Capturing Associations in Graphs

Wenfei Fan[1,2,3] Ruochun Jin[1] Muyang Liu[1] Ping Lu[3] Chao Tian[4] Jingren Zhou[4]

[1]University of Edinburgh [2]Shenzhen Institute of Computing Sciences, Shenzhen University
[3]Beijing Advanced Innovation Center for Big Data and Brain Computing [4]Alibaba Group

{wenfei@inf., ruochun.jin@, muyang.liu@}ed.ac.uk, luping5303@gmail.com, {tianchao.tc, jingren.zhou}@alibaba-inc.com

ABSTRACT

This paper proposes a class of graph association rules, denoted by GARs, to specify regularities between entities in graphs. A GAR is a combination of a graph pattern and a dependency; it may take as predicates ML (machine learning) classifiers for link prediction. We show that GARs help us catch incomplete information in schemaless graphs, predict links in social graphs, identify potential customers in digital marketing, and extend graph functional dependencies (GFDs) to capture both missing links and inconsistencies.

We formalize association deduction with GARs in terms of the chase, and prove its Church-Rosser property. We show that the satisfiability, implication and association deduction problems for GARs are coNP-complete, NP-complete and NP-complete, respectively, retaining the same complexity bounds as their GFD counterparts, despite the increased expressive power of GARs. The incremental deduction problem is DP-complete for GARs versus coNP-complete for GFDs. In addition, we provide parallel algorithms for association deduction and incremental deduction. Using real-life and synthetic graphs, we experimentally verify the effectiveness, scalability and efficiency of the parallel algorithms.

PVLDB Reference Format:
Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, Jingren Zhou. Capturing Associations in Graphs. *PVLDB*, 13(11): 1863-1876, 2020.
DOI: https://doi.org/10.14778/3407790.3407795

1. INTRODUCTION

Association rules [7] have been studied for relational data for decades and proven effective in market basket analysis, Web mining, intrusion detection, continuous production and bioinformatics, among others. When it comes to graphs, the need for studying association rules is more evident.

Example 1: Consider the following real-life examples.

(1) Marketing. Unlike traditional marketing strategies such as TV advertising, e-commerce marketing promotes products by association analysis of purchases and user behaviors, which are often represented as graphs. It has proven important: "the visits where the shopper clicked a recommendation comprise just 7% of visits, but drive an astounding 24% of orders and 26% of revenue" [58]. Moreover, associations play a vital role in recommendation systems [5, 15, 37].

For instance, graph G_1 of Fig. 1 depicts an e-commerce recommendation network [62]. A rule for marketing is as follows: if (a) a shopper Ada follows a store Uniqlo and clicks product Long-Sleeve Hoodie W sold by it, (b) Uniqlo also sells Denim Mini Skirt, which is combined with Long-Sleeve Hoodie W in some package deals, and (c) if the classes of the two products, Hoodie W and Mini Skirt, are correlated in women's shopping activities, then Ada may also be interested in Denim Mini Skirt; the link is not in G_1.

(2) Link prediction. Association rules help us predict links in social networks. People visiting the same place, having common friends and similar interests tend to develop friendship [48, 57]. For example, graph G_2 in Fig. 1 is taken from social network Gowalla [38]. It suggests the following: if (a) two persons Bob and Joe have a common friend Sue, (b) all of them like to visit cafe Beans, and (c) if Bob and Joe share the same interest as Sue, then Bob and Joe are likely to become friends; the link between Bob and Joe is absent in G_2.

(3) Incomplete information. Unlike relational tables, real-life graphs typically do not come with a schema. As a result, it is more common to find information missing from graphs. As indicated by G_3 of Fig. 1, in the knowledge graph adopted by e-commerce platforms [39], there exist clothing items (*e.g.,* Winter Dress) without brand or material. To make them complete items, the missing data need to be added.

(4) Catching both absent links and semantic errors. Graph functional dependencies have been studied [25, 21], referred to as GFDs. Like relational functional dependencies (FDs), GFDs are universal logic sentences to catch semantic inconsistencies. However, GFDs stop short of catching missing links, which have an existential semantics.

On the one hand, GFDs may fail to catch semantic errors when links are missing. Consider graph G_4 taken from DBPedia [35], in which the English and French Chapters return different populations for France. One can use a GFD to catch the inconsistency: if two records x and x' refer to the same nation, then they must have the same population. However, if the equivalent link between x and x' is missing, then this GFD cannot catch the error. On the other hand, with such inconsistencies, conventional logical rules fail to connect the nation records in G_5 of Fig. 1. The scale of the problem is far more staggering in schemaless graphs.

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407795

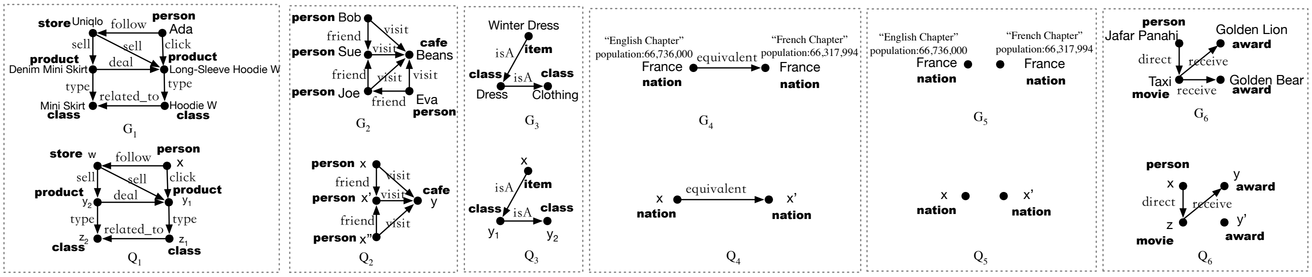


Figure 1: Example associations in real-life graphs

(5) Incorporating machine learning (ML). Association deduction requires both logic-based and ML-based methods. On the one hand, we can use ML classifiers to predict links above between x and x'. On the other hand, we can use logic to interpret ML predictions and help improve its accuracy. For instance, if an ML classifier "predicts" that movie Taxi receives Golden Bear and Golden Lion awards (see G_6 of Fig. 1), then we can conclude that the predication is wrong since the two film festivals require their participants to be "initial release" and no movie receives both awards. □

This example gives rise to several questions. What rules should we use to catch associations? Can we catch missing links and semantic inconsistencies at the same time? Is it possible to extend existing graph dependencies (*e.g.,* GFDs) to meet the requirements while striking a balance between the expressive power and complexity? Better yet, can we incorporate ML classifiers into the rules such that we can uniformly apply rule-based and ML-based methods? Putting these together, above all, can we make practical use of the rules to deduce associations in large-scale graphs?

Contributions & organization. This paper makes an effort to answer these questions, from foundation to practice.

(1) Rules (Section 3). We propose a class of graph association rules, denoted by GARs. GARs extend graph pattern association rules (GPARs) [23] with preconditions, and GFDs [25, 21] with limited existential semantics. Moreover, GARs may take embedding-based machine learning (ML) classifiers for link prediction as predicates, and thus provide a uniform framework to catch missing links and semantic errors in graphs, by *unifying rule-based and ML-based methods.*

(2) Deducing associations (Section 4). We study deducing associations from real-life graphs. We formalize the problem by extending the chase [45] with GARs, to uniformly apply rules and embedding-based ML classifiers. We show that the deduction has the Church-Rosser property, *i.e.,* the chase converges at the same answer no matter in which order the GARs are applied, even though the graphs may mutate.

(3) Complexity (Section 5). We study fundamental problems for graph associations, including (a) satisfiability to decide whether a set of GARs has no conflicts with each other; (b) implication to decide whether a GAR is entailed by a given set of GARs, to reduce redundant GARs; (c) association deduction to infer missing links and missing attributes in a real-life graph; and (d) incremental deduction to deduce changes to the associations in response to updates to graphs.

We show that while GARs increase the expressive power of GFDs, the satisfiability, implication and association deduction problems retain *the same complexity* as their counterparts for GFDs. The incremental deduction problem for GARs is slightly harder than for GFDs, DP-complete vs. coNP-complete, unless P = NP. That is, GARs indeed strike a balance between the complexity and expressivity.

(4) A parallel solution (Section 6). To make practical use of the association analysis, we parallelize the association deduction process by adopting the fixpoint computation model of GRAPE [26]. We show that the parallelization guarantees to converge at the same set of associations deduced.

Moreover, we provide a parallel incremental algorithm in response to updates. Real-life graphs frequently change, and it is costly to re-deduce associations starting from scratch when graphs change. We incrementally compute changes to associations, minimizing unnecessary recomputation.

(5) Experimental study (Section 7). Using real-life and synthetic graphs, we empirically verify the effectiveness, scalability and efficiency of our (incremental) deduction methods. We find the following. (1) GARs are effective in association deduction by unifying rule-based and ML-based methods. Our methods have F-measure above 88.3%, and outperform existing ML and rule-based methods by 21.3% and 28.2% in accuracy, respectively. (2) Our parallel deduction method is efficient and scalable; it is 18.1 times faster than existing deduction methods on graphs of 1.3 billion nodes and edges with 12 processors. (3) Incremental deduction performs better than the batch counterpart even when updates ΔG are up to 25% of G, *e.g.,* 4.3 times faster when $|\Delta G| = 10\%|G|$.

This paper focuses on (incremental) deduction of associations. Besides its immediate applications, the same techniques can be adapted to graph data cleaning [22], fraud detection [46] and annotation analysis [30]. For instance, putting this and [22] together, we can develop a uniform process to fix missing links and inconsistencies with certainty.

Proofs of the results of the paper can be found in [19].

Related work. We categorize related work as follows.

Association rules and graph dependencies. Association rules [7] aim to capture item relationships in transaction data, and have proven effective on relational datasets [60]. Similar rules have been applied on graphs [28] to analyze social networks, by extracting relations [16, 11]. GPARs [23, 24] define association rules directly on graphs, for graph data analysis [27, 50] and knowledge graph search [40].

Different from rule-based solutions, machine learning formalizes graph association analysis as the link prediction problem. Based on statistical learning, link prediction models learn vector embedding of each entity and relation [34]. They predict links over the embedding via additive functions [10], product-based functions [53, 61], or deep neural networks [47]. While these models have shown competitive performance, their prediction errors are unexplainable.

Graph functional dependencies have recently been proposed for RDF [8, 32, 17] and property graphs [25, 21, 20].

Expressed as universal logic sentences, these dependencies have been used to catch semantic inconsistencies in graphs. There has also been work on tuple-generating dependencies (TGDs [4]) on graphs [13, 14], which are defined with both universal and existential logic quantifiers.

The novelty of this work consists of the following.

(1) This work makes a first effort to incorporate ML classifiers into logic rules for association deduction. On the one hand, such rules plug in existing ML classifiers and improve the accuracy of association deduction. On the other hand, they can interpret links predicted by ML classifiers in logic.

(2) We propose a first framework to catch semantic inconsistencies and missing associations in the same process. Indeed, inconsistencies can also be modeled as erroneous associations (Section 4), and should be treated in a uniform framework for associations. That is why we opt to extend GFDs [25, 21] to catch missing links and attributes, rather than to define a class of new rules starting from scratch.

