

Budgeted Interactive Property Graph Repair with GNN

AMEDEO PACHERA, Lyon1 University, CNRS Liris, France

ANGELA BONIFATI, Lyon1 University, CNRS Liris & IUF, France

LAKS V.S. LAKSHMANAN, The University of British Columbia, Canada

ANDREA MAURI, Lyon1 University, CNRS Liris, France

Property graphs are powerful data models, yet they often suffer from inconsistencies caused by violations of integrity constraints. Existing repair strategies are either fully automated—struggling with ambiguous cases that require domain expertise—or human-driven, which lack scalability and overlook budget and user heterogeneity. In this work, we propose a novel approach to property graph repair that integrates automatic inference with user expertise. We introduce a Graph Neural Network (GNN)-based difficulty estimator that jointly captures topological and semantic aspects of violations, enabling the system to distinguish between cases suitable for automatic repair and those requiring human input. Building on this, we design a budgeted, difficulty- and user-aware assignment algorithm that allocates violations to users based on their expertise and capacity while maximizing expected repair quality under resource constraints. We prove the NP-hardness of the problem and provide a Lagrangian relaxation. Extensive experiments demonstrate that our approach significantly improves repair accuracy compared to baselines and substantially reduces the number of involved users compared to both other interactive approaches.

CCS Concepts: • **Information systems** → **Graph-based database models**; **Data cleaning**.

Additional Key Words and Phrases: Property Graphs, Graph Data Cleaning, Graph Neural Networks, Interactive Data Cleaning

ACM Reference Format:

Amedeo Pachera, Angela Bonifati, Laks V.S. Lakshmanan, and Andrea Mauri. 2026. Budgeted Interactive Property Graph Repair with GNN. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 262 (September 2026), 26 pages. <https://doi.org/10.1145/3837100>

1 INTRODUCTION

Graphs are widely recognized as a fundamental data model across numerous fields, including finance, cybersecurity, social networks [2], and bioinformatics [43], among others [60]. Among the various graph models, property graphs [4, 12] stand out due to their ability to associate key-value properties with both multi-labeled nodes and edges, providing a highly expressive framework for representing complex relationships.

However, the integrity of graph-structured data is often compromised by inconsistencies and inaccuracies, often manifest as violations of constraints on graph databases. A well-known class of graph constraints are denial constraints [13]. These violations typically appear as erroneous property values or incorrect labels assigned to nodes and edges, or incorrect presence of edges. The widespread occurrence of such errors has been highlighted in several studies [18, 50, 67], which also point out that data cleaning is one of the most critical and labor-intensive tasks [59].

Authors' Contact Information: Amedeo Pachera, Lyon1 University, CNRS Liris, Lyon, France, amedeo.pachera@univ-lyon1.fr; Angela Bonifati, Lyon1 University, CNRS Liris & IUF, Lyon, France, angela.bonifati@univ-lyon1.fr; Laks V.S. Lakshmanan, The University of British Columbia, Vancouver, Canada, laks@cs.ubc.ca; Andrea Mauri, Lyon1 University, CNRS Liris, Lyon, France, andrea.mauri@univ-lyon1.fr.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/9-ART262

<https://doi.org/10.1145/3837100>

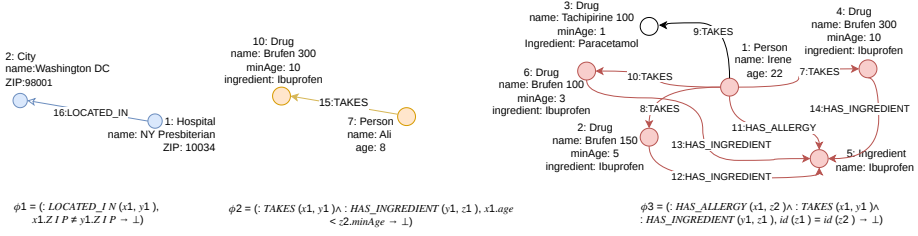


Fig. 1. Three examples of violations caused by denial constraints.

Previous approaches to graph repair have generally followed two orthogonal types of approaches: (i) fully automated methods that leverage constraints and optimization techniques [26, 28, 40, 61], which assume that rules are known in advance and are sufficient to guide repair decisions; and (ii) human-in-the-loop approaches [9, 36, 42, 51, 52], which engage humans in repair validation or suggestion but often oversimplify task assignment, rely on a single oracle, or do not consider budget constraints. Both face key limitations: full automation struggles with ambiguous cases requiring domain expertise [52] to make the right choice, while human-driven solutions lack scalability and often ignore practical concerns such as cost [37], diversity in user expertise, and cognitive load [32].

Fig. 1 shows a property graph for a healthcare scenario with three denial constraints causing different types of violations.

- Constraint ϕ_1 states that a hospital and the city it is located in must have the same ZIP code, producing the violation depicted in blue. This is an instance of a violation that can be solved by an automated method, since the information to fix it can be found in the original graph or learned by inference [57].
- Constraint ϕ_2 states that a person cannot take a drug whose ingredient is age-restricted, while being younger than the minimum age required, resulting in the orange violation. Solving this violation requires domain knowledge. No one except Ali's physician can know whether the source of inconsistency lies in the wrong age or in him taking the Ibuprofen.
- Constraint ϕ_3 states that if a person has an allergy to an ingredient, they should not take a drug that contains that ingredient, producing the red violations. Here, a single constraint causes three violations with two overlapping nodes (Person and Ingredient). In this case, not only is domain knowledge required (e.g., knowing Irene's allergies), but the violations include a complex topological structure with many nodes and edges.

Based on the above observations, one can notice that there are two fundamental problems in graph repair. The first is to identify the *difficulty* of the repair and to factor it into the assignment, second, assigning the repair tasks to either automatic methods based on inference or to users with appropriate expertise. Crowdsourcing approaches assume the difficulty to be a function of the semantics of the elements composing a task [66], while the recent interactive graph repair works assume that the difficulty is a function of the graph topology [52]. By contrast, we argue that there is a need for a *holistic* assessment of the difficulty, considering both topological and semantic aspects of the violation. Previous work in crowdsourcing, while considering human skills and an overall budget, does not consider interconnected tasks like violations in a graph in making assignments. Furthermore, to our knowledge, it does not consider human limitations such as time and fatigue [11, 31, 54, 55].

To address this gap, we propose a novel budget-aware assignment strategy that combines difficulty estimation with user limitations. A key requirement for such an approach is the ability to reason

simultaneously about the structural context of a violation (e.g., how it fits into the surrounding topology) and the semantic properties in nodes and edges (e.g., attribute values and types).

Graph Neural Networks (GNNs) naturally meet these requirements, as they are designed to integrate structural and attribute-based information in a unified representation [1], making them particularly suitable for both predicting violation difficulty and for automating less complex repairs (e.g., edges with low existence probabilities). More precisely, we leverage GNNs, which blend topological dependencies and node/edge feature representations, to predict edge existence probabilities and to extract node embeddings via latent feature representations. These predictions are used to compute a *difficulty score* for each violation, reflecting the model's confidence in resolving the violation automatically. These scores are integrated into a user model, which estimates the likelihood that a user, given their knowledge, will successfully repair a violation.

The same model can automatically resolve simple violations by deleting the corresponding edge. This creates a trade-off between users, who provide higher-quality but costly repairs, and automated methods, which are cost-free but may produce lower-quality results. To manage this trade-off, we propose a tunable assignment strategy that allows to balance violations between users and automated repair based on quality and cost preferences.

Our approach jointly considers violation difficulty and user expertise and capacity (i.e., the amount of work they can perform), with the goals of *maximizing* the expected repair quality, while staying within a specified *budget*, making sure people are not overloaded with repairs.

In sum, we present a novel framework for budgeted interactive repair of property graphs that tightly integrates learning, automation, and optimization. Our key contributions are:

- We introduce a new notion of violation difficulty tailored to graph repair, based on the model's confidence in automatic resolution of violation. Unlike prior work that models difficulty based solely on topology or semantics, we jointly capture both via a GNN and further tailor the representation to the repair task via a constraint-driven feature filtering mechanism (§ 4.3).
- We propose a unified repair pipeline in which the GNN is used both to estimate difficulty and to automatically resolve easy violations via confidence-based pruning. We formalize this process and show that automated repairs commute with human repairs, ensuring that pruning does not affect the correctness or quality of the final repair outcome.
- We formulate the violation assignment problem as a budget-constrained covering problem over a graph of dependent violations, where tasks are inherently interconnected and prove that it is NP-Hard (§ 4.1, § 4.2). To address this, we design a tailored Lagrangian-relaxation-based algorithm that assigns connected subgraphs to users through a seed-based construction, explicitly accounting for structural dependencies, user expertise, and cognitive capacity (§ 4.5). Building on the proposed formulation, we derive a closed-form dual upper bound on the optimal objective, enabling us to quantify the quality of the obtained assignments despite the problem's NP-hardness.
- We validate our approach with an extensive set of experiments evaluating 1) the validity of our difficulty measure and 2) the overall effectiveness of our approach, considering both the quality of the repairs and cost, against interactive and automated graph repair methods and other task assignment strategies (§ 5).

§ 2 discusses related work. § 3 presents the background needed for the rest of the paper. Finally § 6 concludes the paper, discussing limitations and future work.

2 Related Work

Graph dependencies and repairs. In this work, we focus on Graph Denial Constraints (GDCs), one of the most expressive classes of constraints for graph databases [12, 27]. Denial Constraints specify patterns that must not exist in the data, allowing flexible and powerful integrity enforcement. GDCs enable the specification of conditions on edges and (possibly multi-labeled) vertices in Property Graphs through subgraph pattern matching. Several approaches exist for repairing graphs violating integrity constraints, including deletion [16] – remove conflicting data, and the update [64] – modify data to restore consistency. We consider a richer setting that permits both deletes and updates as repair actions. Repair strategies have been studied for different dependency classes [12, 27, 40, 61]. The problem of finding minimal repairs in this context is NP-complete [36], but algorithms achieving constant-factor approximation exist for graphs with bounded maximum degree. A former study [52] proposed an interactive framework for collaboratively repairing a property graph under GDCs, leveraging a dependency graph in the form of a Hypervertex Labeled Property Graph for assigning violations to different users. However, this work does not account for diversity in the user pool w.r.t. skill levels, nor variations in the difficulty levels of repairing different violations. There is a need for a systematic method that assigns violations to users based on their difficulty. In this paper, we focus on a targeted assignment strategy, first leveraging GNNs to estimate the repair difficulty of violations, and then pairing users with violations according to the user’s skill levels and the violations’ difficulties. To our knowledge, such a targeted assignment has not been explored in prior work.

Modeling task difficulty and assignment. A significant body of work has explored task assignment optimization and task difficulty modeling in crowdsourcing [11]. These approaches typically assume tasks to be independent units, and aim to assign them to workers with suitable skills under constraints such as cost, quality, and group compatibility [54]. To solve the assignment problem, they introduce probabilistic or integer programming models to optimally allocate workers based on skill vectors and cost budgets [11, 31, 54]. Other studies rely on scalable heuristics, like greedy algorithms, to iteratively select workers who provide the highest utility per unit cost [55]. Despite their effectiveness in micro-task environments, these approaches fall short in scenarios such as collaborative repairs of property graphs. In our setting, tasks are not independent: each violation is part of a larger, interconnected structure, and repairing one can have cascading effects, resolving or introducing other violations. Existing models do not capture this violation dependency or the semantic interrelation between graph entities, which is central to our task and contributes to the challenge. On the task difficulty modeling side, prior work has focused on estimating task complexity using statistical models that separate worker reliability from inherent task hardness [66]. However, these methods require an initial seed to train the statistical models. Other works consider the cognitive load (e.g., the number of elements involved in the task) as the difficulty measure [65]. However, modeling the difficulty of repairing a graph requires evaluating the semantics of nodes and edges, the violations structure, and the potential side effects of a repair operation. Unlike previous work, our tasks are interdependent and demand reasoning over the neighborhood of given inconsistencies, rendering prior task assignment and difficulty modeling approaches unsuitable for our setting.

3 Preliminaries

We review the notions of property graphs, denial constraints, violations, and repair strategies.

