Imputing Duplicate, Conflict, Stale and Missing Data in Graphs

Wenfei Fan[1,2,3], Ping Lu[2], Kehan Pang[2], Chao Tian[2]
[1]University of Edinburgh [2]Beihang University [3]Shenzhen Institute of Computing Sciences
wenfei@inf.ed.ac.uk,{luping@,pangkh@act.,tianchao@}buaa.edu.cn

ABSTRACT

This paper develops Hercules, a system for entity resolution (ER), conflict resolution (CR), timeliness deduction (TD) and missing value/link imputation (MI) in graphs. We show that the four issues interact with each other; hence, to improve the overall quality of graphs, we need to conduct the four in the same process. We propose GCR$^+$s, a class of graph cleaning rules that support not only predicates for ER and CR, but also temporal orders to deduce timeliness and data extraction to impute missing data. We show that while the implication and satisfiability problems are intractable for GCR$^+$s, it is in PTIME to detect and correct errors with GCR$^+$s. We train a ranking model to predict the temporal orders on attributes, and embed it as a predicate of GCR$^+$s. We present a method for conducting ER, CR, TD and MI in the same process by chasing with GCR$^+$s, with the Church-Rosser property under certain conditions. We also develop algorithms of Hercules for discovering GCR$^+$s and detecting errors with the GCR$^+$s. Using real-life and synthetic graphs, we empirically verify that Hercules is 53% more accurate than the state-of-the-art graph cleaning systems, and is comparable in efficiency.

1 INTRODUCTION

Previous work on the quality of graph-structured data has mostly focused on (a) *entity resolution* (ER), to identify whether two vertices refer to the same real-world entity [23, 28, 30, 31, 55, 58, 61, 64, 92], and (b) *conflict resolution* (CR), to resolve (semantic) inconsistencies between attributes of the entities [23, 28, 30, 31, 35, 45, 80, 87].

Besides ER and CR, there are two other critical issues of data quality. One is *timeliness* (*a.k.a.* data currency), for how up-to-date the information is. The other is *information completeness*, for the availability of data on-hand for analytics. In the real world, data easily becomes obsolete, and stale data is inaccurate or incorrect. It is well known that data-driven decisions based on outdated data can be worse than making decisions with no data [59]. Unfortunately, "82% of companies are making decisions based on stale information", and 85% of the companies witness that "this stale data is leading to incorrect decisions and lost revenue" [12]. Equally damaging is missing data. As revealed by a poll on clinical data management challenges in 2018, 77% of the participants rated missing patient data as a critical problem [75], which may result in incorrect diagnosis [19, 46, 49, 53]. Missing data also poses "the most pervasive challenge in network-based crime analysis" [11], with serious consequences [11, 37, 67]. In reality, individuals attempt to deceive investigators to protect themselves or their associates; just a few missing actors (vertices) and ties (edges) can change the fundamental structure of the network; indeed, 2% of missing data can lead to difference of up to 80% from a real criminal network [21, 37].

The need is evident for (c) *timeliness deduction* (TD), to deduce temporal orders on attribute values, and (d) *missing data imputation* (MI), to fill in missing values/links. Moreover, TD, MI, ER and CR interact with each other: ER and CR help TD and MI, and vice versa.

Example 1: Meta [62] changed its name from "Facebook, Inc." to "Meta Platforms, Inc." in 2021. This yields company entities v_1 and v_2 with distinct titles in knowledge graphs, *e.g.,* DBpedia [3] and Yago [77]. Worse yet, some other information of v_1 is either outdated or missing, *e.g.,* revenue, research labs, type. Similarly, there also exist two product entities u_1 and u_2 for the online social networking service named "Facebook". They carry different values for current status, languages and platforms, but have very few timestamps available. These call for TD and MI in graphs.

TD and MI interact with ER and CR, and can help each other. For example, if TD finds that product u_2 has newer values for languages and platforms than u_1, and if the status of u_1 is active, then u_2 should also be active, *i.e.,* the erroneous status of u_2 can be fixed (CR). After the TD step, companies v_1 and v_2 can be identified (ER) since they have the same CEO, country and in particular, they own active products u_1 and u_2 with the same name. This in turn helps us fill in the missing value of type of v_1, by copying from v_2 (MI; assuming that v_2 is in a more reliable knowledge graph). Moreover, a missing link is restored between v_1 and a research lab Facebook AI created by the same CEO, by copying the edge from reliable v_2 to the lab. □

The example indicates that to improve the overall data quality, we need to conduct ER, CR, TD and MI in the same process. However, already hard for relational data, these issues are more challenging for graphs. For instance, missing values/links are more common in graphs since unlike relational data, real-life graphs typically do not have a schema. We are aware of no prior graph cleaning systems that support TD and MI, let alone their interactions with ER and CR.

To support these, several questions need to be answered. How can we effectively deduce the timeliness of properties of entities (vertices), and impute missing values and links in a graph? Can we make use of both machine learning (ML) and logic rules in the process? How can we conduct ER, CR, TD and MI in the same process and leverage their interactions? Would it be prohibitively expensive to conduct the four tasks altogether? What changes have to be made to existing methods for rule discovery, error detection and error correction when TD and MI are taken into account?

Contributions & Organization. In an effort to answer these questions, we develop Hercules, a system for cleaning graphs. Unlike prior graph cleaning systems, Hercules has several unique features.

(1) GCR$^+$s*: Rules for ER, CR, TD and MI* (Section 2). Hercules proposes GCR$^+$s, a class of rules for specifying ER, CR, TD and MI in graphs. GCR$^+$s are an extension of the graph cleaning rules (GCRs) of [23] for ER and CR, by supporting (a) temporal orders on attribute values and an ML model for temporal ranking, to deduce timeliness, and (b) data extraction from, *e.g.,* knowledge graphs, to fill in missing values and links. They subsume GCRs as a special case, and hence support ER and CR. Like GCRs and graph association rules (GARs) [28], GCR$^+$s may embed ML models as predicates, and unify

ML and logic deduction in the same process of graph cleaning.

(2) Complexity bounds (Section 3). We show that it takes polynomial time (PTIME) to check whether a graph satisfies a set of GCR$^+$s (*validation*). Moreover, error detection and correction with GCR$^+$s remain in PTIME despite the addition of TD and MI, as opposed to their intractability for graph entity rules (GEDs) [31] and graph association rules (GARs) [28]. However, it is NP-complete to decide whether there exists a graph that satisfies a given set of GCR$^+$s (*satisfiability*), and coNP-complete to decide whether a set of GCR$^+$s entails another GCR$^+$ (*implication*). This said, we show that these problems are already intractable for mild extensions of GCRs that are a must for supporting CR and TD. Hence GCR$^+$s strike a balance between the complexity and necessary expressive power.

(3) A temporal ranking model (Section 4). We train the first pairwise ranking model for determining temporal orders on attributes. The model departs from prior models for temporal graph completion and time series prediction. In particular, it gathers evolving temporal data by graph partitioning, embeds correlated information of neighbors and attribute changing behavior, deduces the confidence of temporal ordering, and employs semi-supervised training to progressively expand training data with deduced orders.

(4) Hercules*: A graph cleaning system* (Section 5). Hercules conducts ER, CR, TD and MI in the same process, and leverages their interaction to improve the overall quality of the graphs. We propose a parallel algorithm for Hercules to carry these out by chasing with GCR$^+$s, and accumulating and referencing ground truth. The chase is guaranteed to be Church-Rosser under certain conditions, *i.e.,* it converges at a unique result. We also develop algorithms for discovering GCR$^+$s from real-life graphs, and detecting errors with the mined GCR$^+$s. All these algorithms are parallelly scalable [54], *i.e.,* they guarantee to take less time when given more processors.

(5) Experimental study (Section 6). Using real-life and synthetic graphs, we empirically verify the following. (a) In error detection (resp. correction), Hercules is 54% (resp. 53%) more accurate than the rule-based methods with GCRs. On average, it is 60%, 51%, 67% and 37% more accurate than ML models for CR, ER, TD and MI, respectively. (b) Our temporal ranking model is 17% more accurate than prior ML models. (c) While Hercules takes longer for discovering GCR$^+$s than mining much simpler GCRs [23], it is 16.5× faster than mining GARs [24]. Moreover, with GCR$^+$s, it performs comparably in efficiency to methods with GCRs for error detection and correction; it is also 39× faster than with GARs. (d) Its algorithms for rule mining, error detection and correction are 3×, 3.2× and 3× faster, respectively, when 16 processors are used instead of 4.

Novelty. This work substantially extends the GCRs of [23] with the following: (1) GCR$^+$s with temporal orders for TD and data extraction for MI, (2) complexity bounds for GCR$^+$s, (3) a novel temporal ranking model for graph properties, and (4) the first graph cleaning system that supports ER, CR, TD and MI altogether, beyond [23] for ER and CR only, while retaining PTIME error detection and correction. Its algorithms enhance those of [23] to support TD and MI.

We discuss related work in Section 7 and future work in Section 8. For the lack of space, we provide in [4] the proofs of the results of the paper, and the algorithms for rule mining and error detection.

2 GRAPH CLEANING RULES

In this section we introduce GCR$^+$s. We first present basic notations and define temporal orders (Section 2.1). We then define GCR$^+$s and show how the rules support ER, CR, TD and MI (Section 2.2).

2.1 Temporal Graphs and Temporal Orders

Assume three countably infinite sets of symbols, denoted by Λ, Υ and Ω, for labels, attributes and timestamps, respectively. Here Ω is linearly ordered *w.r.t.* discrete orders $\preceq$ and $\prec$ (see below).

Preliminaries. We consider directed labeled graphs $G = (V, E, L, F_A)$, where (a) V is a finite set of vertices; (b) $E \subseteq V \times \Lambda \times V$ is a finite set of edges, in which $e = (v, l, v')$ denotes an edge from vertex v to v' that is labeled with $l \in \Lambda$; (c) each vertex $v \in V$ has label $L(v)$ from Λ; and (d) each vertex $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity, where $A_i \in \Upsilon$ and a_i is a constant, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$, representing properties. Different vertices may carry different attributes, which are not constrained by a schema like relational databases.

We assume the existence of a special attribute id at each vertex v, denoting its vertex identity, such that for any two vertices v and v' in G, if $v.\mathrm{id} = v'.\mathrm{id}$, then v and v' refer to the same entity.

Paths. A *path ρ from a vertex v_0* in G is a list $(v_0, l_0, v_1, \ldots, v_{n-1}, l_{n-1}, v_n)$ such that (v_{i-1}, l_{i-1}, v_i) is an edge in G ($i \in [1, n]$). We consider *simple* paths on which each vertex appears at most once. A vertex v is called a *child* of u if there exists an edge (u, l, v) in E.

Pattern matching. We now review star-shaped dual patterns [23].

Star patterns. A *star pattern* is $Q[x_0, \bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of pattern vertices (resp. edges) as defined above; (2) L_Q assigns a label of Λ to each vertex in V_Q; (3) $\bar{x}$ is a list of distinct variables, and μ is a bijective mapping from $\bar{x}$ to the vertices of Q; (4) x_0 is a designated variable in $\bar{x}$, referred to as the *center* of Q; and (5) for each $z \in \bar{x}$, there exists a single path from x_0 to z, and z has at most one child, except x_0. For variables $z \in \bar{x}$, we use $\mu(z)$ and z interchangeably if it is clear in the context.

Intuitively, $Q[x_0, \bar{x}]$ has the shape of a star with center x_0. The center x_0 denotes an entity of interest, and it links to a set of characteristic features (*leaf* vertices) without children, each via a path.

Dual patterns. A *dual pattern* is $\boldsymbol{Q}[x_0, y_0] = \langle Q_x[x_0, \bar{x}], Q_y[y_0, \bar{y}] \rangle$, where $Q_x[x_0, \bar{x}]$ and $Q_y[y_0, \bar{y}]$ are disjoint star patterns. *i.e.,* Q_x and Q_y have no common vertices. Intuitively, $\boldsymbol{Q}$ is a pair of patterns to represent two entities x_0 and y_0 with (possibly) heterogeneous structures in a schemaless graph. The star patterns specify features (leaf vertices) for pairwise comparison between x_0 and y_0.

Example 2: Figure 1 depicts a graph G that includes the product entities u_1, u_2 and companies v_1, v_2 (see Example 1). Some vertex labels are given in parentheses. For entities of general type, *i.e.,* object, we omit their vertex labels and show the assigned values (val) directly, *e.g.,* Android. Moreover, four dual patterns are shown in Fig. 2, in which vertex labels for general type entities are also omitted. □

Matches. A *match of a star pattern Q in a graph G* is a homomorphic mapping h from the pattern vertices in Q to G such that (a) for each vertex $u \in V_Q$, $L_Q(u) = L(h(u))$, and (b) for each pattern edge (u, l, u') in Q, $(h(u), l, h(u'))$ is an edge in graph G.

