Discovering Association Rules from Big Graphs

Wenfei Fan[1,2,3], Wenzhi Fu[1], Ruochun Jin[1], Ping Lu[3], Chao Tian[4]
[1]University of Edinburgh [2]Shenzhen Institute of Computing Sciences [3]Beihang University [4]Chinese Academy of Sciences
{wenfei@inf.,wenzhi.fu@,ruochun.jin@}ed.ac.uk,luping@buaa.edu.cn,tianchao@iscas.ac.cn

ABSTRACT

This paper tackles two challenges to discovery of graph rules. Existing discovery methods often (a) return an excessive number of rules, and (b) do not scale with large graphs given the intractability of the discovery problem. We propose an application-driven strategy to cut back rules and data that are irrelevant to users' interests, by training a machine learning (ML) model to identify data pertaining to a given application. Moreover, we introduce a sampling method to reduce a big graph G to a set H of small sample graphs. Given expected support and recall bounds, the method is able to deduce samples in H and mine rules from H to satisfy the bounds in the entire G. As proof of concept, we develop an algorithm to discover Graph Association Rules (GARs), which are a combination of graph patterns and attribute dependencies, and may embed ML classifiers as predicates. We show that the algorithm is parallelly scalable, *i.e.,* it guarantees to reduce runtime when more machines are used. We experimentally verify that the method is able to discover rules with recall above 91% when using sample ratio 10%, with speedup of 61 times.

PVLDB Reference Format:
Wenfei Fan, Wenzhi Fu, Ruochun Jin, Ping Lu, and Chao Tian. Discovering Association Rules from Big Graphs. PVLDB, 15(7): 1479-1492, 2022.
doi:10.14778/3523210.3523224

1 INTRODUCTION

A variety of rules have been studied for graphs, to detect inconsistencies [20, 23], resolve entities [17, 21], reason about knowledge graphs [24, 45], catch network evolution [8, 38], and recommend items to users [22]. There have also been recent graph rules that embed machine learning (ML) classifiers as predicates [19], to deduce associations by unifying rule-based and ML-based methods.

To make practical use of the rules, effective methods have to be in place to discover useful rules from real-life data. Rules on graphs are more complicated than relational rules. For instance, graph functional dependencies (GFDs) [23] are defined as $Q(X \rightarrow Y)$, with a graph pattern Q to identify entities and an attribute dependency $X \rightarrow Y$ to apply to those entities. Given a graph G, rule discovery is to find a set Σ_G of non-redundant rules that can be frequently applied to G (measured by support). Discovery of such rules requires to mine both graph patterns and dependencies, and is more challenging than discovery of relational data quality rules.

Challenges. There are two major challenges to graph rule discovery. (1) *Scalability.* Rule discovery algorithms can hardly scale with large graphs. As shown in [18], for instance, the discovery problem for GFDs subsumes subgraph isomorphism, which is intractable (cf. [25]). As a consequence, it is prohibitively costly to discover GFDs with graph patterns of 7 nodes or more, which cannot finish in 1.66 hours on graphs with 32 million nodes and edges, even when using 8 machines [18]. It is also reported in [18] that mining GFDs with patterns of at most 5 nodes in the same setting already takes 76 minutes. (2) *Excessive rules.* A large number of rules typically hold on a given graph. It is hard for users to inspect the excessive number of discovered rules and identify useful ones to them.

In fact, most existing discovery algorithms for graph rules follow the levelwise search paradigm, *e.g.,* [8, 18, 22, 34, 51]. These methods enumerate candidate rules in an exponential search space, and evaluate each candidate by subgraph matching to check whether it meets the discovery requirement, *e.g.,* support threshold. The latter also incurs exponential cost as mentioned above. Therefore, such methods suffer from poor scalability in mining graph rules with large patterns, and often output excessive rules created from the large search space, while users' interests are not considered in pruning useless candidates. The challenges are already present when discovering relational functional dependencies (FDs) [55], but the problems become more staggering for graph rules.

Is it possible to develop an effective method that is able to discover only rules relevant to users' interests and scale with large graphs, without degradation in the quality of the discovered rules?

Strategies. We explore new approaches to tackling the challenges.

(1) Application-driven rule discovery. Users are often interested only in rules that help their applications. For instance, when a company is promoting sale of an album, it wants rules to identify music fans, and could not care less about rules for suggesting buyers of pickup trucks. In light of this, we propose an application-driven strategy. Given an application $\mathcal{A}$ and a graph G, we train an ML model $\mathcal{M}_{\mathcal{A}}$ to identify nodes, edges and properties in G that pertain to $\mathcal{A}$. We reduce G to a smaller graph $G_{\mathcal{A}}$ with only the data pertaining to $\mathcal{A}$, and discover $\mathcal{A}$*-relevant* rules from $G_{\mathcal{A}}$ instead of from the entire G.

(2) Sampling big graphs. To further reduce discovery cost, we sample a set H of graphs $H(\mathcal{A}, \rho\%)$ from $G_{\mathcal{A}}$, such that their sizes are at most $\rho\%$ of $G_{\mathcal{A}}$. The samples consist of representative data cells in $G_{\mathcal{A}}$ along with their surrounding subgraphs. Denote by Σ_G and Σ_H the set of $\mathcal{A}$-relevant rules discovered from G and H, respectively. We show that given bounds σ and $\gamma\%$, we can deduce H such that (a) at least $\gamma\%$ of rules in Σ_G are covered by Σ_H, and (b) each of these rules can be applied at least σ times on the entire G, *i.e.,* the rules in Σ_G can be mined from H above recall $\gamma\%$ and support σ.

(3) Parallel scalability. We parallelize the discovery process. We show that the algorithm is parallelly scalable, *i.e.,* it guarantees to reduce runtime when more machines are used. In principle, it can scale with large graphs G by using more machines when needed.

Proceedings of the VLDB Endowment, Vol. 15, No. 7 ISSN 2150-8097.
doi:10.14778/3523210.3523224

(4) PoC with GARs. As a proof of concept, we test the strategies with discovering Graph Association Rules (GARs) [19]. GARs subsume GFDs [23], graph entity rules (GEDs) [21] and graph pattern association rules (GPARs) [22] as special cases, and hence are able to identify entities, catch conflicts, detect missing links and deduce associations. Moreover, GARs may plug in existing ML classifiers as predicates, to leverage well-trained ML models for entity resolution, link prediction and similarity checking, among other things.

Putting these together, we propose a 3-step scheme to discover useful rules from a big graph G for a given application $\mathcal{A}$: (a) reducing G to $\mathcal{A}$-relevant $G_{\mathcal{A}}$, (b) sampling a set H of $H(\mathcal{A}, \rho\%)$ from $G_{\mathcal{A}}$, and (c) parallelizing discovery with the parallel scalability.

We reduce irrelevant rules by proposing ML-based reduction (step (a)), and improve the scalability by combining steps (a)-(c). Steps (a)-(b) reduce the problem of rule discovery from large G to much smaller $H(\mathcal{A}, \rho\%)$. We show that the scheme guarantees accuracy bounds. As opposed to prior methods [13, 26, 45] that only sample paths, step (b) can sample general subgraphs.

Contributions and organization. After reviewing GARs in Section 2, we present our new approaches to discovering GARs.

(1) Discovery problem revisited (Section 3). We formulate the discovery problem for GARs driven by applications and based on sampling, and present our three-step discovery scheme.

(2) Application-driven discovery (Section 4). We develop a graph reduction method to deduce $\mathcal{A}$-relevant graph $G_{\mathcal{A}}$ from graph G for a given application $\mathcal{A}$. It trains an ML model $\mathcal{M}_{\mathcal{A}}$ (long short-term memory (LSTM) networks [29]) to label data pertaining to $\mathcal{A}$.

(3) Sampling graphs (Section 5). We propose a sampling method GSRD for reducing $G_{\mathcal{A}}$ to a set H of sample graphs $H(\mathcal{A}, \rho\%)$. We prove probabilistic bounds on recall and support, to determine the number and size of $H(\mathcal{A}, \rho\%)$ such that Σ_H retains the bounds on G.

(4) Parallel discovery algorithm (Section 6). We develop a parallel algorithm to discover GARs from the set H of samples $H(\mathcal{A}, \rho\%)$. We show that the algorithm guarantees the parallel scalability.

(5) Experimental study (Section 7). Using real-life and synthetic graphs, we empirically verify the following. On average, (a) the application-driven reduction method cuts down the graph sizes by 76%, up to 98%. Moreover, 85% of $\mathcal{A}$-relevant GARs can be mined from the reduced graphs. (b) GSRD is effective: the recall of discovered GARs reaches 94% when having 4 sample graphs deduced by GSRD with sample ratio 10%, and the support of each GAR is at least 1000 in the original graphs. (c) Sampling-based discovery is on average 60.6 times faster than mining GARs from the entire graphs, while retaining the recall above 91%. (d) Our parallel algorithm for GAR discovery scales well with the number of machines used.

Related work. We categorize the related work as follows.

Rule discovery. Besides the extensive study on mining relational rules, *e.g.,* [14, 30, 63–65, 75], there have also been several discovery methods for different kinds of graph rules. Following the levelwise search that is widely used in data mining, GFDs [18], GPARs [22], graph temporal association rules [51] and graph differential dependencies [34] can be mined from graphs with different pruning strategies. GERM [8] and LFR-Miner [38] revise pattern mining method gSpan [76] to mine graph evolution rules and link formation rules, respectively. Some rule learners are in place to discover Horn rules of restricted forms from knowledge graphs modeled in RDF [44], *e.g.,* AnyBURL [45] learns rules from paths of various lengths in a bottom-up manner. GPFL [26] is a probabilistic rule learner that optimizes AnyBURL by generalizing paths into templates, to reduce search space. Based on inductive logic programming, top-down rule learners have also been developed, such as AMIE [24] and ScaLeKB [13], which repeatedly produce new rules at each level by specializing the ones derived in the upper level. RuDik [53] discovers acyclic Horn rules by generating the universe of all possible rules, from which it selects rules according to a minimum weighted set cover of the given examples. RNNLogic [58] develops an EM-based method to train a rule generator, and [77] proposes to learn rules through the differentiable model of [16].

Our method differs from the prior work as follows. (1) We propose application-driven reduction and graph sampling strategies with accuracy guarantee to reduce excessive rules and improve efficiency, as opposed to mining rules from the entire graphs. In light of these, the problem and even the notion of support are different. (2) We study the discovery of GARs from general property graphs, without requiring to encode graph data in RDF as knowledge graph rule learners [13, 16, 24, 45, 53]. These rule learners may exhibit poor scalability on RDFs that are transformed from property graphs, since their node attributes often yield a large number of RDF triples. (3) We discover GARs with ML predicates and graph patterns of generic subgraphs. In contrast, no prior methods consider ML predicates, and most of them study path patterns only.

Sampling methods. Sampling has long been studied to facilitate the discovery of association rules and frequent itemsets from relational data. For example, inspired by the VC-dimension theory [71] and Rademacher average [10], [60, 61] establish bounds on the required sample size for finding approximate frequent itemsets. They ensure that for each itemset mined from the sample, the difference between its frequency in the entire dataset and the expected frequency threshold is within a user specified bound. Similarly, [11] gives a theoretical framework to analyze the impact of sample size on the quality of association rules mined from the samples, *i.e.,* their support and confidence in the original relations.

Sampling has also been adopted in the discovery of data cleaning rules from relational tables such as FDs [56] and (approximate) denial constraints (DCs) [9, 42]. In particular, [42] shows how to estimate the number of violations of approximate DCs in sample data that is uniformly drawn from relations, by which it decides the right approximation threshold to use when discovering approximate DCs from the samples. It guarantees that these rules also hold in the entire dataset with a high probability.

When it comes to graphs, the majority of knowledge graph rule learners perform sampling to randomly extract paths from RDF and use the paths to generate restricted forms of Horn clauses, *e.g.,* [26, 35, 36, 45, 46], with variants of random walks.