Definition 3.1 (Property Graph). [6] A property graph is a directed, labeled, and attributed multi-graph $G = (N, E, \eta, \lambda, \nu)$, where both vertices N and edges E are carry labels and key-value

attributes; $\eta : E \rightarrow N \times N$ maps each edge to an ordered pair of vertices; $\lambda : N \cup E \rightarrow \mathcal{P}(\mathcal{L})$ assigns each node/edge a set of labels; and $\nu : (N \cup E) \times \mathcal{K} \rightarrow \mathcal{N}$ assigns key-value properties.

Fig. 1 shows an example of a property graph for a healthcare scenario. Graph constraints and dependencies can fulfill the requirement for enforcing a restricted schema, addressing inconsistencies in a property graph to accomplish data cleaning tasks. In this work, we focus on Graph Denial Constraints (GDCs), which correspond to a highly expressive class of integrity constraints. Many commonly studied constraints in data management, including key constraints, functional dependencies, entity dependencies, and neighborhood constraints, can be naturally expressed as GDCs [20, 22, 25].

GDCs are defined over *graph patterns*, which are directed labeled graphs whose nodes or edges are associated with variables. Graph patterns are matched against a property graph to identify all subgraphs that satisfy their structural and labeling conditions [13, 29]. Pattern matching is typically defined via graph homomorphisms [29], although subgraph isomorphism is also considered in the literature. Our framework is independent of the specific matching semantics and is readily applicable to either matching. For instance, the pattern $Q_1[x_1, y_1] = \text{Hospital}(x_1) \wedge \text{City}(y_1) \wedge \text{located_in}(x_1, y_1)$ selects pairs of hospitals and cities connected by a `located_in` edge. In this pattern, x_1 and y_1 are two node variables and a corresponding match is highlighted in blue in Fig. 1(a).

Definition 3.2 (Graph Denial Constraints [13]). A GDC ϕ is a pair $(Q[\bar{x}], X \rightarrow \perp)$ where $Q[\bar{x}]$ is a graph pattern, $\bar{x}$ is a set of vertex and/or edge variables, X is a set of non-empty θ -literals over $\bar{x}$ and $\perp$ is the empty literal. A θ -literal over $\bar{x}$ has the form either $x.A \theta c$ or $x.A \theta y.B$, where $x, y \in \bar{x}$, $A, B \in \mathcal{K}$ denote property keys, c is a constant, and θ is one of the built-in predicates $=, \neq, >, <, \geq, \leq$. A literal can also be of the form $\text{id}(x) = \text{id}(y)$, where $\text{id}(\cdot)$ denotes vertex or edge identity.

We define a constraint violation as follows:

Definition 3.3 (Violation). For a graph G and a denial constraint $\phi = (Q[\bar{x}], X \rightarrow \perp)$, a violation is a subgraph $v \subseteq G$ such that: (i) v is a match of Q via isomorphism h ; (ii) for each literal $x.A \theta c$ (resp. $x.A \theta y.B$) in X , the corresponding instantiation $h(x).A \theta c$ (resp. $h(x).A \theta h(y).B$) is true.

Notice that the literals must all be satisfied in a violation. A graph G *satisfies* a denial constraint ϕ if it contains no violations of ϕ . G *violates* a set of GDCs Φ if G violates some GDC $\phi \in \Phi$. Otherwise, G *satisfies* Φ .

For instance, the GDC $\phi_1 = (Q_1[x_1, y_1], x_1.\text{ZIP} \neq y_1.\text{ZIP} \rightarrow \perp)$ denies a pattern where a hospital is located in a city with a different zip code than the hospital. This constraint produces the violation depicted in blue in Fig. 1(a). Prior to data repair, violations are detected automatically and efficiently by evaluating the constraints over the graph. Upon new data updates, constraint check can be redone and any detected violations fed back into the pipeline.

Repairing a graph containing violations of given GDCs involves applying a sequence of graph transformations such that the modified graph satisfies the constraints. A graph transformation is an operation that modifies the structure or properties of the graph, resolving constraint violations. As in previous work [44, 52, 57], the candidate transformations are edge deletions and property updates. E.g., consider the GDC ϕ_2 in Fig. 1(b). If we are sure of Ali's age, say by asking their doctor, the only acceptable repair is to delete the edge `TAKES`.

Definition 3.4 (Repair). Given a property graph G that violates a set of GDCs Φ , a repair R is a sequence of graph transformations such that applying R to G yields a graph G' that satisfies Φ .

In this work, we assume that each repair given by users is *minimal*, that is, it includes only the necessary transformations to repair the violation. E.g., if repairing a violation involves updating a specific node property A , no updates on other properties are included in the repair. Given a graph G and a set of constraints Φ , a repair R is *safe* if applying R to G does not introduce new violations of Φ . Safe repairs ensure that the number of violations monotonically decreases during the repair process [36, 52], which guarantees termination and avoids oscillations caused by conflicting or cascading updates. We check safety using a conservative static procedure. First, if R consists only of edge deletions, then R is safe by construction, since edge deletion cannot create a new violation of a GDC [52]. Second, if R contains property updates, we inspect only the constraints whose literals mention at least one updated property. For each such literal, if it is true before the property update or false after it, then the repair cannot introduce a new violation and is accepted as safe. Otherwise, the repair is potentially unsafe and we locally re-evaluate the corresponding constraint queries on the affected neighborhood, before applying it.

Finally, violations may overlap if they share at least one node. To consider this dependency, we leverage the *graph repair dependency graph* (GRDG), a hypergraph that overlays the property graph, where hypernodes are violations and edges between them represent the dependencies. Formally:

Definition 3.5 (Graph Repair Dependency Graph [52]). Given a property graph $G = (N, E, \eta, \lambda, \nu)$ and a set of GDCs Φ , *Graph Repair Dependency Graph* is a pair $\mathcal{G} = (V, F)$ where V is the set of hypernodes corresponding to violations of Φ and F is a set of undirected edges containing an edge (v_i, v_j) whenever violations v_i and v_j overlap, i.e., $v_i \cap v_j \neq \emptyset$.

Fig. 2 shows an example of GRDG with two violations. The hypernodes 20 and 21 overlay two inconsistent subgraphs. The two violations overlap on node 7 and the edge between hypernodes 20 and 21 captures the dependency between the two violations.

Two repairs independently applied to overlapping violations may produce a conflict on the overlapped portion of the graph if they both update the value of the *same property of a node*. This can be reduced to two specific cases. If the two repairs are the best possible ones (i.e., they correspond to the ground-truth), the two repairs perform the same update, thus producing no conflict, because for each graph G , there exists only one ground-truth G^* .

On the other hand, if one of the two repairs does not correspond to the ground-truth, there could be a conflict (e.g., two repairs write different values on the same property). Conflicts can be handled in several ways. Pachera et al. [52] prevent conflicts by locking dependent violations once one is assigned, while crowdsourcing approaches resolve them via aggregation functions [68]. In § 4.1 we elaborate on our strategy for resolving this issue.

4 Graph Repair Framework

We present our property graph repair framework, where a GNN-based module first estimates the difficulty of repairing each violation in a property graph. Second, a task assignment module assigns portions of the graph to users, leveraging the predicted difficulty scores, leaving easy violations for the GNN to repair first, following which users apply repairs tapping into their expertise in handling ambiguity, domain knowledge, and complex interdependencies between violations.

4.1 User Model

Different users vary significantly in their expertise, cognitive ability, and access to resources, which in turn affects their repair quality.

Our user model is motivated by two considerations grounded in prior work. (1) Tasks such as identification and resolution of graph violations impose a cognitive burden on users, which depends on both the structural complexity of the task and the user's domain expertise, as shown by prior

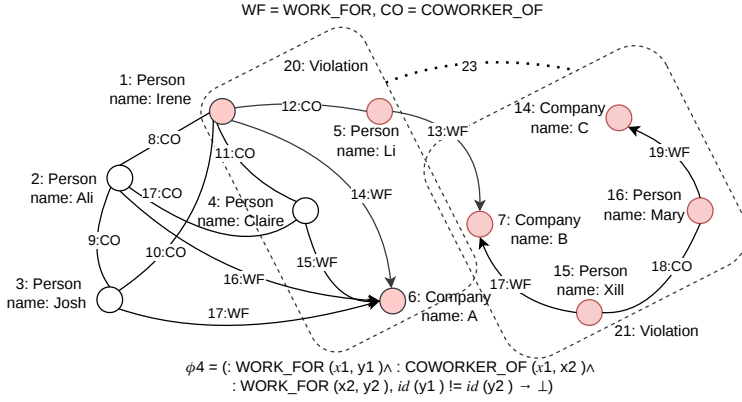


Fig. 2. Example of a GRDG with two violations.

work in cognitive psychology [23] and graph data processing and visualization [24]. They show structural properties of data directly affect cognitive load and user performance. (2) Involving users incurs costs, be it monetary payments or expert's time or an organization's limited workforce. While standard models consider only skill and cost, graph repair tasks impose varying cognitive demands depending on both the size and structure of the violations. Hence, we explicitly model cognitive capacity (CC) as the maximum cumulative difficulty a user can handle, capturing workload limitations, not accounted for in prior task assignment models. We adopt a compact but expressive representation of users as a tuple (K, CC, c) , where $K \in [0, 1]$ denotes skill level, $CC \in \mathbb{R}^+$ captures cognitive capacity, and $c \in \mathbb{R}^+$ models cost. This formulation provides a flexible abstraction that reflects real-world dynamics: **Skill Level** (K): a higher skill level increases the likelihood of a user choosing a high-quality repair for a violation. E.g., medical experts can identify clinically meaningful resolutions more accurately than non-experts. W.l.o.g., we model the skill level as a scalar (e.g., assuming that a property graph models a single domain). The model easily extends to multidimensional skill level vectors, where each component of the vector models the skill level on a specific aspect of the domain. This requires a more expensive data collection in order to train a higher dimensional model. Our approach remains largely unchanged with higher dimensions.

Cognitive capacity (CC): This captures the maximum workload a user can sustain before performance degrades. Prior work models this effort via cognitive load functions based on graph structure [32], but such approaches ignore graph size: a large sparse graph may be less demanding than a small, densely connected one. We therefore adopt an operational notion of cognitive capacity, defined as the maximum cumulative difficulty a user can handle across assigned violations. As task number or difficulty increases, performance may deteriorate due to overload, even for skilled users, making CC a critical constraint in our optimization. We assume that CC is user-provided as in platforms such as Mechanical Turk or Prolific.

Cost (c): Users come with different costs: e.g., crowdsourced workers may be cheaper but less skilled, than domain experts.

With this model, a domain expert (e.g., a medical doctor) may be modeled as $(K = 0.9, CC = 4.0, c = 5.0)$, while a data scientist may be modeled as $(K = 0.6, CC = 8.0, c = 2.0)$, indicating high domain skills and high cost but limited capacity to handle high cognitive load tasks for the domain expert, compared to the data scientist.

The parameters (K, CC, c) can be obtained following standard practices in crowdsourcing systems. The skill level K can be estimated either through questionnaires or by evaluating user performance on repair tasks having ground-truth answers. In some settings, user profiles may be updated online using behavioral metrics and prior performance [55]. The cost c can be either declared by the user or pre-defined by the platform, as commonly done in crowdsourcing settings. Finally, the cognitive capacity CC reflects the maximum workload a user is willing or able to sustain, and can be specified by the user (e.g., indicating how many tasks they are willing to complete), as in platforms such as Prolific [21].

After collecting users' data, we can train a user model to determine which transformation a user would apply to a given violation. Formally, given a user model (K, CC, c) , the probability of selecting a graph transformation T_i to repair a violation v is modeled as: $P_{T_i} = P(T_i \mid D_v, K)$, where D_v is the intrinsic difficulty of the violation, and K is the user's skill level. To train the model, it is possible to leverage known information, users' past experiences, or to collect an initial seed with a preliminary experiment (see § 5.1.2). Finally, we define:

Definition 4.1 (Expected Quality). Given a violation v , a set of candidate transformations $\{T_i\}$ with outcome qualities $\{Q_i\}$, user skill level K , and user-specific selection probabilities $\{P_{T_i}\}$, the expected quality EQ of the user's interaction is defined as:

$$EQ(D_v, K) = \sum_i P_{T_i} \cdot Q_i. \quad (1)$$

This function is component-wise monotonic: for a fixed difficulty D_v , the higher the user skill level, the higher the expected quality; for a fixed skill level K , the higher the difficulty, the lower the expected quality. With this notion of expected quality, we can estimate the performance of users on all the violations. Moreover, we can use the user model to solve conflicts that may manifest when multiple users repair overlapping violations, by applying only the repair done by the user with the highest skill level.