Similarly, a match of a dual pattern $\boldsymbol{Q}[x_0, y_0] = \langle Q_x, Q_y \rangle$ in

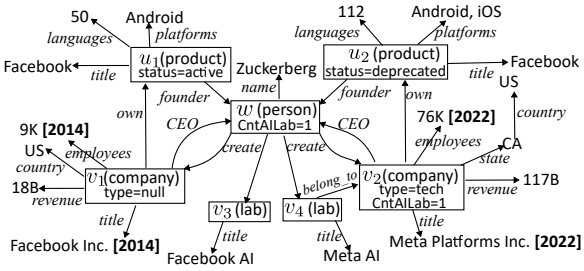


Figure 1: A temporal graph (G, T)

graph G is a homomorphic mapping h from the pattern vertices in $V_{Q_x} \cup V_{Q_y}$ to V of G. Note that $h(x_0)$ and $h(y_0)$ are possibly disconnected vertices in G. These allow us to compare entities that are not connected and may have different topological structures.

It has been shown that it is tractable to check the existence of a match of a star-shaped dual pattern [23] . In contrast, graph pattern matching is NP-complete for generic graph patterns (cf. [44]).

In the sequel, we refer to a star-shaped dual pattern simply as a *dual pattern* or a *pattern* when it is clear in the context.

Temporal orders. We extend graph G with temporal orders.

Temporal graphs. A *temporal graph* is a pair (G, T), where G is a graph as above, and T is a partial function such that for a vertex v and its attribute A in G, $T(v.A)$ is a timestamp in Ω if it is defined.

Intuitively, the timestamp indicates that at the time $T(v.A)$, the A-attribute of vertex v is correct and up-to-date. Note that $T(v.A)$ and $T(v.B)$ may bear different timestamps for different attributes A and B of v, since $v.A$ and $v.B$ may come from different data sources.

To simplify the presentation, we associate *w.l.o.g.* timestamps only with attributes. We may designate an attribute of v to denote the timestamp of vertex v. For an edge $e = (v, l, v')$, we may add a dummy vertex v_e labeled l, replace e with two edges that link v to v_e and v_e to v', and associate the timestamp of edge e with v_e.

Graphs with temporal orders. To deduce the timeliness of attributes, we define a *graph with temporal orders* as $(G, \preceq, \prec)$, where G abbreviates (G, T), $\preceq$ is a partial discrete order such that if $v.A \preceq v'.B$, written as $(v.A, v'.B) \in \preceq$, then $v'B$ is at least as current (up-to-date) as $v.A$; similarly, $v.A \prec v'.B$ if $v'.B$ is more current than $v.A$.

In the sequel, we consider graphs with temporal orders, simply referred to as graphs if it is clear from the context.

In particular, if timestamps $T(v.A)$ and $T(v'.B)$ are both defined (*i.e.,* available) and if $T(v.A) < T(v'.B)$, then $v.A \prec v'.B$, *i.e.,* $v'.B$ is confirmed at a later time and is thus considered more current than $v.A$; similarly, if $T(v.A) \leq T(v'.B)$, then $v.A \preceq v'.B$.

Example 3: Graph G in Fig. 1 is a temporal graph in which timestamps are marked in bold. Since $2014 < 2022$, the number ($76K$) of employees in company v_2 is more current than that ($9K$) in v_1. □

2.2 GCR⁺s for ER, CR, TD and MI

We next extend graph cleaning rules (GCRs) of [23] to GCR⁺s.

Predicates. A *predicate* of a dual pattern $Q[x_0, y_0]$ is defined as

$$p \quad ::= \quad x.A \oplus y.B \mid z.A \oplus c \mid \mathcal{M}(x.\bar{A}, y.\bar{B}) \mid l(z_1, z_2) \mid x.A \preceq y.B \mid x.A \prec y.B,$$

where $\oplus$ is one of $=, \neq, <, \leq, >, \geq$; x, y, z, z_1, z_2 are vertices in the dual pattern Q, *i.e.,* $x, y, z, z_1, z_2 \in \bar{x} \cup \bar{y}$; c is a constant; A and B are attributes in Υ; l is a label in Λ, $l(z_1, z_2)$ indicates an edge from z_1 to z_2 labeled with l; and $x.\bar{A}$ is a list of attributes at x; similarly for $y.\bar{B}$. Here $\mathcal{M}(x.\bar{A}, y.\bar{B})$ is an ML classifier that returns true if and only if (iff) $\mathcal{M}$ predicts true at $(x.\bar{A}, y.\bar{B})$.

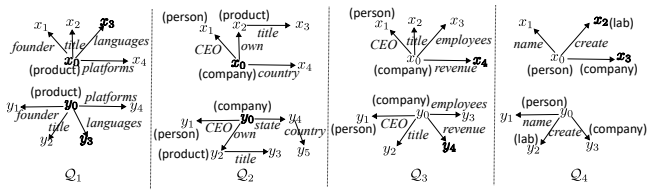


Figure 2: Example dual patterns

We refer to (a) $l(x, y)$ as *edge predicate*, (b) $x.A \preceq y.B$, $x.A \prec y.B$ as *temporal predicate*, and (c) $\mathcal{M}(x.\bar{A}, y.\bar{B})$ as *ML predicate*.

In principle, $\mathcal{M}$ can be any ML model that returns a Boolean value (*e.g.,* $\mathcal{M}_{\geq\theta}$ for a predefined bound θ). To simplify the discussion, we consider only ML models for (a) similarity checking, *e.g.,* pre-trained BERT model [18], (b) entity resolution, *e.g.,* SDEA [91], and (c) temporal ranking $\mathcal{M}_{\mathsf{rank}}(x.A, y.B)$, which returns true iff $\mathcal{M}_{\mathsf{rank}}$ predicts that $x.A \preceq y.B$, *i.e.,* $y.B$ is at least as current as $x.A$; we will train $\mathcal{M}_{\mathsf{rank}}$ in Section 4.

Note that x and y are variables for vertices in Q_x and Q_y, respectively, to pairwisely compare features of centers x_0 and y_0.

Rules. A graph cleaning rule (GCR⁺) φ has the following form:

$$Q[x_0, y_0](X \rightarrow p_0),$$

where $Q[x_0, y_0]$ is a dual pattern, X is a conjunction of predicates of Q, and p_0 is a predicate of Q. Moreover, in precondition X, (1) $x.A \oplus y.B$, $l(x, y)$, $x.A \preceq y.B$, $x.A \prec y.B$ and $\mathcal{M}(x.\bar{A}, y.\bar{B})$ are defined on two leaves x and y in Q_x and Q_y of Q, respectively; and (2) each leaf may carry at most one such predicate (but multiple $z.A \oplus c$).

We refer to Q and $X \rightarrow p_0$ as the *pattern* and *dependency* of φ, and X and p_0 as *precondition* and *consequence* of φ, respectively.

Intuitively, restrictions (1) and (2) above are to strike a balance between expressive power and efficiency (see Section 3). Note that (a) the consequence p_0 is *not* restricted: it can be defined on arbitrary vertices in Q from the same star or different stars. As will be seen shortly, we can leverage $p_0 = l(z_1, z_2)$ on vertices z_1 and z_2 in the same star to fill in missing links. (b) We separate Q and X in order to (i) visualize the topological features of x_0 and y_0 by Q, and (ii) speed up evaluation by leveraging the locality of pattern matching.

Example 4: Using the dual patterns in Fig. 2, below we exemplify some GCR⁺s to improve the data quality as hinted in Example 1. Here vertices involved in the consequences are marked in bold.

(1) $\varphi_1 = Q_1[x_0, y_0](X_1 \wedge x_3.\mathsf{val} \prec y_3.\mathsf{val} \rightarrow x_3.\mathsf{val} \prec y_3.\mathsf{val})$, where X_1 is $x_1.\mathsf{id} = y_1.\mathsf{id} \wedge x_2.\mathsf{val} = y_2.\mathsf{val}$. This GCR⁺ can deduce temporal orders on the number of languages supported by an online service (product), *i.e.,* if two services are created by the same person and have the same title (specified in X_1), then the larger number of the supported languages is more current (for TD).

(2) $\varphi_2 = Q_1[x_0, y_0](X_1 \wedge x_3.\mathsf{val} \prec y_3.\mathsf{val} \wedge \mathcal{M}_{\mathsf{rank}}(x_4.\mathsf{val}, y_4.\mathsf{val}) \wedge x_0.\mathsf{status} = \mathsf{active} \rightarrow y_0.\mathsf{status} = \mathsf{active})$. That is, if service y_0 has newer values for languages (by $x_3.\mathsf{val} \prec y_3.\mathsf{val}$) and platforms (by ranking model $\mathcal{M}_{\mathsf{rank}}$) than x_0, and if x_0 has active status, then with the same condition X_1 as in (1), y_0 should also be active (for CR). Here ML model $\mathcal{M}_{\mathsf{rank}}$ is treated as a predicate in the precondition.

(3) $\varphi_3 = Q_2[x_0, y_0](x_1.\mathsf{id} = y_1.\mathsf{id} \wedge x_2.\mathsf{status} = \mathsf{active} \wedge y_2.\mathsf{status} = \mathsf{active} \wedge x_3.\mathsf{val} = y_3.\mathsf{val} \wedge x_4.\mathsf{val} = y_5.\mathsf{val} \rightarrow x_0.\mathsf{id} = y_0.\mathsf{id})$. This

GCR$^+$ identifies two companies x_0 and y_0 (for ER), if they have the same CEO and country of location, and moreover, if they own *active* products of the same title. Note that the country property is fetched along different paths in the two stars of Q_2.

(4) $\varphi_4 = Q_3[x_0, y_0](x_0.\text{type} = \text{tech} \wedge y_0.\text{type} = \text{tech} \wedge x_1.\text{id} = y_1.\text{id} \wedge \mathcal{M}_s(x_2.\text{val}, y_2.\text{val}) \wedge x_3.\text{val} \prec y_3.\text{val} \rightarrow x_4.\text{val} \prec y_4.\text{val})$. This rule deduces that company y_0 has a newer revenue than x_0 (for TD) if y_0's number of employees is more current and both are tech companies with the same CEO and similar titles (by model $\mathcal{M}_s$).

(5) $\varphi_5 = Q_4[x_0, y_0](x_1.\text{val} = y_1.\text{val} \wedge \mathcal{M}_e(x_2, y_2) \wedge x_3.\text{id} = y_3.\text{id} \wedge y_0.\text{CntAILab} = 1 \wedge y_3.\text{CntAILab} = 1 \rightarrow \text{belong\_to}(x_2, x_3))$. It states that if companies x_3, y_3 and AI labs x_2, y_2 are created by persons x_0 and y_0 of the same name, x_3 and y_3 are the same entity, x_2 and y_2 are identified by ER model $\mathcal{M}_e$, and if y_0 and y_3 pertain to a single AI lab, then x_2 is the unique AI lab belonging to x_3 (for MI). □

Remark. The GCRs of [23] support only ML predicates and logic predicates of the form $x.A = y.B$ and $z.A = c$. They cannot (semantically) express (a) edge predicates $l(x, y)$ and comparisons $<, \leq, >, \geq$, and (b) temporal orders and ranking model $\mathcal{M}_{\text{rank}}$ of GCR$^+$s (in the absence of $<$ and $>$). Moreover, the consequence p_0 of a GCR is defined on the leaves or centers of two stars, while p_0 of a GCR$^+$ can be on any vertices, even in the same star (*e.g.,* φ_5). In fact, none of the rules in Example 4 can be expressed as GCRs except φ_3.

Semantics. We apply GCR$^+$ $Q[x_0, y_0](X \rightarrow p_0)$ to graphs $(G, \preceq, \prec)$ with temporal orders. Denote by $h(\bar{x})$ a match of $Q[x_0, y_0]$ in graph G, also referred to as a *valuation* of the GCR$^+$.

We say that match $h(\bar{x})$ *satisfies* a predicate p, denoted by $h(\bar{x}) \models p$, if (a) when p is $x.A \oplus y.B$, attribute A (resp. B) exists at vertex $h(x)$ (resp. $h(y)$) in G, *i.e.,* the vertex $h(x)$ (resp. $h(y)$) carries attribute A (resp. B), and $h(x).A \oplus h(y).B$; similarly for $z.A \oplus c$; (b) when p is $\mathcal{M}(x.\bar{A}, y.\bar{B})$, the ML model $\mathcal{M}$ predicts true at $(h(x).\bar{A}, h(y).\bar{B})$; (c) when p is $l(z_1, z_2)$, there exists an edge labeled l from $h(z_1)$ to $h(z_2)$ in graph G; and (d) when p is $x.A \preceq y.B$, $h(x).A \preceq h(y).B$ with the partial order $\preceq$ in $(G, \preceq, \prec)$; similarly for $x.A \prec y.B$.

For precondition X, we write $h(\bar{x}) \models X$ if $h(\bar{x})$ satisfies *all* predicates in X. We write $h(\bar{x}) \models X \rightarrow p_0$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models p_0$.