Different from mining relational rules, (1) GAR discovery has to inspect not only dependencies on attributes, but also graph patterns. Thus neither the relational sampling techniques nor their quality analyses can be applied to mining GARs. (2) To the best of our knowledge, we provide the first accuracy guarantee on recall and

support for sampling-based discovery of graph rules, *i.e.,* the percentage of useful and frequent rules *w.r.t.* some support threshold that can be mined from samples of bounded sizes.

Parallel discovery. Parallel algorithms have been developed to mine rules from both relations, *e.g.,* [59, 62], and graphs, *e.g.,* [12, 13, 18, 22, 74]. However, none of these guarantees the parallel scalability except [18, 22]. The parallelly scalable algorithms in [18, 22] perform levelwise search on entire graphs. We extend the discovery algorithm of [18] to cope with sample graphs and ML predicates in mining GARs, without hampering its parallel scalability.

2 GRAPH ASSOCIATION RULES

In this section we review GARs (graph association rules) of [19].

Preliminaries. We assume three countably infinite alphabets Γ, Υ and U of symbols, for labels, attributes and constants, respectively.

Graphs. We consider directed labeled graphs $G = (V, E, L, F)$, where (a) V is a finite set of nodes; (b) $E \subseteq V \times \Gamma \times V$ is the set of edges, and $e = (v, l, v')$ denotes an edge from node v to v' that is labeled with $l \in \Gamma$; (c) each node $v \in V$ has label $L(v)$ from Γ; and (d) each node $v \in V$ carries a tuple $F(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity with $A_i \in \Upsilon$ and $a_i \in U$, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$ for distinct properties. Note that even nodes of the same "type" may have different sets of attributes in a schemaless graph.

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 $L_Q(u) \in \Gamma$ (resp. $L_Q(e) \in \Gamma$) to node $u \in V_Q$ (resp. $e \in E_Q$, *i.e.,* $e = (u, L_Q(e), u')$); (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 node label in $Q[\bar{x}]$. For each variable $x \in \bar{x}$, we use $\mu(x)$ and x interchangeably.

Pattern matching. A *match* of pattern $Q[\bar{x}]$ in a graph G is a homomorphic 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 pattern edge $e = (u, L_Q(e), u') \in E_Q$, $e' = (h(u), L_Q(e), h(u')) $ is in G. Here $L_Q(u) = L(h(u))$ if $L_Q(u)$ is '_', *i.e.,* wildcard can match an arbitrary label. We denote the match as a vector $h(\bar{x})$, consisting of $h(\mu(x))$ for all $x \in \bar{x}$ in the same order as $\bar{x}$.

Predicates. A *predicate* p *of* $Q[\bar{x}]$ has one of the following forms:

$$p \ ::= \ x.A \mid l(x, y) \mid x.A = y.B \mid x.A = c \mid \mathcal{M}(x, y, l),$$

where x, y are variables in $\bar{x}$; $x.A$ denotes an attribute A of pattern node x (for $A \in \Upsilon$); $l(x, y)$ is an edge from x to y labeled with $l \in \Gamma$; c is a constant in U; and $\mathcal{M}(x, y, l)$ is an ML classifier (see below).

We refer to $x.A$, $l(x, y)$, $x.A = y.B$, $x.A = c$ and $\mathcal{M}(x, y, l)$ as *attribute, edge, variable, constant* and *ML predicate*, respectively.

Graph association rules (GARs**)**. A GAR φ is defined as

$$Q[\bar{x}](X \rightarrow p_0),$$

where $Q[\bar{x}]$ is a graph pattern, X is a conjunction of predicates of $Q[\bar{x}]$, and p_0 is a single predicate of $Q[\bar{x}]$. We refer to $Q[\bar{x}]$ and $X \rightarrow p_0$ as the *pattern* and *dependency* of φ, and to X and p_0 as the *precondition* and *consequence* of φ, respectively.

Intuitively, a GAR is a combination of topological constraint Q and logical constraint $X \rightarrow p_0$. The pattern Q identifies entities in a graph, and the dependency $X \rightarrow p_0$ is applied to the entities. Constant and variable predicates $x.A = c$ and $x.A = y.B$ specify

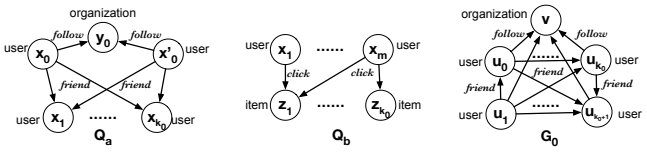


Figure 1: Patterns and graphs

value associations to attributes, which can catch inconsistencies and moreover, identify entities (with $x.\text{id} = y.\text{id}$ when A and B are node ids). Attribute and edge predicates $x.A$ and $l(x, y)$ enforce the existence of attributes and edges, *i.e., attribute and edge associations*, respectively, which can deduce associations and missing links.

ML predicates. One can "plug in" a well-trained ML classifier $\mathcal{M}$ for link prediction, entity matching or similarity checking. That is, $\mathcal{M}(x, y, l)$ is true if $\mathcal{M}$ predicts the existence of a link labeled l from x to y, and false otherwise, by uniformly expressing entity matching and similarity checking as link prediction. Here the label l can indicate (1) a predicted link, (2) the match of x and y as the same entity, linked by an edge with '=' as l, or (3) semantic similarity between x and y linked by an edge with '$\approx$' as l, indicating that x and y are "semantically" close, *e.g.,* "monitor" and "LCD screen". Thus ML predicates can also be regarded as edge predicates. An ML classifier becomes "well-trained" once its training process converges, *e.g.,* the loss of a neural network is stable after epochs.

As shown in [19], GFDs [23], GEDs [21], and GPARs [22] are special cases of GARs without ML, edge and attribute predicates.

Example 1: Embedding ML predicates, GARs are able to predict relationships in professional networks, *e.g.,* colleagues in DingTalk [3], and fraudulent behaviors in e-commerce platforms, as follows.

(a) To establish domestic "colleague" connections, we use a GAR $\varphi_a = Q_a[\bar{x}_a](X_a \rightarrow \text{colleague}(x_0, x_0'))$, where (i) the pattern Q_a is depicted in Fig. 1; and (ii) X_a is $\bigwedge_{i \in [1,k_0]}(x_0.\text{city} = x_i.\text{city} \wedge x_0'.\text{city} = x_i.\text{city}) \wedge \bigwedge_{i,j \in [1,k_0]} \mathcal{M}_a(x_i, x_j, \text{similar\_profile})$. It indicates that two users x_0 and x_0' are likely to be colleagues when they follow the same organization y_0 and have k_0 common friends (*i.e.,* x_1 to x_{k_0}) with the same city attribute, and each pair of common friends have similar profiles, determined by the ML classifier $\mathcal{M}_a$.

(b) Consider GAR $\varphi_b = Q_b[\bar{x}_b](\bigwedge_{i,j \in [1,m]} \mathcal{M}_b(x_i, x_j, \text{one\_group}) \rightarrow \text{click}(x_1, z_{k_0}))$, where pattern Q_b is also shown in Fig. 1. It says that if a set of m users $x_1, \ldots, x_m$ are identified to be within the same community by ML classifier $\mathcal{M}_b$, and if all users in this community except x_1 conduct fake clicks on a set of k_0 items in the e-commerce platform, then x_1 might also perform fake click on item z_{k_0}.

As k_0 and m can vary, these GARs may have large patterns. □

Semantics. To interpret GAR $\varphi = Q[\bar{x}](X \rightarrow p_0)$, denote by $h(\bar{x})$ a match of Q in a graph G, and by p a predicate of $Q[\bar{x}]$. We say that $h(\bar{x})$ *satisfies* a predicate p, denoted by $h(\bar{x}) \models p$, if the following condition is satisfied: (a) when p is $x.A$, node $h(x)$ carries attribute A; (b) when p is $l(x, y)$, there exists an edge with label l from $h(x)$ to $h(y)$; (c) when p is $x.A = y.B$, attributes A and B exist at $h(x)$ and $h(y)$, respectively, and $h(x).A = h(y).B$; (d) when p is $x.A = c$, attribute A exists at $h(x)$, and $h(x).A = c$; and (e) when p is $\mathcal{M}(x, y, l)$, the ML classifier $\mathcal{M}$ predicts an edge $(h(x), l, h(y))$.

For a conjunction X of predicates, we write $h(\bar{x}) \models X$ if match $h(\bar{x})$ satisfies *all* the predicates in X. Note that if X is $\emptyset$ (*i.e.,* true), then $h(\bar{x}) \models X$ for any match $h(\bar{x})$ of Q in G.

Table 1: Notations

Notations	Descriptions
G, $Q[\bar{x}]$, φ	graph, graph pattern, and GAR φ=$Q[\bar{x}](X \to p_0)$, resp.
$\mathcal{A}$	an application that consists of a set of predicates
$G_{\mathcal{A}}$	an $\mathcal{A}$-graph of G
$H(\mathcal{A}, \rho\%)$	a sample graph
Σ_G, Σ_H	GARs mined from G and a set H of sample graphs, resp.
$\text{sup}(\varphi, G)$	the support of GAR φ in graph G

We write $h(\bar{x}) \models X \to p_0$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models p_0$.

We say that a graph G *satisfies* GAR φ, denoted by $G \models \varphi$, if for all matches $h(\bar{x})$ of Q in G, $h(\bar{x}) \models X \to p_0$. Graph G *satisfies* a set Σ of GARs, denoted by $G \models \Sigma$, if $G \models \varphi$ for all $\varphi \in \Sigma$.

Example 2: Consider the graph G_0 depicted in Fig. 1, in which the users u_0 to u_{k_0+1} form a clique with all edges labeled friend, and all the users have the same city attribute value and similar profiles (not shown). Then $G_0 \not\models \varphi_a$ for GAR φ_a of Example 1. This is due to the match h of Q_a in G_0: $x_0 \mapsto u_0$, $x'_0 \mapsto u_1$, $x_i \mapsto u_{i+1} (i \in [1, k_0])$, $y_0 \mapsto v$, which satisfies precondition X_a, but $h \not\models \text{colleague}(x_0, x'_0)$. □

Notations of the paper are summarized in Table 1.

3 A DISCOVERY SCHEME

In this section, we formulate the notions associated with the GAR discovery problem, and propose a new discovery scheme.

GARs to discover. In practice, we want a minimum set of GARs that are non-redundant and nontrivial. Thus we consider only GARs $Q[\bar{x}](X \to p_0)$ in which p_0 does not appear in X, since otherwise the GAR is trivial and not useful. Moreover, we want GARs that are helpful for the downstream applications $\mathcal{A}$, specified as follows.

$\mathcal{A}$-relevant GARs. We model an application $\mathcal{A}$ as a set of predicates, also denoted by $\mathcal{A}$. We say that a GAR $\varphi = Q[\bar{x}](X \to p_0)$ is *$\mathcal{A}$-relevant* if the consequence p_0 is in $\mathcal{A}$. As an example, p_0 can be an edge predicate $\text{buy}(x, y)$, where x denotes a person and y denotes an item; it suggests person x is a potential buyer of item y.

This simple model originated from the observation that an application benefits from a specific class of association rules, whose consequences include entities of some particular "types", *i.e.,* the labels in p_0. For instance, when using association rules in marketing [73] (resp. intrusion detection [68], disease diagnosis [50]), the consequences only need to indicate that a customer buy a product (resp. a signature is generated by an attack attempt, a person is healthy or sick). A similar model has been adopted and shown effective in similarity search in heterogeneous network [67]. It abstracts nodes and edges to labels to define similarity measure.

Support. The support of a GAR $\varphi = Q[\bar{x}](X \to p_0)$ in a graph G should indicate how often φ can be applied to G. We consider connected pattern Q as commonly found in graph rules.

We quantify support of an $\mathcal{A}$-relevant GAR in terms of the number of distinct matches of Q in G that satisfy both the precondition X and consequence p_0, at p_0. More specifically, we assume *w.l.o.g.* that p_0 involves two variables x_{p_0} and x'_{p_0}; the case of one variable is defined similarly. Let $Q(G, Z, p_0) = \{\langle h(x_{p_0}), h(x'_{p_0})\rangle \mid h \in Q(G), h \models Z\}$ be a set of node pairs, where Z is a conjunction of predicates. Then its cardinality $\|Q(G, Z, p_0)\|$ counts the number of matches satisfying Z at the designated variables in the consequence p_0.

