Apply})$.

Note that $\mathcal{A}_{trans}$ allows relabeling into any label from the label set L, even if it might not solve the selected violation. The idea is to allow the users to plan the repair, as while a graph transformation may not fix a violation right away, it can lead to a correct solution later on. For instance, in Figure 3, relabeling *:tea* to *:syrup* will not solve the violation as it does not treat fever, but further relabeling *:fever* to *:cough* achieves the optimal repair.

Proposition 5.11. *$\mathcal{F}_{permissive}$ has the following properties:*

(a) *$\mathcal{F}_{permissive}$ is Oracle Optimal*
(b) *$\mathcal{F}_{permissive}$ may never terminate*
(c) *$difficulty(\mathcal{F}_{permissive}) = O(|L|)$*

Proof. (a) Assume towards contradiction that there is a conflict $e = (u, v)$ such that $A_{transform(e)} \cap R^* = \emptyset$. Then, for G and G^*, $\lambda(u) = \lambda^*(u)$ and $\lambda(v) = \lambda^*(v)$ and $e \in E^*$ by definition of R^*. This implies that G^* does not respect the constraint set, which is a contradiction.

(b) For any conflict $e = (u, v)$, a user can answer to relabel u into $\lambda(u)$, without changing anything, an infinite amount of times.

(c) The set of answers has the same size for all questions, namely $2\alpha + (2|L| + 1)\beta = O(|L|)$. □

This framework no longer forces progress, i.e. a user interaction can result in no change. In terms of question difficulty, this framework asks the same question as $\mathcal{F}_{greedy}$, but the answer space is larger as it grows with the label set L, which may be much larger than the maximum degree of S. Giving more freedom to users results in Oracle Optimality, at the cost of increased difficulty.

On the other hand, a framework can restrict the possible answers to ensure the repair terminates. The framework $\mathcal{F}_{term}$ allows only transformations that strictly decrease the number of violations. This is always possible by proposing a deletion. Limiting the answer space like this requires investigating potential fixes and counting the new violations before applying them.

Let $\mathcal{A}_{decr}$ be the function that given a violation (u, v) returns the aforementioned space of transformations that both (i) repair the violation and (ii) decrease the overall number of violations. This includes the deletion repair and any relabeling r of u or v such that $|violations(G, S)| > |violations(\texttt{Apply}(G, (u, v), r), S)|$.

Proposition 5.12. *$\mathcal{F}_{term}$ has the following properties:*

(a) *$\mathcal{F}_{term}$ is not Oracle Optimal*

(b) $\mathcal{F}_{term}$ *terminates*
(c) $difficulty(\mathcal{F}_{term}) \leq difficulty(\mathcal{F}_{greedy})$

Proof. (a) Same as Proposition 5.10.(a)

(b) Let S be the constraint graph and G_0 be the original graph. Let G_j be the state of the graph after j user interactions. By construction of $\mathcal{A}_{decr}$, the number of violations strictly decreases as j increases. For some $j \leq |violations(G_0, S)|$, the number of violations of G_j is 0. This is a repaired graph and the algorithm terminates.

(c) For any question q, $|\mathcal{A}_{decr}(q)| \leq |\mathcal{A}_{repair}(q)|$. □

Like $\mathcal{F}_{greedy}$, this framework is not Oracle Optimal, but it does terminate. In terms of question difficulty, this framework improves because, while the question part does not change, the answer space is at most as big as in $\mathcal{F}_{greedy}$.

Note that with this, we have a framework that takes at most k interactions to terminate, with k the number of violations in the original graph, for any graph and constraint set.

We can also investigate other frameworks at the extreme of the framework difficulty, both easy and hard. At the hard extreme, we can give the whole problem to the user in a single interaction. $\mathcal{F}_{problem}$ asks the users to repair the graph completely. The question space is thus the input graph as a whole, and the answer space is any repair of the graph. This framework is Oracle Optimal and will terminate in one step. However, the question difficulty is very high, as this framework asks an NP-complete task to the user [38].

On the other side of the spectrum, we may benefit from asking very easy questions to the user. This might require many more interactions than with other frameworks, but the tradeoff with some types of users may still be worthwhile in some cases. $\mathcal{F}_{bool}$ asks a yes/no question, in which users agree to a repair or not. The questions thus incorporate both the violation and the proposed repair, but the answer space is boolean. For $\mathcal{F}_{bool}$, the `SelectQuestion` procedure randomly selects a possible repair of some violation in the graph, and the `Apply` procedure applies the repair if the user's answer is "yes", or does nothing. Regarding the difficulty, assume input as in Figure 4, and a question q = "repair d-f with relabel(G,f,:tea)". The question is harder to understand as it needs to represent both the violation and the proposed repair, 4 vertices in this case. But answering the question is much easier as there are only two possibilities to consider (e.g., yes or no):

$$difficulty(q) = 4\alpha + 2\beta$$

Proposition 5.13. $\mathcal{F}_{bool}$ *has the following properties:*

(a) $\mathcal{F}_{bool}$ *is not Oracle Optimal*
(b) $\mathcal{F}_{bool}$ *may never terminate*
(c) $difficulty(\mathcal{F}_{bool}) = O(1)$

Proof. (a) Any question that does not allow the Oracle to make progress will break Oracle Optimality. This includes any repair that does not go towards the oracle's G^*, and in this case the questions are selected randomly.