(3) GARs strike a balance between the expressivity and complexity, with necessary yet minimum extensions to GPARs and GFDs. It is well known that when universal logic rules and existential rules are put together, their static analyses are often undecidable, *e.g.,* the implication problem for TGDs (cf. [4]), and for functional dependencies and inclusion dependencies put together [12]. GARs enrich GFDs (of universal semantics) with limited existential semantics, while their satisfiability and implication problems are decidable in coNP and NP, respectively, the same as for GFDs.

While the complexity bounds for GARs are similar to GFD counterparts, the proofs are quite different (see Section 5).

(4) GARs extend GPARs [23] with preconditions. This work provides the first formulation of association deduction with chase, and the first fundamental results for reasoning about graph association rules, which were not studied in [23, 24].

Parallel deduction. Several parallel algorithms have been developed for graph pattern matching (*e.g.,* [26, 6, 9, 44, 49, 43, 33, 52, 42, 44]) and GFD checking [25, 22, 20], based on the following: (1) work unit distribution [25, 22, 49]; (2) data replication [20, 26, 9, 43]; (3) pattern decomposition and multiway join [6, 33, 52, 42]; and (4) pattern match expansion by fetching data and verifying edges [44, 55].

This work adopts a different approach. (a) We first develop two *sequential* algorithms for deduction and incremental deduction of associations. We then parallelize the algorithms following the fixpoint model of GRAPE, with convergence guarantees [26]. These depart from the prior algorithms on GFDs [25, 22, 20]. (b) We process a set of GARs at the same time, not a single pattern. Moreover, enforcing GARs may *mutate the topological structure of graphs.* In contrast, prior algorithms assume static graphs; they do not work for association deduction. (c) We propose a strategy to reduce redundant mutual effects between different types of updates in incremental deduction. (d) To the best of our knowledge, the incremental deduction algorithm also yields the first incremental graph repairing algorithm.

2. PRELIMINARIES

We start with a review of basic notations. We assume three countably infinite sets of symbols, denoted by Γ, Υ and U, for labels, attributes and constants, respectively.

Graphs. We consider directed labeled graphs, specified as $G = (V, E, L, F_A)$, where (a) V is a finite set of nodes; (b) $E \subseteq V \times \Gamma \times V$ is the set of edges, where $e = (v, \iota, v')$ denotes an edge from node v to v' that is labeled with $\iota \in \Gamma$; (c) each node $v \in V$ has label $L(v)$ from Γ; and (d) each node $v \in V$ 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 $a_i \in U$, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$, representing properties.

Patterns. A *graph pattern* is $Q[\bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of pattern nodes (resp. edges), (2) L_Q assigns a label $L_Q(u) \in \Gamma$ to each node $u \in V_Q$, (3) $\bar{x}$ is a list of distinct variables; and (4) μ is a bijective mapping from $\bar{x}$ to V_Q, *i.e.,* it assigns a distinct variable to each node v in V_Q. We allow wildcard '_' as a special label in $Q[\bar{x}]$.

For $x \in \bar{x}$, we use $\mu(x)$ and x interchangeably.

Example 2: Six patterns are given in Fig. 1. For example, pattern Q_1 shows that shop w sells products y_1 and y_2 of classes z_1 and z_2, respectively, y_1 and y_2 are linked in a special offer, z_1 and z_2 are related in order activities, and customer x follows shop w and clicks product z_1. Patterns Q_2-Q_6 in Fig. 1 can be interpreted similarly. □

Pattern matching. We adopt the homomorphism semantics following [21, 8, 14]. A *match* of pattern $Q[\bar{x}]$ in graph G is a mapping h from Q to G such that (a) for each node $u \in V_Q$, $L_Q(u) = L(h(u))$; and (b) for each $e = (u, \iota, u')$ in Q, $e' = (h(u), \iota, h(u'))$ is an edge in G. Here $L_Q(u) = L(h(u))$ if $L_Q(u)$ is '_', *i.e.,* wildcard matches an arbitrary label.

We denote the match as a vector $h(\bar{x})$, consisting of $h(x)$ for all $x \in \bar{x}$ in the same order as $\bar{x}$. Intuitively, $\bar{x}$ is a list of entities to be identified, and $h(\bar{x})$ is an instantiation for it.

3. GRAPH ASSOCIATION RULES

We now define graph association rules (GARs).

Literals. A *literal of pattern* $Q[\bar{x}]$ is one of the following: for variables $x, y \in \bar{x}$ and attributes $A, B \in \Upsilon$,

- *attribute literal* $x.A$;
- *edge literal* $\iota(x, y)$, where ι is a label in Γ;
- *ML literal* $\mathcal{M}(x, y, \iota)$, an ML classifier that returns true if and only if it predicts the existence of edge (x, ι, y);
- *variable literal* $x.A = y.B$; and
- *constant literal* $x.A = c$, where $c \in U$ is a constant.

GARs. A *graph association rule* (GAR) φ is defined as

$$Q[\bar{x}](X \to Y),$$

where $Q[\bar{x}]$ is a graph pattern, and X and Y are (possibly empty) conjunctions of literals of $Q[\bar{x}]$. We refer to $Q[\bar{x}]$ and $X \to Y$ as the *pattern* and *dependency* of φ, respectively.

Intuitively, the pattern Q in a GAR identifies entities in a graph, and the dependency $X \to Y$ is applied to the entities. Constant and variable literals $x.A = c$ and $x.A = y.B$ specify *value associations to* attributes, and attribute and edge literals $x.A$ and $\iota(x, y)$ enforce the existence of attributes and edges, *i.e., attribute and edge associations*, respectively.

Moreover, one can "plug in" an existing well-trained ML classifier $\mathcal{M}$ for link prediction, and treat it as a Boolean predicate, *i.e.,* $\mathcal{M}(x, y, \iota)$ is true if $\mathcal{M}$ predicts the existence of a link labeled ι from x to y, and false otherwise. As will be seen shortly, it allows us to employ embedding-based ML classifiers in logic rules, and interpret such classifiers in logic.

Example 3: One can use the GARs below to deduce associations described in Example 1, using patterns Q_1-Q_6 of Fig. 1.

(1) $\varphi_1 = Q_1[x, y_1, y_2, w, z_1, z_2](\emptyset \to Y_1)$, where Y_1 consists of an edge literal $\mathsf{like}(x, y_2)$. It says that if products y_1 and y_2 are sold by the same shop w and are connected in a package deal, their corresponding classes are related in buying activities, and if customer x clicks y_1 and follows shop w (specified in Q_1), then x is also a potential customer of product y_2.

(2) $\varphi_2 = Q_2[x, x', x'', y](X_2 \to Y_2)$, where X_2 is $x.\mathsf{interest} = x'.\mathsf{interest} \wedge x''.\mathsf{interest} = x'.\mathsf{interest}$, $\mathsf{interest}$ is an attribute of person entity, and Y_2 is $\mathsf{friend}(x, x'')$. It states that if x' is a friend of both x and x'', all of x, x' and x'' visit the same cafe y (specified in Q_2), and if the three share common interest (in X_2), then x and x'' are likely to become friends.

(3) $\varphi_3 = Q_3[\bar{x}](y_2.\mathsf{name}=\text{“Clothing”} \to Y_3)$, where Y_3 is defined with attribute literals $x.\mathsf{brand} \wedge x.\mathsf{material}$. It enforces each clothing entity to carry brand and material attributes.

(4) $\varphi_4 = Q_4[x, x'](\emptyset \to Y_4)$, where Y_4 is $x.\mathsf{population} = x'.\mathsf{population}$, and $\mathsf{population}$ is an attribute of nation entity. It says that records about the same nation should have the same population. It is a GFD [25] to catch inconsistencies.

(5) $\varphi_5 = Q_5[x, x'](X_5 \to Y_5)$, where X_5 is $x.\mathsf{name} = x'.\mathsf{name} \wedge \mathcal{M}(x, x', \mathsf{equivalent})$, and Y_5 is $\mathsf{equivalent}(x, x')$. It states that if two nations x and x' have the same name and they are predicted to be equivalent by an ML classifier (link predictor) $\mathcal{M}$, then the link $(x, \mathsf{equivalent}, x')$ should be added. It makes use of existing ML classifiers to catch associations.

(6) $\varphi_6 = Q_6[\bar{x}](X_6 \to \mathsf{false})$, where X_6 is $\mathcal{M}(z, y', \mathsf{receive}) \wedge y.\mathsf{name}=\text{“Golden Bear”} \wedge y'.\mathsf{name}= \text{“Golden Lion”}$. Here false is a Boolean constant expressed as $y.\mathsf{name}=c$ and $y.\mathsf{name}=d$ for distinct constants c and d. Intuitively, it says that a movie cannot receive both Golden Bear and Golden Lion awards. This suggests that if $\mathcal{M}(z, y', \mathsf{receive})$ returns true, then the classifier $\mathcal{M}$ should be further trained. □

Semantics. To interpret GAR $\varphi = Q[\bar{x}](X \to Y)$, we use the following notations. Denote by $h(\bar{x})$ a match of Q in a graph G, and l a literal of $Q[\bar{x}]$. We write $h(\mu(x))$ as $h(x)$, where μ is the mapping in Q from $\bar{x}$ to nodes in Q.

We say that $h(\bar{x})$ *satisfies* a literal l, denoted by $h(\bar{x}) \models l$, if the following condition is satisfied: (a) when l is $x.A$, attribute A exists at $h(x)$; (b) when l is $\iota(x, y)$, there is an edge with label ι from $h(x)$ to $h(y)$; (c) when l is $\mathcal{M}(x, y, \iota)$, the ML classifier $\mathcal{M}$ predicts an edge $(h(x), \iota, h(y))$; (d) when l is $x.A = y.B$, attributes A and B exist at $h(x)$ and $h(y)$, respectively, and $h(x).A = h(y).B$; and (e) when l is $x.A = c$, attribute A exists at $h(x)$, and $h(x).A = c$.

For a set X of literals, we write $h(\bar{x}) \models X$ if match $h(\bar{x})$ satisfies *all* the literals in X. If X (resp. Y) is $\emptyset$ (*i.e.,* true), then $h(\bar{x}) \models X$ (resp. $h(\bar{x}) \models Y$) for any match $h(\bar{x})$ of Q in G. We write $h(\bar{x}) \models X \to Y$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models Y$.

A graph G *satisfies* GAR φ, denoted by $G \models \varphi$, if for all matches $h(\bar{x})$ of Q in G, $h(\bar{x}) \models X \to Y$. Graph G *satisfies* a set Σ of GARs, denoted by $G \models \Sigma$, if $G \models \varphi$ for all $\varphi \in \Sigma$.

Example 4: Consider graph G_2 of Fig. 1 and GAR φ_2 in Example 3. Then $G_2 \not\models \varphi_2$, since there exists a match h_1: $x \mapsto$“Bob”, $x' \mapsto$“Sue”, $x'' \mapsto$ “Joe”, $y \mapsto$“Beans”, such that $h_1(\bar{x}) \models X_2$, but there exists no edge (“Bob”, friend, “Joe”) in G_2. Hence $h_1(\bar{x}) \not\models X_2 \to Y_2$, *i.e.,* $h_1(\bar{x})$ witnesses $G_2 \not\models \varphi_2$. Similarly, $G_i \not\models \varphi_i$ for other $i \in [1, 6]$. □

Special cases. We single out three special cases of GARs.