We define the *support* of GAR $\varphi = Q[\bar{x}](X \to p_0)$ in graph G as:

$$\text{sup}(\varphi, G) = \|Q(G, X \wedge p_0, p_0)\|.$$

Extending the support of [18, 22] that counts the number of matches at a single designated variable from $Q[\bar{x}]$, we treat both x_{p_0} and x'_{p_0} in p_0 as "pivots", to better estimate the effectiveness of $\mathcal{A}$-relevant φ, *e.g.,* whether application $\mathcal{A}$ benefits more from those φ having larger support. Here pivots refer to designated focus nodes representing users' interests. Below we show that this measure has the anti-monotonicity under a well-defined ordering of GARs.

Anti-monotonicity. We say that pattern $Q[\bar{x}] = (V_Q, E_Q, L_Q, \mu)$ *subsumes* pattern $Q'[\bar{x}'] = (V'_Q, E'_Q, L'_Q, \mu')$, denoted as $Q'[\bar{x}'] \sqsubseteq Q[\bar{x}]$, if $V'_Q \subseteq V_Q$, $E'_Q \subseteq E_Q$, and for each node $u \in V'_Q$ (resp. edge $e \in E'_Q$), either $L_Q(u) = L'_Q(u)$ or $L'_Q(u) = \_$ (resp. $L_Q(e) = L'_Q(e)$ or $L'_Q(e) = \_$) and u is paired with the same variable by μ and μ', *i.e.,* $\bar{x}' \subseteq \bar{x}$.

We define a partial order $\preceq$ on GARs. Consider two GARs $\varphi_1 = Q_1[\bar{x}_1](X_1 \to p_0)$ and $\varphi_2 = Q_2[\bar{x}_2](X_2 \to p_0)$ with the same p_0. Then $\varphi_1 \preceq \varphi_2$, referred to as φ_2 *subsumes* φ_1, if $Q_1[\bar{x}_1] \sqsubseteq Q_2[\bar{x}_2]$ and for each predicate p in X_1, p also appears in X_2.

We can see that when GAR φ_2 subsumes another GAR φ_1, both its pattern and precondition subsume their counterparts in φ_1. This results in the following anti-monotonicity.

Lemma 1: *Given two* GARs φ_1 *and* φ_2, *if* $\varphi_1 \preceq \varphi_2$, *then for any graph* G, $\text{sup}(\varphi_1, G) \geq \text{sup}(\varphi_2, G)$. □

With the order $\preceq$, a GAR φ is called *minimum* in graph G if $G \models \varphi$ and there is no other GARs φ' such that $\varphi' \preceq \varphi$ and $G \models \varphi'$. That is, each minimum GAR warrants the minimality of its graph pattern and precondition *w.r.t.* the consequence predicate p_0.

Example 3: Recall GAR φ_a from Example 1. Consider GAR φ'_a revised from φ_a by removing pattern edge $(x_0, \text{friend}, x_{k_0})$ from Q_a and predicate $\mathcal{M}_a(x_1, x_2, \text{similar\_profile})$ from X_a. Then $\varphi'_a \preceq \varphi_a$ and φ'_a induces more matches projected at the consequence. □

Cover. To further reduce redundant GARs, we use another notion. A set Σ of GARs *entails* a GAR φ, denoted by $\Sigma \models \varphi$, if for all graphs G, $G \models \Sigma$ implies $G \models \varphi$. As a special case, $\{\varphi_1\} \models \varphi_2$ if $\varphi_1 \preceq \varphi_2$, since φ_1 is less restrictive. A set Σ of GARs is *equivalent to* another set Σ', denoted as $\Sigma \equiv \Sigma'$, if $\Sigma' \models \varphi$ for any $\varphi \in \Sigma$, and vice versa.

A *cover* of a set Σ of GARs for graph G is a subset Σ^c of Σ such that (1) $\Sigma^c \equiv \Sigma$, (2) each GAR in Σ^c is minimum in G, and (3) $\Sigma^c \not\equiv \Sigma^c \setminus \{\varphi\}$ for any GAR φ in Σ^c, *i.e.,* the subset Σ^c is minimal.

Discovery problem. It is intractable to find a cover of $\mathcal{A}$-relevant GARs from a graph [18], since the problem is already intractable for GFDs and GFDs are a special case of GARs. To speed up this process, we discover GARs from a set H of sample graphs $H(\mathcal{A}, \rho\%)$ extracted from G such that (a) the size $|H(\mathcal{A}, \rho\%)|$ accounts for $\rho\%$ of $|G_{\mathcal{A}}|$ and (b) $H(\mathcal{A}, \rho\%)$ has representative data pertaining to $\mathcal{A}$.

Recall. Observe that H may not cover all the information of G, and hence GARs discovered from H may not include all those GARs that are mined from entire G. To assess the quality of samples $H(\mathcal{A}, \rho\%)$ for GAR discovery, we adapt the notion of recall *w.r.t.* support bounds σ. Denote by Σ_H (resp. Σ_G) the set of $\mathcal{A}$-relevant GARs discovered from H (resp. G). We use *recall* w.r.t. σ, denoted as $\text{recall}(\Sigma_H, \Sigma_G, \sigma)$, to refer to the percentage of the GARs φ in Σ_G that are also in Σ_H with $\text{sup}(\varphi, G) \geq \sigma$, *i.e.,* φ has support at least σ

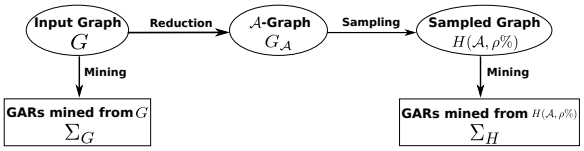


Figure 2: Workflow of GAR discovery

when applied to G. This recall indicates to what extent the required frequent GARs can be mined from the small samples $H(\mathcal{A}, \rho\%)$.

Problem statement. We are now ready to state the discovery problem of GARs *w.r.t.* a given application $\mathcal{A}$ and graph sampling.

- *Input*: A graph G, an application $\mathcal{A}$, a positive integer k, support threshold $\sigma > 0$, and a positive percentage $\gamma\%$ for recall.
- *Output*: A cover Σ_H^c of Σ_H mined from H such that recall(Σ_H, Σ_G, σ) $\geq \gamma\%$ and each φ in Σ_H has at most k pattern nodes.

Here H is the set of sample graphs, and Σ_G (resp. Σ_H) is the set of $\mathcal{A}$-relevant GARs mined from G (resp. H), as described above.

Intuitively, the set Σ_H is accurate, since at least $\gamma\%$ of the $\mathcal{A}$-relevant GARs with support no smaller than σ in the entire graph G are covered by Σ_H. Following [18], we adopt parameter k to balance the complexity of discovery and the interpretability of GARs. It is an input parameter decided by user's demand in practice.

Remark. (1) In practice, we discover GARs from a possibly dirty graph G, hence we cannot expect that the correct GARs are satisfied by G. We only discover GARs that have support above a given threshold and have high confidence, *i.e.,* the percentage of pivots satisfying X that also satisfy p_0, following the common practice of data mining. Domain experts will then examine the discovered GARs before the rules can be applied to association deduction. (2) Discovering GARs involves mining of both their patterns and dependencies. The former is similar to frequent subgraph mining [5], which conducts subgraph enumeration during the discovery. Besides, data dependencies are discovered on the identified entities, by incorporating predicates such as value bindings and ML classifiers.

Discovery scheme. We propose a 3-step scheme to discover GARs, whose workflow is depicted in Figure 2.

Application-driven reduction (Section 4). The first step of the scheme deduces a graph $G_{\mathcal{A}}$ from G, referred to as the $\mathcal{A}$-*graph* of G, by retaining only the data pertaining to the given application $\mathcal{A}$.

Graph sampling (Section 5). Since $G_{\mathcal{A}}$ may still be large, the scheme deduces samples $H(\mathcal{A}, \rho\%)$ from $G_{\mathcal{A}}$ to further reduce the cost. We propose a sampling method to ensure support and recall bounds.

Mining (Section 6). The final step is to discover $\mathcal{A}$-relevant GARs Σ_H from small samples $H(\mathcal{A}, \rho\%)$ by using a parallelly scalable algorithm. It computes and returns a cover Σ_H^c of Σ_H.

4 APPLICATION DRIVEN DISCOVERY

In this section, we show how to reduce big graphs G to smaller $\mathcal{A}$-graphs $G_{\mathcal{A}}$ for a given application $\mathcal{A}$, via an ML-based method. We start with some notations, and then present the method.

Label triplets. A *label triplet* is defined as $\langle l_v, l_e, l'_v \rangle$, where l_v and l'_v are two node labels and l_e is an edge label in between.

We say that an edge $e=(v, l, v')$ *conforms to* a label triplet $t=\langle l_v, l_e, l'_v \rangle$ if $L(v)=l_v$, $l=l_e$ and $L(v')=l'_v$. Here the special wildcard '_' also "equals" any arbitrary label. We refer to $\langle L(v), l, L(v') \rangle$ as the *label triplet* $\mathbb{T}(e)$ *of edge* e. For a set T of label triplets, a graph G *conforms to* T if each edge e in G conforms to a triplet in T.

We also define label triplets for predicates, by abstracting labels from patterns. The *label triplets of a predicate* p of pattern $Q[\bar{x}]$, denoted as $\mathbb{T}(p)$, is (a) $\{\langle L_Q(x), l, L_Q(y) \rangle\}$ when p is $l(x, y)$ or $\mathcal{M}(x, y, l)$; (b) $\{\langle L_Q(x), \_, \_ \rangle, \langle \_, \_, L_Q(x) \rangle\}$ when p is $x.A$ or $x.A{=}c$; and (c) $\{\langle L_Q(x), \_, L_Q(y) \rangle, \langle L_Q(y), \_, L_Q(x) \rangle\}$ when p is $x.A{=}y.B$.

Intuitively, for an application $\mathcal{A}$ modeled as a set of predicates, the label triplets of predicates form a simple abstraction of $\mathcal{A}$. We opt to use language (ML) models to learn and analyze the distribution of label triplets created from application $\mathcal{A}$, which indicates the characteristics of the data pertaining to $\mathcal{A}$.

Example 4: Continuing with Example 1, the label triplets of predicates x_0.city = x_i.city and $\mathcal{M}_a(x_i, x_j,$ similar_profile) in φ_a are $\{\langle$user, _, user$\rangle\}$ and $\{\langle$user, similar_profile, user$\rangle\}$, respectively. □

ML models and graph reduction. Given an application $\mathcal{A}$, *i.e.,* a set of predicates, a well-trained ML classifier $\mathcal{M}(x, y, l)$ for *e.g.,* link prediction, and a graph G, we employ a language model $\mathcal{M}_{\mathcal{A}}$, implemented as long short-term memory (LSTM) networks [29], to deduce the $\mathcal{A}$-graph $G_{\mathcal{A}}$ in the following four stages.

(1) Firstly, we expand graph G to $G_{\mathcal{M}}$ by adding edges predicted by $\mathcal{M}(x, y, l)$. In fact, due to the use of label triplets, two isolated nodes cannot be classified as the data pertaining to $\mathcal{A}$ and retained in $G_{\mathcal{A}}$. However, they may contribute to the support of an $\mathcal{A}$-relevant GAR when it has ML predicate $\mathcal{M}(x, y, l)$ and $\mathcal{M}$ predicts the existence of an edge between the two nodes. In light of this, expanding G with predicted links allows us to discover GARs with ML predicates.

(2) Taking triplets $\mathbb{T}(p)$ of each predicate p in $\mathcal{A}$ as seed input and treating each triplet as a word, we enforce the trained language model $\mathcal{M}_{\mathcal{A}}$ (see below for the training of $\mathcal{M}_{\mathcal{A}}$) to generate a number of sequences of label triplets, denoted as $\Theta_{\mathcal{A}}$. Since the LSTM-based $\mathcal{M}_{\mathcal{A}}$ models the probability of sentence generation, the generated sequences are *semantically* related to $\mathbb{T}(p)$.