We note that, following common practice in the literature [11, 66], we adopt a static setting in which user capacities are fixed during assignment. Modeling temporal effects such as capacity recovery or user fatigue dynamics is an interesting direction for future work.

4.2 Problem Formulation

We formulate the problem of assigning dependent violations to users as a constrained graph covering problem. Consider a GRDG graph $\mathcal{G} = (V, F)$, where each node $v \in V$ represents a violation with an associated difficulty score $D_v \in \mathbb{R}_{\geq 0}$, and each edge $(v_i, v_j) \in F$ indicates a dependency between violations, i.e., they are preferably handled together. The goal is to cover the graph using a subset of users drawn from a pool $U = \{u_1, u_2, \dots, u_N\}$. Each user u_i has a *cognitive capacity* $CC_i \in \mathbb{R}_{\geq 0}$, representing the total difficulty of violations they can handle, a skill level $K_i \in \mathbb{R}_{\geq 0}$, and a cost $c_i \in \mathbb{R}_{\geq 0}$. A user is assigned a group of violations, a connected subset of GRDG, whose total difficulty does not exceed their capacity.¹ To prevent conflicts, we assign only disjoint violation subgraphs to different users. As a result, repairs operate on non-overlapping portions of the graph, ensuring that concurrent updates do not interfere with each other. Combined with the safety property (see § 3), this guarantees that repairs cannot invalidate or undo each other. Moreover, since all repairs are required to be safe, each applied transformation strictly reduces the number of violations without introducing new ones. Therefore, the repair process is guaranteed to terminate after we assigned all the violations. Finally, we have a budget $B \in \mathbb{R}_{\geq 0}$. The objective is to assign violations to a subset of users, maximizing the expected quality of the repairs while

¹We assume that we have sufficiently many users.

respecting the budget constraint and ensuring that each user is assigned a connected subgraph with total difficulty not exceeding their cognitive capacity.

Definition 4.2 (Budget-Constrained GRDG Covering Problem). Given a GRDG graph $\mathcal{G} = (V, F)$ with node difficulties D_v and a set of users $U = \{u_1, \dots, u_N\}$, each user u_i having a skill value K_i , cognitive capacity CC_i , and cost c_i , and a total budget $B \in \mathbb{R}_{\geq 0}$, find a subset of users $U^* \subseteq U$ and an assignment of disjoint subgraphs $\{P_i \subseteq V \mid u_i \in U^*\}$ such that:

$$U^* = \arg \max_{U' \subseteq U} \sum_{u_i \in U'} \sum_{v \in P_i} EQ(D_v, K_i) \quad (2)$$

subject to: (1) Each P_i is connected, (2) $\sum_{v \in P_i} D_v \leq CC_i \quad \forall u_i \in U'$, (3) $P_i \cap P_j = \emptyset \quad \forall u_i \neq u_j$, (4) $\bigcup_{u_i \in U'} P_i = V$, (5) $\sum_{u_i \in U'} c_i \leq B$. where $EQ(D_v, K_i)$ is a component-wise monotonic function defined in Section 4.1, Equation (1), estimating the expected quality of a user u_i with skill level K_i on a violation with difficulty D_v .

The above formulation captures the key aspects of the violation assignment task. Constraint (1) ensures that each user is assigned a connected subgraph of violations, preserving local context and facilitating coherent resolution. Constraint (2) guarantees that the total difficulty of assigned violations does not exceed the user's cognitive capacity, reflecting individual workload limits. Constraint (3) enforces disjointness in the assignment, ensuring each violation is handled by at most one user. Constraint (4) ensures that the assignment covers all necessary violations. Finally, constraint (5) is the budget constraint.

THEOREM 4.3. *The Budget-Constrained GRDG Covering Problem (bud-GCP) is NP-hard.*

PROOF SKETCH. By reduction from the Multiple Knapsack Problem (MKP). Create a violation per item (with difficulty = item weight) and a user per knapsack (with capacity that of the knapsack and unit), plus a dummy user; set the budget to one more than the number of knapsacks and make the graph complete. We can show that the MKP instance admits a packing of profit $\geq P$ iff the bud-GCP instance admits a solution of quality $\geq P$. The full construction and correctness proof are given in the additional material [8]. $\square$

4.3 GNN Difficulty Annotation

To estimate the difficulty of repairing a violation, we propose to leverage a GNN trained on the property graph to predict the likelihood of individual graph components (nodes and edges) conforming to the learned structural and semantic patterns of the graph. Rather than designing a handcrafted heuristic, we follow a standard principle in graph anomaly detection: subgraphs that have low likelihood under the model's learned joint distribution are anomalous and therefore easier to repair [3]. For instance, violation 20 in Fig. 2 is caused by a person being coworker with another one even if working for a different company (e.g. Irene (node 1) is coworker of Li (node 5), but they work for company A and B respectively). However, Irene has multiple COWORKER_OF outgoing edges, each one consistently paired with a WORK_FOR edge (e.g., nodes 2, 3 and 4 work for company A). In this violation, edge 13 deviates from the local pattern and is therefore anomalous (it is the only case where there is a WF edge from a person to a company different from the company that their coworker works for). By contrast, in violation 21, each edge occurs only once, and no clear regularity emerges from the local neighborhood. As a result, violation 21 is less clearly attributable to a specific anomalous component, making it more difficult to repair.

In other words, violations that deviate significantly from learned graph-wide patterns will tend to receive lower confidence scores from the model, indicating lower repair difficulty. On the contrary,

the more a subgraph is aligned with learned graph-wide patterns, the harder it is to fix and may require human expertise.

We adopt a spatial GNN architecture, namely the Heterogeneous Graph Attention Network (hetero-GAT) [63], as it naturally handles subgraphs of varying size and irregular topology. Its attention mechanism further emphasizes relevant neighbors, reducing unnecessary computation and highlighting meaningful relationships.

Because directly computing the joint likelihood of a heterogeneous subgraph is intractable for graph neural networks, we follow the common factorization approach used in prior work [34, 58] on GNN-based anomaly scoring and link prediction. In particular, we approximate the subgraph likelihood by independent node and edge factors, in line with standard independence assumptions used to derive edge likelihoods and node reconstruction scores in graph autoencoders [10, 34] and link prediction models [39]. Under this approximation, the confidence of a subgraph decomposes into the mean confidence of its edges (link prediction) and the mean confidence of its nodes (density-based estimates in the learned embedding space).

To obtain the edge confidence, we employ a hetero-GAT trained specifically for link prediction tasks, which produces edge-level scores by computing the probability of existence for each edge type between heterogeneous node types.

Unlike standard GNN-based anomaly detection, we explicitly tailor both the representation and the scoring mechanism to the repair task. In particular, we restrict node features to those involved in the constraints via a feature filtering step. In the context of property graphs, nodes' features correspond to node property values. Let $h_v \in \mathbb{R}^d$ denote the vector of property values associated with a node v . Constraints typically involve only a subset $\mathcal{K} \subseteq \{1, \dots, d\}$ of these properties. To restrict node representations to constrained properties, we apply a linear projection $F \in \mathbb{R}^{|\mathcal{C}| \times d}$, which selects the components of h_v indexed by $\mathcal{K}$. The resulting representation is then mapped into a shared feature space via a node-type-specific transformation: $h'_v = M_{T_v} F h_v$, where $M_{T_v} \in \mathbb{R}^{d' \times |\mathcal{C}|}$ is the transformation matrix associated with node type T_v .

Then, we compute the embedding z_v^T of a node v for an edge of type T aggregating the neighbor's projected features h'_u : $z_v^T = \sigma \left(\sum_{u \in N_v^T} a_{vu}^T \cdot h'_u \right)$ where σ is the sigmoid activation function, N_v^T is the set of neighbors of node v via edges of type T , and a_{vu}^T is the normalized node-level attention for the edge type T . To be specific, $a_{vu}^T = \text{softmax}(\text{att}_{node}(h'_v, h'_u); T)$, where att_{node} is a deep neural network trained for node-level attention.

The advantage of using type-specific attention is that att_{node} is shared for all the edges of type T . More details about hetero-GAT models can be found in [63]. For an edge $e = (u, v)$ of type T , the **edge confidence** is: $c_{edge}(e) = \sigma(\text{MLP}_T([z_v^T || z_u^T]))$, where z_v^T and z_u^T are the latent embeddings of nodes v and u extracted by the hetero-GAT, $[z_v^T || z_u^T]$ denotes their concatenation, $\text{MLP}_T(\cdot)$ is a relation-specific feed-forward network, and $\sigma(\cdot)$ is the sigmoid function. Moreover, we define the **node confidence** based on the proximity of a node's latent representation to other nodes of the same type in the latent space. Specifically, given a node v of type T_v , we define its node confidence as: $c_{node}(v) = \exp \left(-\text{avg}_{u \in N_{T_v} \setminus \{v\}} \|z_v^T - z_u^T\|_2 \right)$ where N_{T_v} is the set of nodes of type T_v . The intuition is that nodes whose embeddings are closer to similar nodes (i.e., within the same type) in the learned space represent more "typical" or expected cases, while outliers will exhibit larger distances and lower confidence. To estimate the overall difficulty of a violation, a subgraph of the property graph G , we integrate confidence scores of both nodes and edges belonging to it. The *estimated difficulty* of a violation $v = (N_v, E_v)$ is computed as the average of the two contributions:

$$\tilde{D}(v) = \text{avg} \left(\frac{1}{|N_v|} \sum_{v \in N_v} (c_{node}(v)), \frac{1}{|E_v|} \sum_{e \in E_v} (c_{edge}(e)) \right), \quad (3)$$

where N_v and E_v are the node and edge sets of the subgraph v respectively. With this definition, higher confidence for nodes and edges means a higher likelihood of existence for the subgraph and thus higher difficulty. Finally, we annotate every hypernode of the GRDG (denoting a violation) with the estimated difficulty score.

4.4 Automated Repairs with GNN

We extend the use of GNNs beyond difficulty estimation by integrating them directly into the repair process. While some violations require human expertise due to ambiguity or domain-specific knowledge, others arise from edges whose existence is highly unlikely given the structure and semantics of the violation subgraph. Involving users in such cases is unnecessary and inefficient. Specifically, we propose a hybrid repair framework in which the model automatically resolves easy violations based on confidence scores, while delegating harder cases to human experts. Our intuition is that we can leverage the GNN to automatically repair some violations, essentially deleting low-confidence edges, thereby reducing the number of violations to be assigned to humans. Concretely, for a violation $v = (N_v, E_v)$ and a threshold $\theta \in [0, 1]$, we remove an edge $e \in E_v$ whenever its uncertainty $1 - c_{\text{edge}}(e)$ exceeds θ . If multiple edges satisfy this condition, we perform a minimal repair by deleting only the edge with the highest uncertainty (breaking ties at random). These GNN Repairs can be applied synchronously or asynchronously w.r.t. user repairs. For any fixed threshold θ , the relative order of GNN and human repairs does not affect the quality outcome.

LEMMA 4.4 (COMMUTATIVITY OF GNN AND HUMAN REPAIRS). *Given a violation assignment strategy AS that respects the constraints of problem 4.2, we define V_θ as the set of violations that would be automatically solved using GNN by using $1 - \theta$ as the threshold. Applying the GNN repairs before running AS yields the same result as applying GNN repairs after or during running AS on the GRDG $\mathcal{G} = (V \setminus V_\theta, F)$.*

PROOF SKETCH. By construction, GNN repairs consist exclusively of deleting edges belonging to violations whose edge confidence falls below the threshold $1 - \theta$ (i.e., they do not affect nodes). Since dependencies in the GRDG are created only from shared nodes, and node sets are unaffected, GNN repairs neither alter violation overlaps nor introduce new violations. Hence, the user assignment as well as the total expected quality of repairs conducted by users on the V_θ violations won't change if the GNN repairs are applied before, after or even during the AS assignment. $\square$

The threshold θ is the central parameter that controls the tradeoff between automated and manual repairs. Intuitively, large values of θ make the system conservative, delegating most violations to human experts, while smaller values increasingly favor automation by removing low-confidence edges, by raising the threshold for what is deemed "low confidence". As a result, our approach is fully tunable: practitioners can select θ to explicitly balance repair quality, human effort, and computational cost according to application requirements and available budget. Decreasing θ monotonically increases the number of violations resolved automatically by the GNN, and correspondingly reduces the size of the GRDG passed to the assignment strategy. This reduction directly translates into fewer involved users and a smaller optimization problem, yielding significant improvements in runtime and scalability (Section 4.5.2).