(1) GFDs and graph entity dependencies (GEDs) [25, 21] are GARs defined with constant and variable literals only, assuming that node id is a special attribute. That is, GARs extend GFDs and GEDs with the existential semantics for attributes and edges, and by allowing ML classifiers as predicates. For instance, φ_4 of Example 3 is a GFD but all the other GARs there cannot be expressed as GFDs or GEDs. GARs can catch both missing links and semantic errors, as opposed to GFDs and GEDs that detect inconsistencies only.

(2) GPARs [23] are GARs $Q[\bar{x}](\emptyset \to \iota(x, y))$, in which $X \to Y$ specifies no precondition X and Y is a single edge literal $\iota(x, y)$. In contrast to GARs, GPARs do not allow ML classifiers. No GAR in Example 3 is expressible as GPARs.

(3) GARs unify logic and ML methods. On the one hand, $Q[\bar{x}](\mathcal{M}(x, y, \iota) \to \iota(x, y))$ plugs in an ML link predictor $\mathcal{M}(x, y, \iota)$, *e.g.,* GAR φ_5 of Example 3. On the other hand, GARs $Q[\bar{x}](\psi \to \mathcal{M}(x, y, \iota))$ help us interpret why $\mathcal{M}(x, y, \iota)$ predicts true with condition ψ. For instance, the ML classifier $\mathcal{M}$ in φ_6 may be interpreted as a rule like $Q_6[\bar{x}](z.\mathsf{name} = y'.\mathsf{movie\_name} \wedge z.\mathsf{director} = y'.\mathsf{movie\_director} \to \mathcal{M}(z, y', \mathsf{receive}))$, by extracting the attributes from the textual description of movies and awards.

ML classifiers in GARs. GARs support embedding-based ML classifiers for link prediction. Having sets of entities and relations denoted by $\mathcal{E}$ and $\mathcal{R}$, respectively, these ML classifiers view each link in a graph as a triple (h, r, t), where $h \in \mathcal{E}$ is the head, $r \in \mathcal{R}$ is the relation and $t \in \mathcal{E}$ refers to the tail of the triple. Given positive/negative triples as training data, the classifiers apply tensor factorization to learn vector representations of entities and relations. During the learning process, with a predefined similarity function, the positive triples guide the classifier to embed their vectors similar while the negative triples force theirs to become dissimilar. Here all types of entities and relevant information (all relevant attributes and edges) are considered.

Once the training completes, such an ML classifier $\mathcal{M}$ behaves just like a Boolean function. Given two entities h', t' and a relation r', $\mathcal{M}(h', r', t')$ returns a Boolean value. That is, $\mathcal{M}$ maps h', t' and r' to precomputed vectors $v_{h'}$, $v_{t'}$ and $v_{r'}$ as their embedding. Then it feeds these vectors to the similarity function, and returns true (resp. false) if the score is above (resp. below) the threshold. The hypothesis of such ML link predictor is that all entities and relations have been covered by the training data and learned by $\mathcal{M}$ [10]; thus $\mathcal{M}$ can find embedding of h', t' and r', and predicts whether h' is linked to t' with an r'-edge. That is how the state-of-the-art embedding-based SimpIE [34] and CompIEx [53] work.

4. DEDUCING ASSOCIATIONS

One of the central issues of the study is to deduce associations. There are two types of associations: (a) associations between entities (edge literals) and associations of attributes to entities (attribute literals); and (b) associations of values to attributes (variable and constant literals).

We model association deduction by chasing graphs with GARs. Below we first extend the chase [45] from relations to graphs (Section 4.1) and then prove its Church-Rosser property (Section 4.2). Based on these, we will formulate the association deduction problem in the next section.

4.1 Chasing with GARs

Consider a graph $G = (V, E, L, F_A)$ and a set Σ of GARs.

Chase graphs. A *chase graph* G_c is (V, E_c, L, F_{A_c}), where V and L are from G, $E_c = E \cup \Delta E_c$, and $F_{A_c} = F_A \cup \Delta F_{A_c}$. Here ΔE_c includes edges added by ML literals and edge literals during the chase, and ΔF_{A_c} includes attributes added by attribute, constant and variable literals.

Chasing. We define *a chase step of* G *by* Σ *at* G_c as

$$G_c \Rightarrow_{(\varphi,h)} G'_c,$$

where $\varphi = Q[\bar{x}](X \to Y)$ is a GAR in Σ and $h(\bar{x})$ is a match of Q in G_c such that (a) $h(\bar{x}) \models X$, and (b) G'_c extends G_c by enforcing one literal $l \in Y$ if $h(\bar{x}) \models l$ does not yet hold. More specifically, based on l, G'_c is defined as follows.

- If l is $x.A$, then G'_c extends G_c by adding attribute A to $\Delta F_{A_c}(h(x))$ with a special value "#" if $A \notin F_A(h(x))$. If A is already in $F_A(h(x))$, its value remains unchanged.
- If l is $\iota(x, y)$, then G'_c extends G_c with edge $(h(x), \iota, h(y))$.
- If l is $\mathcal{M}(x, y, \iota)$, then G'_c extends G_c by adding edge $(h(x), \iota, h(y))$. As a byproduct, it suggests to set $\mathcal{M}(x, y, \iota)$ true, *i.e.,* it provides feedback to ML predictor $\mathcal{M}$.
- If l is $x.A = y.B$, then G'_c extends G_c by (a) adding attributes A to $\Delta F_{A_c}(h(x))$ and B to $\Delta F_{A_c}(h(y))$ if the attributes are not there, and (b) letting $h(x).A = h(y).B$.
- If l is $x.A = c$, then G'_c extends G_c by adding attribute A to $\Delta F_{A_c}(h(x))$ if $A \notin F_A(h(x))$, and letting $h(x).A = c$.

Consistency. Conflicts may emerge in a chase step. We say that chase step $G_c \Rightarrow_{(\varphi,h)} G'_c$ is *invalid* if when it enforces literal l, either (a) l is $x.A = y.B$, but $h(x).A = c$ and $h(y).B = d$ are in G_c for distinct c and d, or (b) l is $x.A = c$, $h(x).A = d$ is in G_c and $c \neq d$. Otherwise the step is *valid*. We say that G'_c is *inconsistent* if either (a) or (b) happens.

Note that edge and ML literals do not incur inconsistencies as multiple edges can co-exist between a pair of nodes.

Chasing sequences. We start with $G_{c_0} = G$ in which ΔF_{A_c} and ΔE_c are both $\emptyset$. A *chasing sequence* ρ of G by Σ is

$$G_{c_0}, \ \ldots, \ G_{c_k},$$

where for all $i \in [0, k-1]$, there exist a GAR $\varphi = Q[\bar{x}](X \to Y)$ in Σ and a match h of graph pattern Q in G_{c_i} such that $G_{c_i} \Rightarrow_{(\varphi,h)} G_{c_{i+1}}$ is a valid chase step.

The sequence is *terminal* if there exist no GAR $\varphi \in \Sigma$ and match h of pattern Q of φ in G_{c_k} such that chase step $G_{c_k} \Rightarrow_{(\varphi,h)} G_{c_{k+1}}$ is valid and can extend G_{c_k}. More specifically, the chase terminates in one of the following two cases:

(a) G_{c_k} cannot be expanded and G_{c_i} is consistent ($i \in [0, k]$); if so, the chasing sequence is *valid* and its result is G_{c_k}; or

(b) at some step i, G_{c_i} is inconsistent; if so, the chasing sequence is *invalid*, and the result is undefined $\perp$.

Prior work on chasing graphs [21, 22] mainly changes attribute values. In contrast, *the topological structure* of G_c may be changed by new edges and attributes added when chasing GARs. Hence when G_c is extended to G'_c, we have to check new possible matches of graph patterns in GARs.

Example 5: Consider the graph G_2 shown in Fig. 1. Assume that Σ consists of only one GAR φ_2 in Example 3. From $G_{c_0} = G_2$, we have the following chase steps:

(1) $G_{c_0} \Rightarrow_{(\varphi_2,h_1)} G_{c_1}$, where match h_1 is given in Example 4; and G_{c_1} extends G_{c_0} with edge ("Bob", friend, "Joe");

(2) $G_{c_1} \Rightarrow_{(\varphi_2,h_2)} G_{c_2}$, where h_2 is defined as follows: $x \mapsto$ "Bob", $x' \mapsto$ "Joe", $x'' \mapsto$ "Eva", $y \mapsto$ "Beans", and G_{c_2} extends G_{c_1} with edge ("Bob", friend, "Eva") using φ_2. Note that match h_2 exists in the mutated chase graph G_{c_1} *only after* the edge ("Bob",friend,"Joe") is added in step (1). □

4.2 The Church Rosser Property

A major concern is whether the chase always terminates with the same result. Following [4], we say that chasing with GARs is *Church-Rosser* if for all graphs G and all sets Σ of GARs, all chasing sequences of G by Σ are terminal and converge at the same result, regardless of what GARs in Σ are used and in what order the GARs are applied.

The analysis of chasing with GARs is harder than the one in [21], since the ML predictors depend on the structure of the graph, which in turn affect the prediction and the chase.

Theorem 1: *Chasing with* GARs *is Church-Rosser.* □

Proof sketch: The proof consists of two steps. (1) The size of chase graph G_{c_i} is bounded by $|G|^2|\Sigma|$, since between each pair of nodes, the labels of new edges are constrained by GARs in Σ, and hence at most $|\Sigma|$ edges can be added; similarly for attributes added and values changed. Since each chase step makes at most one change, the length of any chasing sequence is at most $4|G|^2|\Sigma|$. (2) All chasing sequences terminate at the same result. If there exist two terminal sequences having different results, then one of them is not terminal, a contradiction. As opposed to the chase with GEDs [21], here we have to show that the prediction of ML classifier remains stable during the chase; to this end, we need to exploit the property of ML classifiers given in Section 3 (*i.e.,* all entities and relations in the graph are covered by the training data, and the embedding vectors of entities and relations will not change after the graph is extended), which justifies our choice of ML classifiers. Moreover, we have to find new matches when the chase graph is expanded with new edges; the graph is no longer static. □

By Theorem 1, we define *the result of chasing* G *by* Σ as the result of any terminal chasing sequence of G by Σ, denoted by $\mathsf{Chase}(G, \Sigma)$. If the sequence is valid, $\mathsf{Chase}(G, \Sigma)$ has the form of G_c. We refer to edges and attributes that are in G_c but not in G as *deduced associations* of G by Σ. Intuitively, they are missing links and attributes. We denote by $\mathsf{deduced}(G, \Sigma)$ the set of all such deduced associations.

As shown in Section 3, we can use deduced associations to retrain $\mathcal{M}$, improve its accuracy and explain its prediction.

5. FUNDAMENTAL PROBLEMS

We next settle the satisfiability, implication, association deduction and incremental deduction problems. Our main conclusion is that for GARs, these problems either retain the same complexity as for GFDs, or are slightly harder than for GFDs, despite the increased expressivity of GARs. However, the proofs are rather different, to cope with, *e.g.,* unexpected conflicts introduced by ML classifiers.

Satisfiability. The *satisfiability* problem is as follows.

- Input: A set Σ of GARs.
- Question: Does there exist a graph G such that $G \models \Sigma$ and for each GAR $Q[\bar{x}](X \to Y) \in \Sigma$, Q has a match in G?

Intuitively, this is to ensure that Σ is sensible and all GARs can be simultaneously applied without conflicts.