(3) We select the top-m frequent triplets from $\Theta_{\mathcal{A}}$ to construct a set $T_{\mathcal{A}}$ of label triplets, referred to as $\mathcal{A}$-*triplets*. Here m is a predefined positive integer. That is, we focus on triplets that are *most closely* related to application $\mathcal{A}$. Such $\mathcal{A}$-triplets and the triplets of the predicates in $\mathcal{A}$ co-occur with high probability. Thus it is very likely that the $\mathcal{A}$-relevant GARs include predicates related to these label triplets, and the (pattern) edges in such GARs also conform to them.

(4) We finally deduce $\mathcal{A}$-graph $G_{\mathcal{A}}$ from $G_{\mathcal{M}}$ by preserving only the edges conforming to $T_{\mathcal{A}}$. In particular, all attributes of a node are kept if one of its adjacent edges is preserved. Filtered by triplets, $G_{\mathcal{A}}$ conforms to $T_{\mathcal{A}}$ and contains only the data pertaining to $\mathcal{A}$.

The reduction takes a time linear to the number of generated triplets to run LSTM model and $O(|G_{\mathcal{M}}|)$ time to filter out irrelevant edges. Here we choose LSTM network since it can effectively model the semantics of labels on paths in knowledge graphs [39–41]. That is, given an edge label l, LSTM generates a path following l with reasonable semantic meaning [47]. Note that $\mathcal{M}_{\mathcal{A}}$ can also be implemented by other language models for sequence modeling [54].

Example 5: Using label triplet $\{\langle$user, colleague, user$\rangle\}$ of the consequence colleague(x_0, x'_0) in GAR φ_a of Example 1 as seed

input, $\mathcal{M}_{\mathcal{A}}$ outputs sequences of triplets, in which the top-4 frequent ones are $\{\langle \text{user}, \_, \text{user}\rangle\}$, $\{\langle \text{user}, \text{similar\_profile}, \text{user}\rangle\}$, $\{\langle \text{user}, \text{friend}, \text{user}\rangle\}$, $\{\langle \text{user}, \text{follow}, \text{organization}\rangle\}$. These make the set $T_{\mathcal{A}}$. Then we preserve edges conforming to $T_{\mathcal{A}}$ for $\mathcal{A}$-graph $G_{\mathcal{A}}$, where irrelevant data, *e.g.,* commute methods is dropped. □

Model training. To train the language model $\mathcal{M}_{\mathcal{A}}$, *i.e.,* LSTM, we prepare a training corpus $\mathcal{D}_T$, which consists of sequences of label triplets that are collected by random walks in graph $G_{\mathcal{M}}$.

More specifically, for each path $(e_1, e_2, \ldots, e_n)$ generated by a random walk in $G_{\mathcal{M}}$, we derive the label triplet of each edge and add sequence $(\mathbb{T}(e_1), \mathbb{T}(e_2), \ldots, \mathbb{T}(e_n))$ to $\mathcal{D}_T$ (a path is a list of consecutive edges e_i=(v_i, l_i, v_{i+1})). The label triplets in each such sequence are semantically related, whose distribution can be learned by $\mathcal{M}_{\mathcal{A}}$. We apply non-backtracking random walks (NBTRW) [37] to sample paths since it restrains the bias towards visiting high-degree nodes and can closely knit communities around seed nodes. This property helps us capture more representative localized structures in $G_{\mathcal{M}}$.

Once $\mathcal{D}_T$ collects adequate sequences, the model $\mathcal{M}_{\mathcal{A}}$ views each label triplet as a word and each sequence of triplets as a sentence. It learns the likelihood of the occurrence of a triplet based on the previous sequences of triplets in $\mathcal{D}_T$. Thus, given label triplets $\mathbb{T}(p)$ as seed input, a well-trained $\mathcal{M}_{\mathcal{A}}$ is able to generate sequences of triplets that are related to $\mathbb{T}(p)$ as mentioned above.

Remark. For each graph $G_{\mathcal{M}}$, this unsupervised model training process needs to be performed only once such that the trained model $\mathcal{M}_{\mathcal{A}}$ can be applicable to different applications.

The training of model $\mathcal{M}_{\mathcal{A}}$ benefits from the usage of label triplets. Without them, $\mathcal{M}_{\mathcal{A}}$ has to inspect the triplets of attribute values, which are more diversified compared to labels. This larger vocabulary in the training corpus results in harder and slower training of $\mathcal{M}_{\mathcal{A}}$. In fact, abstracting edges and predicates as label triplets suffices to characterize the objective of applications, *e.g.,* a recommendation would only concern about recommending "Sports Shoes" to "Players" without considering their specific names.

After this graph reduction, the reliability of the discovered GARs *w.r.t.* application $\mathcal{A}$ is *relative to the accuracy* of model $\mathcal{M}_{\mathcal{A}}$. That is, the more correct $\mathcal{A}$-triplets are returned by $\mathcal{M}_{\mathcal{A}}$, the more $\mathcal{A}$-relevant GARs can be discovered from $G_{\mathcal{A}}$. Here an $\mathcal{A}$-triplet is said to be *correct* if there exists an $\mathcal{A}$-relevant GAR in G having a pattern edge or predicate that conforms to it.

5 SAMPLING BIG GRAPHS

In this section, we develop a sampling method to deduce a set H of sample graphs $H(\mathcal{A}, \rho\%)$ from $\mathcal{A}$-graph $G_{\mathcal{A}}$, such that the GARs mined from H satisfy the expected bounds on support and recall in graph G. We start with an overview of the sampling framework (Section 5.1) and introduce underlying techniques (Section 5.2). We then show the accuracy guarantees offered by it (Section 5.3).

5.1 Overview

For an application $\mathcal{A}$, an $\mathcal{A}$-relevant GAR $\varphi = Q[\bar{x}](X \rightarrow p_0)$ aims to "promote" the action specified by its consequence p_0. Thus when sampling big graphs G, we take distinct matches of variables of p_0 in G as the "pivots". This is justified by the support of φ, which is measured by the number of pivots (see Section 3). Intuitively, sampling the pivots helps us discover GARs with a high support.

Algorithm 1: GSRD

Input: An $\mathcal{A}$-graph $G_{\mathcal{A}}$, a positive integer N, two sampling strategies M_v and M_s, and sample ratios $\rho_v\%$ and $\rho\%$.
Output: A set H of N sample graphs $H(\mathcal{A}, \rho\%)$.

```
1  i ← 0;   H ← nil;
2  repeat
3      C ← nil;
4      foreach predicate p0 involved in A do
5          C ← C ∪ PS(p0, G_A);   /* computing pivot sets */
6      S_A ← PSample(C, M_v, G_A, ρ_v%);
7      H(A, ρ%) ← LSample(S_A, M_s, G_A, ρ%);
8      H ← H ∪ {H(A, ρ%)};
9      i ← i + 1;
   until i = N;
10 return H;
```

Pivot sets. Consider a predicate p of pattern $Q[\bar{x}]$. The pattern $Q_p[\bar{x}_p]$ *induced by* p is the subgraph of $Q[\bar{x}]$ that only contains the corresponding pattern nodes of variables in p without any edge. The *pivot set of* p *in graph* G, denoted as $\text{PS}(p, G)$, is the set of matches of Q_p in G. Therefore, each pivot is *either a single node or a node pair* taken from G that matches the labels in Q_p.

A sampling framework. Based on pivot sets, we propose a Graph Sampling framework for Rule Discovery of GARs, denoted as GSRD. Algorithm 1 shows its main steps. The input of GSRD includes an $\mathcal{A}$-graph $G_{\mathcal{A}}$ (Section 4), the number N of sample graphs, two strategies M_v and M_s for sampling pivots and their surrounding subgraphs, respectively, as well as two sample ratios $\rho_v\%$ and $\rho\%$. It computes a set H of N sample graphs $H(\mathcal{A}, \rho\%)$ such that $|H(\mathcal{A}, \rho\%)| \leq \rho\% \times |G_{\mathcal{A}}|$, in N rounds.

Each round of GSRD deduces a sample graph $H(\mathcal{A}, \rho\%)$ and adds it to H (lines 3-9). It first finds the pivot set of each consequence predicate p_0 of application $\mathcal{A}$ in $\mathcal{A}$-graph $G_{\mathcal{A}}$, and collects all pivots in a set C (lines 3-5). It then deduces $H(\mathcal{A}, \rho\%)$ in two phases (lines 6-7).

(1) The first phase targets the pivot sets. More specifically, GSRD calls procedure PSample to sample pivots from C, stored in a set $S_{\mathcal{A}}$ (line 6). PSample applies an input sampling strategy M_v (to be given in Section 5.2) to compute $S_{\mathcal{A}}$, and ensures that at most $\rho_v\%$ of the nodes from set C appear in the sampled pivots.

(2) In the second phase, GSRD samples substructures of the selected pivots from $G_{\mathcal{A}}$. It picks nodes and edges within k hops from the nodes in $S_{\mathcal{A}}$ to build sample graph $H(\mathcal{A}, \rho\%)$, via procedure LSample (line 7). Such sampled data cells constitute the "substructures" surrounding the pivots; and $H(\mathcal{A}, \rho\%)$ includes all pivots and their substructures. Procedure LSample adopts another input strategy M_s to extract substructures (see Section 5.2), which is a linear time operation in the worst case. In addition, LSample guarantees that the size of $H(\mathcal{A}, \rho\%)$ is at most $\rho\% \times |G_{\mathcal{A}}|$.

The sampling strategies can be randomized methods; hence the samples $H(\mathcal{A}, \rho\%)$ created in multiple rounds are different. More GARs can be mined from such samples as they cover more pivots.

Cost analysis. The cost for computing pivot sets is bounded by $O(|G_{\mathcal{A}}|)$ because it only needs label checking at constant times. Observe that extracting substructures is confined within the small localized areas around the pivots only in GSRD; thus

the two phases for sampling pivots and substructures take at most $O(|H(\mathcal{A}, \rho\%)|^2 \log(|H(\mathcal{A}, \rho\%)|))$ time, including the sorting cost for applying locality-aware sampling (Section 5.2).

5.2 Representative Strategies

We next present sampling methods M_v and M_s adopted by GSRD for selecting pivots and extracting substructures, respectively.

Sampling pivots. We propose a clustering-based strategy as M_v.

Clustering-assisted sampling. We propose to first cluster all pivots in the set C into multiple groups, such that each group consists of semantically similar pivots. We then construct a set of representative pivots by picking elements from *every* group guided by the ratio $\rho_v\%$, by employing one of two sampling strategies. Below we first show how to cluster pivots and then present the sampling strategies.

Since a pivot can also be a node pair, we cannot apply node clustering directly to $\mathcal{A}$-graph $G_{\mathcal{A}}$. In light of this, we convert $G_{\mathcal{A}}$ to an undirected $G'_{\mathcal{A}}$ such that each node pair of a pivot in $G_{\mathcal{A}}$ is contracted into a single node in $G'_{\mathcal{A}}$. In graph $G'_{\mathcal{A}}$, each contracted node has links to (a) the two nodes in the node pair and (b) other contracted nodes if the corresponding pairs have nodes in common.

Example 6: Figure 3(a) depicts a graph conversion process, in which the three directed edges are encoded as nodes r_0, r_1 and r_2 in the undirected graph. Here r_2 is connected to both r_0 and r_1 because of the nodes u_0 and p_1 that are shared by multiple edges. □

Intuitively, clustering allows us to pick pivots with diversified semantics from different groups, and discover useful GARs from samples of bounded size. In other words, it prevents us from discovering semantically homogeneous GARs only.

For efficient clustering, we adopt Lloyd's k-means algorithm [43] with k-means++ seeding [7]. We also use two approaches for extracting node features for clustering. One takes mean word embeddings [57] of the node attributes as the feature, since an application usually involves nodes with similar semantic meanings. The other learns node features with Deep Graph Informax (DGI) [72], where both topological structure and node attributes are considered.