4.5 Lagrangian-Relaxation Assignment

Given its NP-hardness, we solve the budget-constrained GRDG covering problem (Definition 4.2) with a Lagrangian-relaxation framework [30]. Intuitively, Lagrangian relaxation replaces the hard budget constraint by a soft penalty: a multiplier λ charges a price for each unit of budget consumed. For a fixed λ , the problem decomposes into independent user subproblems, each deciding whether

the value of covering a partition outweighs its penalized cost. This relaxation is easier to solve than the original problem, and by adjusting λ with subgradient updates, we balance solution quality and budget usage.

Lagrangian Relaxation of the Budget. Let B be the total budget and c_i the cost of user u_i . We introduce a multiplier $\lambda \geq 0$ and relax the budget constraint $\sum_{u_i \in U'} c_i \leq B$:

$$\mathcal{L}(\lambda) = \max_{\substack{\{P_i\} \text{ connected,} \\ \sum_{v \in P_i} D_v \leq CC_i, \\ P_i \cap P_j = \emptyset, \cup_i P_i = V}} \sum_{u_i} \left(\sum_{v \in P_i} EQ(D_v, K_i) - \lambda c_i \right) + \lambda B. \quad (4)$$

For any $\lambda \geq 0$, $\mathcal{L}(\lambda)$ sets the upper-bound of the optimal value of the objective function of the *primal problem* (e.g. the original budget-constrained covering problem) and any feasible solution to it yields a *lower bound* on the optimal objective. The *dual problem* is to minimize the Lagrangian function (4) over $\lambda \geq 0$, whose value provides an *upper bound* since relaxing the constraint enlarges the feasible set. The gap between these primal and dual bounds quantifies how close the relaxation is to the true optimum.

For a fixed λ , user u_i contributes with a gain $\Phi_i(P_i; \lambda) =_{\text{def}} \sum_{v \in P_i} EQ(D_v, K_i) - \lambda c_i$, with P_i being a connected subgraph assigned to u_i and $\sum_{v \in P_i} D_v \leq CC_i$. Since users' partitions must be disjoint and cover V , we work with a set of candidate subgraph assignments per user and then perform a final selection among the candidates.

Per user assignment. Unlike classical assignment problems where tasks are independent, our setting requires assigning connected subgraphs under structural dependencies. This fundamentally changes the optimization problem and motivates our seed-based candidate generation. Algorithm 1 shows the procedure generating the candidate assignments. For each user u_i , we build a small set $\mathcal{S}_i$ of connected subgraphs $P \subseteq V$, with $\sum_{v \in P} D_v \leq CC_i$. We first identify the violation v that produces the highest gain $g_i(v) \leftarrow EQ(D_v, K_i)$ for the user u_i , then we greedily grow the subgraphs by including nodes that maximize the gain (lines 2-6). Finally, we perform t steps of local improvement by swapping neighboring nodes, if the user's capacity allows it (line 7). We note that in practice, the algorithm uses *estimated* difficulty $\tilde{D}$, but just write D for simplicity.

Algorithm 1 BuildCandidatesSet(i): candidate subgraphs for u_i

Require: GRDG $\mathcal{G} = (V, F)$, $g_i(v) \leftarrow EQ(D_v, K_i)$, capacities CC_i , difficulties D_v

Ensure: A set $\mathcal{S}_i$ of candidate subgraphs.

```

1:  $\mathcal{S}_i \leftarrow \emptyset$ 
2: for each violation  $v$  in Top- $V$  vertices by  $p_i(v)$  do
3:    $P \leftarrow \{v\}$ 
4:   while there exists  $u \in \text{Neighbors}(P)$  with  $\sum_{v \in P} D_v + D_u \leq CC_i$  and  $EQ(D_u, K_i) > 0$  do
5:     add to  $P$  the  $u$  with max  $g_i(u)$ 
6:   end while
7:   Apply  $t$  steps of local improvement (swap/add/drop neighbors if feasible)
8:    $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{P\}$ 
9: end for
10: return  $\mathcal{S}_i$ 

```

Assignment under a budget. Algorithm 2 shows how to select the candidate subgraphs for each user. Given the set of candidate subgraphs $\{\mathcal{S}_i\}$ for each user u_i , we now select which users to activate and which single assignment each of them will take. We proceed greedily. Let A be the set of violations already assigned (empty in the beginning). For any pair (i, P) with $P \in \mathcal{S}_i$, we compute its current marginal gain as $\Delta g(i, P) = \sum_{v \in P \setminus A} EQ(D_v, K_i) - \lambda c_i$, i.e., the expected quality contributed by solving violations in P that are not yet covered (line 4). At each step we choose the feasible pair with the largest ratio $\Delta g(i, P)/c_i$ (line 6). We add P to the assignment, update

$A \leftarrow A \cup P$, and remove (i) all other candidates of user i (at most one per user) and (ii) any pair (j, P') with $P' \cap A \neq \emptyset$ (to avoid overlaps). We stop when the budget $\sum_{i \in U'} c_i$ would be exceeded or when no positive-gain pair remains.

Algorithm 2 BudgetAssignment($\{\mathcal{S}_i\}, B, \lambda$)

Require: Sets of candidates assignments $\mathcal{S}_i$, Budget B , multiplier λ .

Ensure: The best subgraph assigned to each user.

```

1:  $U' \leftarrow \emptyset, A \leftarrow \emptyset, \mathcal{I} \leftarrow \{(i, P) : i \in U, P \in \mathcal{S}_i\}$ 
2: while  $\exists (i, P) \in \mathcal{I}$  with  $c_i + \sum_{j \in U'} c_j \leq B$  do
3:   for each feasible  $(i, P)$  with  $P \cap A = \emptyset$  do
4:     compute  $\Delta g(i, P) = \sum_{v \in P} g_i(v) - \lambda c_i$ 
5:   end for
6:   Select  $(i^*, P^*) = \arg \max_{(i, P)} \frac{\Delta g(i, P)}{c_i}$  with  $\Delta g(i, P) > 0$ 
7:   if no positive-gain item exists then
8:     break
9:   end if
10:   $U' \leftarrow U' \cup \{i^*\}$ 
11:  assign  $P_{i^*} \leftarrow P^*, A \leftarrow A \cup P^*$ 
12:  Remove all items  $(i^*, \cdot)$  and all items overlapping  $A$  from  $\mathcal{I}$ 
13: end while
14: return  $(U', \{P_i\}_{i \in U'})$ 

```

Full algorithm. A key challenge is that the inner solver (Algorithm 2) is a greedy priced selector (i.e. $\Delta g(i, P) = \sum_{v \in P} g_i(v) - \lambda c_i$). This solver does not come with a worst-case guarantee on the Lagrangian objective $\mathcal{L}(\lambda)$. We cannot guarantee an upper bound of the optimal solution because we are optimizing the total priced gain $\sum_i \left(\sum_{v \in P_i} EQ(D_v, K_i) - \lambda c_i \right)$, and not a pure monotone sub-modular one (like in a sub-modular knapsack problem [35]). We therefore certify solution quality with a decoupled dual upper bound. Conceptually, we relax all the constraints on the assignment (overlaps, “one-per-user”, and coverage) and let each user contribute at most their best candidate value in isolation. For each user i , let $\text{Val}_i = \max_{P \in \mathcal{C}_i} \sum_{v \in P} EQ(D_v, K_i)$ be the best value they could achieve in isolation on a candidate subgraph. The decoupled upper bound is: $UB^{\text{dec}}(\lambda) = \sum_{i=1}^N \max\{0, \text{Val}_i - \lambda c_i\} + \lambda B$, $\lambda \geq 0$. This relaxation is guaranteed to sit above the true Lagrangian and therefore above the optimal objective, so it provides a valid certificate regardless of how the priced greedy selector behaves. A key practical advantage is that this bound admits a closed-form minimizer via a simple breakpoint computation after sorting users by value-per-cost; it is inexpensive, deterministic, and independent of implementation details (e.g., tie-breaking or step sizes). To compute the certified upper bound, we first take, for each user i , their best candidate P_i with value Val_i and compute the ratio $r_i = \text{Val}_i / c_i$ where c_i is the cost of the user. We sort users by r_i in decreasing order, carrying along (Val_i, c_i) , and define the prefix sums $C_k = \sum_{j \leq k} c_{(j)}$ and $Sg_k = \sum_{j \leq k} g(j, P_j)$. If the budget covers all users ($B \geq C_N$), the minimizer is $\lambda^* = 0$ and the certificate is $UB_{\text{cert}} = \sum_i g(i, P_i)$. Otherwise, letting k^* be the first index with $C_{k^*} \geq B$, we set $\lambda^* = r_{(k^*)}$ obtaining $UB_{\text{cert}} = Sg_{k^*-1} - \lambda^* C_{k^*-1} + \lambda^* B$. The decoupled bound can be written as $UB^{\text{dec}}(\lambda) = \sum_{i: r_i > \lambda} (\text{Val}_i - \lambda c_i) + \lambda B$, which is convex and piecewise linear in λ with kinks exactly at the ratios $\{r_i\}$. Between two consecutive ratios, the slope equals $B - \sum_{i: r_i > \lambda} c_i$, i.e., the budget minus the total cost of users whose ratio exceeds the current threshold. As λ increases, this cumulative cost drops in steps when we pass a ratio, so the minimum occurs when the slope first changes sign—precisely at the smallest prefix whose cost reaches the budget. Plugging $\lambda^* = r_{(k^*)}$ and separating the top $(k^* - 1)$ users yields the closed form above; the case $B \geq C_N$ corresponds to $\lambda^* = 0$.

Algorithm 3 shows the LARA, the full Lagrangian Relaxation Assignment algorithm. We first run BUILD_CANDIDATES_SET to obtain the assignment candidates for each user (lines 2-4). We then

compute the certified upper bound with the method described above before running the outer loop for T iterations. In the loop, we call `BUDGETASSIGNMENT` at price λ obtaining a feasible assignment $U'(\lambda)$ with value Z (the primal) and cost $\text{cost}(\lambda) = \sum_{i \in U'(\lambda)} c_i$. We then compute the dual upper bound UB_t at iteration t (lines 7-9). We maintain the best feasible solution $(U^*, \{P_i^*\})$ with value Z^* seen so far, and a running dual upper bound $UB = \min_t UB_t$ (lines 10-13). We then update the Lagrange multiplier by a projected subgradient step using the difference between the cost and the budget: $\lambda \leftarrow \max\{0, \lambda + \alpha_t((\sum_{i \in U'} c_i - B))\}$, so λ increases when the assignment exceeds the budget and decreases otherwise (typical choices are diminishing steps $\alpha_t = \alpha_0/\sqrt{t}$).

We track the best feasible value Z_t returned by the greedy assignment and stop once the certified gap $1 - \frac{Z_t}{UB_{\text{cert}}} \leq \varepsilon$.

The outer loop stops if the multiplier stabilizes ($|\lambda_t - \lambda_{t-1}| \leq \varepsilon_\lambda$ for k iterations) or if the maximum T is reached.