(b) The framework may ask the same question infinitely if the answer is always "no".

(c) If the question includes a relabeling, then $difficulty(q) = 4\alpha+2\beta$. Otherwise, the question includes a deletion and $difficulty(q) = 4\alpha + 2\beta$. Both cases are $O(1)$. □

Although this QAR framework loses Oracle Optimality and Termination properties, it greatly improves on difficulty, as the framework complexity does not grow with the size of the input graph.

Table 1 summarizes the QAR frameworks designed up to this point. The oracle optimality correlated with expressiveness of the questions and the user's freedom of choice. However, as expected, this is at the cost of having more difficult questions.

Finally, notice that our theoretical model is general and can be used to create many other framework designs with expressive powers ranging between and beyond these extremes. Their study and comparison are left as future work.

6 EXPERIMENTS

In this section, we present an extensive experimental study devoted to gauging the effectiveness of our approach. The datasets we used are comprised of the ground truth G^*, the instance graph G, and the constraint graph S. For each dataset, the result of the repair produces a repair graph G'. The different datasets are presented in Section 6.1 and Table 2 summarizes their characteristics. We analyze our interactive approach under the different QAR frameworks in terms of the quality of the repair, the efficiency of the repair, and question difficulty (more details on the metrics are in Section 6.3).

All the material necessary to reproduce the results is available on our online repository [2]. The experiments were run on a desktop machine with 16GB of main memory and an Intel i7 processor in a Jupyter environment using Python 3. For each experiment parameter combination, 20 independent experiments were run to deal with randomness.

6.1 Datasets and Ground Truth

We evaluate our approach on four different datasets, one synthetic and three real-world ones. In the following, we explain how they were generated or adapted to our experiment, how we obtained the ground-truth, the instance, and the constraint graphs. Table 2 summarizes each dataset's properties. Our choice of the datasets and their sizes are motivated by the fact that users can provide interactions on datasets of small or medium sizes.

6.1.1 Synthetic Dataset. To run extensive experiments in a controlled environment, we generated several ground truths having different sizes and densities, for both the instance and constraint graphs. We generate ground truth $G^* = (V, E, \lambda)$ and $S = (L, N)$ using the following hyperparameter combinations:

- size $|V| \in (20, 100, 500)$
- G density $\frac{|E|}{|V|} \in (1, 2, 4)$
- label-to-vertex ratio $\frac{|L|}{|V|} \in (0.2, 0.5, 1)$
- density of the constraint graph $\frac{|N|}{|L|} \in (1, 2, 3)$

For each combination, 20 graphs were randomly generated. Size influences the amount of repairing work to be done and hence tests the scalability of our approach. The density of the graph G influences the interplay of constraints and relabeling. A high-density graph will incur more constraints on a given vertex. The label-to-vertex ratio influences how many labels exist and the chance of a vertex being assigned a label. Finally, the constraint density influences how strict the constraints are: a small density value makes strict constraints, as there is a little number of allowed labels for a given node. On the other hand, a complete constraint graph doesn't result in any constraint. Because the constraints include self-edges by definition, the density of the constraint graph must be greater than or equal to 1.

Notice that the sizes of synthetic graphs are relatively small as user interactions are assumed to occur sequentially on each question-answer pair. The issue of parallelizing user interactions and dealing with larger datasets is orthogonal to our work and interesting for future investigation.

For the analysis, we aggregate the results by the size of G. Since all parameters are proportional to $|V|$, these groups are diverse but comparable, since only the graph scale varies.

6.1.2 Patient trajectory process mining (sepsis). Process mining is the task of extracting a process based on event data. In [27], the authors analyzed the trajectories of patients in a Dutch hospital from their registration in the emergency room until their discharge. From event logs, i.e. a list of chronologically ordered events, they extracted a process diagram: a directed graph of the allowed sequence of events. The process diagram can thus be used as constraints for the patients' trajectories. The result of process mining is inherently noisy, as the process (constraints) is defined downstream to sampling. To run our experiments, we use the process diagram reported in [27] as our S. As ground truth G^*, we extract the patient paths that satisfy S. Because the instance graph is composed of paths, the density of the instance graph is very low. More details on how we processed the data can be found in our Git repository [2].