Uniform sampling. We may select pivots from each group in *a uniform manner*, by randomly selecting each pivot independent of the others. Note that when sampling the pivots of edge predicates, it only considers those node pairs that are connected by edges in the $\mathcal{A}$-graph $G_{\mathcal{A}}$. By the semantics of GARs, only such pivots help us discover GARs that have edge predicates as the consequences.

After uniform sampling, it is assured that the selected pivots cover all the semantics pertaining to the given application $\mathcal{A}$ and more pivots are picked out from larger groups.

Locality-aware sampling. Alternatively, we may *greedily* choose pivots such that their substructures maximally overlap. More specifically, for each pivot, we estimate the "scope" of its substructure in $G_{\mathcal{A}}$ using a fixed substructure extraction scheme (see below). Then each time we pick a pivot such that the inclusion of its substructure leads to minimum size increase of the sample graph.

Compared to uniform sampling, this strategy creates more compact sample graphs when combined with substructure extraction. As another consequence, more pivots can be included in sample graphs of a fixed size, from which we can mine more GARs.

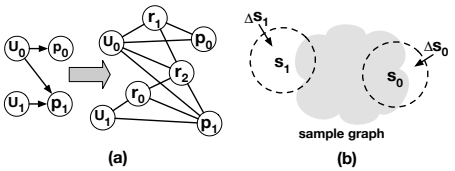


Figure 3: Demonstration of different strategies in GSRD

Example 7: The locality-aware sampling prefers the pivot s_0 to s_1 shown in Fig. 3(b). This is because the changes Δs_0 induced by s_0 to the sample graph is smaller than that induced by s_1, *i.e.,* Δs_1. □

Substructure extraction. We can use breadth-first search (BFS) as strategy M_s for extracting substructures of the pivots selected by M_v above. Starting from a sampled pivot v, the BFS proceeds up to the fixed depth k, and fetches the k-hop neighborhood of v as its substructure, guided by ratio $\rho\%$. We also remark that there exist other optimized extraction strategies for M_s (see Section 7).

5.3 Accuracy Guarantee

We now study the quality of the graphs H that are sampled by GSRD for GAR discovery. Given desired bounds on recall and support of GARs in the entire graph G, we show how to decide the number N of samples and their sample ratios $\rho\%$ for running GSRD accordingly, and deduce the support threshold for mining GARs from H.

Characterization. We start with an observation. As observed in [52], real-life graphs often have a power-law degree distribution. The application of GARs is analogous, *i.e.,* a small number of nodes are involved in the matches of patterns in most GARs. In practice, nodes with larger degrees are more likely to be matched by the patterns. Thus to have a high recall when mining GARs from sample graphs, we need to sample such critical nodes as many as possible.

We next formalize this observation to estimate the recall.

Formalization. Given a GAR $\varphi = Q[\bar{x}](X \to p_0)$ and a node v in graph G, we say that v is a *pivot* of φ and *contributes to* the support of φ in G if v appears in the pivot set $\mathsf{PS}(\varphi, G)$. Here $\mathsf{PS}(\varphi, G)$ is defined as $Q(G, X \wedge p_0, p_0)$ (see Section 3). It helps analyze the accuracy bound below and is different from the previous notion of pivot set defined *w.r.t.* a single predicate p.

Let $\Sigma_{G_{\mathcal{A}}}$ be the set of $\mathcal{A}$-relevant GARs with support at least σ in the $\mathcal{A}$-graph $G_{\mathcal{A}}$, and $\gamma\%$ be an expected recall value, *i.e.,* we want to mine $\gamma\% \times \|\Sigma_{G_{\mathcal{A}}}\|$ many $\mathcal{A}$-relevant GARs in $\Sigma_{G_{\mathcal{A}}}$ from the sample graphs. We use two variables $\rho_{\max}\%$ and $\rho_{\min}\%$ to model the power-law distribution *w.r.t.* nodes and GARs. Here $\rho_{\max}\%$ (resp. $\rho_{\min}\%$) denotes the maximum (resp. minimum) percentage of the nodes in $G_{\mathcal{A}}$ that can contribute to the support of $\gamma\% \times \|\Sigma_{G_{\mathcal{A}}}\|$ many $\mathcal{A}$-relevant GARs from $\Sigma_{G_{\mathcal{A}}}$. Then the recall $\gamma\%$ satisfies the following:

$$\gamma\% = \left(\frac{\rho_{\max}\%}{\rho_{\min}\%}\right)^{-\Delta}.$$

The exponent Δ can be estimated by using parameter estimation methods for power-law distribution [15]. Intuitively, the larger Δ is, the fewer critical nodes can contribute to the support of most GARs. Moreover, for each node v sampled in the first phase and any GAR φ with $\mathsf{sup}(\varphi, G_{\mathcal{A}}) \geq \sigma$, $v \in \mathsf{PS}(\varphi, G_{\mathcal{A}})$ if and only if $v \in \mathsf{PS}(\varphi, H_v)$. Here H_v denotes the substructure of v extracted via BFS.

When the ML model $\mathcal{M}_{\mathcal{A}}$ used for deducing $G_{\mathcal{A}}$ is accurate, the $\mathcal{A}$-graph $G_{\mathcal{A}}$ in the analyses below can be replaced by graph G.

Accuracy bound. Denote by $V_{\mathcal{A}}$ the node set of the $\mathcal{A}$-graph $G_{\mathcal{A}}$, and by V_H the set of nodes sampled by GSRD as pivots in each sample $H(\mathcal{A}, \rho\%)$, respectively. We have the following.

Theorem 2: *For an $\mathcal{A}$-graph $G_{\mathcal{A}}$, an expected recall value $\gamma\%$, support threshold σ for required $\mathcal{A}$-relevant* GARs $\Sigma_{G_{\mathcal{A}}}$ *in $G_{\mathcal{A}}$, and a constant $\varepsilon \in (0, 1)$, if the support threshold for $\mathcal{A}$-relevant* GARs *is $\sigma' = \lceil \frac{\|V_H\|}{\|V_{\mathcal{A}}\|}\sigma(\gamma\%)^{1/\Delta} + 1 \rceil$ in sample graphs, then after creating a set H of $N = \lceil \ln\varepsilon \,/ \left(1 - \exp(-\frac{(\gamma\%)^{1-\frac{1}{\Delta}}\left(\|V_H\|\sigma(\gamma\%)^{\frac{1}{\Delta}} - \|V_{\mathcal{A}}\|\sigma'\right)^2}{3\|V_H\|\|V_{\mathcal{A}}\|\sigma}\right)\rceil$ sample graphs by* GSRD *with BFS as substructure extraction strategy,* $\mathsf{recall}(\Sigma_H, \Sigma_{G_{\mathcal{A}}}, \sigma) \geq \gamma\%$ *with probability* $1 - \varepsilon$. □

Proof sketch: We prove this in two steps: (1) when discovering $\mathcal{A}$-relevant GARs with the computed support threshold σ' in a sample graph deduced via GSRD, $\mathsf{recall}(\Sigma_H, \Sigma_{G_{\mathcal{A}}}, \sigma)$ is no smaller than $\gamma\%$ with a specific probability p_x based on Chernoff bound [49]; and (2) after mining GARs from $N = \lceil \frac{\ln(1-\varepsilon)}{\ln(1-p_x)} \rceil$ sample graphs deduced via GSRD in the same setting, the probability reaches $1 - \varepsilon$. □

Remark. (1) Note that the percentage of $\|V_H\|$ in $\|V_{\mathcal{A}}\|$ provides a guideline to determine the sample ratio $\rho_v\%$, guided by Theorem 2. (2) One can verify that for each GAR φ in Σ_H that has support σ' in sample graph $H(\mathcal{A}, \rho\%)$ of H, φ has support of at least σ' in the entire graph G, since each $H(\mathcal{A}, \rho\%)$ is essentially a subgraph of G.

Example 8: Consider an $\mathcal{A}$-graph $G_{\mathcal{A}}$ deduced from the citation network DBLP [1], which includes $\|V_{\mathcal{A}}\| = 16M$ nodes. Suppose that the expected support threshold σ is 50 on $G_{\mathcal{A}}$, we need a recall of 90% *w.r.t.* support 50, and $\frac{\rho_{\min}\%}{\rho_{\max}\%} = 0.09$. By Theorem 2, we can see that to achieve this expected recall value, it suffices to create $N = 9$ sample graphs by GSRD such that $\|V_H\| = 4.8M$, and set the support threshold $\sigma' = 3$ for sample graphs. □

6 PARALLEL DISCOVERY

In this section, we develop a parallel algorithm for discovering GARs from the sample graphs in H, with the parallel scalability.

Sequential mining. To see the challenges inherent to GAR discovery, we start with a sequential algorithm for mining GARs, denoted as GARMine, by extending the GFD discovery algorithm of [18]. GARMine processes the N sample graphs in H one by one to mine GARs and returns their union. On each sample graph, it interleaves levelwise *pattern expansion* and *dependency expansion* to generate patterns Q and dependencies $X \rightarrow p_0$ for candidate GARs, respectively, following [18]. Apart from the constant and variable predicates of GFDs, here dependency expansion also includes new ML, attribute and edge predicates in GARs (see details shortly). GARMine returns those candidate GARs having support above the threshold σ', which is determined by Theorem 2.

GARMine takes exponential time in the worst case due to graph homomorphism needed in GAR validation (cf. [25]). To speed it up, we parallelize the discovery process. To measure the effectiveness of the parallelization, we review the notion of parallel scalability.

Parallel scalability. We adapt the parallel scalability of [32] to characterize the effectiveness of parallel algorithms for GAR discovery. Denote by $T_{\mathsf{seq}}(|H|, k, \sigma')$ the worst-case cost of a sequential GAR discovery algorithm $\mathbb{A}$, which finds GARs from H with support threshold σ' and bound k on pattern node numbers. We say that a parallel algorithm $\mathbb{A}_p$ for GAR discovery is *parallelly scalable relative to* $\mathbb{A}$ if with n processors,

$$T_{\mathsf{par}}(|H|, k, \sigma', n) = O\Big(\frac{T_{\mathsf{seq}}(|H|, k, \sigma')}{n}\Big),$$

where $T_{\mathsf{par}}(|H|, k, \sigma', n)$ is the parallel cost of $\mathbb{A}_p$. Intuitively, a parallelly scalable $\mathbb{A}_p$ "linearly" reduces the cost of $\mathbb{A}$ when n increases.

The main result of this section is the following.

Theorem 3: *There exists an algorithm* ParGARMine *for* GAR *discovery that is parallelly scalable relative to* GARMine. □

Below we give a constructive proof by presenting ParGARMine.

Algorithm 2: ParGARMine

Input: A set H of N sample graphs $H(\mathcal{A}, \rho\%)$, processors $P_1, \ldots, P_n$, positive integer k and support threshold σ'.
Output: A set Σ_H of all minimum GARs in H such that each has at most k pattern nodes and a support at least σ' in H.

1 distribute the sample graphs to n processors;
2 $\Sigma_H \leftarrow nil$; $\ell_q \leftarrow 1$; $Q^0 \leftarrow nil$;
3 **while** $\ell_q \leq k^2$ **do**
4 $Q^{\ell_q} \leftarrow \mathsf{QExpand}(\ell_q, Q^{\ell_q - 1})$;
5 parallel matching of patterns in Q^{ℓ_q}; adjust Q^{ℓ_q} *w.r.t.* σ';
6 compute the maximum size ℓ_p^m for preconditions *w.r.t.* Q^{ℓ_q};
7 $\ell_p \leftarrow 0$; $\Sigma^{-1} \leftarrow nil$;
8 **while** $\ell_p \leq \ell_p^m$ **do**
9 $\Sigma^{\ell_p} \leftarrow \mathsf{PExpand}(\ell_p, \Sigma^{\ell_p - 1}, Q^{\ell_q})$;
10 parallel validation of the GARs in Σ^{ℓ_p};
11 extend Σ_H with the verified GARs *w.r.t.* σ' ;
12 $\ell_p \leftarrow \ell_p + 1$;
13 $\ell_q \leftarrow \ell_q + 1$;
14 **return** Σ_H;

Parallel mining. Algorithm ParGARMine conducts levelwise expansion of patterns and dependencies simultaneously at a designated coordinator. It validates candidate patterns and GARs in parallel with n workers, since the expensive subgraph matching in validation dominates the cost of the discovery process. It extends parallel GFD discovery [18] by (a) partitioning multiple sample graphs, (b) supporting attribute, edge and ML predicates, and (c) applying new pruning strategies during the expansions.