We say that graph $(G, \preceq, \prec)$ with temporal orders *satisfies* GCR$^+$ $\varphi = Q[x_0, y_0](X \rightarrow p_0)$, denoted by $(G, \preceq, \prec) \models \varphi$, if for all matches $h(\bar{x})$ of $Q[x_0, y_0]$ in G, $h(\bar{x}) \models X \rightarrow p_0$. We say that $(G, \preceq, \prec)$ *satisfies* a set Σ of GCR$^+$s, denoted by $(G, \preceq, \prec) \models \Sigma$, if for all GCR$^+$s $\varphi \in \Sigma$, $(G, \preceq, \prec) \models \varphi$, *i.e.,* $(G, \preceq, \prec)$ satisfies every GCR$^+$ in Σ.

Graph cleaning. As shown in [23], GCRs extended with $\preceq, \prec, \geq, >$ support the primitives of relational data quality rules: conditional functional dependencies (CFDs) [26], denial constraints (DCs) [7], matching dependencies (MDs) [25]. GCR$^+$s subsume GCRs as a special case, and support all the primitives of CFDs, DCs and MDs.

Moreover, GCR$^+$s can express rules for ER, CR, TD and MI. Consider a GCR$^+$ $\varphi = Q[x_0, y_0](X \rightarrow p_0)$, and a graph with temporal orders $(G, \preceq, \prec)$. Assume the availability of a knowledge graph G_{KB}.

(1) When p_0 is $x_0.\text{id} = y_0.\text{id}$, the GCR$^+$ deduces that vertices $h(x_0)$ and $h(y_0)$ refer to the same entity for a match h of Q in G, for ER.

(2) If a match $h(\bar{x}) \models X$ but $h(\bar{x}) \not\models x.A \oplus y.B$ (*i.e.,* p_0), then it catches conflicts between $h(x)$ and $h(y)$, for CR; similarly for $x.A \oplus c$ as p_0.

(3) When p_0 is $x.A \preceq y.B$, it deduces temporal order on $h(x).A$ and $h(y).B$ for any match h of Q in G, for TD; similarly for $x.A \prec y.B$.

(4) For MI, GCR$^+$s imputes missing values and links by using G_{KB}.

(a) If p_0 is $x_0.A = y_0.B$, the GCR$^+$ adds missing A attribute to entity $h(x_0)$ for match h of Q in G, by possibly extracting the corresponding property of $h(y_0)$ in G_{KB}; similarly for $x_0.A = c$ as p_0.

(b) If p_0 is $l(z_1, z_2)$, $Q[x_0, y_0] = \langle Q_x, Q_y \rangle$, and if h maps z_1 and z_2 to vertices in G, matches Q_x in graph G and Q_y in knowledge graph G_{KB}, the GCR$^+$ adds edge $(h(z_1), l, h(z_2))$ to G; we treat G and G_{KB} as a "single disconnected" graph G'; over G', the GCR$^+$ fills in a missing link of G by referencing G_{KB}; here the consequence p_0 is defined on z_1 and z_2 in the same star mapped to G. This leverages the property that x_0 and y_0 of a dual pattern $Q[x_0, y_0]$ may be disconnected, and the two stars Q_x and Q_y in Q may find their matches in *different* G and G_{KB}, respectively. Note that the addition of $(h(z_1), l, h(z_2))$ changes the topological structure of G.

Example 5: We apply the GCR$^+$s from Example 4 to improve the quality of graph G in Example 2. Assume that all timestamps, language, title and country values, and entities u_1, v_2, v_4, w and their links in-between are taken from a reliable knowledge graph G_{KB}.

(a) Initially, we deduce that u_2's number (112) of supported languages is more current than that (50) of u_1 by φ_1 (TD).

(b) Then a conflict between the status values of u_1 and u_2 is found by φ_2, and u_2's status can be rectified to be active (CR).

(c) We next identify companies v_1 and v_2 by using φ_3 (ER), where x_4 and y_5 are mapped to vertices with value US.

(d) The type at v_1 is then copied from the reliable counterpart v_2 in G_{KB} (MI), as a byproduct of ER.

(e) The venue of v_2 is deduced more current than v_1, via φ_4 (TD).

(f) Finally, we deduce a missing link by φ_5, *i.e.,* the AI lab v_3 belongs to v_1 (MI); it references the link between v_4 and v_2 in G_{KB}.

Note that ER step (c), CR step (b), TD steps (a) and (e), and MI steps (d) and (f) interact with each other in this process. For instance, step (b) is applicable only after we deduce temporal orders for supported languages in (a); and step (e) needs the results of ER and MI in (c)-(d), which decide the right type value for company v_1. □

3 THE COMPLEXITY FOR GCR$^+$s

In this section, we settle the complexity for reasoning about GCR$^+$s.

Classic problems. We study three problems associated with rules.

Validation. We first study the *validation* problem.

- Input: A graph $(G, \preceq, \prec)$ and a set Σ of GCR$^+$s.
- Question: Does $(G, \preceq, \prec) \models \Sigma$?

This is to settle the complexity of error detection (Section 5).

Satisfiability. The *satisfiability* is to check whether GCR$^+$s have no conflicts with each other and can be applied at the same time.

- Input: A set Σ of GCR$^+$s.
- Question: Is there a graph $(G, \preceq, \prec)$ such that $(G, \preceq, \prec) \models \Sigma$ and for each GCR$^+$ $Q[x_0, y_0](X \rightarrow Y)$ in Σ, Q has a match in G?

Implication. A set Σ of GCR$^+$s *implies* another GCR$^+$ φ, written as $\Sigma \models \varphi$, if for any $(G, \preceq, \prec)$, if $(G, \preceq, \prec) \models \Sigma$ then $(G, \preceq, \prec) \models \varphi$.

The *implication problem* also concerns static analyses.

- Input: A set Σ of GCR$^+$s and another GCR$^+$ φ.
- Question: Does $\Sigma \models \varphi$?

This can remove redundant rules that are implied by the others.

Complexity. The validation problem is in PTIME for GCRs [23], but coNP-complete for GFDs and GARs [28, 35]. We show that the problem remains tractable for GCR$^+$s despite that GCR$^+$s support edge predicates, temporal predicates and comparison predicates beyond GCRs (see [4] for a proof). That is, one can efficiently detect errors with GCR$^+$s. A PTIME algorithm for error detection, the complement of validation, is given in Section 5.1 and [4]. We assume that it takes $f(|x.\bar{A}|, |y.\bar{B}|)$ time to check whether the prediction $\mathcal{M}(x.\bar{A}, y.\bar{B})$ is true for some polynomial function f; here $|x.\bar{A}|$ (resp. $|y.\bar{B}|$) is the size of the values in $x.\bar{A}$ (resp. $y.\bar{B}$).

Theorem 1: *For* GCR$^+$s*, the validation problem is in* PTIME. □

However, their static analyses become intractable, as opposed to PTIME for GCRs [23]. Here we assume *w.l.o.g.* that for ML models $\mathcal{M}$ in GCR$^+$s, there exists a small range such that any $x.\bar{A}$ and $y.\bar{B}$ can be mapped to values $\bar{V}_1$ and $\bar{V}_2$ in the small range, and $\mathcal{M}(x.\bar{A}, y.\bar{B}) = \mathcal{M}(\bar{V}_1, \bar{V}_2)$, as commonly found in practice [23, 29].

Theorem 2: *For* GCR$^+$s*, the satisfiability problem is* NP*-complete, and the implication problem is* coNP*-complete.* □

Proof sketch: The upper bound for satisfiability is verified by a small model property: if Σ is satisfiable, then there exists a graph $(G_s, \preceq, \prec)$ such that its size is bounded by $|\Sigma|^2(|\Sigma|+|\mathcal{M}|)$ and $(G_s, \preceq, \prec) \models \Sigma$; similarly for implication. The proofs make use of the small range property. The lower bound follows from Theorem 3 below. □

It is known that the satisfiability and implication analyses are coNP-complete and NP-complete for GFDs [35] and GARs [28], respectively. In contrast, the two problems are tractable for GCRs. However, the tractability is quite fragile. Indeed, the two problems become intractable even for mild extensions of GCRs with either built-in comparison predicates $\leq, <, \geq, >$ for CR, or temporal orders $\preceq, \prec$ that are necessary for TD (see [4] for a detailed proof).

Theorem 3: *The satisfiability problem is* NP*-hard and the implication problem is* coNP*-hard even for* GCRs *extended with either (a) comparison predicates* $\geq, <$*, or (b) temporal orders* $\preceq, \prec$*.* □

Proof sketch: We show that the satisfiability problem is NP-hard by reduction from the 3SAT problem, which is NP-complete (cf. [44]). The reduction is nontrivial: it encodes 3SAT formulas with $\geq, <$ for GCRs extended with comparison predicates, and with $\prec, \preceq$ for temporal order. Similarly, we show that the implication problem is coNP-hard by reduction from the complement of 3SAT. These are quite different from the proofs of [23], which use reductions with tree patterns, rather than temporal orders. □

4 A TEMPORAL RANKING MODEL

In this section, we train the temporal pairwise ranking model $\mathcal{M}_{\mathsf{rank}}$. Given vertices v and v' in a graph G along with their attributes A and B, $\mathcal{M}_{\mathsf{rank}}$ aims to determine whether $v.A \preceq v'.B$ or $v.A \prec v'.B$.

Limitation of existing models. One may want to adopt existing ML models for temporal graph completion (*e.g.,* ReGCN [57], TA-DistMult and TA-TransE [43]) or time series prediction (*e.g.,* EvoNet [48]). Unfortunately, such models stop short of the task for the following reasons. (1) The completion models focus on predicting missing entities/links in temporal knowledge graphs at explicitly specified timestamps, but do not compare numerical values (timestamps); hence timeliness deduction cannot be directly treated as a completion task. (2) Time series prediction models aim to predict future events in time-series data based on the historical ones, not to decide temporal orders between attributes within arbitrary timespan. (3) Prior methods for detecting outdated facts [47, 79] are not applicable either, since they rely heavily on external data and human intervention. In addition, (4) to train a good ranking model, a large amount of labeled orders are often a must [56]. However, it is hard to find sufficient timestamps in real-life graphs for training; manual labeling by *e.g.,* crowd-sourcing [74], is often too costly.

Overview. To address the above challenges, we propose temporal ranking model $\mathcal{M}_{\mathsf{rank}}$ with the following properties.

(a) A *graph partitioning strategy* is introduced to divide a graph based on available timestamps. All data pertaining to close timestamps (and same attributes) is grouped together, upon which $\mathcal{M}_{\mathsf{rank}}$ can learn the temporal features of attributes, including the structural information. The partitioning also serves as a bridge to connect timestamps sequentially, making it possible to incorporate the characteristics of attribute changes in temporal features (see below).

(b) A new type of embeddings is adopted to represent the temporal features of attributes. Observe that some temporal orders can be decided regarding the neighborhood structure, *e.g.,* if a person got a bachelor degree from university v (encoded as an edge) and is awarded a PhD from another university v' in the same field, then the year she graduated from v' should be later than that from v. On the other hand, the attribute changes also help deduce timeliness, *e.g.,* if we learn from the temporal graph that newly built subway lines in a city v are named with monotonically increasing numbers over the time, then the opening date of a subway line in v having the largest number should be the most current. In light of these, extending previous latent features [71], we take both the graph topological structures and the attribute changes over time into account to design the embeddings. Moreover, we also consider useful correlations among entities (*i.e.,* frequencies of co-occurrences). For instance, if a blogger updates daily while another one updates monthly, then it is highly possible that the latest post by the former blogger is more up-to-date.

(c) We apply semi-supervised learning to enrich the training set. That is, certain samples verified by the ranking model are included in a progressive manner during the model training.

Model. As shown in Figure 3, given $v.A$ and $v'.B$, $\mathcal{M}_{\mathsf{rank}}$ returns the likelihood of $v.A \preceq v'.B$ in three steps; similarly for $v.A \prec v'.B$.

(1) Based on the available timestamps, it first partitions graph G into multiple fragments with an *attribute-driven* strategy. This ensures that neighborhood information of vertices related to the same type of attributes and close timestamps is collected together.

(2) For two attributes $v.A$ and $v'.B$, $\mathcal{M}_{\mathsf{rank}}$ generates their embeddings via an extended 3-layer RGCN [71] model. It computes the embeddings starting from the first fragment with the earliest timestamp, and adopts the embeddings of $v.A$ and $v'.B$ *w.r.t.* the fragment having the latest timestamp as their final embeddings. Here RGCN is chosen because of its ability to capture relation-aware structural information in graphs with heterogeneous edges (relations). Despite the simplicity, RGCN has shown to perform well in various

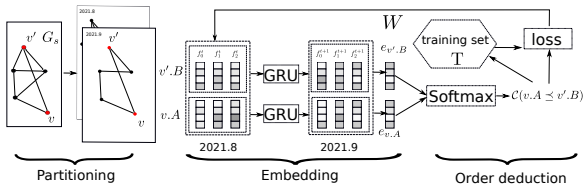


Figure 3: The network architecture of $\mathcal{M}_{\mathsf{rank}}$

tasks on knowledge graphs, *e.g.,* entity classification [82]. This said, one can also adopt other models capable of representing contextual features in heterogeneous graphs, *e.g.,* r-GAT [15].