6.1.3 Restaurants similarity network (restaurants). We use a database of restaurant records [40], which contains duplicates and inconsistencies. Using the same methodology as [38], we translate a Differential Dependency (DD) on the relational data to Neighborhood Constraints on graph data. We consider a DD

$$(Address, City \rightarrow AreaCode, [0, 6], [0, 0], [0, 0])$$

which states that two restaurants with similar addresses (within an edit distance of 6), and the same city, should have the same area code. A distance of 6 is selected as a middle ground between constraint strictness and graph sparseness. Analogously, we construct a similarity network based on (*Address*, *City*) similarity, yielding an edge between similar - in terms of *Address* and *City* - restaurants. The label of a node v is the *area code* for restaurant v. The neighborhood constraints express the need for *area code* equality, such that if two restaurants are connected in the instance graph, but do not have the same *Area Code*, it is a violation. We build S as the set of *area code* as isolated vertices with self edges, G^* contains restaurants as vertices and the edges connect restaurants that are similar and satisfy the constraints.

6.1.4 Coauthor network (coauthors). The co-author network example is inspired by an entity resolution task presented in [18]. The procedure to apply neighborhood constraints is inspired by [38]. The co-author network is a network where each publication is represented by a fully-connected component of authors. Vertices are labeled with authors' names and surnames. Name overlaps result in ambiguous labels that can be considered noise. The entity resolution task is to disambiguate these overlapping names. In our case, publications on CiteSeer [1] have ambiguous author labels whereas publications on DBLP [3][1] also use affiliation. Hence, we can use DBLP as the neighborhood constraint graph S with disambiguated labels only, and "repair" the CiteSeer with the disambiguated labels to achieve entity resolution. For a more concrete example, let *Lei Chen* be an ambiguous label. The task is, from the coauthor network from CiteSeer, to relabel *Lei Chen* vertices with their corresponding affiliations from the constraints-the DBLP coauthor network-, e.g. *Lei Chen (HKUST)*. In our experiments, we constructed co-author networks from dumps of both websites with publications corresponding to *Lei Chen*, which is distinguished into five entities on DBLP. To obtain the ground truth G^*, we prune the instance graph of intractable publications for which there is no corresponding label in the constraint graph.

	G^* = (V, E, λ)		S = (L, N)	
Dataset	|V|	|E|/|V|	|L|	|N|/|L|
synthetic	[20,100,500]	[1, 2, 4]	|V|×[0.2, 0.5, 1]	[1, 2, 3]
sepsis	7382	0.86	14	3.57
restaurants	846	6.19	11	1.0
coauthors	54	1.02	688	3.33

Table 2: Datasets characteristics

6.1.5 Noise Injection. To obtain a noisy instance graph G from G^*, we must inject noise. Since the datasets have different sizes, the share of the noise needs to be relative to the graph size, in terms of labels and number of vertices, to yield meaningful comparisons. We define noise ratio as $\#violations/|E|$.

We introduce noise by randomly relabel vertices into an *invalid* label. Note that this only injects noise that is detected by the neighborhood constraints. Moreover, note that relabeling a vertex may introduce more than one violation. To cope with this, noise injection works on noise intervals. For example, in a graph with 30 vertices and 100 edges, a noise ratio of (0.27, 0.33) implies that among the 100 edges, between 27 and 33 are violations. This number alone gives little information on the number of vertices that were relabeled to achieve this noise ratio, however, the transformations can still be recovered by comparing G to G^*. We injected noise in the (0.27,0.33) noise ratio interval, selected as a middle ground between a reasonable amount of repairing to do and repairable data.

6.2 User simulation

We decided to model the behavior of the users by considering the two extremes of a user modeling spectrum and a user model in the middle. The objective is to evaluate how the different frameworks perform at the best and worst cases, and with a knowledgeable user in the middle. In our experiments, we simulate three different user behaviors that are representative of the entire spectrum.

`Random`: This user chooses an answer randomly with uniform probability. We opted for a random strategy in the selection of the answer as this is quite a common baseline with real users. Indeed, statistical data has indicated that when involving humans, up to 50% of users may provide random answers for various reasons, such as lack of comprehension or intentionally providing random responses to expedite task completion [31, 43].

`MaxFix`: This user exhaustively tries all possible answers, and selects the best in terms of resulting violations. This user understood how the constraints work and how to remove them efficiently, but without proper domain expertise or context comprehension - as it does not know the ground truth.

`Oracle`: This user has knowledge of the ground truth and knows the sequence of answers needed to reach it. He might choose answers that do not fix the graph right away, but work towards reaching the most correct result.

6.3 Metrics

We evaluate our approach in terms of *efficiency*, i.e. number of interaction, result *quality*, with $f1$-score, and question difficulty.

Mean Resolution Ratio The resolution ratio measures how efficient an answer is in repairing the graph. It consists of the number

[1]This dataset is courtesy of the authors of [38]

of violations fixed per question: $resolution\ ratio = k/|interactions|$ with k the number of violations in G. The higher the resolution, the more efficiently the graph was repaired. We consider 1 as the baseline value, as it means that on average each interaction fixes 1 violation. Its value can be below 1 if some relabeling adds violations. In addition, notice that the maximum value achievable by a framework depends on the structure of the underlying graph.