The details of ParGARMine are shown in Algorithm 2. It starts by evenly allocating computing resources to sample graphs (line 1), such that each sample is assigned a distinct set of $\lfloor \frac{n}{N} \rfloor$ processors except one that takes all the rest. We fragment and distribute each sample graph across $\lfloor \frac{n}{N} \rfloor$ workers via vertex-cut partitioning [78].

Following the BSP model [70], ParGARMine next works in k^2 rounds to generate and validate GARs (lines 3-13). As k pattern nodes result in at most k^2 edges, each one of the k^2 rounds intends to find GARs with a specific number of pattern edges in $[1, k^2]$.

Pattern expansion. In each round ℓ_q, ParGARMine expands patterns at level ℓ_q by creating a set Q^{ℓ_q} of patterns with ℓ_q edges via procedure QExpand at coordinator P_c (line 4). QExpand generates Q^{ℓ_q} by expanding each pattern in $Q^{\ell_q - 1}$ with a single new edge; initially the edges in Q^1 should conform to the triplets

of predicates in application $\mathcal{A}$ (Section 4). ParGARMine then computes the matches of such patterns in sample graphs using the parallel pattern matching strategy of [18]; it prunes from Q^{ℓ_q} all patterns that have less than σ' matches in each sample (line 5).

Dependency expansion. Given patterns Q^{ℓ_q}, ParGARMine expands dependencies $X \rightarrow p_0$ at level ℓ_p to produce candidate GARs, in ℓ_p^m iterations (lines 8-12). Here ℓ_p^m denotes the maximum number of predicates in X, estimated by combinations of possible predicates *w.r.t.* Q^{ℓ_q}. In each iteration ℓ_p, procedure PExpand is called at the coordinator to compute a set Σ^{ℓ_p} of GARs, such that each one has a pattern from Q^{ℓ_q} and ℓ_p predicates in precondition X (X=$\emptyset$ when ℓ_p=0) (line 9), where X is expanded from a counterpart in Σ^{ℓ_p-1} with one new predicate. To speed up the process, PExpand associates each pair of nodes with a set of predicates that they satisfy, similar to the evidence sets for discovering DCs [42]. ParGARMine validates GARs in Σ^{ℓ_p} by extending the parallel method of [18]. It reserves those GARs that meet support bound σ' and have high confidence (lines 10-11).

Handling edge and ML predicates. In dependency expansion, procedure PExpand also generates possible attribute, edge and ML predicates that are unique to GARs. More specifically, for each candidate pattern Q and attribute dependency $X \rightarrow p_0$, PExpand expands X with $x.A$, $l(x, y)$ or $\mathcal{M}(x, y, l)$ for all x and y in Q, l in the label set Γ, A in the attribute set Υ, and ML classifiers $\mathcal{M}$ if applicable. Recall that $\mathcal{A}$-graph $G_{\mathcal{A}}$ already incorporates edges predicted by ML model (Section 4); hence when discovering $\mathcal{A}$-relevant GARs, we can treat ML predicates as edge predicates in the samples of the $\mathcal{A}$-graph. All such expanded dependencies will be validated.

For GFDs, the parallel validation is only conducted on matches computed for patterns Q^{ℓ_q} [18]. In contrast, due to the edge and ML predicates introduced by GARs, we have to inspect the existence of additional edges, which may reside at different workers. Thus, if a match h at processor P_i involves two nodes v and v' and if we need to check the existence of an edge e from v to v' together with h, ParGARMine transmits e to P_i from other workers if it exists before the local checking, using an additional superstep of BSP.

Cover. We revise the parallel implication checking algorithm for GFDs [18] to compute the cover Σ_H^c of GARs Σ_H returned by ParGARMine, based on a characterization of GAR implication [19].

Example 9: Consider GAR φ_a of Example 1 and suppose that its consequence predicate is covered by application $\mathcal{A}$. By the number of pattern edges, ParGARMine can find φ_a in round $2k_0 + 2$. More specifically, after generating the pattern Q_a in this round, it validates combinations of the predicates of Q_a to mine dependencies, including the one of φ_a that consists of $2k_0$ variable predicates, k_0^2 ML predicates and an edge predicate colleague(x_0, x_0'). Note that the ML predicates predict similar_profile links, which have been added to the $\mathcal{A}$-graph in graph reduction. ParGARMine performs similarly in checking the ML and edge predicates of φ_a, *i.e.,* inspecting similar_profile and colleague links, with necessary communication when the links and matches of Q_a are not at the same worker. □

Note that when the sample graphs are small enough to be deployed at a single processor, *e.g.,* some real-life graphs in Section 7, ParGARMine can also be implemented with another setting, where each sample is replicated at its assigned processors and the computation is evenly partitioned following the techniques in [23].

Pruning strategies. To reduce unnecessary expansion of patterns and dependencies that cannot yield minimum GARs *w.r.t.* support threshold σ', procedures QExpand and PExpand employ new pruning strategies in addition to those adopted in GFD discovery [18].

(a) *Incremental dependency expansion.* Given a pattern Q' expanded from patterns Q, PExpand only generates those dependencies $X' \rightarrow p_0'$ such that there exists $X \rightarrow p_0$ at a prior level whose predicates are covered by X' and p_0', and the support of $Q[\bar{x}](X \rightarrow p_0)$ exceeds the bound σ'. That is, ParGARMine maintains the valid dependencies in the prior levels to prevent from producing dependencies starting from scratch at the current level.

(b) *Interleaved pruning.* If the support of GAR $Q[\bar{x}](X \rightarrow p_0)$ is less than σ' for all $X \rightarrow p_0$, then QExpand only expands Q with edges having new nodes when Q is a path. This pattern pruning makes use of the information obtained during dependency expansion.

Both strategies leverage the anti-monotonicity of the support of GARs (Lemma 1). Without it, the algorithm easily expands a GAR φ' from φ such that $\varphi \preceq \varphi'$ but the support of φ cannot reach σ', which has already been verified in prior levels, *i.e., φ'* is useless.

Analyses. To see that algorithm ParGARMine is correct, observe the following. (a) The parallel matching method of [18] ensures that the matches of candidate patterns computed at n processors are the same as that deduced sequentially. (b) All the data related to validating edge and ML predicates in a GAR is sent to the same processor in advance. One can also verify its parallel complexity by analyzing the parallel cost incurred in each round.

7 EXPERIMENTAL STUDY

We experimentally evaluated (1) the effectiveness of application-driven graph reduction, (2) the quality of sample graphs produced by GSRD, (3) the speedup of GAR discovery with sample graphs, (4) the (parallel) scalability of algorithm ParGARMine, (5) the quality of the discovered GARs; and conducted (6) an ablation study for discovery.

Experimental setting. We start with the experimental setting.

Datasets. We used five real-life graphs: (1) DBLP [1], a real-life citation network with $0.2M$ nodes and $0.3M$ edges, where the attributes constitute bibliographic records of research papers in computer science; (2) YAGO [66], a knowledge graph with $3.5M$ nodes and $7.4M$ edges; (3) DBpedia [2], a larger knowledge graph with $5.2M$ nodes and $17.5M$ edges; its attribute values indicate various types of facts related to the entities (nodes); (4) IMDB [4], a graph database that includes attributes for the information of movies, directors and actors, having $5.1M$ nodes and $5.2M$ edges; and (5) movieLens [27], a movie recommendation network, with 10K nodes and 0.1M edges.

We also designed a graph generator to evaluate the scalability of the methods. The synthetic graphs have up to $7M$ nodes and $21M$ edges, with labels, attributes and values drawn from 70 symbols.

To find more practical GARs, following the observation in [6], we mainly discovered GARs having patterns with diameter at most 3, and imposed a bound on the number of cycles within the patterns. When constructing constant predicates, we used 5 most frequent values from the active domain of the attributes in the graphs.

Algorithms. We implemented the following, all in C++. (1) The graph reduction method (Section 4). (2) Various graph sampling approaches $\text{GSRD}_{\text{y+z}}$ in framework GSRD (Section 5), where y denotes the strategy for sampling pivots, including CA (cluster-assisted) and LC (locality-aware); and z denotes the strategy for extracting substructures, including OB (BFS), WB (BFS with bounded width) and RW (random walk). Here WB is a variant of OB, which takes an additional bound on the number of neighbors that each BFS step explores. It helps us mine GARs with patterns of a large diameter. RW also takes two parameters: the depth k of random walk, and the size of substructure. It performs random walk from a sampled pivot and is able to extract irregular substructures for pivots. (3) The parallel GAR discovery algorithm ParGARMine (Section 6). (4) A variant $\text{ParGARMine}_{\text{w}}$ of ParGARMine that discovers GARs from the entire graphs in parallel. (5) The parallel GFD discovery algorithm DisGFD [18], for efficiency comparison.

We also implemented three baseline graph sampling methods. (6) UniNode uniformly samples nodes with a ratio. It returns the subgraph induced by all the selected nodes as a sample graph. (7) UniEdge extracts edges from graphs in a uniform manner to build samples. (8) PRA, which samples paths with a linear path ranking model to select nodes that are mostly related to the query nodes [36]; we picked query nodes by uniform node sampling, preserving all edges connected to the sampled nodes. In addition, we compared with (9) AnyBURL [45] for efficiency in mining Horn rules.

ML models. We used the SimplE model [31] as the ML classifier in GARs due to its high accuracy and efficiency. To train SimplE, we adopted the default configurations in [31], and took 85% and 15% of each graph as the training set and validation set, respectively. The trained model is used to recover missing information.

The LSTM model used for graph reduction was implemented as [48] with its default training configuration and two 650 wide layers. We targeted the discovery of $\mathcal{A}$-relevant GARs for a specific application $\mathcal{A}$. By default we considered 7 predicates in a single $\mathcal{A}$.

The algorithms were deployed on a cluster of up to 16 machines connected by 10Gbps links. Each machine has 2 processors powered by Intel Xeon 2.2 GHz and 64 GB memory. All the experiments were repeated 5 times and the average is reported here.

Experimental results. We next report our findings.

Exp-1: Effectiveness of application-driven reduction. We first evaluated the performance of our graph reduction strategy. The ML-based method selects the top-m frequent label triplets from a candidate set generated by $\mathcal{M}_{\mathcal{A}}$, which are most closely relevant to the application $\mathcal{A}$ (Section 4). We studied the impact of m on the effectiveness of graph reduction by varying m from 3 to 10. We fixed the upper bound $k = 6$ for pattern nodes, and set the same support threshold as 1000 when mining GARs from both the graphs G and $\mathcal{A}$-graphs $G_{\mathcal{A}}$ deduced by the reduction. Here movieLens is omitted, since it only has 10 types of label triplets for scoring and ML models can hardly discern their semantics. The results on the other four real-life graphs are reported in Table 2. We find the following.

(1) The graph reduction ratio (reduc.), measured as the ratio of removed data to the entire graph, *i.e.,* $\frac{|G|-|G_{\mathcal{A}}|}{|G|}$, becomes smaller when m increases, as expected since all data conforming to the m triplets is preserved in $\mathcal{A}$-graph $G_{\mathcal{A}}$. In particular, the reduction is very effective for YAGO and DBpedia, with an average ratio of 94% when $m = 7$. This is because that most data in these comprehensive knowledge graphs is irrelevant to a given specific application.

Table 2: Effectiveness of ML-based graph reduction

Graphs	top-3		top-7		top-10	
	Reduc.	Recall	Reduc.	Recall	Reduc.	Recall
DBLP	57%	58%	53%	67%	50%	100%
IMDB	71%	71%	67%	100%	63%	100%
YAGO	98%	73%	96%	83%	86%	91%
DBpedia	94%	78%	92%	100%	90%	100%

(2) Over the four real-life graphs, the recall of GARs discovered from $G_{\mathcal{A}}$ is on average 87% (resp. 98%) when m is 7 (resp. 10).