Algorithm 3 LARA($B, \{EQ, CC_i, c_i\}$)

```

1: Initialize  $\lambda \geq 0$ , step sizes  $\{\alpha_t\}$ , best feasible solution  $(U^*, \{P_i^*\}) \leftarrow \emptyset$ , best primal value  $Z^* \leftarrow -\infty$ , best dual upper bound  $UB \leftarrow +\infty$ 
2: for each user  $i$  do
3:    $C_i \leftarrow \text{BuildCandidatesSet}(i)$ 
4: end for
5:  $(UB_{\text{cert}}, \hat{\lambda}) \leftarrow \text{BreakpointUB}(\{\text{Val}_i\}, \{c_i\}, B)$ 
6: for  $t = 1, 2, \dots, T$  do
7:    $(U', \{P_i\}) \leftarrow \text{BudgetAssignment}(\{C_i\}, B, \lambda)$ 
8:    $Z \leftarrow \sum_{i \in U'} \sum_{v \in P_i} EQ(D_v, K_i)$ 
9:    $UB_t \leftarrow (\sum_{i \in U'} \sum_{v \in P_i} EQ(D_v, K_i) - \lambda \sum_{i \in U'} c_i) + \lambda B$ 
10:  if  $Z > Z^*$  then
11:     $(U^*, \{P_i^*\}, Z^*) \leftarrow (U', \{P_i\}, Z)$ 
12:  end if
13:   $UB \leftarrow \min\{UB, UB_t\}, \lambda_{\text{old}} \leftarrow \lambda$ 
14:   $\lambda \leftarrow \lambda + \alpha_t(\sum_{i \in U'} c_i - B)$ 
15:   $s \leftarrow \begin{cases} s + 1, & \text{if } |\lambda - \lambda_{\text{old}}| \leq \varepsilon_\lambda \\ 0, & \text{otherwise} \end{cases}$ 
16:  if  $\frac{UB_{\text{cert}} - Z^*}{UB_{\text{cert}}} \leq \varepsilon_{\text{gap}}$  or  $s \geq k$  then
17:    break
18:  end if
19: end for
20: return  $(U^*, \{P_i^*\})$ , LowerBound =  $Z^*$ , UpperBound =  $UB$ 

```

4.5.1 Guarantees. To assess the quality of the solutions returned by LARA, we need two ingredients. First, we show that the decoupled relaxation introduced in the previous section provides a valid dual upper bound on the optimal objective, regardless of the greedy solver behavior. Second, we use this bound to certify that whenever the algorithm terminates by the certified stopping rule, the returned solution has a relative optimality gap at most ε with respect to this dual upper bound. That is, the algorithm certifies near-optimality with respect to the Lagrangian relaxation, though not necessarily w.r.t. the original NP-hard problem. The following theorem formalizes the validity of the certified upper bound.

THEOREM 4.5 (CERTIFIED UPPER BOUND). *For all $\lambda \geq 0$, $UB^{\text{dec}}(\lambda) \geq \mathcal{L}(\lambda) \geq \text{OPT}$, hence $UB_{\text{cert}} \equiv \min_{\lambda \geq 0} UB^{\text{dec}}(\lambda)$ is a valid dual upper bound on the optimal value of the original problem.*

PROOF SKETCH. $UB^{\text{dec}}(\lambda)$ drops all the constraints for the assignments, enlarging the feasible set of the inner maximization that defines the Lagrangian. This can only increase the value, so $UB^{\text{dec}}(\lambda) \geq \mathcal{L}(\lambda)$. The weak duality gives $\mathcal{L}(\lambda) \geq \text{OPT}$. $\square$

Table 1. Dataset statistics.

Dataset	# Nodes	# Edges	# Viol.	# Dep.	Density	Avg Diff.
FAERS	~ 11K	~ 62K	~ 2.4K	~ 127K	0.0220	0.7595 (0.1139)
Finbench	~ 56K	~ 311K	~ 9K	~ 31K	0.0003	0.7666 (0.1395)
ICIJ	~163K	~364K	~ 33K	~ 24M	0.0220	0.8100 (0.1135)
SNB	~ 358K	~728K	~ 70K	~ 5.9M	0.0120	0.7241 (0.0955)

Given the certified dual upper bound UB_{cert} from Theorem 4.5, we monitor the best feasible primal value Z_t . If $1 - \frac{Z_t}{UB_{\text{cert}}} \leq \varepsilon$, then Z_t is guaranteed to be within a relative gap ε of this dual bound, i.e., $Z_t \geq (1 - \varepsilon) \cdot UB_{\text{cert}}$. This provides a computable certificate of near-optimality with respect to the dual bound. In § 5.3 we show how different numbers of GNN repairs affect the achieved value and the certified gap with respect to UB_{cert} .

4.5.2 Time complexity. Let $V := |V|$ be the number of nodes in the GRDG, $U := |U|$ the number of users, and T the number of outer Lagrangian iterations actually executed. The dominant cost is BUILD_CANDIDATES_SET, which, for each user, grows seed candidate sets and performs a few local-improvement steps. Up to constants, the complexity depends on how many neighbors each candidate set can see and how many seed sets and local steps we allow. Formally, the hidden constants in our bounds are controlled by: (i) the graph's average degree Δ , (ii) the typical candidate size X , (iii) the number of seeds per user S , and (iv) the local-search steps t . Larger Δ , S , t , or X make candidate construction heavier, while the assignment phase remains roughly linear in the total size of all candidates and is not the bottleneck.

Worst case. In dense graphs ($\Delta = \Theta(|V|)$) with many seeds and long local search ($S, t = \Theta(|V|)$) and large candidate sets ($X = \Theta(|V|)$), the per-user time is $\Theta(|V|^4)$, hence $T_{\text{total}} = O(T |U| |V|^4)$.

In practice. GRDGs are sparse, as shown in Table 1. The number of violations is strictly lower than the number of nodes [14, 41], and hence also the number of overlaps (i.e., number of dependency edges). With sparse GRDGs ($\Delta = O(1)$), capped seeds and local steps ($S, t = O(1)$) and candidate subgraphs much smaller than the graph ($X \ll |V|$), the per-user cost is near-linear, yielding $T_{\text{total}} \approx O(T |U| |V| \log |V|)$, with small constant factors. Intuitively: bigger or denser graphs, more seeds, and longer local search increase the constant; sparse graphs and capped neighborhoods keep the runtime close to linear in V per user per outer iteration. Early stopping simply replaces T by the observed $T' \leq T$. For a more detailed complexity analysis, refer to the supplementary material [8]. Notice that this corresponds to the most common scenarios as we show in Table 1 and it is also reflected in the results of our experimental study (see § 5.3).

5 Experiments

In this section, we experimentally evaluate the effectiveness of our approach over different datasets and settings. We used Neo4j [49] (version 5.12) as the underlying graph database, while the GNN was implemented using PyTorch. All materials needed to reproduce the results and train the GNN are publicly available [7].

5.1 Settings

5.1.1 Datasets. The datasets used in our experiments are illustrated in Table 1 and include FAERS [47] which contains real world data from a food and drug adverse event reporting system, Finbench [53], a synthetic dataset that simulates real-world financial transactions, SNB [5], a synthetic dataset that simulates a realistic social network scenario, with users, posts, comments and likes. Additionally, we used ICIJ [46], a real-world dataset with the investigation on Paradise Papers led by the International Consortium of Investigative Journalists. We manually derived real

world denial constraints for each dataset using different pattern lengths. As previously done in the literature [14, 15, 41, 52] we introduced $\sim 20\%|V|$ violations in each dataset, where $|V|$ is the number of nodes in the graph. The list of constraints is reported in the supplementary material [8].

5.1.2 Behavior of the Simulated Users. We model user behavior as described in Section 4.1, where each user is represented as a tuple (K, CC, c) , denoting their skill, cognitive capacity, and cost respectively. We created the users by drawing from a uniform probability their value of skill $K \in [0, 1]$, cognitive capacity $CC \in [5, 20]$, and the cost of each user $c \in [1, CC - 1]$ is computed as a linear function of the skill: the more skilled a user is, the more it costs.

To validate the model and ensure it captures real-world behavior, we conducted a user study with 20 users repairing the *Star Wars* dataset [48]. Participants first self-assessed their domain knowledge (familiarity with *Star Wars*) and technical skill (experience with graph data structures) on a Likert scale from 1 (none) to 5 (expert). These values were normalized and mapped to the user-model parameter K . Then, participants were presented with 6 distinct graph violations extracted from the dataset. For each one, they were asked to select a repair from a predefined list of candidate repairs. The details are in the suppl. materials [8].

We collected success/failure outcomes for each user–violation pair. We then applied a 1-Parameter Logistic Item Response Theory model [56] to estimate the latent difficulty of each violation. These ground-truth difficulty values were then used to compute the users’ perceived difficulties for each dataset, as a linear combination of node and edge confidences. These perceived difficulties are used as input to our user model in order to simulate user behavior.

Since training a user model to predict the exact repair a user would choose requires a large amount of data, we decided instead to predict the *class* of a repair according to their quality (F1 score). Different repairs indeed lead to the same F1 score (e.g. all the repairs corresponding to the ground truth lead to $F1=1$ or all the repairs that delete the wrong edge lead to $F1=0$, etc.). We trained a Naive Bayes classifier to predict the most likely repair based on user skill and the violation’s perceived difficulty. While assignments rely on *estimated* difficulty, user behavior is driven by *perceived* difficulty. Lastly, to ensure all the violations are repaired, even in the worst case (i.e., each user can repair only one violation), we consider the initial number of users equal to the number of violations. This choice is an experimental control to ensure that all compared methods are evaluated under identical conditions. It does not imply that all users are used by the repair process: the assignment algorithm selects only a subset of users subject to the budget and capacity constraints.

5.2 Validating the difficulty estimation

This experiment aims to validate our GNN confidence based difficulty estimate (ConfGNN), obtained using the method described in § 4.3, by comparing it with alternative heuristics.

Specifically, we consider two families of baselines: structural heuristics such as node Degree and PageRank, assuming the GRDG topology influences task difficulty as in [52], and cognitive load (CL), computed as a simple function of the number of entities and relations involved in a violation, following the approach used in crowdsourcing [66]. For the PageRank and Degree baselines, we replace the ConfGNN difficulty score with a topology-based score computed from the underlying graph. Specifically, we aggregate node-level PageRank (resp. degree) scores over the nodes involved in each violation to obtain a scalar difficulty value, which is then used unchanged as input to the assignment algorithm. The second family belongs to the anomaly detection score-based methods. Radar [38] is residual analysis based anomaly detection method for attributed networks. HO-GAT [33] is a GAT based method that combines attributes information with structural properties of the graph. SmoothGNN [19] is a smoothing aware unsupervised GNN method for graph anomaly scoring.

For this experiment, all other components of the pipeline, including the user model, assignment strategy, and repair process, remain unchanged.

First, we evaluated the accuracy of each estimate by comparing it to the real difficulty scores, using the Mean Squared Error (MSE) and the Kullback-Leibler Divergence (which measures the relative entropy between distributions). We use KL-divergence to check that the predicted difficulty distribution matches the true one, not just pointwise accuracy. This matters because rare high-difficulty violations disproportionately affect assignment quality, so preserving them in the predicted distribution is critical. As shown in Table 2, our method yields the most accurate estimates, indicating that indeed the GNN approach blends topology and the semantics of the graph to better estimate the difficulty of the violations. While SmoothGNN achieves lower MSE on two datasets, ConfGNN attains lower KL-Divergence, better capturing outliers. Second, we ran the full graph repair pipeline to assess how each difficulty estimate affects the assignment of violations and the resulting repair quality. To isolate the effect of the difficulty estimate, we disabled the GNN’s edge pruning capability by setting the pruning threshold to 1.0, ensuring that no edges would be removed (see § 4.4). Each difficulty function was integrated into our assignment algorithm, and we evaluated the resulting repairs using the F1 Score, ROC-AUC, and Mean Reciprocal Rank (MRR). While the F1 Score measures the correctness of the final repairs, ROC-AUC measures the ability of the assignment strategy to discriminate between better and worse repair choices, and MRR evaluates how highly the correct repair action is ranked for each violation. Table 2 shows that our estimator consistently achieves the best overall performance across datasets and metrics. In particular, ConfGNN obtains the highest ROC-AUC, and MRR scores in nearly all cases, indicating that it not only improves the final repair quality, but also produces substantially better repair rankings during assignment. The strong MRR results show that the correct repair choices are typically ranked near the top, while the high ROC-AUC values indicate a clear separation between good and poor repair candidates. CL is the second-best method for FAERS, but it fails to deliver good results on the larger datasets, suggesting that estimations based solely on the size of the violations are insufficient to capture their difficulty. Degree and PageRank fail to achieve satisfactory results, indicating that the local topology alone does not correctly capture the complexity of repairing a graph. Moreover, Radar and HO-GAT do not scale to larger datasets due to prohibitive matrix operations and feature expansion, respectively.