For GFDs, the satisfiability problem is coNP-complete [21]. We next show that this problem is no harder for GARs.

Theorem 2: *The satisfiability problem is* coNP*-complete.* □

Proof sketch: (1) For the upper bound, given a set Σ of GARs, we define a canonical graph G_Σ by combining all patterns in Σ into one. We show that Σ is satisfiable if and only if $\mathsf{Chase}(G_\Sigma, \Sigma)$ is consistent and G_Σ is no larger than Σ. As opposed to the proofs for GEDs [21] and other extensions of GFDs [20], (a) when constructing G_Σ, we have to take special care of wildcards to avoid conflicts introduced by predictions of ML classifiers; and (b) take newly deduced edges into account when checking the consistency of $\mathsf{Chase}(G_\Sigma, \Sigma)$. Based on the characterization, we give an NP algorithm to check whether Σ is not satisfiable. (2) The lower bound follows from the coNP-completeness of the satisfiability problem for GFDs [21], since GFDs are a special case of GARs. □

Implication. A set Σ of GARs *implies* a GAR φ, denoted by $\Sigma \models \varphi$, if for all graphs G, if $G \models \Sigma$ then $G \models \varphi$. That is, φ is a logical consequence of Σ and hence, is redundant.

The *implication problem* is stated as follows.

○ Input: A set Σ of GARs and a GAR φ.
○ Question: $\Sigma \models \varphi$?

The need for studying this problem is evident, to remove redundant rules and hence speed up deduction process.

The good news is that the implication analysis of GARs has the same complexity as its counterpart for GFDs [21], as opposed to TGDs [4]. This is because (1) chasing with GARs does not generate new nodes; (2) while GARs enforce the existence of edges and attributes, the new additions are confined to those specified in GARs only. Taken together, these ensure that chasing with GARs will end up with a finite graph. In contrast, chasing with TGDs [4, 13, 14] may lead to infinite graphs and hence may not terminate.

Theorem 3: *The implication problem is* NP*-complete.* □

Proof sketch: (1) It is NP-hard since GFD implication is NP-complete [21] and GARs subsume GFDs as a special case. (2) For the upper bound, given a set Σ of GARs and a GAR $Q[\bar{x}](X \rightarrow Y)$, we build another canonical graph G_Q with Q, and show that $\Sigma \models \varphi$ if and only if either X is inconsistent or all literals in Y can be deduced from $\mathsf{Chase}(G_Q, \Sigma)$. Similar to the proof of Theorem 2, here we also deal with possible conflicts introduced by ML classifiers and graph mutation during the chase. Based on the characterization we then develop an NP algorithm to check whether $\Sigma \models \varphi$. □

Deduction. To simplify the discussion, we focus on deducing missing attributes and missing links, although the techniques developed in this paper can be readily used to deduce all associations, including values associated to attributes. That is, GARs can deduce missing links/attributes and correct inconsistencies in the same framework.

Consider a graph $G = (V, E, L, F_A)$. For a node $v \in V$ and an attribute $A \in \Upsilon$, if $v.A$ does not exist in G, we refer to $v.A$ as a *candidate attribute of* v *in* G. Similarly, for nodes $v_1, v_2 \in V$ and label $\iota \in \Gamma$, if (v_1, ι, v_2) is not in G, we refer to it as a *candidate edge of* G. We refer to such $v.A$ and (v_1, ι, v_2) as *candidate associations* of G, denoted by α.

The *association deduction* problem is stated as follows.

○ Input: Graph G, GARs Σ, and a candidate association α.
○ Question: Is α a deduced association of G by Σ, *i.e.,* whether $\alpha \in \mathsf{deduced}(G, \Sigma)$?

This problem is to settle the complexity of computing $\mathsf{deduced}(G, \Sigma)$, the set of all links and attributes that are missing from graph G and are deduced by the set Σ of GARs.

A similar problem is studied in [22], to deduce value associations $v.A = c$ or $v.A = v'.B$ using GFDs [21]. That problem is known NP-complete [22]. Below we show that deducing associations with GARs is no harder.

Theorem 4: *The association deduction problem for* GARs *is* NP*-complete.* □

Proof sketch: (1) We give an NP algorithm that guesses a chasing sequence $G_{c_0}, \ldots, G_{c_k}$ of bounded length, and checks whether α is in G_{c_k}. Its correctness follows from the bound on chasing sequences given in the proof of Theorem 1. (2) We show that the problem is NP-hard by reduction from the 3-colorability problem, which is NP-complete [29]. The latter problem is to decide, given an undirected graph $G_1 {=} (V_1, E_1)$, whether there exists a 3-coloring ν of G_1 such that for each edge $(u, v) \in E_1$, $\nu(u) \neq \nu(v)$. □

Incremental deduction. We consider *batch updates* ΔG to graph G, which are sequences of *unit updates*:

○ edge insertion ($\mathsf{insert}\ e$), possibly with new nodes, and
○ edge deletion ($\mathsf{delete}\ e$), along with endpoints of degree 0.

These can simulate modifications of *e.g.,* edge labels.

We use $G \oplus \Delta G$ to denote the graph G updated by ΔG.

We use $\mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$ to denote the set of *changes* to the set $\mathsf{deduced}(G, \Sigma)$ of associations in response to updates ΔG, *i.e.,* associations that are either in $\mathsf{deduced}(G, \Sigma)$ but not in $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$, or vice versa.

The *incremental deduction* problem is stated as follows.

○ Input: A graph G, a set Σ of GARs, batch updates ΔG to G, and a candidate association α of G or $G \oplus \Delta G$.
○ Question: Is $\alpha \in \mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$?

The need for studying this problem is evident. It is costly to compute $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$ starting from scratch, by Theorem 4. Hence we want to incrementally compute the changes to $\mathsf{deduced}(G, \Sigma)$ such that $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$ = $\mathsf{deduced}(G, \Sigma) \oplus \mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$ by making maximum reuse of $\mathsf{deduced}(G, \Sigma)$. When ΔG is small, often so is $\mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$, which is less costly to compute.

A related problem was studied for GFDs, to decide whether a match $h(\bar{x})$ is a violation of GFDs in $G \oplus \Delta G$ but not in G, or vice versa [20]. It is shown coNP-complete. But the incremental deduction for GARs is slightly harder.

Theorem 5: *The incremental deduction problem is* DP*-complete for* GARs*, and remains* DP*-hard when either graph* G *or updates* ΔG *to* G *has a constant size.* □

The increased complexity arises from the following. Given a match $h(\bar{x})$ in graph G (resp. $G \oplus \Delta G$), we can check if $h(\bar{x})$ is an old (resp. new) violation of GFDs in PTIME by directly inspecting $h(\bar{x})$ in $G \oplus \Delta G$ (resp. G). In contrast, for an association α in $\mathsf{deduced}(G, \Sigma)$ (resp. $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$) with GARs, we need to inspect the entire chasing sequence to verify that α is not in $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$ (resp. $\mathsf{deduced}(G, \Sigma)$), which requires an NP step and a coNP step.

Proof sketch: (1) To check if $\alpha \in \mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$, an algorithm is as follows: (a) check whether $\alpha \in \mathsf{deduced}(G, \Sigma)$ or $\alpha \in \mathsf{deduced}(G \oplus \Delta G, \Sigma)$; if so, continue; otherwise, return false; (b) check whether $\alpha \notin$ deduced(G, Σ) or $\alpha \notin$ deduced$(G \oplus \Delta G, \Sigma)$; if so, return true; otherwise, return false. The correctness follows from the statement of the incremental deduction problem and the following property of set theory: $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B) = (A \cup B) \cap (\overline{A} \cup \overline{B})$, where A and B are two sets. The algorithm is in DP as step (a) is in NP and step (2) is in coNP by Theorem 4.

(2) We show the DP-hardness by reduction from the critical 3-colorability problem, which is DP-complete [41]. The latter is to decide, given an undirected graph G_1, whether G_1 is not 3-colorable, but deleting any node makes G_1 3-colorable (see proof of Theorem 4 for 3-colorability). The reduction uses a constant-size G and ΔG of a single edge insertion. □

6. PARALLEL DEDUCTION ALGORITHM

In this section we show how to deduce associations with GARs in parallel by using the computation model of GRAPE [26]. We first review the GRAPE model (Section 6.1). We then provide algorithms for parallel association deduction (Section 6.2) and incremental deduction (Section 6.3).

6.1 Graph Centric Parallelization

Employing a *master* P_0 and a set of n *workers* (processors) $P_1, \ldots, P_n$, GRAPE operates on a graph G that is fragmented into $(F_1, \ldots, F_n)$ by a partitioner picked by users. For $i \in [1, n]$, each worker P_i maintains a fragment F_i in G.

PIE program. To answer a class $\mathcal{Q}$ of queries on graphs, GRAPE takes a PIE *program* (PEval, IncEval, Assemble) that consists of three (existing) sequential algorithms as follows.

- PEval is a sequential algorithm for $\mathcal{Q}$ that given query $Q \in \mathcal{Q}$ and graph G, computes answers $Q(G)$ to Q in G.
- IncEval is a sequential incremental algorithm for $\mathcal{Q}$ that given Q, G, $Q(G)$ and updates M to G, computes changes ΔO to $Q(G)$ such that $Q(G \oplus M) = Q(G) \oplus \Delta O$.
- Assemble collects partial answers computed locally at each worker by PEval and IncEval, and combines them into a complete answer; it is typically simple.

The only additions to existing sequential algorithms are the following. (1) PEval declares a set $\bar{x}_i$ of *update parameters* for each fragment F_i, which are status variables of "border nodes" of F_i, *e.g.,* nodes having edges from or to another fragment F_j (assuming edge-cut partition). (2) PEval also defines an aggregate function f_{aggr} to resolve conflicts, when the status variable of a node is given multiple values by different workers. These parameters are shared with IncEval.

Parallel computation. Given a query $Q \in \mathcal{Q}$, GRAPE posts the same Q to all workers. Then a PIE program is executed in supersteps under BSP model [54], as follows.

*(1) Partial evaluation (*PEval*)*. In the first superstep, PEval computes $Q(F_i)$ at each worker P_i on F_i locally, in parallel for all $i \in [1, n]$. Then, each worker generates a message consisting of update parameters $\bar{x}_i$ and sends it to master P_0.