(3) Compared to mining GAR from entire graphs G, on average the discovery of GARs from $G_{\mathcal{A}}$ achieves a speedup of 7.6 times when $m = 7$, using the parallel algorithm $\text{ParGARMine}_{\text{w}}$ (not shown). In addition, such $G_{\mathcal{A}}$ can be constructed in 320 seconds on average.

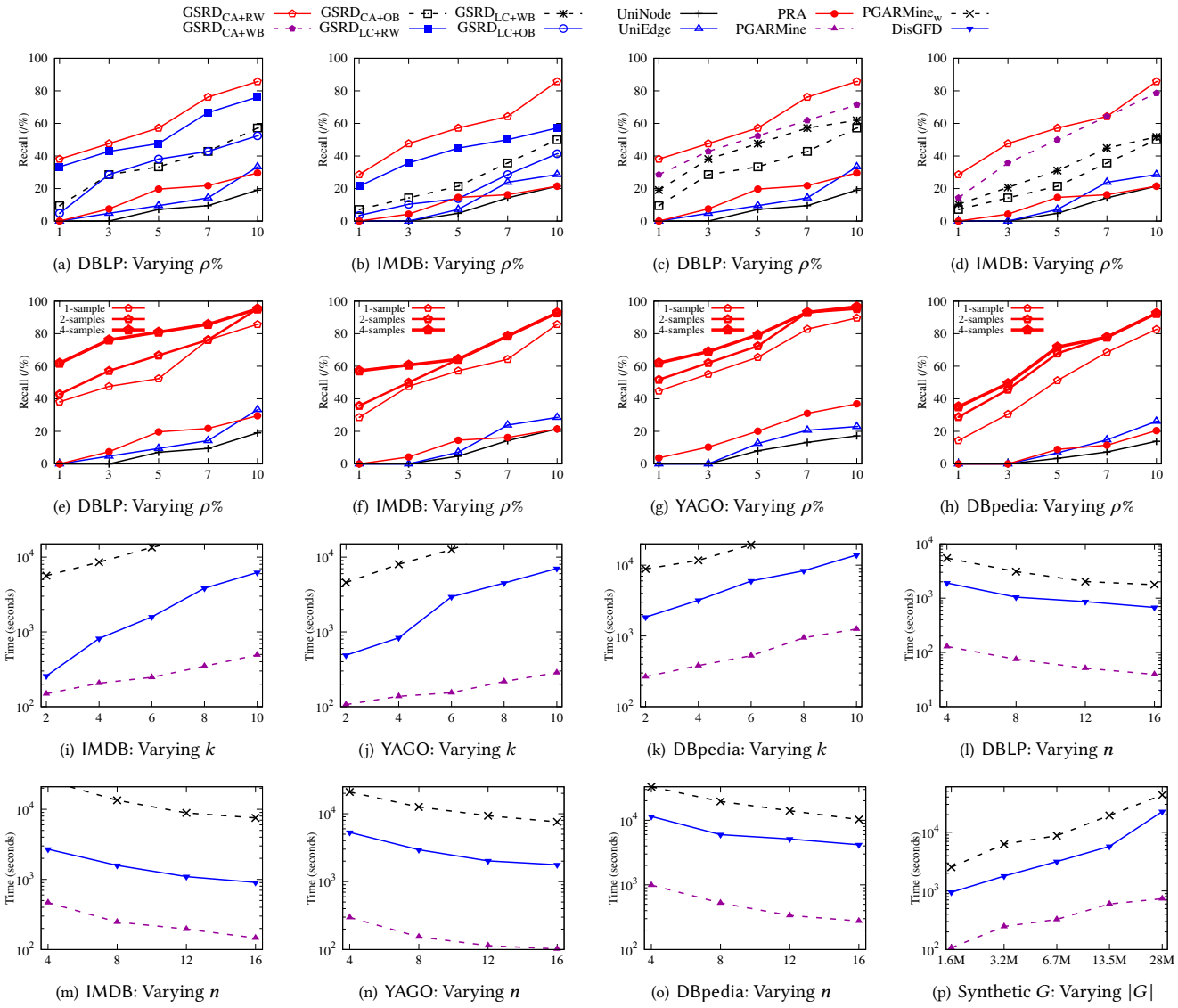
These verify the effectiveness of the ML-based graph reduction. For all the other experiments, m was set to be 7 by default.

Exp-2: Effectiveness of graph sampling. We next evaluated the quality of samples $H(\mathcal{A}, \rho\%)$ deduced by GSRD. Varying the sample ratio $\rho\%$, we assessed the impact of different (a) pivot sampling policies, (b) substructure extraction methods, and (c) the number N of sample graphs on the recall of GARs discovered. The support thresholds σ on $\mathcal{A}$-graphs $G_{\mathcal{A}}$ were set as 1000 in these experiments. In addition, when enforcing BFS with bounded width (resp. random walk) for substructure extraction, the bound on width (resp. size of substructure) was set as 3 (resp. 30) by default.

(1) Impact of pivot sampling. Fixing $k = 8$ and $N = 1$, we varied $\rho\%$ from 1% to 10% on DBLP and IMDB. Here the support thresholds σ' for discovery in sample graphs were determined by following the formula in Theorem 2, which is unaffected by the specific substructure extraction strategy and hence is applicable to both WB and RW as well. As shown in Figures 4(a) and 4(b), (a) $\text{GSRD}_{\text{CA+RW}}$ consistently performs the best among all the methods. (b) It outperforms $\text{GSRD}_{\text{LC+RW}}$ (resp. $\text{GSRD}_{\text{LC+OB}}$) by 24% (resp. 35%) on average in the recall of the discovered GARs, validating the need of node clustering in sampling pivots. We also find that clustering-assisted sampling exhibits little difference with different types of node features. It means that word embeddings, *i.e.,* node attributes, suffice to distinguish application-related pivots.

(2) Impact of substructure extraction. In the same setting, Figures 4(c) and 4(d) report the recall of GARs mined from the samples deduced by different strategies from DBLP and IMDB, respectively. We can see that when combing clustering with various approaches for substructure extraction, $\text{GSRD}_{\text{CA+RW}}$ still offers the highest recall and $\text{GSRD}_{\text{CA+WB}}$ performs better than $\text{GSRD}_{\text{CA+OB}}$. We examine the substructures extracted by RW and find that the semantics related to them, *e.g.,* node labels, is more diversified than those extracted by OB and WB, making it possible to find more GARs. That is, while the theoretical bounds (Theorem 2) are proved for $\text{GSRD}_{\text{CA+OB}}$, $\text{GSRD}_{\text{CA+RW}}$ achieves better bounds in practice since OB and WB introduce a bias towards high-degree nodes [33].

(3) Impact of N. Using the same k, σ and range of $\rho\%$ as in Exp-2(1), Figures 4(e) to 4(h) report the results with different number



(a) DBLP: Varying ρ% (b) IMDB: Varying ρ% (c) DBLP: Varying ρ% (d) IMDB: Varying ρ%

(e) DBLP: Varying ρ% (f) IMDB: Varying ρ% (g) YAGO: Varying ρ% (h) DBpedia: Varying ρ%

(i) IMDB: Varying k (j) YAGO: Varying k (k) DBpedia: Varying k (l) DBLP: Varying n

(m) IMDB: Varying n (n) YAGO: Varying n (o) DBpedia: Varying n (p) Synthetic G: Varying $|G|$

Figure 4: Performance evaluation

N of samples deduced from four real-life graphs (the results on movieLens are consistent and are not shown). Here we only tested GSRD$_{CA+RW}$ for framework GSRD, which has been verified to be the best combination in Exps(2)(a) and (2)(b) and this also holds when N>1. We find that (a) all sampling methods perform better with more sample graphs, as expected. (b) GSRD$_{CA+RW}$ on average beats UniNode (resp. UniEdge and PRA) by 8.0 (resp. 5.3 and 4.3) times when ρ% varies from 7% to 10% and using 2 samples, which is consistent with Figures 4(a) to 4(d). (c) The recall offered by GSRD$_{CA+RW}$ "converges" fast as N increases. For instance, on DBpedia, the recall already reaches 92% when N=2 and ρ%=10%. These validate the effectiveness of GSRD for GAR discovery.

We also find that if GSRD$_{CA+RW}$ is applied on original graphs with k = 6, N = 2 and ρ% = 10%, then an average recall of 72% is achieved (not shown). Compared with Exp-1, this suggests graph reduction contributes more to a better accuracy of GAR discovery.

Exp-3: Efficiency. Using n = 8 machines, we tested the efficiency of ParGARMine in finding GARs from sample graphs, and compared it with ParGARMine$_w$ and DisGFD that operate on the entire graphs. We took the sample graphs deduced by GSRD$_{CA+RW}$ from IMDB, YAGO and DBpedia with sample ratio 10% and N = 2; the support thresholds were decided along the same lines as that in Exp-2(1). Figures 4(i) to 4(k) report the runtime for mining GARs with different upper bounds k for patterns. The corresponding time for building samples is on average 198 seconds. We find the following.

(1) ParGARMine constantly outperforms both ParGARMine$_w$ and DisGFD, although GARs include edge, attribute and ML predicates beyond GFDs. ParGARMine is on average 60.6 (resp. 10.5) times faster than ParGARMine$_w$ (resp. DisGFD). DisGFD is faster than ParGARMine$_w$ since mining edge and ML predicates of GARs has to check additional edges beyond the matches of patterns.

(2) It is feasible for ParGARMine to discover GARs with large patterns. From $N = 2$ sample graphs of YAGO (resp. IMDB), it takes ParGARMine 285 (resp. 492) seconds to find GARs with $k = 10$, while ParGARMine$_w$ cannot terminate in 7 (resp. 16) hours.

(3) When ParGARMine is used to discover GFDs only, it outperforms DisGFD by 52.7 times when using $N = 2$ sample graphs.

Results on DBLP and movieLens are consistent and not shown.

(4) With 8 machines, we also compared the runtime of ParGARMine with AnyBURL in mining 100 Horn rules from DBLP and DBpedia. Here we target finding 100 rules with support above 1000 and path length of at most 3, since AnyBURL cannot return the complete set of rules. We find that ParGARMine is 3.0 (resp. 12.3) times faster than AnyBURL in Horn rule discovery on DBLP (resp. DBpedia).

Exp-4: Scalability. We tested the scalability of GAR discovery.

(1) Parallel scalability. Fixing $k = 6$ and employing the same ρ%, σ and N as in Exp-3, we varied the number n of machines used from 4 to 16. As reported in Figures 4(l) to 4(o) on the four real-life graphs, respectively, (a) ParGARMine and ParGARMine$_w$ are on overage 3.2 and 3 times faster, respectively, *i.e.,* they scale well with n. (b) ParGARMine outperforms ParGARMine$_w$ by 58.3 times on average. These empirically verify Theorem 3 and further show the effectiveness of sampling-based GAR discovery.

(2) Larger graphs. Fixing $k = 6$, $n = 8$, $\sigma = 1000$, $\rho\% = 10\%$ and $N = 2$, we varied the size $|G| = |V| + |E|$ of synthetic graphs G from $1.6M$ to $28M$. The results in Fig. 4(p) show that sampling-based GAR discovery can scale with large graphs, *e.g.,* when $|G| = 28M$, it takes 735 seconds to mine GARs from two sample graphs.

Exp-5: Effectiveness of GARs. We evaluated the effectiveness of the discovered $\mathcal{A}$-relevant GARs with different applications $\mathcal{A}$.