(3) Using a softmax layer, it next combines the final embeddings of $v.A$ and $v'.B$, and generates a confidence score for $v.A \preceq v'.B$, denoted by $\mathcal{C}(v.A \preceq v'.B)$, indicating how likely $v.A \preceq v'.B$ holds.

In the following, we first show the details of these steps, and then describe how we train the model with a progressive procedure.

Graph partitioning. Figure 4 illustrates the graph partitioning procure. It partitions the temporal graph G into m fragments $F_1, \ldots, F_m$.

One might want to partition G solely based on timestamps, *i.e.,* attributes having close timestamps in the same interval are grouped in a single fragment. This can reduce the number of parameters and network depth of the model, but may generate excessive sparse partitions due to the imbalanced distribution of attributes over time, *e.g.,* there is an explosive growth of academic papers after the year 2000.

To alleviate this problem, we apply *attribute-driven partitioning*. It groups the attributes $v.A$ in G by their types (line 2), *i.e.,* the pair $[L(v), A]$ of vertex label $L(v)$ and attribute name A; and evenly partitions each group $S[L(v), A]$ *w.r.t.* the timestamps (lines 3-4). That is, it sorts the attributes by their available timestamps and divides the sorted sequence into m consecutive sub-sequences of roughly the same size, where each one corresponds to a subset of $S[L(v), A]$, and all attributes within the same subset have close timestamps. Finally, each output fragment F_i consists of the attributes in the i-th partition of *every* group and their adjacent edges (lines 5-6). As the partitioning is achieved at a more fine-grained level, it guarantees that all the resulting fragments have comparable sizes.

In Section 6 we will report the benefit of attribute-driven partitioning and the impact of parameter m on the accuracy of $\mathcal{M}_{\mathsf{rank}}$.

RGCN-based embedding. We extend a 3-layer RGCN model with co-occurrence probabilities to compute the embeddings of vertices and attributes in different fragments, and connect consecutive embeddings to incorporate the attribute changes.

Given a vertex or an attribute x, denote by $P(y|x)$ (resp. $P(l|x)$) the probability that vertex or attribute y (resp. edge label l) appears as a neighbor of x. For each such an x in fragment $F_t (t \in [1, m])$, the RGCN model deduces its embedding at each layer $j+1$ by aggregating the embeddings of its neighbors at layer j via message passing [71]. It adopts relation-aware transformation, *i.e.,* messages passed over different types of relations (edges) are treated separately. Differing from the original RGCN, at aggregation, here the embeddings are further multiplied by the frequencies of co-concurrences with the neighbors in G (*i.e.,* the probabilities $P(y|x)$ and $P(l|x)$). This is to include the impact of correlations among entities in the embeddings, as stated above. More specifically, the embedding of x at layer $j+1$ *w.r.t.* fragment F_t is computed as follows:

$$f_{j+1}^t(x) = \sigma\left(W_j^t f_j^t(x) + \sum_{l \in R}\left(P(l|x) W_j^t(l) \sum_{y \in \mathcal{N}_l^t(x)} P(y|x) f_j^t(y)\right)\right).$$

Input: A temporal graph G and a positive integer m.
Output: Fragments $F_1, \ldots, F_m$ of G.
1. initialize $F_1, \ldots, F_m$ as empty graphs; $S := nil$;
2. collect in $S[L(v), A]$ for every attribute $v.A$ from G;
3. **for each** attribute set $S[L(v), A]$ **do**
4. evenly partition $S[L(v), A]$ into m subsets based on consecutive timestamps of the attributes;
5. **for each** $i \in [1, m]$ **do**
6. add $v.A$ in the i-th subset of $S[L(v), A]$ and adjacent edges to F_i;
7. **return** $F_1, \ldots, F_m$;

Figure 4: Graph partitioning for training $\mathcal{M}_{\mathsf{rank}}$

Here (1) $j \in [0, 2]$ denotes the j-th layer of neural network; (2) $f_j^t(x)$ is the embedding of x at the j-th layer *w.r.t.* fragment F_t; (3) σ is the activation function (*e.g.,* ReLU); (4) R is the set of edge labels in G; (5) $\mathcal{N}_l^t(x)$ is the set of all vertices and attributes that appear as neighbors of x and are connected to x by edges labeled with l in F_t; and (5) both W_j^t and $W_j^t(l)$ are weight matrices at layer j. In particular, it utilizes a BERT-style pre-trained language model, *e.g.,* SimCSE [42], to initialize the embeddings *w.r.t.* the first fragment.

Moreover, for two consecutive fragments F_t and F_{t+1}, the embedding of x *w.r.t.* F_{t+1} starts with the final embedding *w.r.t.* F_t based on the GRU model [16], *i.e.,* $f_0^{t+1}(x)$=GRU$(f_3^t(x), h_3^t(x))$, where h_3^t is the hidden state of the third layer at time t. That is, we also learn temporal orders by connecting consecutive embeddings, to leverage relevant timestamps and capture attribute changes over the time.

Order deduction. After generating the final embeddings $r_{v.A}$ and $r_{v'.B}$ of attributes $v.A$ and $v'.B$, *i.e.,* $f_3^m(v.A)$ and $f_3^m(v'.B)$, where m denotes the identifier of last fragment, it exploits the softmax function to deduce the likelihood score $\mathcal{L}(v.A \preceq v'.B)$ for the temporal order $v.A \preceq v'.B$. More specifically, $\mathcal{L}(v.A \preceq v'.B)$ is

$$\mathcal{L}(v.A \preceq v'.B) = \frac{e^{r_{v'.B} - r_{v.A}}}{e^{\gamma} + e^{r_{v'.B} - r_{v.A}}}.$$

Here parameter γ is a weight decay to avoid overfitting.

Loss function. Let D_T be the set of training data in graph G (labeled temporal orders $v.A \preceq v'.B$). Denote by D_T^+ (resp. D_T^-) the sets of positive (resp. negative) examples in D_T. We train ranking model $\mathcal{M}_{\mathsf{rank}}$ by minimizing the cross-entropy loss:

$$\text{loss} = -\sum_{o \in D_T^+} \ln(\mathcal{L}(o)) + \sum_{o \in D_T^-} \ln(1 - \mathcal{L}(o)).$$

Progressive training. To cope with inadequate training data, we adopt a semi-supervised training strategy to progressively expand training data with high-quality temporal orders deduced by $\mathcal{M}_{\mathsf{rank}}$, and iteratively repeat the training for N times to refine $\mathcal{M}_{\mathsf{rank}}$ (see Section 6 for the impact of N on model training).

A temporal order $v.A \preceq v'.B$ is classified as high-quality if (1) its likelihood score is relatively high, which is marked as *a positive example*; or (2) its score is relatively low *i.e., a negative example*; intuitively, it indicates that $v'.B \succ v.A$ is of high confidence. By default, we set confidence score ranges $[0.6, 0.8]$ and $[0.2, 0.4]$ for defining these two classes, respectively. To avoid trivial orders, we do not choose those having either very high or very low scores beyond the two ranges. Such orders cannot help $\mathcal{M}_{\mathsf{rank}}$ capture detailed and unbiased characteristics for timeliness deduction [17].

Remark. (1) The trained model $\mathcal{M}_{\mathsf{rank}}$ can simply adapt to graph updates, *e.g.,* vertex and edge insertions/deletions. In fact, if the size of updates is very small, it is safe to reuse $\mathcal{M}_{\mathsf{rank}}$ directly since small

updates do not change the underlying characteristics of temporal graphs. If a large amount of updates are applied with more current timestamps, a common scenario in practice, it suffices to resume the training with newly introduced fragments that contain the updates. In case that historical changes are made, we maintain the states of previous training and retrain the model starting from the first fragment that is affected by the updates. Moreover, incremental learning techniques like [66] are employed to speed up the process.

(2) As users are often interested in timeliness deduction among attributes of a few types only, we can train the ranking model within subgraphs that include attributes of the requested types and their local structures. This makes the training scalable with growing temporal graphs; and it does not affect the expressive power of the model because RGCN only inspects localized information. Besides, users can adjust the configurable partition number, *i.e.,* parameter m, to reduce the size of the model and improve its scalability.

(3) Employing the model $\mathcal{M}_{\text{rank}}$, we can do timeliness deduction on any graph with partially ordered timestamps. Moreover, with the recent progress on graph pre-training and self-supervised learning, the parameters of $\mathcal{M}_{\text{rank}}$ can be efficiently and accurately transferred to timeliness deduction over other unseen graphs having similar characteristics, avoiding the extensive retraining [93].

5 HERCULES: A GRAPH CLEANING SYSTEM

In this section, we present the graph cleaning system Hercules (Section 5.1), and develop its underlying algorithms (Section 5.2).

5.1 The Architecture of Hercules

As depicted in Fig. 5, Hercules takes a graph G as input. It maintains a knowledge base G_{KB}, a store of training graphs of different types, and ground truth Γ (see below). Given G, it first mines a set Σ of GCR$^+$s. Users may then opt to detect or correct errors in G with the GCR$^+$s and accumulated ground truth, with the following modules.

Rule discovery. The module mines a set Σ of *high-quality* GCR$^+$s from a training graph of the same "type" (*e.g.,* Web) as G in its graph store. It also trains ranking model $\mathcal{M}_{\text{rank}}$ (Section 4); $\mathcal{M}_{\text{rank}}$ and other pre-trained ML models are embedded in the GCR$^+$s learned.

Error detection. Given a graph G, Hercules uses the mined GCR$^+$s, and catches errors in G as violations of the GCR$^+$s. For a GCR$^+$ $\varphi = Q[x_0, y_0](X \rightarrow p_0)$, a *violation* is $h(p_0)$, where h is a valuation of φ in G such that $h \models X$ but $h \not\models p_0$, and $h(p_0)$ is p_0 in which each variable x is instantiated with $h(x)$. Hercules is to detect all violations in G, such as duplicates (u.id=v.id), semantic inconsistencies ($u.A = v.B$, $u.A = c$, $v.B = d$ but $c \neq d$), conflicting timeliness ($u.A \preceq v.B$ and $v.B \prec u.A$) and missing values/links ($u.A = c$ or $l(u, v)$).

Error correction. This module enforces GCR$^+$s in Σ on graph G via revised chase [5], to deduce fixes (see below). Besides ER and CR, it determines temporal orders on graph properties, and fills in missing values and links by referencing knowledge base G_{KB} and ground truth Γ. It updates graph G with the deduced fixes, *e.g.,* new edges. Moreover, it adds the fixes to Γ for subsequent corrections.

Properties. Hercules has the following unique functionalities.

(1) Hercules is able to catch duplicates (ER), fix conflicts (CR), deduce timeliness (TD), and fill in missing values and links (MI), by applying GCR$^+$s that target ER, CR, TD and MI simultaneously. It leverages their interaction to improve the overall quality of graphs.

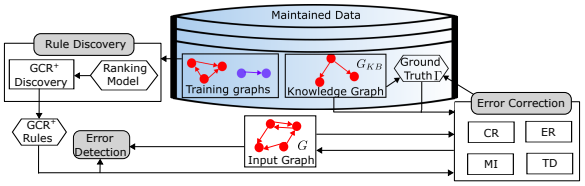


Figure 5: The architecture of system Hercules

(2) In principle Hercules is able to scale with large graphs. It detects errors and fixes errors in PTIME, and its modules for rule mining, error detection and correction are provably parallelly scalable, *i.e.,* they guarantee to reduce runtime when more processors are used.

(3) It ensures that the fixes to errors are the *unique* logical consequences of the set Σ of GCR$^+$s and ground truth Γ under certain conditions on the ML models embedded in the GCR$^+$s of Σ.

Algorithms. We present the error correction algorithm below, and the algorithms for rule mining and error detection in [4]. These algorithms extend the counterparts of [23] to deal with new challenges introduced by TD and MI, as exemplified in Section 5.2.

5.2 Parallel ER, CR, TD and MI Altogether

Extending the chase [5], we next present the parallel algorithm for Hercules to conduct ER, CR, TD and MI in the same process for error correction. We start with the fixes and review the procedure of chasing with graph rules to deduce fixes [28, 31, 32].

Fixes. We model each *fix* as a triple (s_1, op, s_2), where s_1 and s_2 are attributes $v.A$ or vertices v, and one of them can also be a constant c; op is a comparison operator, a temporal order or an edge label supported by GCR$^+$s. The fix can encode various *valid facts*: (a) if op is one of $=, \neq, \leq, <, \geq, >$, s_1 op s_2 enforces ER or CR; (b) if op is $\preceq$ or $\prec$, s_1 op s_2 is for TD; (c) if op is an edge label, then s_1 op s_2 indicates an edge labeled op from s_1 to s_2 in graph G, for MI.

We correct errors in graph G by enforcing the fixes as in Section 2.2. More specifically, (1) in CR, if s_2 (resp. s_1) is a constant c and op is $=$, we change the value of attribute s_1 (resp. s_2) to c; for all other cases of CR including $\neq$ comparison, the fix is recorded for further validating the preconditions of graph rules (see below), without changing the graph; (2) in ER, we merge s_1 and s_2 into a single vertex; (3) in MI, we add a new edge (s_1, op, s_2) if it does not exist; (4) in TD, we just keep s_1 op s_2 for subsequent verification.