*(2) Incremental computation (*IncEval*)*. In the following supersteps, the partial answers $Q(F_i)$'s are iteratively updated by IncEval. More specifically, (a) master P_0 applies f_{aggr} to messages from the last superstep, which resolves conflicts. Then these aggregated values are routed to relevant workers. (b) Upon receiving the message M_i, worker P_i *incrementally* computes $Q(F_i \oplus M_i)$ with IncEval in parallel for $i \in [1, n]$, by *treating M_i as updates*. At the end of each superstep, worker P_i sends a message to P_0 that consists of *changes* to the update parameters $\bar{x}_i$ of F_i just like in PEval.

```
Input: Fragment $F_i = (V_i, E_i, L_i, F_{A_i})$ and a set $\Sigma$ of GARs.
Output: Set $Q(F_i)$ of missing links and attributes of $F_i$ by $\Sigma$.
Declaration: for each node $v \in V_i$, two variables $v$.link and
             $v$.attr; and an additional variable $F_i.H$;
$\Psi \leftarrow \Sigma$;   $\mathcal{C}_V \leftarrow V_i$;   $F_i.H \leftarrow \emptyset$;
repeat
    $\Delta F_c \leftarrow \emptyset$;
    for each GAR $\varphi = Q[\bar{x}](X \rightarrow Y) \in \Psi$ do
        extract a set $\mathcal{T}$ of partial matches $h(\bar{x}_p)$ for $Q$
            s.t. $X$ (resp. $Y$) can be (resp. cannot be) satisfied;
        $(A_c, H_p) \leftarrow$ ExpandAssoc$(\varphi, \mathcal{T}, \mathcal{C}_V, F_i)$;
        $\Delta F_c \leftarrow \Delta F_c \cup A_c$;   $F_i.H \leftarrow F_i.H \cup H_p$;
    update $F_i$ with $\Delta F_c$;
    adjust $\mathcal{C}_V$ using nodes of $\Delta F_c$;   $\Psi \leftarrow$ SuccGAR$(\Sigma, \Delta F_c)$;
until $\Delta F_c = \emptyset$
$Q(F_i)$ stores the deduced associations;
```

Figure 2: PEval for program PDeduce

(3) Termination. The process proceeds until it reaches a fixpoint, *i.e.,* no more changes to update parameters. Assemble is then invoked to combine all partial answers into $Q(G)$.

PIE programs guarantee to converge at correct answers under a monotone condition as long as the sequential PEval, IncEval and Assemble are correct [26].

6.2 Parallel Association Deduction

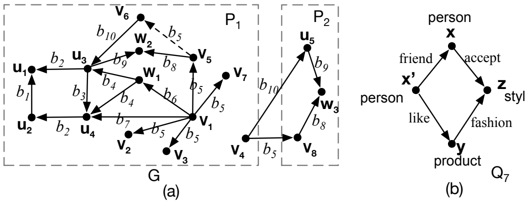
We next provide a PIE program, denoted by PDeduce. Given a fragmented graph G and a set Σ of GARs, it computes deduced(G, Σ). We give its PEval, IncEval and Assemble, which are parallelized as described in Section 6.1.

Challenges. As indicated in Section 4, a major task for deducing associations is to compute homomorphic mappings. Most subgraph matching methods preprocess graphs to build static indices, and enumerate matches by accessing candidates in the indices. However, these do not work in our setting for the following reasons. (1) During the chase, graphs are *mutated* and new matches are introduced at runtime, as opposed to static graphs and indices. (2) Prior methods often take a single graph pattern as input and find its matches. In contrast, the chase handles a set of GARs, and PDeduce has to decide which GARs to use and in what order the GARs are applied. (3) Even for a single pattern in a GAR, PDeduce needs to identify only a subset of matches that make missing associations, not all the matches.

In light of these, we propose to (1) compute matches only for patterns from *active* GARs, in an *incremental* manner; (2) use a *dynamic* matching order and simple indices that are *dynamically* maintained; and (3) employ an *association-guided* strategy to prune matches. These help us avoid checking useless matches that do not contribute to deduced(G, Σ).

To simplify the discussion, we assume that graphs are partitioned via edge-cut and all the patterns are connected.

PEval. PEval of PDeduce is given in Fig. 2. It takes a set Σ of GARs and a fragment F_i of graph G as input, and deduces a set $Q(F_i)$ of associations pertaining to F_i with Σ. It employs two status variables v.link and v.attr for each node v in F_i, recording v's adjacent edges and attribute values, respectively. It also uses a "global" status variable $F_i.H$ to store *partial matches* of the patterns in Σ that involve nodes residing at other workers, where a partial match maps only



(a) (b)

Figure 3: Example graph and pattern

a subset of pattern nodes. The update parameters of F_i include (a) $F_i.H$ to pass partial matches to other workers, and (b) $v.\mathsf{link}$ and $v.\mathsf{attr}$ of border nodes v to reconcile values, where border nodes are those that are within $\max_{Q\in\Sigma}|Q|$ hops of the nodes on edges crossing different fragments.

Algorithm PEval iteratively applies *active* GARs in Σ, guided by *active* nodes in fragment F_i (lines 2-10). Here a GAR (resp. node) is *active* if it can be enforced (resp. involves in the mapping) in a chase step for deducing *new* associations in the current iteration. The active GARs and nodes are collected in sets Ψ and $\mathcal{C}_V$, initially Σ and V_i, respectively (line 1). For each active GAR, it first extracts a set $\mathcal{T}$ of partial matches under certain conditions (line 5), and then completes them and deduces associations A_c via procedure ExpandAssoc (line 6). At the end of each iteration, it updates F_i with the new associations ΔF_C that are accumulated during this iteration (line 8), and adjusts Ψ and $\mathcal{C}_V$ for the next iteration (line 9). The iterations proceed until no new associations can be deduced (line 10). The associations deduced in the process are stored in $Q(F_i)$.

PEval employs the following new techniques.

Indices. We maintain (a) an index on each pattern node label ι (except wildcard) that occurs in Σ to fetch nodes labeled with ι in F_i; (b) an index on triples $\langle v, \iota, \eta\rangle$ to fetch edges incident to node v that are labeled ι and link to nodes labeled η. The index on the triples is *dynamically* updated when newly deduced edges are added to fragment F_i.

Match extraction. For an active GAR $Q[\bar{x}](X \to Y)$, PEval maps pattern nodes $\bar{x}_p$ ($\bar{x}_p \subseteq \bar{x}$) in literals X and Y to nodes in F_i, to extract partial matches $h(\bar{x}_p)$ (line 5), so that $h(\bar{x}_p)$ can (resp. cannot) satisfy X (resp. Y). This is conducted by using the index on pattern node labels and choosing nodes within $|Q|$ hops of the active nodes $\mathcal{C}_V$, by the locality of pattern matching. One can verify that only partial matches of this form can contribute to new associations.

Match completion. Procedure ExpandAssoc completes partial match $h(\bar{x}_p)$ by iteratively mapping the remaining $\bar{x}\setminus\bar{x}_p$ to nodes in F_i, following a *dynamic* candidate-size order [31] (line 6). That is, each time it maps a pattern node u that is connected to one of the matched pattern nodes, and currently has the *minimum* number of candidates. The candidates are inspected using the index on relevant triples, and each extended partial match should not satisfy $X \to Y$.

Once the partial $h(\bar{x}_p)$ is extended to a complete match $h(\bar{x})$ and $h(\bar{x})$ includes active nodes of $\mathcal{C}_V$, it deduces relevant associations directly and prunes all subsequent extensions of $h(\bar{x}_p)$ when pattern nodes in Y are already mapped in $h(\bar{x}_p)$ (*i.e., association-guided* pruning). ExpandAssoc also returns a set H_p of partial matches that involve border nodes and hence need to be expanded at other workers. The status variable $F_i.H$ is extended with partial matches H_p (line 7).

Active GARs and nodes. After each iteration, we revise $\mathcal{C}_V$ with those nodes involved in the newly deduced associations ΔF_c and derive active GARs by procedure SuccGAR for the next iteration (line 9). Extending templates that generalize nodes to their labels [22], SuccGAR picks such active GARs that have the same templates in their preconditions (or pattern edges) as that of the associations in ΔF_c. For instance, a GAR becomes active if it has a literal $x.A = y.B$ in its X and there is a new association $v.A = 1$ with $L(v) = L_Q(x)$.

```
Input: Fragment $F_i=(V_i, E_i, L_i, F_{A_i})$, GARs $\Sigma$, message $M_i$.
Output: Missing associations $Q(F_i \oplus M_i)$ deduced.
Declaration: Message $M_i = \{v.A, (v, \iota, v') \mid v, v' \in V_i,\ v.A$ and
             $(v, \iota, v')$ changed$\} \cup \{h(\bar{x}_p) \mid h(\bar{x}_p)$ is a partial match
             involving nodes in $V_i\}$
1.  collect nodes (resp. changes) of $M_i$ into $\mathcal{C}_V$ (resp. $\Delta F_c$);
2.  $\Psi \leftarrow \mathsf{SuccGAR}(\Sigma, \Delta F_c) \cup \{\varphi \mid \exists h(\bar{x}_p) \in M_i, h(\bar{x}_p)$ is
      a partial match of the pattern of $\varphi\}$;    $F_i.H \leftarrow \emptyset$;
3.  update $F_i$ with $\Delta F_c$;
4.  apply active $\Psi$ on $F_i$ iteratively to deduce new associations;
5.  $Q(F_i \oplus M_i)$ stores the deduced associations that are
      accumulated over supersteps;
```

Figure 4: IncEval for program PDeduce

Example 6: A fragmented graph G is shown in Fig. 3(a) (excluding dotted edge), where v_1 to v_8 denote persons, u_1 to u_2 are classes, u_3 to u_5 are products, w_1 denotes a shop and w_2 to w_3 are styles; labels b_1 to b_{10} are related_to, type, deal, sell, friend, follow, click, accept, fashion and like, respectively.

Consider a set Σ of GARs including φ_1 of Example 3 and $\varphi_7 = Q_7[\bar{x}](\emptyset \to \mathsf{like}(x, y))$, where Q_7 is depicted in Fig. 3(b).

Given G and Σ, a partial match h_1' of Q_1 from φ_1 is extracted by PEval at worker P_1 in the first iteration, where $x \mapsto v_1$, $w \mapsto w_1$, $y_1 \mapsto u_4$, $y_2 \mapsto u_3$, $z_1 \mapsto u_2$ and $z_2 \mapsto u_1$. Since h_1' is already a complete match and $h_1' \not\models Y_1$, it adds association $(v_1, \mathsf{like}, u_3)$ at P_1. The other partial matches with $x \mapsto v_1$ and $y_2 \mapsto u_3$ are dropped by association-guided pruning.

Then φ_7 is treated as an active GAR for the second iteration since Q_7 has a pattern edge (x', like, y) sharing the same template with the newly deduced association. PEval next extracts a partial match h_2' for Q_7 that maps x' (resp. y) to active node v_1 (resp. u_3). When completing h_2' by procedure ExpandAssoc, z of Q_7 is mapped ahead of x since z has only one candidate w_2 whereas x has four (v_2, v_3, v_5, and v_7). In fact, only a singe complete match of Q_7 is finally expanded from h_2' and it yields a new association $(v_5, \mathsf{like}, u_3)$.

PEval also finds a partial match h_3' of Q_7 in the first iteration at worker P_1. It maps x to v_8, x' to v_4 and y to u_5. Since h_3' involves v_8 and u_5 that reside at worker P_2 (crossing-edges are maintained by both workers), the partial match h_3' will be sent to P_2 for further completion. □

At the end of PEval, master P_0 collects the status variables of border nodes v from all fragments. It applies aggregate function f_{aggr} (not shown) to reconcile $v.\mathsf{link}$ and $v.\mathsf{attr}$, and routes the aggregation and partial matches to relevant workers as messages. If conflicts emerge in attributes $v.\mathsf{attr}$ of any node, P_0 terminates the process immediately with $\perp$.

IncEval. As shown in Fig. 4, IncEval of PDeduce also deduces new associations incrementally. At worker P_i, it is triggered by message M_i that includes all the changes to the status variables of the border nodes in fragment F_i, and a set of partial matches to be further expanded at F_i.