Mean Question difficulty: we compute the average question difficulty of a framework as defined in Def. 5.6. For the experiments, we use $\alpha = \beta = 0.5$ in all cases to have meaningful comparisons.

$f1$*-measure:* we compute the f1-score as defined in [38] with respect to ground truth. Namely, true positives (tp) are vertices correctly relabeled, false positives (fp) are both vertices relabeled in an incorrect label and relabeled vertices that were not meant to. Lastly, false negatives (fn) are injected noise that was not relabeled.

Note that runtimes are not reported, as the overall time to achieve a repair is largely dominated by the time a real user will take to answer any given question (at least seconds). Indeed, even for big graphs, the time to produce a new question (apply the previous answer, select a new question, and ask it) takes microseconds, for all the frameworks we evaluate, which is negligible.

6.4 Results and Analysis

In the following set of plots in Figure 5 and 6, we present the results for $f1$-score and resolution ratio, grouped by the framework (column) and user (row). A row-by-row comparison shows how a given framework performs with different users. A column-by-column comparison shows how frameworks compare given a fixed user. While Figure 5 showcases results for the variety of real-life datasets described above, Figure 6 benefits from the controlled environment with synthetic datasets to investigate the relation between the quality of the results and scalability.

First, we compare the results based on real-world datasets. When the bars are missing, it means that this particular experimental setup is *non terminating*: in a majority of cases, no repaired graph was produced. In practice, since we cannot know in advance if the experiment will terminate or not, we halt the experiment after a time-out given by $2\times|V|$ interactions, which should leave plenty of iterations for a functioning setup to terminate.

For instance, the case of having users `Random` repairing a big graph with frameworks that do not have the *Termination* property may never finish (cf. Section 5). On the other hand, the framework designed for this case, $\mathcal{F}_{term}$, terminates in all cases, even with bigger graphs. Interestingly, in this case also the `Oracle` user fails to terminate with the $\mathcal{F}_{bool}$ framework and the restaurant dataset. This is because this framework, by proposing random violation and repair pairs, has a low chance to show the oracle repair, resulting in the user rejecting all the options and getting stuck.

Further, note that even when the bars are present, some experimental setups might have failed. In these cases, the minority of failures are ignored (with failure rates ranging between 0.2 and 0.4 in precise cases, cf [2] for details). The presented results are thus subject to a survivor bias [20]: experiments terminated early are not accounted for in the presented averages.

For a given user, our experiments show repair quality depends not only on the framework but also on the dataset characteristics. In particular, it seems the *restaurants* dataset is easier to repair well. This might be explained by the high density of G and the very low density of S for this dataset: there are not many possibilities for each relabeling, so getting the right one is easier. Except for $\mathcal{F}_{bool}$, repair quality is almost perfect with `Oracle` users (averaged over dataset, $f1$-measure with standard deviation: $\mathcal{F}_{bool}$ 0.722±0.244, $\mathcal{F}_{greedy}$ 0.988±0.009, $\mathcal{F}_{permissive}$ 0.995±0.006, $\mathcal{F}_{term}$ 0.964±0.042). Notably, this is also the case with frameworks that are not Oracle Optimal (such as $\mathcal{F}_{greedy}$). This suggests that although it is not true in general, most contexts do not require oracle optimality to reach good repair quality. Also, both the users `MaxFix` and `Random` approach the `Oracle`'s behavior with $\mathcal{F}_{permissive}$ and F_{term} on some datasets (see the *restaurants* dataset in Figure 5). This demonstrate that our method is applicable in real world cases. In fact, even users not as knowledgeable as an oracle, that in real-world does not exist or are costly to engage, achieve good repair quality.

For resolution ratio, the dataset dependency is striking, in particular for the *restaurant* dataset again. Intuition is that along with the framework, graph shape have a stron influence on resolution ratio. Notice that $\mathcal{F}_{bool}$'s resolution ratio is very low - even sometimes below 1 - compared to other frameworks. This is because this QAR framework may propose any possible graph transformation for a given violation. With many of them being wrong and thus refused by the users, it results in no change. Interesting to notice that with $\mathcal{F}_{term}$, the `Random` user not only achieves the repair but also reaches a resolution ratio of at least 1 for all datasets. For `MaxFix` and `Oracle`, the resolution ratio is always above 1.

Next, Figure 6 shows the obtained results with our synthetic data depending on the input size $|V|$. The figure follows the same configuration. Recall that (i) both graph densities and the size of S also grow with $|V|$ and (ii) for each graph size, the experiments are run on a variety of input configurations.