Chasing. The chase [5] has been revised to deduce a set $\mathcal{F}$ of fixes with different kinds of graph rules [28, 31, 32]. Consider a graph G, a set Σ of graph rules and another set Γ of ground truth (initial valid facts). A *chasing sequence* of G by (Σ, Γ) is defined as

$$(\mathcal{F}_0, G_0) \Rightarrow_{(\varphi_1, h_1)} (\mathcal{F}_1, G_1) \Rightarrow \cdots \Rightarrow (\mathcal{F}_{t-1}, G_{t-1}) \Rightarrow_{(\varphi_t, h_t)} (\mathcal{F}_t, G_t).$$

Here each $\mathcal{F}_i$ ($i \in [1, t]$) is a set of fixes and $\mathcal{F}_0$ is initialized as Γ; and G_i is a graph, where $G_0 = G$. Furthermore, $(\mathcal{F}_{i-1}, G_{i-1}) \Rightarrow_{(\varphi_i, h_i)} (\mathcal{F}_i, G_i)$ is a *chase step* for each $i \in [1, t]$, where

(a) h_i is a valuation of a graph rule φ_i (in the form of $Q(X \rightarrow p_0)$) of Σ in graph G_{i-1}, such that every predicate p' in precondition X is *validated* by $\mathcal{F}_{i-1}$ at h_i, *i.e.,* the fact $h_i(p')$ is encoded by an existing fix in $\mathcal{F}_{i-1}$; here $h_i(p')$ is obtained from p' by instantiating

each vertex x in p' by its match $h_i(x)$; and

(b) $\mathcal{F}_i$ expands $\mathcal{F}_{i-1}$ with the fix encoding $h_i(p_0)$, and graph G_i is obtained from G_{i-1} by enforcing the fixes in $\mathcal{F}_i$ as described above.

Intuitively, each chase step validates the precondition of a graph rule by using the fixes that *have already been verified* (initially from Γ). By induction on the number of chase steps, we know that all fixes deduced via the chase are the logical consequences of Σ and Γ. Moreover, the G_i's evolve the input G with changed attribute values, improving its quality at each chase step. Ground truth Γ can also accumulate more fixes during the process.

The chasing sequence terminates when no more rules in Σ can be enforced; and $(\mathcal{F}_t, G_t)$ is referred to as *the result of the chase*. It has been shown that chasing the graph with previous graph rules in [28, 31, 32] has the *Church-Rosser property*. That is, all chasing sequences of G by (Σ, Γ) converge at the same result, no matter what rules in Σ are used and in what order they are applied.

Adapting to GCR$^+$s. The chase above, however, cannot be directly applied to GCR$^+$s, since the introduction of MI and TD complicates the chasing procedure. Indeed, (a) since MI may add new edges to the graph, mutating the topological structures is needed in the chase, beyond attribute changes. Hence we have to perform repeated matching of the same pattern in updated graphs to find valuations. (b) TD introduces fixes with new operators $\prec$ and $\preceq$, making it possible to entail more fixes than the single $h_i(p_0)$ deduced in each chase step, *e.g.,* $x.A \prec y.B$ and $y.B \prec z.C$ imply $x.A \prec z.C$; similarly for other comparison operators in GCR$^+$s; deducing $h_i(p_0)$ alone may miss valid fixes that can be used for verification. (c) On the other hand, TD can help resolve conflicts as the conflicts often indicate that an attribute has conflicting values at different time, *e.g.,* v.name for the same college v was Hendrix College in 1889-1929, Hendrix-Henderson College in 1929-1931 and Trinity College in 1931. (d) The Church-Rosser property may be affected because the ML models in GCR$^+$s are applied on evolving graphs.

To cope with these issues, we extend the chasing procedure above by incorporating (1) incremental GCR$^+$ enforcement to avoid redundant computation caused by MI; (2) fix implication to exhibit the effects of comparison operators introduced by, *e.g.,* TD; and (3) temporal order-based conflict resolution. Below we present a sequential algorithm that implements the chase *w.r.t.* GCR$^+$s with such new strategies, followed by its parallelization. We also establish the condition on ML models to ensure the Church-Rosser property.

Algorithm. We develop an algorithm, denoted by GPlusFix, to correct errors with GCR$^+$s via revised chase, in Figure 6. Apart from graph G, a set Σ of GCR$^+$s and ground truth Γ, it additionally takes the ranking model $\mathcal{M}_{\mathsf{rank}}$ (Section 4) for resolving conflicts. Initially, GPlusFix assigns the set $\mathcal{F}$ of fixes as ground truth Γ and graph G_0 as input G; it also keeps a set $\Delta\mathcal{F}$ of *new fixes*, which is initialized as Γ (line 1). It iteratively computes the fixes and updates graph (lines 2-6). In each step, it selects an appropriate set Σ_i of GCR$^+$s that can be enforced to deduce new fixes, and performs the deduction *incrementally* (line 3). Then $\mathcal{F}$ is expanded with the newly derived fixes in current round and graph G_i is "repaired" with the fixes after conflict resolution (lines 5-6). GPlusFix returns $\mathcal{F}$ and the final repaired version of G when no new fix is deduced (line 7).

Incremental enforcement. GPlusFix employs procedure IncSelect to

Algorithm GPlusFix
Input: A graph G, a set Σ of GCR$^+$s and a set Γ of ground truth.
Output: A set $\mathcal{F}$ of fixes and the repaired graph G_t.
1. $\mathcal{F} := \Gamma$; $G_0 := G$; flag := *true*; $\Delta\mathcal{F} := \Gamma$; $i := 0$;
2. **while** flag $\neq$ *false* **do**
3. $\Sigma_i :=$ IncSelect(Σ, $\Delta\mathcal{F}$); $\Delta\mathcal{F} :=$ IncDeduce(Σ_i, G_i, $\mathcal{F}$); update := *false*;
4. **if** $\Delta\mathcal{F} \neq \emptyset$ **then**
5. flag := *true*; $\mathcal{F} := \mathcal{F} \cup \Delta\mathcal{F}$;
6. $\mathcal{F}_r :=$ Resolve(G_i, $\mathcal{F}$, $\mathcal{M}_{\mathsf{rank}}$); $G_{i+1} :=$ Update(G_i, $\mathcal{F}_r$); $i := i + 1$;
7. **return** ($\mathcal{F}$, G_i);

Figure 6: Correction algorithm

choose appropriate GCR$^+$s such that they have certain precondition predicates or pattern edges whose modes coincide with that of some fixes in $\Delta\mathcal{F}$. Here the *mode* of a fix just abstracts vertices to their labels; similarly for predicates and edges. By the semantics of the chase, only such GCR$^+$s are able to deduce new fixes in the current iteration. In particular, rules with ML predicates $\mathcal{M}(x.\bar{A}, y.\bar{B})$ in preconditions are included as long as there exist vertices that have the same labels as x or y in $\Delta\mathcal{F}$; and the modes of pattern vertices are considered to handle the merged vertices regarding ER.

GPlusFix then conducts incremental rule enforcement via procedure IncDeduce. When applying a selected GCR$^+$ in the chase step, it matches patterns in an incremental manner to find valuations [33], by treating fixes in $\Delta\mathcal{F}$ as vertex/edge insertions, *i.e.,* the computation is confined within the neighborhood of fixes. This helps reduce unnecessary computation that do not lead to new fixes. Note that we do not pre-validate the ML predicates in the chase step in light of graph evolution. Moreover, IncDeduce infers more fixes based on the ones deduced via GCR$^+$ enforcement, by solving the linear inequalities formed by subsets of fixes [52]. That is, it also adds those fixes that can be implied by the deduced ones, and their comparison operators and constants must appear in Σ and $\mathcal{F}$.

Conflict resolution. Using temporal ranking model $\mathcal{M}_{\mathsf{rank}}$, procedure Resolve determines the most current value of attributes in $\mathcal{F}$, to resolve conflicts such as $(x.A, =, c)$ and $(x.A, =, c')$ with $c \neq c'$. More specifically, for each group of fixes $(x.A, =, a_1), \ldots, (x.A, =, a_n)$ with the same attribute $x.A$ from $\mathcal{F}$, procedure Resolve applies $\mathcal{M}_{\mathsf{rank}}$ to pairwisely deduce temporal orders among the fixes and finds the corresponding latest value a_j for $x.A$. It only includes $(x.A, =, a_j)$ in the set $\mathcal{F}_r$, which is to be returned, while other fixes without conflicts are simply copied from $\mathcal{F}$ to $\mathcal{F}_r$. Then algorithm GPlusFix updates graph G_i with $\mathcal{F}_r$; in contrast, the entire set $\mathcal{F}$ of fixes is adopted to execute the chase in IncDeduce.

Intuitively, we update the graph with the latest values of attributes resolved from fixes in each chase step *w.r.t.* GCR$^+$s. This departs from the previous chase in [28, 31, 32], which terminates immediately with "undefined" ($\perp$) after encountering conflicts.

Example 6: The process described in Example 5 essentially demonstrates the run of GPlusFix over the graph G of Example 2 with the GCR$^+$s in Example 4 and a set of ground truth (initialized with a small set of validated facts from knowledge graph in Example 5). It applies incremental GCR$^+$ enforcement in every step. For instance, after changing u_2's status to active by enforcing GCR$^+$ φ_2, GPlusFix only inspects those potential valuations of the selected GCR$^+$ φ_3 in the next iteration, where x_2 or y_2 in Q_2 is mapped to u_2.

Suppose that there is another GCR$^+$ φ_2' that revises φ_2 by replacing its consequence predicate with y_0.status = preview. Then φ_2 and φ_2' are enforced in the same iteration of GPlusFix, yielding conflicts for u_2's status, *i.e.,* active and preview. Both are included in $\mathcal{F}$ for extending the chase; nonetheless, u_2's status is changed to active since it is more current as verified by model $\mathcal{M}_{\mathsf{rank}}$. □

Complexity. GPlusFix takes $O(F_m|\Sigma|^2|G|^3(|G|+|\Gamma|)^2)$ time because (1) there exist at most $O((|G|+|\Gamma|)^2)$ iterations, and (2) in each iteration, deducing new fixes and resolving conflicts is in $O(F_m|\Sigma|^2|G|^3)$ time based on the algorithm for validating GCR$^+$s (proof of Theorem 1, see [4]), where F_m is the cost of checking ML predicates.

Church-Rosser. We show that if the ML models in GCR$^+$s are *robust to graph perturbations* [10, 81], *i.e.,* if no perturbation (*e.g.,* edge insertions/deletions and attribute changes) changes the predictions of the ML models embedded in the GCR$^+$s, then chasing with GCR$^+$s still has the Church-Rosser property as in [28, 31, 32] (see [4] for a proof). In fact, when perturbations are small, *i.e.,* when the deduced fixes do not change the graph very much, it is easy to train robust models [10], including our ranking model $\mathcal{M}_{\mathsf{rank}}$. Such robust models can be trained via randomized smoothing [81], learning clean graph structure from perturbed graphs [50], worst-case margin [10] or robust cross-entropy loss [83], among other things.

Corollary 4: *Chasing with* GCR$^+$s *is Church-Rosser if the ML models in* GCR$^+$s *are robust to graph perturbations.* □

Parallelization. We parallelize GPlusFix with the guarantee of parallel scalability [54], to scale with large graphs.

Parallel Scalability. We adapt the parallel scalability of [54] to characterize the effectiveness of parallel algorithms. Denote by $t_s(|I|)$ the worst-case cost of a sequential algorithm $\mathcal{A}$ related to GCR$^+$s , where $|I|$ denotes the size of the input instance for $\mathcal{A}$. We say that a parallel algorithm $\mathcal{A}_p$ is *parallelly scalable relative to* $\mathcal{A}$ if $\mathcal{A}_p$ and $\mathcal{A}$ output the same result *w.r.t.* input instance I and when using n processors, its parallel cost can be expressed as

$$t_{\mathsf{p}}(|I|, n) = O\Big(\frac{t_{\mathsf{s}}(|I|)}{n}\Big).$$

Intuitively, the parallel scalability measures the speedup over a yardstick sequential algorithm $\mathcal{A}$ by parallelization. A parallelly scalable $\mathcal{A}_p$ "linearly" reduces the cost of $\mathcal{A}$ when n increases.

Parallel error correction. We next outline the parallelization of GPlusFix. Intuitively, parallelizing every iteration of GPlusFix is essentially parallelizing the procedure for computing violations of GCR$^+$s, *i.e.,* validating GCR$^+$s (proof of Theorem 1, see details in [4]). This includes a bottom-up process to match pattern vertices of GCR$^+$s starting from the leaves and aggregating matches at the centers; and a top-down process for consequence verification.