Unlike PEval that initially makes the set Σ of GARs and the set V_i of nodes in F_i active, IncEval determines initial active nodes and GARs according to the changes and partial matches passed over in message M_i (lines 1-2). It treats the received changes directly as ΔF_c and updates fragment F_i with ΔF_c (line 3). After that, IncEval applies active GARs iteratively to deduce new associations pertaining to F_i (line 4), along the same lines as that in PEval, *i.e.,* lines 2-10 of Fig 2. The difference is that it also considers the partial matches in M_i, which are expanded just like extracted partial matches. IncEval stores the deduced associations that are accumulated over iterations as partial result $Q(F_i \oplus M_i)$.

At the end of IncEval, *changes* to the status variables of border nodes in F_i are sent to P_0. Master P_0 then aggregates the changes and sends messages just like in PEval.

Example 7: Continuing with Example 6, upon receiving partial match h'_3 at worker P_2, IncEval completes it by mapping the only remaining pattern node z of Q_7 to w_2. It then yields a missing link (v_8, like, u_5) deduced as a new association. This is a local edge for worker P_2 and it will be used to update both the fragments and indices at P_2. □

Assemble. When no more associations can be deduced, Assemble takes the union of partial results $Q(F_i \oplus M_i)$ from all workers P_i, *i.e.,* associations deduced from all fragments.

Correctness. Although PEval and IncEval compute associations simultaneously on multiple workers, the correctness of this parallel association deduction method is warranted.

Proposition 6: PIE *program* PDeduce *correctly computes the result* deduced(G, Σ) *of chasing G by Σ in parallel.* □

Proof: The result returned by PDeduce is in deduced(G, Σ). This follows from the definition of the chase, since no associations are deduced by PDeduce until partial matches become complete and $X \to Y$ is not satisfied (lines 5-6 in PEval and line 4 in IncEval). Conversely, by induction on the chase steps, it can be verified that all associations in deduced(G, Σ) are computed by PDeduce as PEval and IncEval inspect all candidate (partial) matches that can contribute to deduction of new associations (lines 6 and 4, respectively). □

6.3 Incremental Deduction of Associations

As remarked in Section 5, real-life graphs frequently change and association deduction is costly over large-scale graphs. These highlight the need for incremental association deduction, *e.g.,* in updating the the recommendation of products in e-commerce. We next develop a parallel algorithm for incremental deduction, denoted by IncDeduce.

Challenges. Essential to incremental deduction is analyzing different impacts of the inserted and deleted edges on deduced(G, Σ). Inserted edges could trigger the generation of new associations, while deleted ones make some old associations invalid, which hence have to be removed.

We say that a deduced association α' is *affected by* an edge e in graph G (resp. another deduced association α) if e (resp. α) is involved in the homomorphic mapping or precondition checking of a chase step in the chasing sequence that leads to the deduction of α'. Then an invalid association must be affected by some deleted edges e. However, *the opposite does not always hold.* That is, there exist deduced associations that are affected by edges e in ΔG but remain valid after updating graphs, since the associations can be deduced by other chasing sequences without the need of e.

Instead of first removing all the associations affected by deleted edges and then recovering those valid ones, algorithm IncDeduce reduces redundant computation by checking each affected association as soon as it is encountered and stopping further propagation from the valid ones to others.

```
Input: Fragmented chase graph $G_c$ with auxiliary information,
       a set $\Sigma$ of GARs and batch update $\Delta G = (\Delta G^+, \Delta G^-)$.
Output: The changes deduced$_\Delta(G, \Delta G, \Sigma)$.
1.  update $G_c$ with $\Delta G$; deduced$_\Delta^+ := \emptyset$;
2.  deduced$_\Delta^- \leftarrow$ DisAssoc$^-(G_c, \Sigma, \Delta G^-)$;
3.  update $G_c$ with deduced$_\Delta^-$;
4.  $A_c \leftarrow$ RefineAssoc$(G_c, \Sigma, \Delta G)$;
5.  refine deduced$_\Delta^+$ and deduced$_\Delta^-$ by $A_c$;
6.  update $G_c$;
7.  return deduced$_\Delta(G, \Delta G, \Sigma)$ = (deduced$_\Delta^+$, deduced$_\Delta^-$);
```

Figure 5: Algorithm IncDeduce

Auxiliary structures. In addition to the indices of PDeduce, for each edge e (resp. deduced association α), IncDeduce maintains a set d(e) (resp. d(α)) to store associations α' if the last chase steps for deducing α' include e (resp. α) in their mappings or precondition checking. Here e (resp. α) is also in d(e) (resp. d(α)). Note that these structures can be readily obtained when running PDeduce; their sizes are polynomial in $|G|$ and $|\Sigma|$ (the proof of Theorem 1).

Algorithm. As shown in Fig. 5, IncDeduce takes as input Σ, ΔG and moreover, the chase graph G_c and the corresponding auxiliary structures that are cached after the batch execution of PDeduce and are distributed across workers. Denote by ΔG^+ and ΔG^- the inserted and deleted edges in ΔG, respectively. IncDeduce computes the changes deduced$_\Delta(G, \Delta G, \Sigma)$ to the old associations deduced.

After adjusting G_c with update ΔG (line 1), IncDeduce computes the changes in two steps. (1) It first invokes procedure DisAssoc$^-$ to find a set deduced$_\Delta^-$ of associations that newly become invalid in response to deletions ΔG^- (line 2). (2) It then refines deduced$_\Delta^+$, *i.e.,* newly introduced associations due to insertions, and deduced$_\Delta^-$ by using the associations A_c derived via procedure RefineAssoc; it updates the corresponding parts in G_c (lines 4-6). The pair (deduced$_\Delta^+$, deduced$_\Delta^-$) is returned as the output (line 7).

We next show that each of the two steps can be implemented as a PIE program by revising PDeduce.

(1) Catching invalid associations. DisAssoc$^-$ identifies invalid associations in response to deletions ΔG^-, by extending PDeduce. In contrast to deducing new associations, here we need to find affected associations that may become invalid, and check whether they can be deduced by other chasing sequences as soon as possible in PEval and IncEval.

More specifically, PEval selects nodes in affected associations as initial active nodes, which are fetched from d(e) for each deleted edge e. It initializes active GARs with those having the same templates in their consequences Y as those of affected associations d(e). PEval iteratively inspects affected associations and enforces active GARs to check their validity. In addition, when examining affected associations, extracted partial matches must involve nodes of affected ones, such that they satisfy Y of active GARs. Moreover, only original parts of the graph and those associations that have been confirmed valid are accessed to construct matches.

If an affected association α can still be deduced, PEval marks α valid and removes it from the set of affected associations, *i.e.,* further checking of d(α) is avoided. Otherwise α is marked invalid and associations in d(α) except α are taken as affected associations for further inspection.

Algorithm IncEval is extended analogously. Note that the master worker monitors and coordinates the progress of the checking of the same affected association α at different workers, via message passing. It notifies the designated worker that maintains association α if all deduction attempts fail. After all the affected associations have been validated, the other deduced ones are also marked valid by IncEval.

(2) Refinement. Procedure RefineAssoc deduces new associations in response to inserted edges. It revises PDeduce as follows: (a) active nodes in PEval are initialized with vertices in ΔG^+ and those in the invalid associations, from which initial active GARs are derived; and (b) the associations in $\mathsf{deduced}^-_\Delta$ are filtered out when extracting and completing partial matches, unless they have been deduced in RefineAssoc. Intuitively, modification (a) limits "the scope" of active nodes and GARs by *treating inserted edges as new associations*. Modification (b) is to reduce false positives, as those old associations may become invalid due to edge deletions. RefineAssoc returns both newly introduced and valid affected ones, which are used to adjust the output of prior steps. RefineAssoc ensures that each deleted (resp. inserted) edge e is added to $\mathsf{deduced}^+_\Delta$ (resp. $\mathsf{deduced}^-_\Delta$) if e is marked as valid (resp. is deduced as an old association).

Example 8: Recall graph G and GARs Σ from Example 6. Consider ΔG that inserts $(v_5, \mathsf{friend}, v_6)$ and deletes $(v_1, \mathsf{friend}, v_5)$. IncDeduce first checks association affected by the deletion, which is $(v_5, \mathsf{like}, u_3)$. Since this link can be deduced with the insertion in a way similar to Example 6, it remains valid and IncDeduce stops further checking of associations depending on it. Besides, no new association is deduced during the refinement phase in this case. Hence the result of batch PDeduce (Example 6) remains stable. □

We have the following about algorithm IncDeduce.

Proposition 7: *The associations in* $\mathsf{deduced}(G \oplus \Delta G, \Sigma) \setminus \mathsf{deduced}(G, \Sigma)$ *are computed without any unnecessary invalid attempts in algorithm* IncDeduce. □

Proof: Since each association in $\mathsf{deduced}(G \oplus \Delta G, \Sigma) \setminus \mathsf{deduced}(G, \Sigma)$ must involve inserted edges or is a recovery of one deleted edge, it is computed by IncDeduce in step (2), using updated graph and valid associations. Moreover, once an association is confirmed valid, it cannot become invalid any more, since the validations are conducted iteratively by capturing all prior impacts of edge deletions in step (1). Thus no invalid new association is derived in IncDeduce. □

7. EXPERIMENTAL STUDY

Using real-life and synthetic graphs, we evaluated the accuracy, efficiency and scalability of our (incremental) association deduction algorithms. We also conducted a case study to demonstrate the effectiveness of GARs with real-life data.

Experimental setting. We used six real-life graphs as summarized in Table 1. In particular, Orkut is a large social network without informative attributes that can be used by GARs. We evaluated the efficiency of enforcing various GARs on it, and randomly included 20 attributes in Orkut.

We also generated synthetic graphs with size up to 300 million vertices and a billion edges, to test scalability.

Updates. We generated random updates ΔG for real-life and synthetic graphs, controlled by the size $|\Delta G|$ and the ratio τ of edge deletions to insertions. We set τ to 1 by default, *i.e.,* the sizes of graphs remain stable after the updates.

Table 1: Real-life graphs

Dataset	Type	Vertices	Edges
DBpedia [1]	knowledge base	6.2M	33.4M
YAGO2 [51]	knowledge base	2M	5.7M
Pokec [3]	social network	1.6M	30.6M
Patent [36]	citation network	3.7M	16.5M
IMDB [2]	knowledge graph on movies	16.7M	43.2M
Orkut [56]	social network	3M	117M

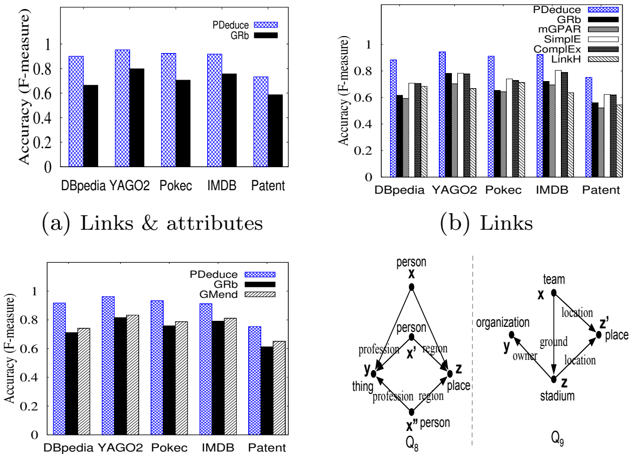
ML classifiers. We adopted SimplE [34] and ComplEx [53] to implement the ML classifier $\mathcal{M}$ for link prediction. We followed the protocol of [53, 34] to prepare training data; we obtained positive triples from original graphs and negative ones by combining entities and relations randomly. We created on average two negative samples per positive one for training, using 55% edges of each graph. We followed the PyTorch framework, the hyper-parameter search strategy and training settings of [53, 34] to train classifier $\mathcal{M}$.