Table 2. Comparison of our method with different difficulty estimations. ME = Memory Error. Bold indicates the best. FinB.=Finbench

Metric	Mean Square Error				KL-divergence				F1 Score				ROC AUC				MRR			
	FAERS	FinB.	ICIJ	SNB	FAERS	FinB.	ICIJ	SNB	FAERS	FinB.	ICIJ	SNB	FAERS	FinB.	ICIJ	SNB	FAERS	FinB.	ICIJ	SNB
CL	0.059	0.146	0.118	0.178	0.229	2.777	0.252	8.313	0.923	0.944	0.596	0.515	0.889	0.938	0.914	0.343	0.820	0.805	0.917	0.346
Degree	0.180	0.216	0.210	0.151	4.044	6.059	3.955	2.975	0.677	0.511	0.592	0.533	0.890	0.499	0.616	0.390	0.820	0.466	0.516	0.374
PageRank	0.057	0.116	0.086	0.038	3.201	5.865	1.772	1.423	0.678	0.437	0.406	0.475	0.886	0.748	0.772	0.937	0.820	0.718	0.754	0.918
Radar	0.098	0.213	ME	ME	0.068	0.336	ME	ME	0.570	0.814	ME	ME	0.576	0.545	ME	ME	0.607	0.482	ME	ME
HO-GAT	0.029	0.033	ME	ME	0.063	0.037	ME	ME	0.844	0.648	ME	ME	0.833	0.995	ME	ME	0.829	0.942	ME	ME
SmoothGNN	0.064	0.027	0.016	0.030	0.080	0.016	0.028	0.037	0.571	0.995	0.972	0.944	0.794	0.995	0.998	0.576	0.782	0.942	0.988	0.511
ConfGNN	0.024	0.093	0.063	0.019	0.003	0.004	0.003	0.002	1	0.9996	0.9996	0.9994	0.993	0.9989	0.9996	0.99986	0.962	0.996	0.9991	0.9994

5.3 Balancing human involvement

In this experiment, we analyzed the influence of GNN involvement and the available budget on the final repair quality.

We vary θ from 0 (the GNN performs all the repairs) to 1 (only humans repair the graph). The budget depends on the dataset, and it varies from 0 (no human can be involved), to the amount necessary to involve all the humans in the repair of all the violations.

Fig. 3 illustrates the results. Since the datasets show similar behaviors, and for space limitations, we only report two of them, with the rest shown in the supplementary material [8]. Each graph

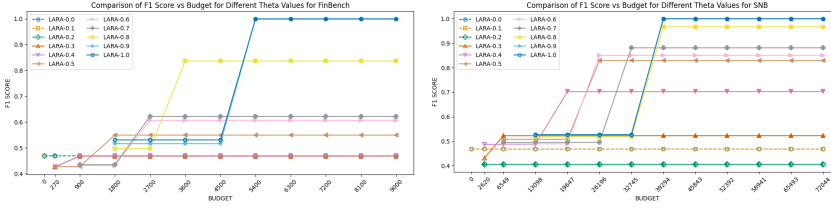


Fig. 3. F1-Score of our approach using different θ across different datasets at different budget values.

shows the F1 score as a function of the budget, with different lines corresponding to the employed θ values (e.g., LARA- $\theta.1$ corresponds to $\theta = 0.1$). The reported F1 score is computed as the average between the scores obtained by the GNN and the users. The dashed lines represent the values of θ where the GNN repairs all the violations. We vary the budget from 0 to maximum to cover all the users. Each point thus represents a run of the assignment with that budget.

We can observe a common behavior across the different datasets where the charts are divided into four regions. The plot exhibits four regions. The infeasible area contains missing points, corresponding to settings where the budget and GNN involvement are insufficient to assign all violations. This occurs rarely and only at higher θ values requiring more human involvement (e.g., the first two or three steps for LARA- $\theta.8$). The feasible region is where all violations are assigned within the available budget, but the more expert (and costly) users cannot be involved. The inflection point marks the budget needed to achieve the highest-quality assignment, where high-skilled users become affordable. Finally, the plateau area is where additional budget no longer improves repair quality; solutions here always fall within the optimality gap (shown as filled dots on the line, versus empty dots for solutions outside the gap).

For instance, let us consider the case of LARA- $\theta.8$ applied to FinBench (Fig. 3). When the budget is set to 0 or 900, the algorithm is unable to allocate a sufficient number of users to repair all the violations; this situation corresponds to the *infeasible* region. When the budget increases to values between 1800 and 2700, the graph can be repaired; however, not all highly skilled users are involved in the process. As a consequence, the F1 score, remains around 0.5, which characterizes the *feasible* region. The *inflection point* is reached at a budget of 3600, where the involvement of additional skilled users results in an improvement of the repair quality, leading to an F1 score of 0.92. Beyond it, increasing the budget does not produce significant gains (e.g. the budget is sufficient to assign the enough expert users). It is interesting to observe that for all the datasets, the plateau is reached while under 70% of the total maximum budget is consumed. In fact, for F1 scores > 0.8 the plateau is reached within three to four successive budget increments prior to the maximum budget.

Notice that some values of θ result in a flat line. This occurs for lower values (e.g., LARA- $\theta.2$ on FinBench in Fig. 3(a)), where the violations predominantly repaired with GNN-Repairs. Because the GNN already fixes the majority, assigning the remaining violations to experts has little impact on overall quality.

In the supplementary material [8] we report the runtime of LARA for each θ under the maximum budget. As discussed in § 4.5.2, sparse graphs lead to near log-linear complexity in the number of nodes. Runtime increases with θ due to the larger number of assigned violations, but remains moderate.

5.4 Comparison with baselines

We compared our method with several baselines from the literature, covering crowdsourcing-based assignment strategies, fully automated approaches, and interactive graph repair techniques.

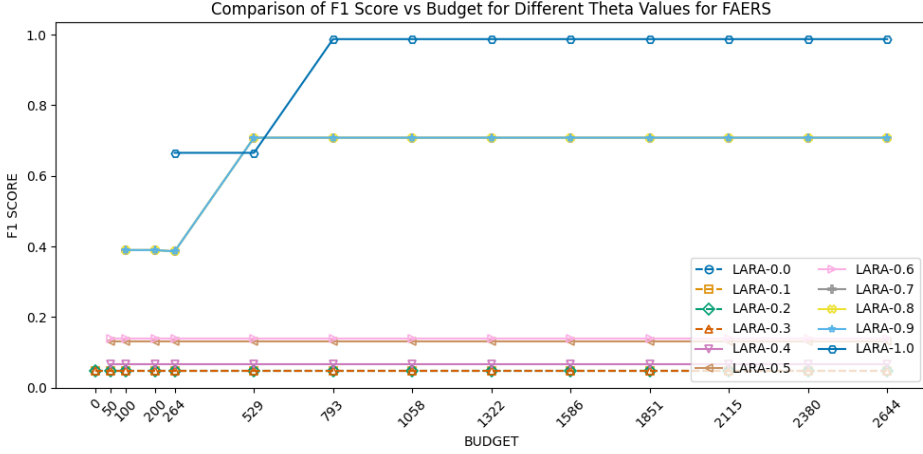
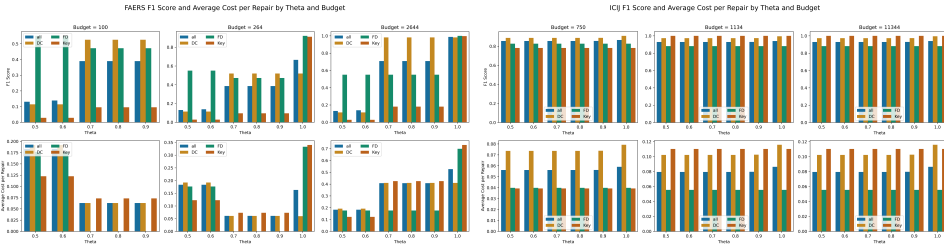
Crowdsourcing-based assignment methods aim to allocate tasks to human users based on models of skill, cost, and expected task success. Although not specifically designed for graph data, these approaches are well-established in the crowdsourcing literature. The similarity-based assignment strategy (SIM) embeds users and violations in a shared feature space and assigns tasks based on vector similarity [17]. We use violation difficulty and user skill as features, with normalized difference as the similarity metric. Another baseline is heterogeneous GNN-based label inference (LI) [65]. We adapt it to graph repair by predicting the most suitable repair using at most 10% of the available users.

We also compared LARA with the interactive graph repair method uQAR [52], which assigns violations based on structural characteristics (e.g., node degree). It uses a dependency graph with a locking mechanism to avoid conflicting repairs. Finally, we evaluated automated graph repair methods based on preferred repairs [62]. These approaches make repair decisions heuristically, without user involvement. Pref-D (Preferred Delete): always deletes edges or nodes to resolve violations first. Pref-U (Preferred Update): preferentially modifies node or edge properties instead of deletions. Pref-S (Preferred Schema): selects transformations according to a schema, favoring safer operations.

We report the F1 score of all methods, cost as the sum of the costs of all involved users, and number of users involved for the interactive repair methods. While Table 2 evaluates the quality of the difficulty estimators in isolation, the results in this section assess their impact on the end-to-end repair pipeline when used within the assignment strategy. As shown in Table 3, LARA outperforms all baselines across all datasets, achieving the highest F1 scores consistently. In particular, the row corresponding to LARA-x in Table 3 reflects the full pipeline performance using the difficulty estimates evaluated in Table 2, allowing us to assess how estimation quality translates into downstream repair effectiveness (in particular, the f1 score reported for LARA-1 coincide with the ones for the ConfGNN estimation in Table 2. The SIM strategy performs reasonably well on smaller datasets but fails to scale to ICIJ and SNB due to computational overhead. LI also suffers from “memory exceeded” (ME) errors on larger datasets. Among the automated methods, Pref-S yields the best results, suggesting schema-based heuristics can serve as a viable fallback in the absence of human expertise. Still, all automated methods lag significantly behind LARA.

Table 3. Comparison of F1 score, total cost, and total number of users for each dataset across all methods. TO = Timeout, ME = Memory Exceeded. Bold values indicate where LARA beats the baselines.

Method	F1 Score				Total Cost				Total Users			
	FAERS	FinBench	ICIJ	SNB	FAERS	FinBench	ICIJ	SNB	FAERS	FinBench	ICIJ	SNB
SIM	0.8061	0.7926	TO	TO	1326	4931	TO	TO	2403 (100%)	8998 (100%)	TO	TO
LI	0.4172	ME	ME	ME	264.3	ME	ME	ME	480 (20%)	ME	ME	ME
uQAR	0.8345	0.7926	0.7678	0.7783	1326	4931	18246	39362	2403 (100%)	8998 (100%)	32948 (100%)	71443 (100%)
Pref-D	0.5000	0.3833	0.1996	0.2984	-	-	-	-	-	-	-	-
Pref-U	0	0.6908	0.6002	0.4104	-	-	-	-	-	-	-	-
Pref-S	0.6253	0.7912	0.6409	0.7874	-	-	-	-	-	-	-	-
LARA-0.0	0.8691	0.4689	0.5127	0.3964	0	0	0	0	0	0	0	0
LARA-0.1	0.8691	0.4689	0.5127	0.3964	0	0	0	0	0	0	0	0
LARA-0.2	0.8691	0.4689	0.5127	0.4052	2.102	0	0	477.4	6 (0.24%)	0	0	253 (0.35%)
LARA-0.3	0.8691	0.4681	0.5127	0.4065	23.72	395.9	0	6236	29 (1.2%)	4 (0.04%)	0	925 (1.29%)
LARA-0.4	0.8691	0.4686	0.5127	0.7026	135.7	402.9	0	14692	65 (2.7%)	14 (0.15%)	0	1455 (2.04%)
LARA-0.5	0.8691	0.5494	0.5127	0.8291	273.1	1238	0.506	22590	114 (4.7%)	425 (4.7%)	1 (0.003%)	1952 (2.73%)
LARA-0.6	0.8691	0.6066	0.5126	0.8383	367.7	1844	5.073	25675	150 (6.2%)	774 (8.6%)	2 (0.006%)	2373 (3.32%)
LARA-0.7	0.8691	0.6219	0.6202	0.8858	612.3	1998	5312	27772	205 (8.5%)	856 (9.5%)	263 (0.7%)	2830 (3.96%)
LARA-0.8	0.9237	0.8321	0.9489	0.9669	913.0	3258	17840	33401	278 (12%)	918 (10%)	932 (2.83%)	3444 (4.82%)
LARA-0.9	0.9266	0.9982	0.9933	0.9996	1014	4849	18701	35977	329 (14%)	1261 (14%)	941 (2.85%)	3863 (5.41%)
LARA-1.0	1	0.9996	0.9996	0.9994	1477	5093	20090	37437	446 (19%)	1640 (18%)	1162 (3.5%)	4014 (5.62%)

Fig. 4. F1-Score of our approach using different θ for FAERSFig. 5. F1-Scores and total costs of our approach using $\theta > 0.5$ across FAERS and ICIJ at different budget values.

uQAR ranks second among interactive methods but lacks adaptation to heterogeneous user skills and violation complexity, resulting in suboptimal performance compared to LARA.