(1) Knowledge graph completion. We first applied GARs to restore missing information, including edges and attributes, in knowledge graphs YAGO and DBpedia. We picked edge and attribute predicates for application $\mathcal{A}$, *e.g.,* member_of(x, y) and x.education. To evaluate the performance, for each graph, we constructed a test set by randomly selecting $10K$ application-related edges and attributes that conform to the predicates of $\mathcal{A}$ in the original graph as positive samples, and picking $10K$ nonexistent links and attributes as negative ones. Then we applied the GARs discovered with GSRD$_{\text{CA+RW}}$ and $N = 2$, $\rho\% = 10\%$, $\sigma = 1000$, $k = 8$ and $m = 7$, referred to as the default setting, to classify the information in test set, *i.e.,* whether they should be restored. F-measure was employed as the accuracy metric, and link prediction models SimplE [31] and ComplEx [69] were treated as baselines. We find that enforcing the $\mathcal{A}$-relevant GARs consistently achieves high accuracy above 0.87, and on average it beats SimplE and ComplEx by 9.2% and 11.5%, respectively.

(2) Recommendation. As another case study, we tested the effectiveness of the $\mathcal{A}$-relevant GARs mined with default setting, in recommendation. We used movieLens and specified application $\mathcal{A}$ with predicate 5−star_rating(x, y), indicating that movie y should be recommended to person x. The configuration of train/test dataset split and training follows [28] for a fair comparison. We find that applying the GARs performs better than recommendation model LightGCN [28], with top-20 recommendation recall of 0.83 vs. 0.71.

Table 3: Ablation study on the efficiency of GAR discovery

Graphs	No graph reduction	No sampling	Full method
DBpedia	31.3s	1663.0s	8.0s
YAGO	117.6s	541.6s	18.6s

(3) Inconsistency detection. We also studied error detection. This experiment was conducted over DBpedia, with an application $\mathcal{A}$ consisting of predicate same_kingdom(x, y). That is, the "kingdom" of two species should be the same under certain conditions, *e.g.,* two species are in the same "class". We randomly drew 7% of species entities from DBpedia and changed their kingdom values, to form the test set. The $\mathcal{A}$-relevant GARs discovered with default setting were used to detect the erroneous kingdom values. We find that such GARs perform comparably to the entire set of GFDs mined from the same graph [18], with F-measure above 0.96.

Exp-6: Ablation study. Observe that Exp-1 and Exp-2 already show graph reduction is more important for achieving high recall in GAR discovery. We next preformed an ablation study using DBpedia and YAGO to investigate how each stage influences the efficiency. Fixing k=3, n=8, σ=1000, ρ%=10% and N=1, we omitted one of graph reduction and sampling stages. As shown in Table 3, the discovery time significantly increases when sampling is left out, indicating that sampling is more critical for improving efficiency.

Summary. We find the following. (1) The application-driven graph reduction method is effective. It reduces the graphs by 76% on average, while achieving recall of 85% for discovered $\mathcal{A}$-relevant GARs. (2) The sample graphs deduced by framework GSRD are of high quality. On 4 such samples with sample ratio 10%, more than 94% of the GARs in the $\mathcal{A}$-graphs $G_{\mathcal{A}}$ can be mined, with support at least 1000 in $G_{\mathcal{A}}$. (3) The sampling-based discovery scheme is efficient. It speeds up mining from the entire graphs by 60.6 times on 2 sample graphs with sample ratio 10%, while retaining recall above 91%. It speeds up algorithm DisGFD of [18] by 52.7 times for GFD discovery. (4) Algorithm ParGARMine is parallelly scalable: on average it is 3.2 times faster when n varies from 4 to 16.

8 CONCLUSION

We have explored a new approach for discovering rules from big graphs G, consisting of (1) a graph reduction scheme to deduce a smaller graph $G_{\mathcal{A}}$ of data pertaining to a given application $\mathcal{A}$, (2) a method to sample a set H of small graphs from $G_{\mathcal{A}}$, such that the rules mined from H satisfy given support and recall bounds in G, and (3) an algorithm with the parallel scalability to mine rules from small H instead of from the entire G, for GARs that may embed ML predicates and subsume GPARs and GEDs as special cases. We have experimentally verified that the approach is promising in reducing excessive number of rules and scaling with large graphs.

One topic for future work is to explore strategies to reduce the impact of noise in real-life graphs on rule discovery. Another topic is to develop incremental discovery algorithms on dynamic graphs.

ACKNOWLEDGMENTS

This work was supported in part by ERC 652976, Royal Society Wolfson Research Merit Award WRM/R1/180014, and NSFC 61902274.

REFERENCES

[1] 2021. DBLP collaboration network. *https://snap.stanford.edu/data/com-DBLP.html.*
[2] 2021. DBpedia. *http://wiki.dbpedia.org.*
[3] 2021. DingTalk. *https://www.dingtalk.com.*
[4] 2021. IMDB. *https://www.imdb.com/interfaces.*
[5] Charu C. Aggarwal, Mansurul Bhuiyan, and Mohammad Al Hasan. 2014. Frequent Pattern Mining Algorithms: A Survey. In *Frequent Pattern Mining*.
[6] Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. 2011. An Empirical Study of Real-World SPARQL Queries. *CoRR* abs/1103.5043 (2011).
[7] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *SODA*.
[8] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. 2009. Mining Graph Evolution Rules. In *ECML/PKDD*.
[9] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *Proc. VLDB Endow.* 11, 3 (2017), 311–323.
[10] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. 2005. Theory of Classification: a Survey of Some Recent Advances. *ESAIM: probability and statistics* 9 (2005), 323–375.
[11] Venkatesan T. Chakaravarthy, Vinayaka Pandit, and Yogish Sabharwal. 2009. Analysis of sampling techniques for association rule mining. In *ICDT*.
[12] Yang Chen, Sean Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. 2016. Ontological Pathfinding. In *SIGMOD*.
[13] Yang Chen, Daisy Zhe Wang, and Sean Goldberg. 2016. ScaLeKB: scalable learning and inference over large knowledge bases. *VLDB J.* 25, 6 (2016), 893–918.
[14] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509.
[15] Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. 2009. Power-Law Distributions in Empirical Data. *SIAM Rev.* 51, 4 (2009), 661–703.
[16] William W. Cohen. 2016. TensorLog: A Differentiable Deductive Database. *CoRR* abs/1605.06523 (2016).
[17] Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. 2015. Keys for Graphs. *Proc. VLDB Endow.* 8, 12 (2015), 1590–1601.
[18] 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.
[19] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Capturing Associations in Graphs. *Proc. VLDB Endow.* 13, 11 (2020), 1863–1876.
[20] 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.
[21] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.
[22] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *Proc. VLDB Endow.* 8, 12 (2015), 1502–1513.
[23] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *SIGMOD*.
[24] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*.
[25] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.
[26] Yulong Gu, Yu Guan, and Paolo Missier. 2020. Towards Learning Instantiated Logical Rules from Knowledge Graphs. *CoRR* abs/2003.06071 (2020).
[27] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2016), 19:1–19:19.
[28] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*.
[29] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[30] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (1999), 100–111.
[31] Seyed Mehran Kazemi and David Poole. 2018. SimplE Embedding for Link Prediction in Knowledge Graphs. In *NeurIPS*.
[32] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A Complexity Theory of Efficient Parallel Algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.
[33] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. 2011. Towards Unbiased BFS Sampling. *IEEE J. Sel. Areas Commun.* 29, 9 (2011), 1799–1809.
[34] 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). *Proc. VLDB Endow.* 12, 6 (2019), 653–666.
[35] Ni Lao, Einat Minkov, and William W. Cohen. 2015. Learning Relational Features with Backward Random Walks. In *ACL*.
[36] Ni Lao, Tom M. Mitchell, and William W. Cohen. 2011. Random Walk Inference and Learning in A Large Scale Knowledge Base. In *EMNLP*.
[37] Chul-Ho Lee, Xin Xu, and Do Young Eun. 2012. Beyond random walk and metropolis-hastings samplers: Why you should not backtrack for unbiased graph sampling. In *SIGMETRICS*.
[38] Cane Wing-ki Leung, Ee-Peng Lim, David Lo, and Jianshu Weng. 2010. Mining interesting link formation rules in social networks. In *CIKM*.
[39] Manling Li, Qi Zeng, Ying Lin, Kyunghyun Cho, Heng Ji, Jonathan May, Nathanael Chambers, and Clare Voss. 2020. Connecting the Dots: Event Graph Schema Induction with Path Language Modeling. In *EMNLP*.
[40] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2018. Multi-Hop Knowledge Graph Reasoning with Reward Shaping. In *EMNLP*.
[41] Yankai Lin, Zhiyuan Liu, Huan-Bo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015. Modeling Relation Paths for Representation Learning of Knowledge Bases. In *EMNLP*.
[42] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *Proc. VLDB Endow.* 13, 10 (2020).
[43] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.
[44] Frank Manola, Eric Miller, Brian McBride, et al. 2004. RDF Primer. *W3C recommendation* 10, 6 (2004), 1–107.
[45] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *IJCAI*.
[46] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. 2018. Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion. In *ISWC*.
[47] Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the State of the Art of Evaluation in Neural Language Models. In *ICLR*.
[48] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and Optimizing LSTM Language Models. In *ICLR*.
[49] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.
[50] Jesmin Nahar, Tasadduq Imam, Kevin S. Tickle, and Yi-Ping Phoebe Chen. 2013. Association rule mining to detect factors which contribute to heart disease in males and females. *Expert Syst. Appl.* 40, 4 (2013), 1086–1093.
[51] Mohammad Hossein Namaki, Yinghui Wu, Qi Song, Peng Lin, and Tingjian Ge. 2017. Discovering Graph Temporal Association Rules. In *CIKM*.
[52] Mark EJ Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary physics* 46, 5 (2005), 323–351.
[53] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. 2018. Robust Discovery of Positive and Negative Rules in Knowledge Bases. In *ICDE*.
[54] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. 2021. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Trans. Neural Networks Learn. Syst.* 32, 2 (2021), 604–624.
[55] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proc. VLDB Endow.* 8, 10 (2015), 1082–1093.
[56] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD*.
[57] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*.
[58] 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*.
[59] Sanjay Rathee and Arti Kashyap. 2018. Adaptive-Miner: an efficient distributed association rule mining algorithm on Spark. *J. Big Data* 5 (2018), 6.
[60] Matteo Riondato and Eli Upfal. 2014. Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees. *ACM Trans. Knowl. Discov. Data* 8, 4 (2014), 20:1–20:32.
[61] Matteo Riondato and Eli Upfal. 2015. Mining Frequent Itemsets through Progressive Sampling with Rademacher Averages. In *SIGKDD*.
[62] Hemant Saxena, Lukasz Golab, and Ihab F. Ilyas. 2019. Distributed Discovery of Functional Dependencies. In *ICDE*.
[63] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *ACM Trans. Database Syst.* 45, 3 (2020), 1–33.
[64] Philipp Schirmer, Thorsten Papenbrock, Sebastian Kruse, Felix Naumann, Dennis Hempfing, Torben Mayer, and Daniel Neuschäfer-Rube. 2019. DynFD: Functional Dependency Discovery in Dynamic Datasets. In *EDBT*.
[65] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *Proc. VLDB Endow.* 11, 2 (2017), 189–202.
[66] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *WWW*.
[67] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4, 11 (2011), 992–1003.
[68] James J. Treinen and Ramakrishna Thurimella. 2006. A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures.

In *RAID*.
[69] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *ICML*.
[70] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (1990), 103–111.
[71] Vladimir N Vapnik and A Ya Chervonenkis. 2015. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. In *Measures of complexity*.
[72] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.
[73] Ke Wang, Senqiang Zhou, Qiang Yang, and Jack Man Shun Yeung. 2005. Mining Customer Value: From Association Rules to Direct Marketing. *Data Min. Knowl. Discov.* 11, 1 (2005), 57–79.
[74] Xander Wilcke, Maurice de Kleijn, Victor de Boer, Henk J. Scholten, and Frank van Harmelen. 2020. Bottom-up Discovery of Context-aware Quality Constraints for Heterogeneous Knowledge Graphs. In *KDIR*.
[75] Catharine Wyss, Chris Giannella, and Edward Robertson. 2001. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances - Extended Abstract. In *DaWaK*.
[76] Xifeng Yan and Jiawei Han. 2002. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*.
[77] Fan Yang, Zhilin Yang, and William W. Cohen. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *NIPS*.
[78] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph Edge Partitioning via Neighborhood Heuristic. In *SIGKDD*.