Mostly of the previous comments apply to this scenario as well. Namely, user `Random` has a hard time terminating, except with $\mathcal{F}_{term}$. Important to notice is that the $f1$-score does not decrease with the number of vertex, but, on the other hand, it seems to increase. We believe the rationale for this phenomenon is that it is easier to reach an incorrect repair on small graphs. Like other datasets, user `Oracle` achieves an extremely high f1-score, stable across input sizes. In the case of the resolution ratio, it similarly does not decrease but increases with graph size. This can be because, by increasing the number of violations, it's more probable that a transformation fix many of them. The fact our performances do not degrade by increasing the graph size demonstrates that our approach promises to be applicable with bigger graphs.

As shown in Figure 7, question difficulty is very dependent on the dataset characteristics: the y-axis ranges differ by orders of magnitude. This does not hold for $\mathcal{F}_{bool}$ though, as expected from Proposition 5.13, since its question difficulty is constant. The results show that $\mathcal{F}_{permissive}$ is exploding the question complexity, and $\mathcal{F}_{term}$ always improving on $\mathcal{F}_{greedy}$: the more a framework allows for user decisions, the more difficult it is for them.

6.5 Lessons Learned

To confront QAR frameworks and different users in a simple metric, we use Pareto Efficiency [10] as a multi-criteria analysis tool. Specifically, for a fixed user and dataset, a QAR framework is Pareto Efficient if no others improve on all the metrics, i.e. $f1$-score, resolution ratio, and mean question difficulty.

Table 3 reports the results of Pareto Efficiency analysis for the mean result of each framework for all (user, dataset) pairs. A T

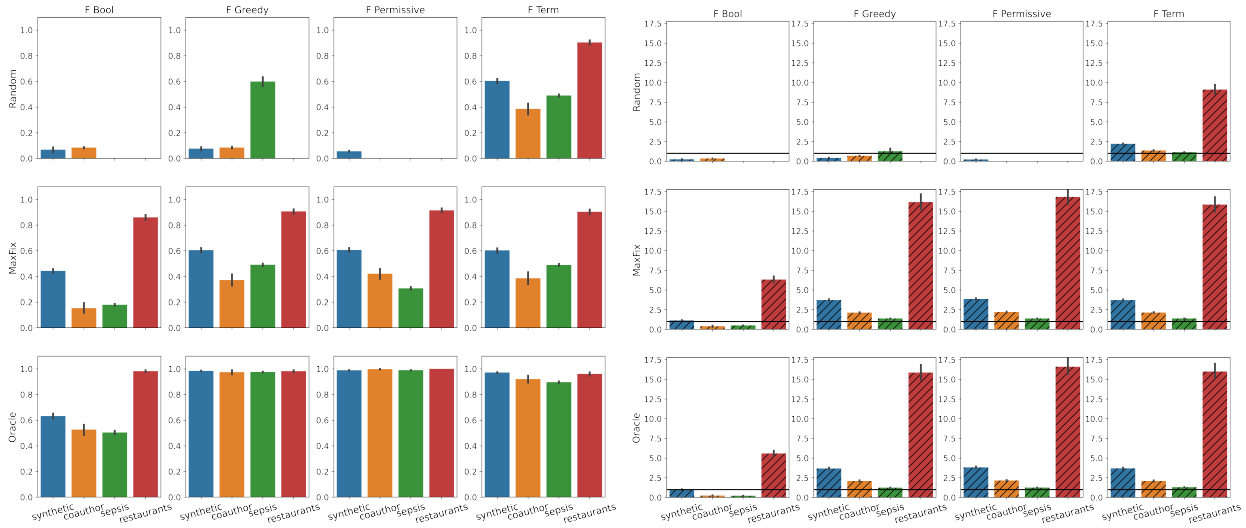


Figure 5: Experiment result for $f1$-score (left, full) and *resolution ratio* (right, dashed) on the different datasets. The table showcases behavior by user (row) and by framework (column), aggregated by dataset (x-axis). Absent bars signify non-termination of the experiment.