GAR *generator*. For each graph, we generated GARs using the training data in three steps. (1) We first added all missing links predicted by the ML classifier between the nodes covered by training data. (2) We next applied an extension of the discovery algorithm for GFDs [18] on the subgraph pertaining to updated training data to derive GARs. Starting from frequent single-node patterns, the algorithm in [18] interleaves vertical spawning to extend the patterns and horizontal spawning to find attribute dependencies. Apart from constant and variable literals considered in [18], we removed some edges from the discovered patterns and included them as edge literals in GARs. Attribute literals were added with attributes that appear in the matches. (3) After these, we replaced certain edge literals $\iota(x, y)$ with ML literals $\mathcal{M}(x, y, \iota)$ in the GARs mined, such that $\mathcal{M}$ predicts the existence of missing edges (v, ι, v') in the training data.

We discovered 200 (resp. 150, 100, 200, 200, 200 and 100) GARs from DBpedia (resp. YAGO2, Pokec, Patent, IMDB, Orkut and synthetic graph). These GARs are satisfied by the subgraphs pertaining to training data; they have at most 7 pattern nodes and 4.6 literals on average.

Evaluation. The accuracy is evaluated over the test set of each real-life graph, *i.e.,* the graph excluding the training data. It is to evaluate the quality of associations deduced. Following [13, 25], we treated the original graphs as "correct" and introduced noises by randomly removing 3% edges and 3% attributes of each test set, since the quality of real-life graphs is unknown [59]. We measured the accuracy by precision, recall and F-measure, which are defined as (1) the ratio of removed associations deduced to all associations deduced by the methods, (2) the ratio of associations correctly deduced to all the associations removed, and (3) $2 \cdot (\mathsf{precision} \cdot \mathsf{recall})/(\mathsf{precision} + \mathsf{recall})$, respectively. As remarked earlier, we used Orkut only to test efficiency.

Baselines. Apart from implementing PDeduce (Section 6.2) and IncDeduce (Section 6.3) in C++, we also compared with the following baselines. (1) A variant $\mathsf{PDeduce_N}$ of PDeduce, without enforcing association-guided pruning; and a variant $\mathsf{IncDeduce_N}$ of IncDeduce without early checking of affected associations. (2) The sequential repairing method of [13], which deduces missing links and attributes, denoted as GRb. (3) ML link predictors SimplE [34] and ComplEx [53]; they are trained and tested with same data as above. (4) The link



(a) Links & attributes

(b) Links

(c) Attributes

(d) Real-life GARs

Figure 6: Effectiveness

deduction algorithm in [23] with GPARs, denoted as mGPAR; and GMend of [22] with an extension of GFDs, which deduces certain fixes to graphs, on deduction of missing attributes. (5) A sequential algorithm LinkH that finds missing links with the Horn rules discovered by AMIE [28].

Among these, GRb, mGPAR, GMend and LinkH are also rule-based methods. To get a fair comparison, besides the subclasses of GARs they support, we mined additional rules using their corresponding discovery methods to make all rule-based ones employ the same amount of rules.

The experiments were conducted on GRAPE [26], deployed on an HPC cluster of up to 10 machines connected by 10Gbps links. From each machine we used 2 processors powered by Intel Xeon 2.2GHz and 64G memory. Each experiment was run 5 times. The average is reported here.

Experimental results. We next report our findings.

Exp-1: Accuracy. We first tested the accuracy of PDeduce with all GARs mined. Figures 6(a) to 6(c) report the F-measure for deducing both missing links and attributes, missing links only and missing attributes only, respectively, over five real-life graphs on average. As shown there, PDeduce consistently outperforms other methods.

(1) It beats rule-based methods GRb, mGPAR, GMend and LinkH by 29.6%, 40.4%, 17.2% and 36.4% on average, respectively. It does better than mGPAR since it uses (a) GARs instead of GPARs, and (b) the chase as opposed to a single "chase step". It outperforms GRb and GMend by supporting ML literals. It beats LinkH for both reasons above.

(2) On average PDeduce is 20.8% and 22.1% more accurate than ML-based SimplE and ComplEx in deducing missing links, respectively. The impact of plugging which of the two ML classifiers into PDeduce is not substantial (not shown).

(3) We also conducted experiments to evaluate the accuracy of detecting semantic inconsistencies by using the same amount of GARs and GFDs. The result tells us that GARs outperforms GFDs by 42% in recall (not shown).

These verify that rules and ML methods put together work much better than each of them taken separately.

Exp-2: Efficiency. We next evaluated the efficiency of PDeduce and IncDeduce versus the variants and GRb. The number $\|\Sigma\|$ of GARs, the average size $|\Sigma_Q|$ of the patterns in Σ, the size $|\Delta G|$ of updates for incremental deduction, and the number n of processors, *i.e.,* workers for parallel algorithms were fixed as 120 for DBpedia (90 for YAGO2, 60 for Pokec, 120 for Patent, 120 for IMDB and 120 for Orkut), 4.8, $10\%|G|$ and 12, respectively, unless otherwise stated.

Varying $\|\Sigma\|$. Varying $\|\Sigma\|$ from 40 to 200 and 30 to 150, Figures 7(a)-7(b) report the results on DBpedia and YAGO2, respectively. We can see that (1) the more rules are used, the longer all methods take, as expected. (2) PDeduce is on average 2.2 (resp. 14.3) times faster than $\mathsf{PDeduce_N}$ (resp. GRb), validating the effectiveness of association-guided pruning.

Varying $|\Sigma_Q|$. We varied $|\Sigma_Q|$ from 3 to 7 over DBpedia and YAGO2. As shown in Figures 7(c)-7(d), (a) all algorithms take longer on larger $|\Sigma_Q|$. (b) PDeduce and IncDeduce are feasible with real-life GARs, *e.g.,* they take 17.7s and 4.2s over DBpedia when $|\Sigma_Q| = 5$, as opposed to 304.5s by GRb and 33.9s by $\mathsf{PDeduce_N}$. (c) PDeduce outperforms other batch algorithms, consistent with Figures 7(a) and 7(b).

Incremental deduction. Varying $|\Delta G|$ from 5% up to 35% of $|G|$, Figures 7(e)-7(i) report the following over DBpedia, YAGO2, Pokec, Patent and Orkut, respectively. (1) IncDeduce is 6.3 to 1.6 (resp. 5.1 to 1.4, 4.8 to 1.3, 4.7 to 1.6 and 9.5 to 1.7) times faster than PDeduce over the five real-life graphs, respectively, when $|\Delta G|$ varies from 5% to 20%. (2) IncDeduce beats PDeduce even when $|\Delta G|$ is up to 25% of $|G|$. This justifies the need for incremental deduction. (3) All incremental methods take longer for larger $|\Delta G|$, while the batch ones are indifferent to $|\Delta G|$.

The results on other graphs are similar (not shown).

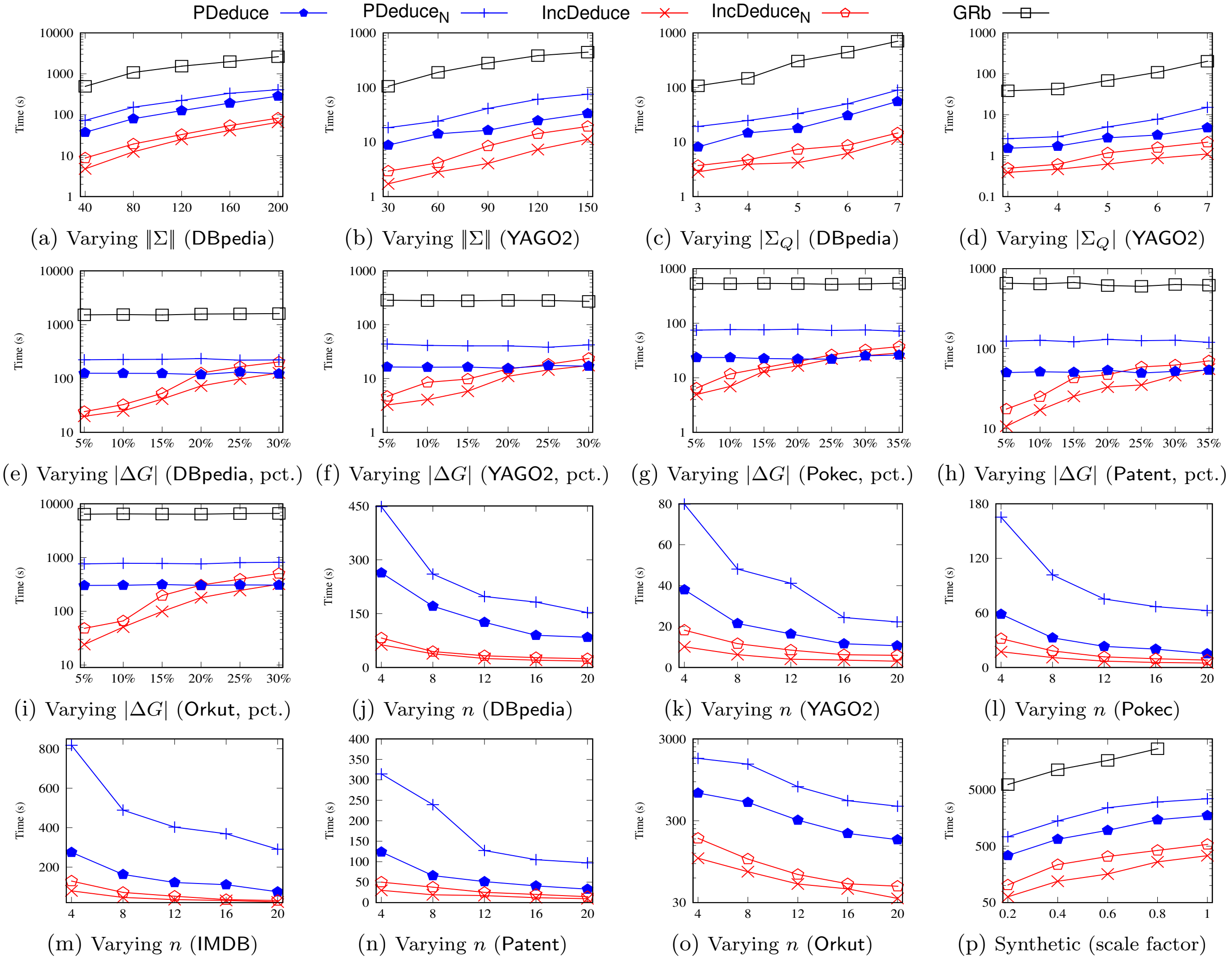
Exp-3: Scalability. In the same default setting as Exp-2, we next evaluated the scalability of deduction approaches.