Since the cost of bottom-up/top-down step is proportional to the number of initial "triggers", *i.e.,* leaves and aggregated matches, we evenly partition the paths with their leaves decomposed from each selected GCR$^+$ as workloads, which are executed at multiple processors in parallel. After that, result aggregation is done at a coordinator. In the next step, we distribute the aggregated matches evenly to processors and perform top-down verification in parallel.

Corollary 5: *There exists an error correction algorithm that is parallelly scalable relative to the sequential algorithm* GPlusFix. □

Proof: The parallel scalability of GPlusFix is proven based on the even workload partition of the bottom-up and top-down procedure for validating GCR$^+$s, the step with the dominating cost (see [4]). □

Graph	#vertices	#edges	type
YAGO [60]	$3.5M$	$7.4M$	knowledge graph
IMDB [2]	$5.1M$	$5.2M$	movie database
DBLP [1]	$8.6M$	$65M$	citation network
Douban [89]	$13.7K$	$214K$	MI benchmark

Table 1: Datasets

6 EXPERIMENTAL STUDY

Using real-life and synthetic graphs, we experimentally evaluated Hercules for its (a) accuracy, (b) efficiency/(parallel) scalability, and (c) effectiveness in error correction, error detection and rule mining.

Experimental setting. We start with the experimental setting.

Experimental data. We used four real-life graphs, summarized in Table 1: (1) YAGO [60], a knowledge graph with temporal facts (*e.g.,* birthday); (2) IMDB, a movie database modeled as graph, including temporal attributes (*e.g.,* release year) [2]; (3) DBLP, a citation network with temporal attributes (*e.g.,* publication year of a paper) [1]; and (4) Douban [89], a benchmark dataset for link prediction (MI), which induces manually labeled *real missing links*.

We also generated synthetic graphs for scalability, having up to $0.6B$ vertices and $0.4B$ edges with 10000 timestamps. Their labels, attributes and values were drawn from a domain of 100 symbols.

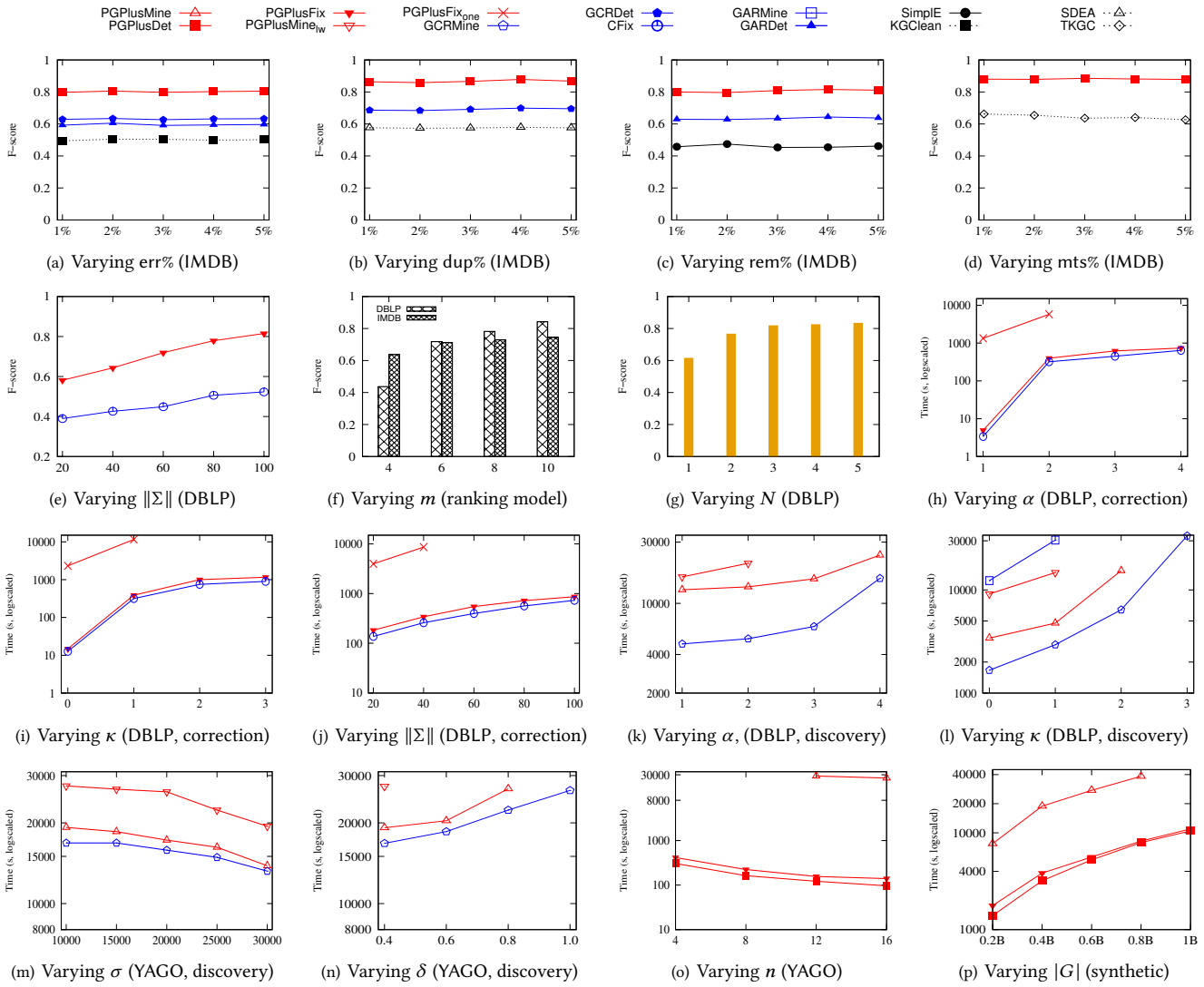
Baselines. We implemented the three parallel algorithms for error correction, error detection and rule discovery (Section 5) in C++, denoted as PGPlusFix, PGPlusDet and PGPlusMine, respectively. In PGPlusMine, we used SVM-L [36] to extract 5% constants and 6 attributes for mining predicates (see [4]). The ML predicates in GCR$^+$s include SDEA [91] for ER, HSRL [38] and SimplE [51] for link prediction, SimCSE [42] and DBSCAN [20] for similarity checking and clustering of textual sentences, and our model $\mathcal{M}_{\mathsf{rank}}$ in Section 4.

We compared with the following baselines: (1) PGPlusFix$_{\mathsf{one}}$ that randomly selects and enforces GCR$^+$s to correct errors; (2) CFix, a graph repairing method that revises [32] by using GCRs instead of GCR$^+$s, without conflict resolution; GCRDet [23] and GARDet [28], error detection algorithms with GCRs and GARs, respectively; (3) PGPlusMine$_{\mathsf{lw}}$, a levelwise method to mine GCR$^+$s; and GCRMine [23] and GARMine [24] for mining GCRs and GARs, respectively; and (4) AMIE+ [39], which mines Horn rules for link prediction. All these are parallel algorithms except AMIE+.

We also tested sequential (5) ML-based SimplE, a link predictor for MI; (6) KGClean [45], an ML model for CR; (7) SDEA [91], an ML ER approach; and (8) TKGC [73], an ML model for temporal link predication; we are not aware of any prior model that is specifically designed for TD over graph attributes; hence we treat pairwise temporal orders as relations, which are predicted by TKGC for TD.

For each graph, we sampled 30% as training data by default to train the models and discover graph rules. To evaluate PGPlusFix, we picked 7% of the data in the original graphs as ground truth Γ.

Effectiveness evaluation. We adopted the conventional F-score to measure the accuracy of different methods for error detection and correction. For MI, the *real* test sets of Douban was ex-



(a) Varying err% (IMDB) (b) Varying dup% (IMDB) (c) Varying rem% (IMDB) (d) Varying mts% (IMDB)

(e) Varying $\|\Sigma\|$ (DBLP) (f) Varying m (ranking model) (g) Varying N (DBLP) (h) Varying α (DBLP, correction)

(i) Varying κ (DBLP, correction) (j) Varying $\|\Sigma\|$ (DBLP, correction) (k) Varying α, (DBLP, discovery) (l) Varying κ (DBLP, discovery)

(m) Varying σ (YAGO, discovery) (n) Varying δ (YAGO, discovery) (o) Varying n (YAGO) (p) Varying $|G|$ (synthetic)

Figure 7: Performance evaluation

tracted as in [89]. For the other real-life graphs, we (1) removed edges (controlled by rem%, *i.e.,* the ratio of removed edges to all edges), and tested the accuracy *w.r.t.* rem%. In CR and ER, following [23, 32, 45, 91], we treated the original graphs as correct and manually (2) modified values of attributes (controlled by err%, *i.e.,* the ratio of the updated values to all attribute values), and (3) added duplicated entities (controlled by dup%, *i.e.,* the ratio of duplicates to all entities). For TD, all temporal attributes and their available timestamps in the graphs are ground truth. Thus we (4) masked available timestamps (controlled by mts%, *i.e.,* the ratio of masked ones to all timestamps) along the same lines as in [34].

We conducted the experiments on a cluster of up to 16 machines, each powered by 12 Intel Xeon 2.2GHz cores with 128 GB memory. Each experiment was run 5 times, and the average is reported here.

Experimental results. We next report our findings.

Exp-1: Accuracy. We first tested PGPlusDet for CR, ER, MI and TD on IMDB, and PGPlusFix for error correction on DBLP by taking the four tasks together. The results on the other graphs are consistent. We used the same number (100) of mined rules for rule-based methods, and the same amount of ground truth (7% of the data) for error correction. By default, we set err%=dup%=rem%=mts%=5%.

Error detection. We varied err% (resp. dup%, rem%, mts%) from 1% to 5% to test the effectiveness for each of the four tasks CR, ER, MI and TD. As shown in Figures 7(a)–7(d), PGPlusDet consistently performs the best for the four tasks. It is (1) 27% (resp. 60%) more accurate than rule-based GCRDet (resp. ML model KGClean) in CR; (2) 25% (resp. 51%) more accurate than GCRDet (resp. SDEA) in ER; (3) it beats SimpIE by 67% in MI; and (4) outperforms TKGC by 37% in TD. This is because PGPlusDet conducts CR, ER, TD and MI simultaneously, and each individual task can benefit from the other three. Note that GCRDet only supports limited MI with predicates $x.A = c$ as consequence in GCRs (hence not shown for MI), and neither GCRDet nor GARDet is able to perform TD.

Putting CR, ER, MI and TD together, PGPlusDet is 54% and 62% more accurate than GCRDet and GARDet, respectively. These justify the need for GCR$^+$s to improve the overall quality of graphs.

Methods	PGPlusDet	GARDet	SimpIE
Accuracy	0.773	0.608	0.472

Table 2: Accuracy for MI on benchmark dataset Douban

Benchmark results. We also conducted MI on benchmark dataset Douban and the accuracy results are given in Table 2. We find that PGPlusDet beats SimpIE and GARDet by 64% and 27% in detecting *real* missing links, respectively. This is consistent with Figure 7(c).

Varying $\|\Sigma\|$. We varied the number $\|\Sigma\|$ of GCR$^+$s from 20 to 100 to check its impact on error correction. As shown in Figure 7(e) on DBLP, (1) when $\|\Sigma\|$ gets larger, PGPlusFix becomes more accurate, as expected. (2) On average, it beats CFix by 53%. This again verifies the effectiveness of the interaction among CR, ER, TD and MI. Moreover, compared to CFix, PGPlusFix benefits from the conflict resolution strategy since CFix terminates immediately when encountering a conflict, without deducing more useful fixes.

Temporal ranking model. We tested the accuracy of our proposed ranking model $\mathcal{M}_{\mathsf{rank}}$ for TD on IMDB and DBLP. Figure 7(f) depicts the accuracy of $\mathcal{M}_{\mathsf{rank}}$ that is trained with different partition numbers m. As shown there, larger m help produce a better $\mathcal{M}_{\mathsf{rank}}$ with higher accuracy for TD on both datasets. This is because $\mathcal{M}_{\mathsf{rank}}$ can learn more delicate representations for temporal features when m grows. However, it also takes much longer to train $\mathcal{M}_{\mathsf{rank}}$ for larger m. We find that attribute-driven partitioning for $\mathcal{M}_{\mathsf{rank}}$ (Section 4) performs better than the one that partitions graphs based on timestamps only, *e.g.,* 0.842 vs. 0.519 on DBLP when $m = 10$. This justifies the design of our partitioning strategy.

We also varied the number N of training iterations to check its impact on the accuracy. As reported in Figure 7(g), the accuracy of $\mathcal{M}_{\mathsf{rank}}$ improves with larger N, as more training data can be used in a progressive manner. Moreover, a few rounds of iterations suffice to make $\mathcal{M}_{\mathsf{rank}}$ accurate, *e.g.,* its F-score reaches 0.8 when $N = 3$ over DBLP. On average, $\mathcal{M}_{\mathsf{rank}}$ outperforms TKGC by 17% for TD.

Exp-2: Efficiency and scalability. We next tested the efficiency and scalability of PGPlusFix, PGPlusDet and PGPlusMine. We used $n = 8$ processors by default, unless stated otherwise.