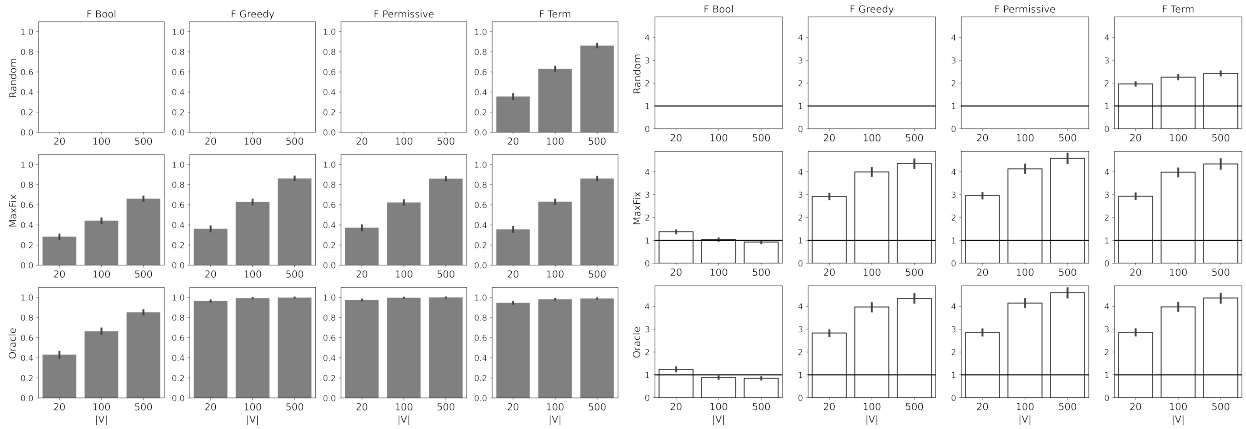


Figure 6: Experiment result for $f1$-score (left, full) and resolution ratio (right, empty) on the synthetic data. The table showcases behavior by user (row), by framework (column), and G size (x-axis). Absent bars signify non-termination of the experiment.

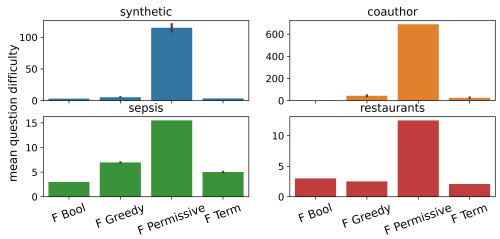


Figure 7: Mean Question Difficulty for different datasets and QAR frameworks. Note that y-axes have different scales.

in a cell means that for that specific user and dataset, the corresponding framework is Pareto efficient. For instance, considering `MaxFix` users and the dataset *coauthor*, the framework $\mathcal{F}_{bool}$ is Pareto efficient, meaning that it is not dominated by any other frameworks on all the criteria - most probably because $\mathcal{F}_{bool}$ has the lowest mean question difficulty. On the other hand, in the same case, $\mathcal{F}_{greedy}$ is not Pareto efficient, meaning that there is at least another framework that improves on all the criteria.

User	Dataset	$\mathcal{F}_{bool}$	$\mathcal{F}_{greedy}$	$\mathcal{F}_{permissive}$	$\mathcal{F}_{term}$
MaxFix	coauthor	T	F	T	T
	synthetic	T	T	T	T
	restaurants	F	T	T	T
	sepsis	T	T	F	T
Oracle	coauthor	T	T	T	T
	synthetic	T	T	T	T
	restaurants	F	T	T	T
	sepsis	T	T	T	T
Random	coauthor	T	F	NA	T
	synthetic	F	F	F	T
	restaurants	F	F	NA	T
	sepsis	T	F	F	T

Table 3: Truth value for Pareto Efficiency of mean performance for each QAR framework and (user,dataset) pair. N/A means the experiments did not terminate.

We can draw several conclusions from this table. First, considering the number of Pareto Efficient options, what is most evident is which options to not take - i.e., when a framework is not Pareto Efficient or worse when it never finishes. For example, using $\mathcal{F}_{greedy}$ or $\mathcal{F}_{permissive}$ with `Random` users or $\mathcal{F}_{bool}$ with

the *restaurant* datasets, and probably with datasets that share the same characteristic.

Further, $\mathcal{F}_{term}$ is always Pareto Efficient. Indeed, from previous analysis as well, this QAR framework is hitting an efficient middle ground by always terminating and having low question difficulty, while maintaining a good resolution ratio and $f1$-score when users are good. In particular, the $\mathcal{F}_{term}$ QAR framework achieves satisfactory results with `Random` user both in terms of $f1$-score and resolution ratio, in both the synthetic and real-world datasets, making it the perfect choice when the users have no knowledge of the domain nor of how graphs work.

Interestingly, $\mathcal{F}_{bool}$ is often Pareto Efficient due to its very low question difficulty. In practice, this means that in a context where it's crucial to minimize difficulty for the users, $\mathcal{F}_{bool}$ remains a good choice. However, consider also that our experiments show that $\mathcal{F}_{bool}$ does not allow the `Oracle` user to achieve good results.

The overall conclusion is that, while none of the QAR frameworks may be optimal, it is important to design and choose the proper framework according to the dataset and user needs.

7 CONCLUSION

In this paper, we introduced and formalized the problem of interactive graph repair under neighborhood constraints and studied its complexity. We defined a theoretical interaction model (QAR - Question, Answer, Repair) and explored different instantiations and their properties. We treated users as more than oracles by considering question difficulty and simulating various behaviors. To validate our approach, we evaluated the QAR frameworks on synthetic and real-world datasets with simulated users, demonstrating its effectiveness. Furthermore, we provided recommendations for selecting the best framework in different situations. Future research should focus on involving multiple real users to enable the repair of large graphs. This includes investigating methods to integrate different answers for the same violation, handling multiple fixes for a graph, and ensuring ease of use for general users [16, 21, 32]. Additionally, future work could automate the exploration of the solution space by developing optimization/heuristic or machine learning-based methods for selecting the most suitable framework given a dataset and set of constraints [9].