LARA achieves high-quality repairs with lower cost and less human effort. At $0.8-0.9$, it consistently outperforms baselines in quality and budget. The only exceptions are LI on FAERS and one uQAR case on ICIJ, both however resulting in lower F1 scores.

Finally, Table 3 also reports the number of users involved in each baseline, both as an absolute count and as a fraction of the initial pool (equal to the number of violations). Our method not only outperforms the others, but also consistently reduces the number of involved users by more than 80%. This suggests that the framework remains applicable when the number of detected violations is large but the effective number of available users is limited.

Furthermore, in the supplementary material [8] we report the number of repairs performed by users and the GNN, showing how the tunable parameter θ impacts the trade-off between quality and costs highlighted in Table 3. In general, for all the datasets, $\theta \in \{0.8, 0.9\}$ represents the best trade-off between automated repairs and user intervention, suggesting that selecting a θ close to the average difficulty value results in a good quality/cost balance.

Table 4. Comparison of F1 score and total cost for each dataset with different values of θ and LLM involvement. FinB.=Finbench.

Mode	θ	F1 Score			Total Cost		
		FinB.	ICIJ	SNB	FinB.	ICIJ	SNB
G-L	0.5	0.245 ↓ 55%	0.629 ↑ 23%	0.445 ↓ 47%	0	0	0
G-L	0.8	0.277 ↓ 67%	0.613 ↓ 36%	0.439 ↓ 55%	0	0	0
G-L	1	0.262 ↓ 74%	0.605 ↓ 39%	0.445 ↓ 55%	0	0	0
G-L-U	0.5	0.785 ↑ 42%	0.900 ↑ 75%	0.807 ↓ 3%	866.7	0.385	18229
G-L-U	0.8	0.787 ↓ 5%	0.867 ↓ 9%	0.835 ↓ 14%	2313	12844	23714
G-L-U	1	0.762 ↓ 24%	0.897 ↓ 10%	0.857 ↓ 15%	3514	14464	26580

5.5 LLMs for Data Repairs

To further investigate the repair process, we evaluate whether Large Language Models (LLMs) can partially or fully replace human workers for repairing violations, analyzing the trade-off between repair quality and cost. We consider three values of $\theta \in \{0.5, 0.8, 1.0\}$, corresponding to LARA- θ . For each θ , we evaluate two repair strategies: (i) assigning all remaining violations to the LLM (G-L), and (ii) splitting in half the remaining violations between the LLM and human users according to the violation difficulty score, assigning the easiest to the LLM and the hardest to the users (G-L-U). The LLM used in our experiments is Llama-3-8B [45]. Table 4 reports the resulting F1-score and total repair cost for each dataset and configuration. We additionally report the relative increase (↑) or decrease (↓) in F1-score w.r.t. LARA without LLM-based repairs. We kept LLM cost as 0 (as for the GNN), while for users we maintained the same settings described in § 5.1. Overall, the results show that fully replacing users with LLMs substantially reduces the repair cost, but at the expense of repair quality. In almost all configurations, the F1-score drops compared to LARA without LLM-based repairs, with drops ranging from moderate to severe depending on the dataset and the value of θ . Only a few configurations with $\theta = 0.5$ achieve improvements over the original approach, i.e., on FinBench and ICIJ, suggesting LLMs can sometimes effectively repair medium-difficulty violations when the GNN already filters out the easier cases. Hybrid strategies combining LLMs and users consistently outperform fully LLM-based repairs, but are largely dominated by the original fully human-assisted configuration.

5.6 Testing Different Constraint Types

To evaluate the generality of our approach, we conducted an additional study using different classes of constraints. We considered three representative categories: a general denial constraint (DC), a non-key functional dependency (FD), and a key constraint (Key). These constraints were instantiated on the two real-world datasets FAERS and ICIJ. The complete list of constraints is reported in the supplementary material [8]. We reran the full end-to-end repair pipeline while varying both θ and budget. Fig. 4 shows the F1-score on FAERS (the plot for ICIJ shows the same trend, and is reported in [8]). Overall, the results confirm the trends observed in § 5.3, with a plateau reached for higher values of $\theta (> 0.7)$ and without using the full available budget. Fig. 5 instead, breaks down the results for each type of constraint reporting the F1-score for $\theta > 0.5$ and budget values corresponding to the plateau regions observed in Fig. 4. For FAERS, denial and key constraints benefit most from larger budgets, achieving near-perfect repair quality with sufficient resources. FDs show more stable behavior across parameter settings, with moderate F1-scores and limited sensitivity to θ . Higher-quality repairs generally incur higher repair costs, particularly for key constraints at large budgets, reflecting the complexity of resolving identity inconsistencies. For ICIJ, performance is even more stable across parameter configurations. Denial and key constraints achieve very high F1-scores for medium and large budgets, while FDs maintain slightly lower

<p>DC: (a:Age)←[r:HAS_AGE]-(c:Case)-[r1:FALLS_UNDER]→(ag:AgeGroup) WHERE a.value < ag.minAge OR a.value > ag.maxAge</p> <p>Inst.:(a:Age 'value': '11') ← [r:HAS_AGE]-(c:Case 'id': 43956) -[r1:FALLS_UNDER]→(ag:AgeGroup 'ageGroup': 'Adolescent', 'maxAge': 18, 'minAge': 13)</p> <p>ConfGNN = 0.987 Degree = 0.082 PageRank = 0.706</p>	<p>FD: (c1:Country)←[r1:REGISTERED_IN]-(n:Entity)-[r2:ADDRESS]→(a:Address), (a)-[l:LOCATED_IN]→(c:Country) WHERE c1.code ≠ c.code</p> <p>Inst.:(c1:Country 'code': 'BMU;HKG')←[r1:REGISTERED_IN]-(e:Entity 'id': 68202.0,...) (e)-[r2:ADDRESS]→(a:Address 'value': '83 Front Street') -[l:LOCATED_IN]→(c:Country 'code': 'BMU')</p> <p>ConfGNN = 0.345 Degree = 0.8 PageRank = 0.437</p>	<p>Key: (e:Entity)-[r1:JURIS]→(j:Jurisdiction)←[r2:JURIS]-(e1:Entity) WHERE e ≠ e1 AND e.inc_date = e1.inc_date AND e.name = e1.name</p> <p>Inst.:(e:Entity 'id': 81155,'inc_date': '01-03-14','name': 'Salamander Inc',...), (e)-[r1:JURIS]→ (j:Jurisdiction 'code': 'KY', 'description': 'Cayman Islands'), (j)←[r2:JURIS]-(e1:Entity 'id': 78094, 'inc_date': '01-03-14', 'name': 'Salamander Inc',...)</p> <p>ConfGNN = 0.87 Degree = 0.11 PageRank = 0.3</p>
---	---	---

Fig. 6. Three examples of constraints with violations and difficulty estimations.

but highly consistent performance. Unlike FAERS, repair costs remain relatively stable across θ , suggesting that the assignment strategy adapts effectively to different constraint semantics without substantial additional effort. Overall, key constraints appear harder to solve, since high F1-scores are reached only with the full user budget ($\theta = 1$), and they are also the most expensive. FDs require smaller budgets but converge to lower accuracy, suggesting that GNNs may underestimate their difficulty, leading to lower-quality repairs. Overall, these results indicate that the proposed framework is not limited to a single type of integrity constraint. Although the experiments use GDCs as a unifying formalism, the assignment and repair pipeline remains robust across keys, FDs, and general DCs.

5.7 Illustrative Examples

Fig. 6 illustrates the practical utility of our framework with three representative violations, one per constraint type, together with the difficulty scores assigned by our method (ConfGNN) and by Degree and PageRank (3 more examples in the supplementary material [8]). For readability, constraints are shown as Cypher-like queries.

Example 5.1. [DC (FAERS)] A case whose recorded age (11) falls outside the range of its assigned age group (*Adolescent*, 13–18). Repairing this violation requires knowing either the true age or the correct age group of the specific case, information that cannot be reliably inferred from the graph. Accordingly, our method estimates a high difficulty (0.987) and, for $\theta = 0.8$, routes the violation to a user. The degree-based estimator assigns a very low score (0.082) and would incorrectly delegate the repair to the GNN. The Pref-* repair approaches in this case would fail, because we cannot solve specific violations with a globally tuned algorithm.

Example 5.2 (FD (ICI)). An entity has a registered country (BMU; HKG) different from the country of its registered address (BMU). Such inconsistencies over country codes are well-studied in the database literature and can typically be repaired with high reliability [20, 22]. Our method estimates a low difficulty (0.345), correctly routing the violation to the GNN for $\theta = 0.5$. PageRank yields a comparable score (0.437), but the degree-based estimator assigns a much higher value (0.8), unnecessarily favoring human repair. Both the Pref-* methods and LARA would choose the correct repair.

Example 5.3 (Key (ICI)). Two entities are sharing the same name and incorporation date in the same jurisdiction. Key-constraint violations are typically hard, since deciding whether the records refer to the same real-world object or to distinct objects with identical attributes generally

requires external knowledge (e.g., access to registries). Consistent with this, our method assigns a high difficulty (0.87), while PageRank and degree return much lower scores (0.3 and 0.11). Pref-* strategies could be tuned to always delete one of the two conflicting records, but such a fixed policy cannot determine whether the apparent duplicate is actually erroneous.

These examples illustrate that topological estimators alone are insufficient for capturing repair difficulty: violations that appear structurally simple may require substantial semantic knowledge, while others can be repaired automatically despite involving larger graph structures. They also highlight the limitations of preferred-repair approaches, which rely on a fixed strategy that cannot adapt to the semantics of individual violations. Our framework adapts automatically through the difficulty estimator and threshold θ , delegating to users only the cases where automatic repair is unreliable.

6 Conclusions

In this paper, we address property-graph repair by combining automated inference with human expertise. We introduce a GNN-based difficulty estimator that fuses topological and semantic cues to predict whether a violation can be fixed automatically; this feeds a user model (skill, capacity, cost) and drives LARA, a difficulty- and capacity-aware assignment algorithm that batches violations and allocates them to maximize expected repair quality. Across datasets, our approach outperforms fully interactive and fully automated baselines in both accuracy and cost-effectiveness. Future work includes learning/optimizing θ from dataset and budget signals, extending the GNN to support more expressive repairs, and enabling human collaboration including, e.g., knowledge sharing between users and partial, system-guided suggestions. Moreover, our settings use a one-user-per-violation pool size as a controlled setup for fair comparison. Studying dynamic settings with substantially smaller user pools, user arrival over time, or repeated assignment rounds is an important direction for future work.

Acknowledgments

The work was partially funded by the Data4Health ANR/CPJ Lyon 1 grant. Lakshmanan's research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (Grant Number RGPIN-2020-05408).