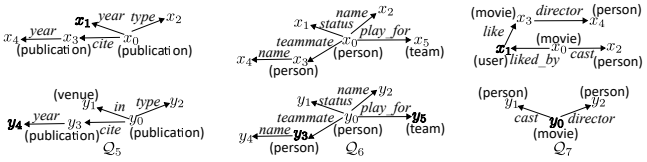
(1) Error correction. We first tested PGPlusFix and compared with PGPlusFix$_{\mathsf{one}}$ and CFix. We fixed a set Σ of rules with $\|\Sigma\| = 100$, α=3 (the maximum number of predicates in each GCR$^+$), and κ=2 (the number of edges in each path of GCR$^+$) unless stated otherwise.

Varying α. As shown in Figure 7(h) on DBLP, (a) PGPlusFix beats PGPlusFix$_{\mathsf{one}}$ by 82.4× on average when α is varied from 1 to 4, since it incrementally enforces GCR$^+$s. (b) PGPlusFix is comparable to CFix, although GCR$^+$s are more complex than GCRs; both adopt star-shaped patterns, and pattern matching dominates the cost.

Varying κ. The results in Figure 7(i) show that (a) PGPlusFix consistently outperforms the baselines with various κ values on DBLP, except CFix. (b) PGPlusFix gets slower when κ increases, since it takes longer to conduct path matching when given more pattern edges.

Varying $\|\Sigma\|$. Varying $\|\Sigma\|$ from 20 to 100 on DBLP, Figure 7(j) shows that (a) all algorithms take longer with larger $\|\Sigma\|$, as expected. (b) PGPlusFix performs comparably to CFix, and it is efficient to correct errors with GCR$^+$s, *e.g.,* PGPlusFix takes 546s using 60 GCR$^+$s.

(2) Error detection. We next tested the efficiency of PGPlusDet on DBLP in the default setting of Exp-2(1). We find the following.



$\mathcal{Q}_5$ $\mathcal{Q}_6$ $\mathcal{Q}_7$

Figure 8: Patterns in real-life GCR$^+$s

(a) PGPlusDet beats baselines PGPlusDet$_{\mathsf{gm}}$ and GARDet by 30× and 39×, respectively. This is because PGPlusDet employs dynamic programming to match star-shaped patterns [4], while the latter two apply conventional graph matching, which is more costly. (b) It is comparable to GCRDet although GCR$^+$s are more expressive than GCRs, and PGPlusDet not only detects duplicates and conflicts, but also deduces temporal orders and imputes missing values/links.

The impact of parameters α, κ and $\|\Sigma\|$ on the efficiency of PGPlusDet is similar to its counterparts on PGPlusFix (not shown).

(3) Rule discovery. We also evaluated the discovery methods, including PGPlusMine. Since it is too costly to find the entire set of rules by levelwise approaches, we set a target to find 100 rules, *i.e.,* the algorithm terminates after it mines 100 rules. We fixed $\alpha = 3$, $\kappa = 3$, $\sigma = 10000$ (support threshold) and $\delta = 0.4$ (confidence threshold), unless stated otherwise. The latter two are input parameters for rule discovery, which characterize high-quality rules in terms of frequency and reliability, respectively (see [4]).

Impact of α *and* κ. As shown in Figures 7(k)–7(l) on DBLP, (a) all approaches get slower with either larger α or κ, as expected. However, PGPlusMine is still at least 1.8× (resp. 16.5×) faster than PGPlusMine$_{\mathsf{lw}}$ (resp. GARMine). (b) PGPlusMine is slower than GCRMine, since it takes longer to generate candidate GCR$^+$s with predicates for TD and MI that are not supported by GCRs. (c) Compared to PGPlusMine and GCRMine, the other methods are more sensitive to α and κ, since the number of their candidate rules grows exponentially with α and κ, and they enumerate all possibles paths without using ML models. In particular, AMIE+ and GARMine fail to terminate in 12 hours in most of the cases (hence not shown).

Impact of σ *and* δ. The impact of support and confidence thresholds is reported in Figures 7(m)–7(n) on YAGO. (a) All levelwise algorithms get faster when σ gets larger, due to the anti-monotonicity of support (see [4]). (b) In contrast, they take longer as confidence threshold δ grows, since more rules are generated and verified with higher δ. These can serve as a guideline to select appropriate threshold values for mining rules for different applications. Again, AMIE+ and GARMine cannot terminate in 12 hours in all the cases tested.

(4) Parallel scalability. Varying the number n of processors from 4 to 16, we tested the parallel scalability of PGPlusFix, PGPlusDet and PGPlusMine. Here the default settings in Exps-2(1) to 2(3) were adopted for the three algorithms, respectively. As shown in Figure 7(o) on YAGO, all algorithms get faster when n increases. For instance, PGPlusDet is 3.2× faster when n varies from 4 to 16.

(5) Synthetic data. We finally tested the scalability of PGPlusFix, PGPlusDet and PGPlusMine with large synthetic graphs G, in the same setting as in Exp-2(4) but fixing $n = 8$. Here $|G| = |V| + |E|$. As shown in Fig. 7(p), (1) all the algorithms get slower when G becomes larger, as expected. (2) The three algorithms work well on large graphs. PGPlusFix (resp. PGPlusDet and PGPlusMine) takes 3838s (resp. 3199s and 18862s) on G with 0.4 billion vertices and edges.

Exp-3: Effectiveness. Below are three example GCR$^+$s mined and the real errors they caught. Their patterns are shown in Figure 8.

(a) In DBLP, we found a GCR$^+$ $\varphi_6 = Q_5[x_0, y_0](x_2.\text{val} = y_2.\text{val} \wedge \mathcal{M}_{\text{rank}}(y_1.\text{status}, x_4.\text{val}) \rightarrow y_4.\text{val} \preceq x_1.\text{val})$ for TD. It indicates that if two publications x_0 and y_0 are of the same type, *e.g.,* conference paper, and if y_0 is published earlier than a reference x_3 of x_0 (this is checked by using $\mathcal{M}_{\text{rank}}$ on the publication year of x_3 and the status of y_0's venue since y_0's year may be missing), then any paper y_4 cited by y_0 should also be published earlier than x_0. This GCR$^+$ detected an error that a paper published in 2018 was cited by another paper earlier than 2016 in DBLP, which was missed by other approaches like GCRDet, GARDet and KGClean.

(b) In YAGO, a GCR$^+$ $\varphi_7 = Q_6[x_0, y_0](x_2.\text{val} = y_2.\text{val} \wedge x_4.\text{val} = y_4.\text{val} \wedge \mathcal{M}_{\text{rank}}(x_1.\text{val}, y_1.\text{val}) \rightarrow \text{play\_for}(y_3, y_5))$ was mined for MI. This rule says that the current team (y_5) of a player (y_3) can be deduced from the up-to-date registration status (y_1) of his teammate (y_0). Here φ_7 checks players' names and applies the ranking model $\mathcal{M}_{\text{rank}}$ to determine the temporal orders for player status. Note that its consequence predicate links two vertices within the same star, *i.e.,* TD helps MI. It deduced the team LA Lakers of basketball player Anthony Davis from the current status of his teammate LeBron James in YAGO, which GARDet and SimpIE failed to find.

(c) In Douban, we discovered a GCR$^+$ $\varphi_8 = Q_7[x_0, y_0](\mathcal{M}_e(x_2, y_1) \wedge \mathcal{M}_e(x_4, y_2) \rightarrow \text{like}(x_1, y_0))$. It asserts that if a user x_1 likes to watch two movies having actor x_2 and director x_4, respectively, then movie y_0 pertaining to the same pair of actor and director simultaneously (checked by ER model $\mathcal{M}_e$) should be recommended to x_1. That is, users are interested in movies with the same actors and directors as the ones they previously watched. This simple GCR$^+$ φ_8 can restore missing links overlooked by SimpIE, *e.g.,* a movie with ID $m4344$ was recommended to user $u2942$.

Summary. We find the following. (1) Using GCR$^+$s, PGPlusDet (resp. GPlusFix) is 54% (resp. 53%) more accurate than their counterparts using GCRs, by conducting ER, CR, TD and MI in the same process, as opposed to ER and CR only with GCRs. PGPlusDet also beats GARDet by 62%. Moreover, it is 60%, 51%, 67% and 37% more accurate than ML-based KGClean, SDEA, SimpIE and TKGC in CR, ER, MI and TD, respectively. (2) Our ranking model outperforms TKGC by 17% in TD. (3) PGPlusMine is 16.5× faster than the algorithm for mining GARs. (4) PGPlusFix and PGPlusDet perform comparably to their counterparts with GCRs [23] in efficiency, despite that GCR$^+$s conduct not only ER and CR, but also TD and MI. These two and PGPlusMine are parallelly scalable, *i.e.,* they are 3×, 3.2× and 3× faster when 16 processors are used instead of 4.

7 RELATED WORK

ER, CR, TD and MI. While there has been a large body of work on ER [23, 28, 30, 31, 55, 58, 64, 92] and CR [23, 28, 30, 31, 35, 45, 69, 87], TD and MI remain longstanding challenges even for relational data [13, 84, 90]. (1) For TD, related are [47, 79] on detecting whether a fact is up-to-date in knowledge graphs, by consulting external data and human; TD over arbitrary attributes is not studied there; similarly in the models for temporal graph completion [43, 57] and time series prediction [48]. (2) For MI, especially on knowledge graphs [70], link prediction was explored by rules [6, 28, 29, 39] and ML [28, 29, 51, 72]. MI was also approached by validating shapes [69], matrix completion (missing-link inference) via Generative Adversarial Nets (GAN) [85], Graph Neural Network (GNN) [76, 86], optimal transport theory [63] and analytic functions [22].

In contrast, this work (1) proposes rules for ER, CR, TD and MI in graphs, by embedding ML models as predicates; it promotes the interactions of the four issues to improve the overall quality of graphs; (2) ensures the tractability of error detection and correction; and (3) develops the first temporal ranking model on graph properties.

Algorithms for graph cleaning. Algorithms for mining graph rules have been studied in [14, 24, 27, 29, 40, 41, 65, 68], based on either levelwise search-based mining that iteratively enumerate candidate rules [24, 27, 40, 65], or generation-based rule learners [14, 29, 41, 68] that train various generators to create promising rules based on embeddings. Error detection methods have been developed to catch conflicts [35], numeric errors [30] and semantic associations [28]. Error corrections are studied for graphs in [32], for ER and CR only.

This work (1) provides the first algorithms of rule mining, error detection and correction for not only ER and CR, but also TD and MI. (2) We show that the algorithms retain the following properties of their counterparts with GCRs, despite that GCR$^+$s handle TD and MI, beyond GCRs: (a) chasing with GCR$^+$s is Church-Rosser under certain conditions of the embedded ML models, (b) error detection and correction remain tractable with GCR$^+$s, and (c) the mining, detection and correction algorithms retain the parallel scalability. (3) We develop new techniques beyond the methods with GCRs [23], *e.g.,* temporal models for TD, and "incremental" rule enforcement to cope with the mutation of topological structures for MI. (4) We settle the complexity of the classical problems associated with GCR$^+$s.

Graph cleaning systems. There have been systems for cleaning knowledge graphs [8, 9, 45, 78, 88], via graph embedding [45, 88], edit history [78], language models [9], and user-guided annotations [8] to catch and fix erroneous subject-predicate-object triples. Unfortunately, none of the systems supports TD or MI.

To the best of our knowledge, Hercules is the first system that supports ER, CR, TD and MI altogether, and makes use of their interaction to improve the overall quality of graphs. It is also the first graph cleaning system that unifies ML models and logic deduction. In principle, it can scale with big graphs when given more resources.

8 CONCLUSION

The novelty of Hercules consists of (1) its capability of supporting (the interaction of) ER, CR, TD and MI for improving the overall quality of graphs; (2) a class of GCR$^+$s for uniformly expressing rules for ER, CR, TD and MI, and the complexity of the classical problems for GCR$^+$s; (3) a unique temporal ranking model for graph properties; and (4) the algorithms of Hercules with the parallel scalability, the tractability for detecting errors and fixing errors, and the Church-Rosser property of the chase under certain conditions. We have experimentally verified that Hercules is promising in practice.

One topic for future work is to equip Hercules with continuous and incremental training of ML models embedded in GCR$^+$s using idle resources, with newly added data and feedback. Another topic is to give a full treatment of online incremental algorithms for rule mining, error detection and error correction in response to updates.