ACKNOWLEDGMENTS

This work was partially supported by the VeriGraph (ANR-21-CE48-0015) and Data4Health ANR projects. We thank Riccardo Tommasini for the insightful discussions on the analysis of the results.

REFERENCES

[1] [n. d.]. CiteSeer. https://citeseerx.ist.psu.edu/. accessed 30th Jan 2023.
[2] [n. d.]. Code and artifacts repository for Interactive Graph Repair. https://github.com/PaulJuillard/InteractiveGraphRepair.
[3] [n. d.]. DBLP. https://dblp.uni-trier.de/db/. accessed 30th Jan 2023.
[4] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2023. PG-Schema: Schemas for Property Graphs. *Proc. ACM Manag. Data* 1, 2, Article 198 (jun 2023), 25 pages. https://doi.org/10.1145/3589778
[5] Abdallah Arioua and Angela Bonifati. 2018. User-guided repairing of inconsistent knowledge bases. In *EDBT: Extending Database Technology*. OpenProceedings. org, 133–144.
[6] Akansha Bhardwaj, Jie Yang, and Philippe Cudré-Mauroux. 2022. Human-in-the-Loop Rule Discovery for Micropost Event Detection. *IEEE Transactions on Knowledge and Data Engineering* (2022).
[7] Shahzad Sarwar Bhatti, Xiaofeng Gao, and Guihai Chen. 2020. General framework, opportunities and challenges for crowdsourcing techniques: A Comprehensive survey. *Journal of Systems and Software* 167 (2020), 110611. https://doi.org/10.1016/j.jss.2020.110611
[8] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. Querying graphs. *Synthesis Lectures on Data Management* 10, 3 (2018), 1–184.
[9] Marco Brambilla, Stefano Ceri, Andrea Mauri, and Riccardo Volonterio. 2015. An explorative approach for crowdsourcing tasks design. In *WWW '15 Companion: Proceedings of the 24th International Conference on World Wide Web*. 1125–1130.
[10] Altannar Chinchuluun and Panos M Pardalos. 2007. A survey of recent developments in multiobjective optimization. *Annals of Operations Research* 154, 1 (2007), 29–50.
[11] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Information and Computation* 197, 1-2 (2005), 90–121.
[12] Qianhao Cong, Jing Tang, Yuming Huang, Lei Chen, and Yeow Meng Chee. 2022. Cost-Effective Algorithms for Average-Case Interactive Graph Search. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 1152–1165. https://doi.org/10.1109/ICDE53745.2022.00091
[13] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2013. Using the Crowd for Top-k and Group-by Queries. In *Proceedings of the 16th International Conference on Database Theory* (Genoa, Italy) *(ICDT '13)*. Association for Computing Machinery, New York, NY, USA, 225–236. https://doi.org/10.1145/2448496.2448524
[14] Wenfei Fan and Ping Lu. 2019. Dependencies for graphs. *ACM Transactions on Database Systems (TODS)* 44, 2 (2019), 1–40.
[15] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 61–72. https://doi.org/10.1145/1989323.1989331
[16] Ujwal Gadiraju, Gianluca Demartini, Ricardo Kawase, and Stefan Dietze. 2019. Crowd anatomy beyond the good and bad: Behavioral traces for crowd worker modeling and pre-selection. *Computer Supported Cooperative Work (CSCW)* 28 (2019), 815–841.
[17] Frank Harary and Robert Z Norman. 1960. Some properties of line digraphs. *Rendiconti del circolo matematico di palermo* 9 (1960), 161–168.
[18] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J Miller. 2009. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1282–1293.
[19] Oana Inel, Khalid Khamkham, Tatiana Cristea, Anca Dumitrache, Arne Rutjes, Jelle van der Ploeg, Lukasz Romaszko, Lora Aroyo, and Robert-Jan Sips. 2014. Crowdtruth: Machine-human computation framework for harnessing disagreement in gathering annotated data. In *International semantic web conference*. Springer, 486–504.
[20] John PA Ioannidis. 2005. Why most published research findings are false. *PLoS medicine* 2, 8 (2005), e124.
[21] Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. 2010. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*. 64–67.
[22] David R. Karger, Sewoong Oh, and Devavrat Shah. 2013. Efficient Crowdsourcing for Multi-Class Labeling. *SIGMETRICS Perform. Eval. Rev.* 41, 1 (jun 2013), 81–92. https://doi.org/10.1145/2494232.2465761
[23] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. 2011. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 43–52.
[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.
[25] Peng Lin, Qi Song, Yinghui Wu, and Jiaxing Pi. 2020. Repairing entities using star constraints in multirelational graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 229–240.
[26] Michael Loster, Davide Mottin, Paolo Papotti, Jan Ehmüller, Benjamin Feldmann, and Felix Naumann. 2021. Few-Shot Knowledge Validation Using Rules. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3314–3324. https://doi.org/10.1145/3442381.3450040
[27] Felix Mannhardt and Daan Blinde. 2017. Analyzing the Trajectories of Patients with Sepsis using Process Mining. *RADAR+ EMISA@ CAiSE* 1859 (2017), 72–80.
[28] Adam Marcus, Eugene Wu, David R Karger, Samuel Madden, and Robert C Miller. 2011. Crowdsourced databases: Query processing with people. Cidr.
[29] A Mauri and A Bozzon. 2021. Towards a human in the loop approach to preserve privacy in images. In *CEUR Workshop Proceedings*, Vol. 2947.
[30] Jasper Oosterman, Archana Nottamkandath, Chris Dijkshoorn, Alessandro Bozzon, Geert-Jan Houben, and Lora Aroyo. 2014. Crowdsourcing knowledge-intensive tasks in cultural heritage. In *Proceedings of the 2014 ACM conference on Web science*. 267–268.
[31] Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, and Karl Aberer. 2013. An evaluation of aggregation techniques in crowdsourcing. In *Web Information Systems Engineering–WISE 2013: 14th International Conference, Nanjing, China, October 13-15, 2013, Proceedings, Part II 14*. Springer, 1–15.