Varying n. We varied the number n of processors from 4 to 20. As shown in Figures 7(j) to 7(o), (a) PDeduce scales well: the improvement is 3.1 (resp. 3.6, 3.9, 3.7, 3.6, 3.8) times over DBpedia (resp. YAGO2, Pokec, IMDB, Patent, Orkut) when n varies from 4 to 20. (b) IncDeduce works well on real-life graphs: it takes only 3.1s to process 10% updates on YAGO2 using 20 processors; the results on other graphs are consistent. (c) On average, PDeduce beats $\mathsf{PDeduce_N}$ by 2.7 times, up to 4.1 times. (d) Early checking of affected associations is effective for incremental association deduction: IncDeduce beats $\mathsf{IncDeduce_N}$ by 1.5 times on average.

Synthetic graphs. Varying the scale factor from 0.2 to 1.0, we tested (incremental) association deduction on synthetic graphs. As shown in Fig. 7(p), (a) all the batch and incremental algorithms take longer over larger G, as expected. (b) PDeduce is feasible on large graphs, taking 1756.5s using 100 GARs on graphs with 300 million nodes and a billion edges; in contrast, GRb ran out-of-memory.

Exp-4: Case study. Figure 6(d) shows the patterns of two GARs discovered in the real-life datasets we used.

(1) In Pokec, GAR $\varphi_8 = Q_8[\bar{x}](\mathcal{M}(x, x', \mathsf{friend}) \wedge x.\mathsf{hobbies} = x'.\mathsf{hobbies} \wedge x'.\mathsf{hobbies} = x''.\mathsf{hobbies} \rightarrow \mathsf{friend}(x, x''))$ suggests that if three people have the same profession, region and hobbies, and two of them are predicted as friends by ML classifier, then another friend relationship should also be established. It identifies a link between two people (IDs:



(a) Varying $\|\Sigma\|$ (DBpedia)
(b) Varying $\|\Sigma\|$ (YAGO2)
(c) Varying $|\Sigma_Q|$ (DBpedia)
(d) Varying $|\Sigma_Q|$ (YAGO2)
(e) Varying $|\Delta G|$ (DBpedia, pct.)
(f) Varying $|\Delta G|$ (YAGO2, pct.)
(g) Varying $|\Delta G|$ (Pokec, pct.)
(h) Varying $|\Delta G|$ (Patent, pct.)
(i) Varying $|\Delta G|$ (Orkut, pct.)
(j) Varying n (DBpedia)
(k) Varying n (YAGO2)
(l) Varying n (Pokec)
(m) Varying n (IMDB)
(n) Varying n (Patent)
(o) Varying n (Orkut)
(p) Synthetic (scale factor)

Figure 7: Efficiency and scalability

361348, 361341) because of another one (ID: 361273), where all three like football and live in Kolarovo.

(2) In DBpedia, GAR $\varphi_9 = Q_9[\bar{x}](\mathcal{M}(x, y, \text{association}) \rightarrow \text{tenant}(z, x))$ predicts associations between stadiums and sport teams. If a team uses a stadium as its ground at the same location, and the stadium is owned by an organization that is predicted to be the association of the team by ML classifier, then φ_9 deduces that the team is a tenant of the stadium. It deduces edge (Chichibunomiya Rugby Stadium, tenant, Sunwolves) in DBpedia, although the link between the owner Japan Sport Council and Sunwolves is missing.

Summary. We find the following. (1) GARs are effective in association deduction. On average our algorithms outperform existing methods for link prediction and deducing missing attributes by 29.1% and 19.4% in accuracy, respectively, and are 21.3% and 28.2% better than ML-based and rule-based methods alone. (2) GARs capture 42% more semantic errors than GFDs in real-life graphs. (3) PDeduce scales well with large graphs; it beats existing deduction methods by 18.1 times on graphs with 1.3 billion nodes and edges. (4) It scales well with the number of processors. (5) Incremental IncDeduce beats batch PDeduce by 4.3 times when $|\Delta G|$ is $10\%|G|$ and works better even when $|\Delta G|$ is up to $25\%|G|$. (6) Our optimization strategies improve batch and incremental deduction by 2.7 and 1.5 times, respectively.

8. CONCLUSION

We have proposed GARs to catch missing links/attributes and semantic inconsistencies in a uniform framework, by unifying rule-based and ML-based methods. We have settled the classical problems for GARs by establishing their upper and lower bounds, all matching. We have developed graph-centric algorithms for deduction and incremental deduction of associations in parallel. Our experimental study has verified that the methods are promising on real-life graphs.

One topic for future work is to explore applications of GARs by treating GARs as soft rules. Another topic is to find non-embedding-based ML classifiers for GARs. A third topic is to develop parallel algorithms for discovering GARs.

Acknowledgments. Fan is supported in part by ERC 652976, Royal Society Wolfson Research Merit Award WRM/R1/180014, EPSRC EP/M025268/1, Shenzhen Institute of Computing Sciences, and Beijing Advanced Innovation Center for Big Data and Brain Computing. Lu is supported in part by NSFC 61602023. This work is also supported in part by Alibaba Innovative Research (AIR) Program. Tian is the corresponding author.

9. REFERENCES

[1] DBpedia. *http://wiki.dbpedia.org*.

[2] IMDB. *https://www.imdb.com/interfaces*.

[3] Pokec. *http://snap.stanford.edu/data/soc-pokec.html*.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[5] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.

[6] F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using Map-Reduce. In *ICDE*, 2013.

[7] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.

[8] W. Akhtar, A. Cortés-Calabuig, and J. Paredaens. Constraints in RDF. In *SDKB*, 2010.

[9] B. Bhattarai, H. Liu, and H. H. Huang. CECI: Compact embedding cluster index for scalable subgraph matching. In *SIGMOD*, 2019.

[10] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[11] L. Cagliero and A. Fiori. Discovering generalized association rules from twitter. *Intelligent Data Analysis*, 17(4):627–648, 2013.

[12] A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985.

[13] Y. Cheng, L. Chen, Y. Yuan, and G. Wang. Rule-based graph repairing: Semantic and efficient repairing methods. In *ICDE*, 2018.

[14] A. Cortés-Calabuig and J. Paredaens. Semantics of constraints in RDFS. In *AMW*, 2012.

[15] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. V. Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. The YouTube video recommendation system. In *RecSys*, 2010.

[16] F. Erlandsson, P. Bródka, A. Borg, and H. Johnson. Finding influential users in social media using association rule learning. *Entropy*, 18(5):164, 2016.

[17] W. Fan, Z. Fan, C. Tian, and X. L. Dong. Keys for graphs. *PVLDB*, 8(12):1590–1601, 2015.

[18] W. Fan, C. Hu, X. Liu, and P. Lu. Discovering graph functional dependencies. In *SIGMOD*, 2018.

[19] W. Fan, R. Jin, M. Liu, P. Lu, C. Tian, and J. Zhou. *Capturing Associations in Graphs*, 2020. `http://homepages.inf.ed.ac.uk/s1837143/publication/gar/GAR.pdf`.

[20] W. Fan, X. Liu, P. Lu, and C. Tian. Catching numeric inconsistencies in graphs. In *SIGMOD*, 2018.

[21] W. Fan and P. Lu. Dependencies for graphs. In *PODS*, 2017.

[22] W. Fan, P. Lu, C. Tian, and J. Zhou. Deducing certain fixes to graphs. *PVLDB*, 12(7):752–765, 2019.

[23] W. Fan, X. Wang, Y. Wu, and J. Xu. Association rules with graph patterns. *PVLDB*, 8(12):1502–1513, 2015.

[24] W. Fan, Y. Wu, and J. Xu. Adding counting quantifiers to graph patterns. In *SIGMOD*, 2016.

[25] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *SIGMOD*, 2016.

[26] W. Fan, J. Xu, Y. Wu, W. Yu, J. Jiang, Z. Zheng, B. Zhang, Y. Cao, and C. Tian. Parallelizing Sequential Graph Computations. In *SIGMOD*, 2017.

[27] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[28] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

[29] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[30] P. H. Guzzi, M. Milano, and M. Cannataro. Mining association rules from gene ontology and protein networks: Promises and challenges. In *ICCS*, 2014.

[31] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *SIGMOD*, 2019.

[32] J. Hellings, M. Gyssens, J. Paredaens, and Y. Wu. Implication and axiomatization of functional and constant constraints. *Ann. Math. Artif. Intell.*, 76(3-4):251–279, 2016.

[33] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *PVLDB*, 4(11):1123–1134, 2011.

[34] S. M. Kazemi and D. Poole. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*, 2018.

[35] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[36] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*, 2005.

[37] W. Lin, S. A. Alvarez, and C. Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data Min. Knowl. Discov.*, 6(1):83–105, 2002.

[38] Y. Liu, W. Wei, A. Sun, and C. Miao. Exploiting geographical neighborhood characteristics for location recommendation. In *CIKM*, 2014.

[39] X. Luo, L. Liu, L. Bo, Y. Cao, J. Wu, Q. Li, Y. Yang, K. Yang, and K. Q. Zhu. AliCoCo: Alibaba e-commerce cognitive concept net. In *SIGMOD*, 2020.

[40] M. H. Namaki, Y. Wu, Q. Song, P. Lin, and T. Ge. Discovering graph temporal association rules. In *CIKM*, 2017.

[41] C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[42] L. Qin, J. X. Yu, L. Chang, H. Cheng, C. Zhang, and X. Lin. Scalable big graph processing in MapReduce. In *SIGMOD*, 2014.

[43] R. Raman, O. van Rest, S. Hong, Z. Wu, H. Chafi, and J. Banerjee. PGX. ISO: Parallel and efficient in-memory engine for subgraph isomorphism. In *GRADES*, 2014.

[44] X. Ren, J. Wang, W.-S. Han, and J. X. Yu. Fast and robust distributed subgraph enumeration. *PVLDB*, 12(11):1344–1356, 2019.

[45] F. Sadri and J. D. Ullman. The interaction between functional dependencies and template dependencies. In *SIGMOD*, 1980.

[46] D. Sánchez, M. A. V. Miranda, L. Cerda, and J. Serrano. Association rules applied to credit card fraud detection. *Expert Syst. Appl.*, 36(2):3630–3640, 2009.

[47] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

[48] S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo. Socio-spatial properties of online location-based social networks. In *ICWSM*, 2011.

[49] Y. Shao, B. Cui, L. Chen, L. Ma, J. Yao, and N. Xu. Parallel subgraph listing in a large-scale graph. In *SIGMOD*, 2014.

[50] Q. Song, Y. Wu, P. Lin, L. X. Dong, and H. Sun. Mining summaries for knowledge graph search. *TKDE*, 30(10):1887–1900, 2018.

[51] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[52] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.

[53] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[54] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[55] Z. Wang, R. Gu, W. Hu, C. Yuan, and Y. Huang. BENU: Distributed subgraph enumeration with backtracking-based framework. In *ICDE*, 2019.

[56] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[57] S.-H. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha. Like like alike: Joint friendship and interest propagation in social networks. In *WWW*, 2011.

[58] H. Young. Personalized product recommendations drive just 7% of visits but 26% of revenue, 2017. *https://www.salesforce.com/blog/2017/11/personalized-product-recommendations-drive-just-7-visits-26-revenue.html*.

[59] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, and J. Lehmann. User-driven quality evaluation of DBpedia. In *ISEM*, 2013.

[60] C. Zhang and S. Zhang. *Association rule mining: models and algorithms*. Springer-Verlag, 2002.

[61] S. Zhang, Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embeddings. In *NIPS*, 2019.

[62] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou. Aligraph: A comprehensive graph neural network platform. *PVLDB*, 12(12):2094–2105, 2019.