REFERENCES

[1] 2021. DBLP collaboration network. *https://snap.stanford.edu/data/com-DBLP.html.*
[2] 2021. IMDB. *https://www.imdb.com/interfaces.*
[3] 2022. DBpedia. *http://www.dbpedia.org.*
[4] 2023. Full version. https://drive.google.com/file/d/1xvkkU8y-9vYwkaq3ZG2_JxZechXWRKIC/view?usp=drive_link.
[5] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[6] Naser Ahmadi, Viet-Phi Huynh, Venkata Vamsikrishna Meduri, Stefano Ortona, and Paolo Papotti. 2020. Mining Expressive Rules in Knowledge Graphs. *ACM J. Data Inf. Qual.* 12, 2 (2020), 8:1–8:27.
[7] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
[8] Abdallah Arioua and Angela Bonifati. 2018. User-guided Repairing of Inconsistent Knowledge Bases. In *EDBT*.
[9] Hiba Arnaout, Trung-Kien Tran, Daria Stepanova, Mohamed Hassan Gad-Elrab, Simon Razniewski, and Gerhard Weikum. 2022. Utilizing language model probes for knowledge graph repair. In *Wiki Workshop 2022*.
[10] Aleksandar Bojchevski and Stephan Günnemann. 2019. Certifiable Robustness to Graph Perturbations. In *NeurIPS*.
[11] David A. Bright, Russell Brewer, and Carlo Morselli. 2021. Using social network analysis to study crime: Navigating the challenges of criminal justice records. *Soc. Networks* 66 (2021), 50–64.
[12] Businesswire. 2022. Over 80 Percent of Companies Rely on Stale Data for Decision-Making. *https://www.businesswire.com/news/home/20220511005403/en/Over-80-Percent-of-Companies-Rely-on-Stale-Data-for-Decision-Making.*
[13] Yang Cao, Wenfei Fan, and Wenyuan Yu. 2013. Determining the relative accuracy of attributes. In *SIGMOD*. 565–576.
[14] Lihan Chen, Sihang Jiang, Jingping Liu, Chao Wang, Sheng Zhang, Chenhao Xie, Jiaqing Liang, Yanghua Xiao, and Rui Song. 2022. Rule mining over knowledge graphs via reinforcement learning. *Knowl. Based Syst.* 242 (2022).
[15] Meiqi Chen, Yuan Zhang, Xiaoyu Kou, Yuntao Li, and Yan Zhang. 2021. r-GAT: Relational Graph Attention Network for Multi-Relational Graphs. *CoRR* abs/2109.05922 (2021).
[16] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*. 1724–1734.
[17] Pádraig Cunningham and Sarah Jane Delany. 2020. Underestimation Bias and Underfitting in Machine Learning. In *TAILOR*. 20–31.
[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.
[19] Jonathan A. Edlow and Peter J. Pronovost. 2023. Misdiagnosis in the Emergency Department: Time for a System Solution. *JAMA* (2023).
[20] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*. 226–231.
[21] Christos Faloutsos, Danai Koutra, and Joshua T. Vogelstein. 2013. DELTACON: A Principled Massive-Graph Similarity Function. In *SDM*. 162–170.
[22] Jicong Fan, Yuqian Zhang, and Madeleine Udell. 2020. Polynomial Matrix Completion for Missing Data Imputation and Transductive Learning. In *IAAI*. 3842–3849.
[23] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Muyang Liu, Ping Lu, and Chao Tian. 2023. Making It Tractable to Catch Duplicates and Conflicts in Graphs. *SIGMOD* 1, 1 (2023), 86:1–86:28.
[24] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Ping Lu, and Chao Tian. 2022. Discovering Association Rules from Big Graphs. *PVLDB* 15, 7 (2022), 1479–1492.
[25] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.
[26] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* 33, 1 (2008), 6:1–6:48.
[27] Wenfei Fan, Chunming Hu, Xueli Liu, and Ping Lu. 2020. Discovering Graph Functional Dependencies. *ACM Trans. Database Syst.* 45, 3 (2020), 15:1–15:42.
[28] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Capturing Associations in Graphs. *PVLDB* 13, 11 (2020), 1863–1876.
[29] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards Event Prediction in Temporal Graphs. *PVLDB* 15, 9 (2022), 1861–1874.
[30] Wenfei Fan, Xueli Liu, Ping Lu, and Chao Tian. 2020. Catching Numeric Inconsistencies in Graphs. *ACM Trans. Database Syst.* 45, 2 (2020), 9:1–9:47.
[31] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.
[32] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing Certain Fixes to Graphs. *PVLDB* 12, 7 (2019), 752–765.
[33] Wenfei Fan and Chao Tian. 2022. Incremental Graph Computations: Doable and Undoable. *ACM Trans. Database Syst.* (2022).
[34] Wenfei Fan, Resul Tugay, Yaoshu Wang, Min Xie, and Muhammad Asif Ali. 2023. Learning and Deducing Temporal Orders. *PVLDB* 16, 8 (2023), 1944–1957.
[35] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. 1843–1857.
[36] Nausheen Fatma, Manoj Chinnakotla, and Manish Shrivastava. 2017. The unusual suspects: Deep learning based mining of interesting entity trivia from knowledge graphs. In *AAAI*.
[37] Annamaria Ficara, Lucia Cavallaro, Francesco Curreri, Giacomo Fiumara, Pasquale De Meo, Ovidiu Bagdasar, Wei Song, and Antonio Liotta. 2021. Criminal networks analysis in missing data scenarios through graph distances. *PLoS one* 16, 8 (2021), e0255067.
[38] Guoji Fu, Chengbin Hou, and Xin Yao. 2019. Learning topological representation for networks via hierarchical sampling. In *IJCNN*. 1–8.
[39] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.* 24, 6 (2015), 707–730.
[40] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*.
[41] Kun Gao, Katsumi Inoue, Yongzhi Cao, and Hanpin Wang. 2022. Learning First-Order Rules with Differentiable Logic Program Semantics. In *IJCAI*. 3008–3014.
[42] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *EMNLP*. 6894–6910.
[43] Alberto García-Durán, Sebastijan Dumancic, and Mathias Niepert. 2018. Learning Sequence Encoders for Temporal Knowledge Graph Completion. In *EMNLP*. 4816–4821.
[44] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.
[45] Congcong Ge, Yunjun Gao, Honghui Weng, Chong Zhang, Xiaoye Miao, and Baihua Zheng. 2020. KGClean: An Embedding Powered Knowledge Graph Cleaning Framework. *CoRR* abs/2004.14478 (2020).
[46] Mahboubeh Haddad, Fereshte Sheybani, HamidReza Naderi, Mohammad Saeed Sasan, Mona Najaf Najafi, Malihe Sedighi, and Atena Seddigh. 2021. Errors in Diagnosing Infectious Diseases: A Physician Survey. *Frontiers in Medicine* (2021), 2243.
[47] Shuang Hao, Chengliang Chai, Guoliang Li, Nan Tang, Ning Wang, and Xiang Yu. 2023. HOFD: An Outdated Fact Detector for Knowledge Bases. *TKDE* (2023), 1–14.
[48] Wenjie Hu, Yang Yang, Ziqiang Cheng, Carl Yang, and Xiang Ren. 2021. Time-Series Event Prediction with Evolutionary State Graph. In *WSDM*.
[49] Witold Jacak and Karin Pröll. 2011. Neural Networks Based System for Cancer Diagnosis Support. In *EUROCAST*.
[50] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph Structure Learning for Robust Graph Neural Networks. In *KDD*. 66–74.
[51] Seyed Mehran Kazemi and David Poole. 2018. SimplE: Embedding for Link Prediction in Knowledge Graphs. In *NeurIPS*. 4289–4300.
[52] Anthony C. Klug. 1988. On conjunctive queries containing inequalities. *J. ACM* 35, 1 (1988), 146–160.
[53] Lingzhen Kong, Lina Wang, Wenwen Gong, Chao Yan, Yucong Duan, and Lianyong Qi. 2022. LSH-aware multitype health data prediction with privacy preservation in edge environment. *World Wide Web* 25, 5 (2022), 1793–1808.
[54] Clyde P Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of efficient parallel algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.
[55] Selasi Kwashie, Jixue Liu, Jiuyong Li, Lin Liu, Markus Stumptner, and Lujing Yang. 2019. Certus: An Effective Entity Resolution Approach with Graph Differential Dependencies (GDDs). *PVLDB* 12, 6 (2019), 653–666.
[56] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[57] Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. 2021. Temporal Knowledge Graph Reasoning Based on Evolutional Representation Learning. In *SIGIR*.
[58] Ying Lin, Han Wang, Jiangning Chen, Tong Wang, Yue Liu, Heng Ji, Yang Liu, and Premkumar Natarajan. 2021. Personalized entity resolution with dynamic heterogeneous knowledge graph representations. *CoRR* abs/2104.02667 (2021).
[59] Ashley Little. 2020. Outdated Data: Worse Than No Data? *https://info.aldensys.com/joint-use/outdated-data-is-worse-than-no-data.*
[60] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. 2015. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*.
[61] Nour Mekki and Djamel Berrabah. 2022. Entity Resolution in graph databases: comparison study. In *EDiS*. 111–116.
[62] Meta. 2023. *https://about.meta.com.*
[63] Boris Muzellec, Julie Josse, Claire Boyer, and Marco Cuturi. 2020. Missing Data Imputation using Optimal Transport. In *ICML*.
[64] Daniel Obraczka, Jonathan Schuchart, and Erhard Rahm. 2021. EAGER: Embedding-Assisted Entity Resolution for Knowledge Graphs. *CoRR* abs/2101.06126 (2021).
[65] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. 2018. Robust Discovery of Positive and Negative Rules in Knowledge Bases. In *ICDE*. 1168–1179.

[66] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*. 5363–5370.

[67] Heiko Paulheim. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web* 8, 3 (2017), 489–508.

[68] Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. 2021. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. In *ICLR*.

[69] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. 2023. Extraction of Validating Shapes from very large Knowledge Graphs. *PVLDB* 16, 5 (2023), 1023–1032.

[70] Tara Safavi and Danai Koutra. 2020. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *EMNLP*. 8328–8350.

[71] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC*.

[72] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. 2019. ConvTransE implementation. *https://github.com/JD-AI-Research-Silicon-Valley/SACN*.

[73] Pengpeng Shao, Dawei Zhang, Guohua Yang, Jianhua Tao, Feihu Che, and Tong Liu. 2022. Tucker decomposition-based temporal knowledge graph completion. *Knowl. Based Syst.* 238 (2022), 107841.

[74] Victor S. Sheng and Jing Zhang. 2019. Machine Learning with Crowdsourcing: A Brief Summary of the Past Research and Future Directions. In *AAAI*. 9837–9843.

[75] Julie Smiley. 2016. Missing data and its impact on clinical research. *https://blogs.oracle.com/health-sciences/post/missing-data-and-its-impact-on-clinical-research*.

[76] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* 129 (2020), 249–260.

[77] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *WWW*. 697–706.

[78] Thomas Pellissier Tanon and Fabian M. Suchanek. 2021. Neural Knowledge Base Repairs. In *ESWC*.

[79] Thong Tran and Tru H. Cao. 2013. Automatic Detection of Outdated Information in Wikipedia Infoboxes. *Res. Comput. Sci.* 70 (2013), 211–222.

[80] Trung-Kien Tran, Mohamed H. Gad-Elrab, Daria Stepanova, Evgeny Kharlamov, and Jannik Strötgen. 2020. Fast Computation of Explanations for Inconsistency in Large-Scale Knowledge Graphs. In *WWW*. 2613–2619.

[81] Binghui Wang, Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. 2021. Certified Robustness of Graph Neural Networks against Adversarial Structural Perturbation. In *KDD*. 1645–1653.

[82] Tobias Weller and Heiko Paulheim. 2021. Evidential relational-graph convolutional networks for entity classification in knowledge graphs. In *CIKM*. 3533–3537.

[83] Eric Wong and J. Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *ICML*. 5283–5292.

[84] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In *MLSys*.

[85] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *ICML*. 5675–5684.

[86] Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel J. Kochenderfer, and Jure Leskovec. 2020. Handling Missing Data with Graph Representation Learning. In *NeurIPS*.

[87] Ge Zhang, Jia Wu, Jian Yang, Amin Beheshti, Shan Xue, Chuan Zhou, and Quan Z. Sheng. 2021. FRAUDRE: Fraud Detection Dual-Resistant to Graph Inconsistency and Imbalance. In *ICDM*.

[88] Qinggang Zhang, Junnan Dong, Keyu Duan, Xiao Huang, Yezi Liu, and Linchuan Xu. 2022. Contrastive Knowledge Graph Error Detection. In *CIKM*.

[89] Jing Zheng, Jian Liu, Chuan Shi, Fuzhen Zhuang, Jingzhi Li, and Bin Wu. 2017. Recommendation in heterogeneous information network via dual similarity regularization. *Int. J. Data Sci. Anal.* 3 (2017), 35–48.

[90] Zheng Zheng, Tri Minh Quach, Ziyi Jin, Fei Chiang, and Mostafa Milani. 2019. CurrentClean: Interactive Change Exploration and Cleaning of Stale Data. In *CIKM*. 2917–2920.

[91] Ziyue Zhong, Meihui Zhang, Ju Fan, and Chenxiao Dou. 2022. Semantics Driven Embedding Learning for Effective Entity Alignment. In *ICDE*. 2127–2140.

[92] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock. 2016. Unsupervised entity resolution on multi-type graphs. In *ISWC*.

[93] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph Contrastive Learning with Adaptive Augmentation. In *WWW*. 2069–2080.