References

- [1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. 2021. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–38.
- [2] Charu C. Aggarwal and B. Haixun Wang (Eds.). 2010. *Managing and Mining Graph Data*. Advances in Database Systems, Vol. 40. Springer.
- [3] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29, 3 (2015), 626–688.
- [4] Renzo Angles. 2018. The Property Graph Database Model. In *AMW (CEUR Workshop Proceedings, Vol. 2100)*. CEUR-WS.org.
- [5] Renzo Angles, János Benjamin Antal, Alex Averbuch, Altan Birlir, Peter Boncz, Márton Búr, Orri Erling, Andrey Gubichev, Vlad Haprian, Moritz Kaufmann, Josep Lluís Larriba Pey, Norbert Martínez, József Marton, Marcus Paradies, Minh-Duc Pham, Arnau Prat-Pérez, David Püroja, Mirko Spasić, Benjamin A. Steer, Dávid Szakállas, Gábor Szárnyas, Jack Waudby, Mingxi Wu, and Yuchen Zhang. 2024. The LDBC Social Network Benchmark. arXiv:2001.02299 [cs.DB] <https://arxiv.org/abs/2001.02299>
- [6] 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. doi:10.1145/3589778

- [7] Anon. 2026. *Code repository*. Retrieved January 16, 2026 from <https://anonymous.4open.science/r/GNN-Based-Interactive-Property-Graph-Repairs-2BFA>
- [8] Anon. 2026. *Supplementary Material*. Retrieved January 16, 2026 from https://anonymous.4open.science/r/GNN-Based-Interactive-Property-Graph-Repairs-2BFA/supplementary_material.pdf
- [9] Abdallah Arioua and Angela Bonifati. 2018. User-guided repairing of inconsistent knowledge bases. In *EDBT: Extending Database Technology*. OpenProceedings.org, 133–144.
- [10] Sambaran Bandyopadhyay, Lokesh N, Saley Vishal Vivek, and M Narasimha Murty. 2020. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the 13th international conference on web search and data mining*. 25–33.
- [11] Senjuti Basu Roy, Ioanna Lykourantzou, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. 2015. Task assignment optimization in knowledge-intensive crowdsourcing. *The VLDB Journal* 24, 4 (2015), 467–491.
- [12] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. Querying graphs. *Synthesis Lectures on Data Management* 10, 3 (2018), 1–184.
- [13] Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs*. Morgan & Claypool Publishers. doi:10.2200/S00873ED1V01Y201808DTM051
- [14] Yurong Cheng, Lei Chen, Ye Yuan, and Guoren Wang. 2018. Rule-Based Graph Repairing: Semantic and Efficient Repairing Methods. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 773–784. doi:10.1109/ICDE.2018.00075
- [15] Yurong Cheng, Lei Chen, Ye Yuan, Guoren Wang, Boyang Li, and Fusheng Jin. 2022. Strict and Flexible Rule-Based Graph Repairing. *IEEE Transactions on Knowledge and Data Engineering* 34, 7 (2022), 3521–3535. doi:10.1109/TKDE.2020.3019817
- [16] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Information and Computation* 197, 1-2 (2005), 90–121.
- [17] Djellal Eddine Difallah, Gianluca Demartini, and Philippe Cudré-Mauroux. 2013. Pick-a-crowd: tell me what you like, and i'll tell you what to do. In *Proceedings of the 22nd international conference on World Wide Web*. 367–374.
- [18] Xin Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmman, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (Eds.). ACM, 601–610. doi:10.1145/2623330.2623623
- [19] Xiangyu Dong, Xingyi Zhang, Yanni Sun, Lei Chen, Mingxuan Yuan, and Sibao Wang. 2025. SmoothGNN: Smoothing-aware GNN for Unsupervised Node Anomaly Detection. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) (WWW '25). Association for Computing Machinery, New York, NY, USA, 1225–1236. doi:10.1145/3696410.3714615
- [20] Abdulhakim Qahtan et al. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.
- [21] Benjamin Douglas et al. 2023. Data quality in online human-subjects research: Comparisons between MTurk, Prolific, CloudResearch, Qualtrics, and SONA. *Plos one* 18, 3 (2023).
- [22] Ester Livshits et al. 2020. Computing optimal repairs for functional dependencies. *ACM TODS* 45, 1 (2020), 1–46.
- [23] Ouhaio Chen et al. 2023. A Cognitive Load Theory Approach to Defining and Measuring Task Complexity Through Element Interactivity. *Educational Psych. Rev.* 35, 2 (2023).
- [24] Vahan Yoghoudjian et al. 2021. Scalability of Network Visualisation from a Cognitive Load Perspective. *IEEE Trans. Visualization & Computer Graphics* 27, 2 (2021), 1677–1687.
- [25] Xu Chu et al. 2016. Data cleaning: Overview and emerging challenges. In *SIGMOD*. 2201–2206.
- [26] Wenfei Fan and Ping Lu. 2017. Dependencies for Graphs. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) (PODS '17). Association for Computing Machinery, New York, NY, USA, 403–416. doi:10.1145/3034786.3056114
- [27] Wenfei Fan and Ping Lu. 2019. Dependencies for graphs. *ACM Transactions on Database Systems (TODS)* 44, 2 (2019), 1–40.
- [28] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 1843–1857. doi:10.1145/2882903.2915232
- [29] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 1843–1857. doi:10.1145/2882903.2915232
- [30] Marshall L Fisher. 1981. The Lagrangian relaxation method for solving integer programming problems. *Management science* 27, 1 (1981), 1–18.

- [31] Danula Hettiachchi, Vassilis Kostakos, and Jorge Goncalves. 2022. A survey on task assignment in crowdsourcing. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–35.
- [32] Kai Huang, Huey Eng Chua, Sourav S Bhowmick, Byron Choi, and Shuigeng Zhou. 2019. CATAPULT: data-driven selection of canned patterns for efficient visual graph query formulation. In *Proceedings of the 2019 International Conference on Management of Data*. 900–917.
- [33] Ling Huang, Ye Zhu, Yuefang Gao, Tuo Liu, Chao Chang, Caixing Liu, Yong Tang, and Chang-Dong Wang. 2021. Hybrid-order anomaly detection on attributed networks. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2021), 12249–12263.
- [34] Yihong Huang, Liping Wang, Fan Zhang, and Xuemin Lin. 2023. Unsupervised graph outlier detection: Problem revisit, new insight, and superior method. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2565–2578.
- [35] Rishabh K Iyer and Jeff A Bilmes. 2013. Submodular optimization with submodular cover and submodular knapsack constraints. *Advances in neural information processing systems* 26 (2013).
- [36] Paul Juillard, Angela Bonifati, and Andrea Mauri. 2024. Interactive Graph Repairs for Neighborhood Constraints. In *Proceedings 27th International Conference on Extending Database Technology (EDBT 2024) Paestum, Italy, March 25 - March 28*. OpenProceedings.org, 2:175–2:187.
- [37] Guoliang Li, Yudian Zheng, Ju Fan, Jiannan Wang, and Reynold Cheng. 2017. Crowdsourced data management: Overview and challenges. In *Proceedings of the 2017 ACM international conference on Management of Data*. 1711–1716.
- [38] Jundong Li, Harsh Dani, Xia Hu, and Huan Liu. 2017. Radar: Residual analysis for anomaly detection in attributed networks.. In *IJCAI*, Vol. 17. 2152–2158.
- [39] Shuming Liang, Yu Ding, Zhidong Li, Bin Liang, Siqi Zhang, Yang Wang, and Fang Chen. 2025. Can gnns learn link heuristics? A concise review and evaluation of link prediction methods. *IEEE Transactions on Big Data* (2025).
- [40] Peng Lin, Qi Song, Yinghui Wu, and Jiaxing Pi. 2020. Repairing entities using star constraints in multi-relational graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 229–240.
- [41] Peng Lin, Qi Song, Yinghui Wu, and Jiaxing Pi. 2020. Repairing Entities using Star Constraints in Multi-relational Graphs. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 229–240. doi:10.1109/ICDE48307.2020.00027
- [42] 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. doi:10.1145/3442381.3450040
- [43] Steven Maere, Karel Heymans, and Martin Kuiper. 2005. BiNGO: a Cytoscape plugin to assess overrepresentation of Gene Ontology categories in Biological Networks. *Bioinform.* 21, 16 (2005), 3448–3449. doi:10.1093/BIOINFORMATICS/BTI551
- [44] Mohammad Mahdavi and Zia Wasch Abedjan. 2021. Semi-Supervised Data Cleaning with Raha and Baran.. In *CIDR*.
- [45] Meta AI. 2024. Meta Llama 3. <https://huggingface.co/meta-llama/Meta-Llama-3-8B>. Model card and weights.
- [46] Inc. Neo4j. 2021. ICJ offshore leaks. Retrieved October 2, 2024 from <https://github.com/neo4j-graph-examples/icj-paradise-papers>
- [47] Inc. Neo4j. 2022. FDA Adverse Event Reporting System. Retrieved May 16, 2025 from <https://github.com/neo4j-graph-examples/healthcare-analytics>
- [48] Inc. Neo4j. 2022. Star Wars. Retrieved March 26, 2024 from <https://github.com/neo4j-graph-examples/star-wars>
- [49] Inc. Neo4j. 2024. Neo4j Graph Database. Retrieved October 18, 2024 from <https://neo4j.com/product/neo4j-graph-database/>
- [50] M. Tamer Özsu. 2016. A survey of RDF data management systems. *Frontiers Comput. Sci.* 10, 3 (2016), 418–432. doi:10.1007/S11704-016-5554-Y
- [51] Amedeo Pachera, Angela Bonifati, and Andrea Mauri. 2024. Towards User-Centric Graph Repairs. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. IEEE Computer Society, Los Alamitos, CA, USA, 375–376. doi:10.1109/ICDEW61823.2024.00057
- [52] Amedeo Pachera, Angela Bonifati, and Andrea Mauri. 2025. User-Centric Property Graph Repairs. *Proc. ACM Manag. Data* 3, 1, Article 85, 27 pages. doi:10.1145/3709735
- [53] Shipeng Qi, Heng Lin, Zhihui Guo, Gábor Szárnyas, Bing Tong, Yan Zhou, Bin Yang, Jiansong Zhang, Zheng Wang, Youren Shen, Changyuan Wang, Parviz Peiravi, Henry Gabb, and Ben Steer. 2023. The LDBC Financial Benchmark. arXiv:2306.15975 [cs.DB] <https://arxiv.org/abs/2306.15975>
- [54] Habibur Rahman, Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. 2015. Task assignment optimization in collaborative crowdsourcing. In *2015 IEEE International Conference on Data Mining*. IEEE, 949–954.
- [55] Habibur Rahman, Saravanan Thirumuruganathan, Senjuti Basu Roy, Sihem Amer-Yahia, and Gautam Das. 2015. Worker skill estimation in team-based tasks. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1142–1153.
- [56] Mark D Reckase. 2006. 18 Multidimensional item response theory. *Handbook of statistics* 26 (2006), 607–642.

- [57] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).
- [58] Amit Roy, Juan Shu, Jia Li, Carl Yang, Olivier Elshocht, Jeroen Smeets, and Pan Li. 2024. Gad-nr: Graph anomaly detection via neighborhood reconstruction. In *Proceedings of the 17th ACM international conference on web search and data mining*. 576–585.
- [59] 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. doi:10.1007/s00778-019-00548-x
- [60] 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. doi:10.1145/3434642
- [61] 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.
- [62] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. 2012. Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence* 64, 2 (2012), 209–246.
- [63] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The world wide web conference. 2022–2032*.
- [64] Jef Wijsen. 2005. Database repairing using updates. *ACM Transactions on Database Systems (TODS)* 30, 3 (2005), 722–768.
- [65] Hanlu Wu, Tengfei Ma, Lingfei Wu, Fangli Xu, and Shouling Ji. 2021. Exploiting heterogeneous graph neural networks with latent worker/task correlation information for label aggregation in crowdsourcing. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 16, 2 (2021), 1–18.
- [66] 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.
- [67] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo c, Jens Lehmann, and Sören Auer. 2013. Quality Assessment Methodologies for Linked Open Data. *Semantic Web Journal* (11 2013).
- [68] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. 2017. Truth inference in crowdsourcing: is the problem solved? *Proc. VLDB Endow.* 10, 5 (Jan. 2017), 541–552. doi:10.14778/3055540.3055547

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026