[32] Bahareh Rahmanian and Joseph G. Davis. 2014. User Interface Design for Crowdsourcing Systems. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces* (Como, Italy) *(AVI '14)*. Association for Computing Machinery, New York, NY, USA, 405–408. https://doi.org/10.1145/2598153.2602248

[33] Jorge Ramírez, Burcu Sayin, Marcos Baez, Fabio Casati, Luca Cernuzzi, Boualem Benatallah, and Gianluca Demartini. 2021. On the State of Reporting in Crowdsourcing Experiments and a Checklist to Aid Current Practices. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 387 (oct 2021), 34 pages. https://doi.org/10.1145/3479531

[34] Fereidoon Sadri and Jeffrey D. Ullman. 1980. The Interaction between Functional Dependencies and Template Dependencies. In *Proceedings of the 1980 ACM SIGMOD International Conference on Management of Data* (Santa Monica, California) *(SIGMOD '80)*. Association for Computing Machinery, New York, NY, USA, 45–51. https://doi.org/10.1145/582250.582258

[35] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2020. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.* 29, 2-3 (2020), 595–618. https://doi.org/10.1007/s00778-019-00548-x

[36] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei R. Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. 2021. The Future is Big Graphs: A Community View on Graph Processing Systems. *Commun. ACM* 64, 9 (aug 2021), 62–71. https://doi.org/10.1145/3434642

[37] IP Samiotis, S Qiu, A Mauri, CCS Liem, C Lofi, and A Bozzon. 2020. Microtask crowdsourcing for music score Transcriptions: an experiment with error detection. In *21st International Society for Music Information Retrieval Conference*.

[38] Shaoxu Song, Boge Liu, Hong Cheng, Jeffrey Xu Yu, and Lei Chen. 2017. Graph repairing under neighborhood constraints. *The VLDB Journal* 26, 5 (2017), 611–635.

[39] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. 2012. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.* 64, 2-3 (2012), 209–246. https://doi.org/10.1007/s10472-012-9288-8

[40] Sheila Tejada. [n. d.]. Restaurant - a collection of restaurant records from the Fodor's and Zagat's restaurant guides that contains duplicate. https://www.cs.utexas.edu/users/ml/riddle/data.html. accessed 30th Jan 2023.

[41] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. 2012. Max Algorithms in Crowdsourcing Environments. In *Proceedings of the 21st International Conference on World Wide Web* (Lyon, France) *(WWW '12)*. Association for Computing Machinery, New York, NY, USA, 989–998. https://doi.org/10.1145/2187836.2187969

[42] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. 2008. recaptcha: Human-based character recognition via web security measures. *Science* 321, 5895 (2008), 1465–1468.

[43] Jeroen Vuurens, Arjen P de Vries, and Carsten Eickhoff. 2011. How much spam can you take? an analysis of crowdsourcing results to increase accuracy. In *Proc. ACM SIGIR Workshop on Crowdsourcing for Information Retrieval (CIR'11)*. 21–26.

[44] Kerri Wazny. 2018. Applications of crowdsourcing in health: an overview. *Journal of global health* 8, 1 (2018).

[45] Jef Wijsen. 2005. Database repairing using updates. *ACM Transactions on Database Systems (TODS)* 30, 3 (2005), 722–768.

[46] Jie Yang, Judith Redi, Gianluca Demartini, and Alessandro Bozzon. 2016. Modeling task complexity in crowdsourcing. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, Vol. 4. 249–258.