Capturing More Associations by Referencing External Graphs

Wenfei Fan
Shenzhen Institute of Computing Sciences
University of Edinburgh
Beihang University
wenfei@inf.ed.ac.uk

Muyang Liu
University of Edinburgh
muyang.liu@ed.ac.uk

Shuhao Liu
Shenzhen Institute of Computing Sciences
shuhao@sics.ac.cn

Chao Tian*
Beihang University
tianchao@buaa.edu.cn

ABSTRACT

This paper studies association rule discovery in a graph G_1 by referencing an external graph G_2 with overlapping information. The objective is to enrich G_1 with relevant properties and links from G_2. As a testbed, we consider Graph Association Rules (GARs). We propose a notion of graph joins to enrich G_1 by aligning entities across G_1 and G_2. We also introduce a graph filtering method to support graph joins, by fetching only the data of G_2 that pertains to the entities of G_1, to reduce noise and the size of the fused data. Based on these we develop a parallel algorithm to discover GARs across G_1 and G_2. Moreover, we provide an incremental GAR discovery algorithm in response to updates to G_1 and G_2. We show that both algorithms guarantee to reduce parallel runtime when given more processors. Better yet, the incremental algorithm is bounded relative to the batch one. Using real-life and synthetic data, we empirically verify that the methods improve the accuracy of association analyses by 30.4% on average, and scale well with large graphs.

PVLDB Reference Format:
Wenfei Fan, Muyang Liu, Shuhao Liu, and Chao Tian. Capturing More Associations by Referencing External Graphs. PVLDB, 17(6): 1173 - 1186, 2024.
doi:10.14778/3648160.3648162

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/MuyangLiu/RefExternalKG.git.

1 INTRODUCTION

Association analyses have been studied for discovering regularities between entities in graphs [40, 47, 48, 80], and have proven effective in online recommendation, link prediction, fraud detection and drug discovery, among other things. Gartner predicts that "graph analytics will double annually" (cf. [26]).

Compared to association analyses in relational data, graph association analyses are more challenging. A real-life graph G_1 often does not come with a schema. It is more common to find the data in G_1 "incomplete", witnessed by missing properties and links.

A natural question arises: Can we fill in the information missing from G_1 with data from external graphs G_2? There exist graphs G_2 that have accumulated the semantics of entities and expert knowledge, notably knowledge graphs, *e.g.,* FreeBase [23], Yago [59], Wikipedia [3] and DBpedia [70]. Why not enrich G_1 and improve the accuracy of its association analyses by referencing such G_2?

Example 1: Consider a real case of money laundering reported by [25]. Four persons make small deposits regularly and the money is then wired to a merchant account for purchasing an asset, *e.g.,* a boat. The transactions are stored in graph G_1 at a bank. With G_1 alone, the usual monitoring process of the bank would not raise an alarm. However, consider a graph G_2, which records the addresses of the users and the ATMs they use. Referencing the external G_2, the bank may suspect that such a group of four users divide transactions into small deposits for money laundering if they use the same ATMs and share addresses. We will see that this is an example of association analysis with a rule, which helps the bank identify potentially illegal activities and label suspicious users as high-risk. The question is how to discover association rules across graphs G_1 and G_2?

Another question concerns incremental rule discovery. In the real world, graphs G_1 and G_2 evolve constantly [68]. The association rules often require dynamic adjustment, *e.g.,* to catch new fraud rings when new transactions and users are added to G_1 and G_2, or to improve the accuracy of fraud detection when high-risk labels are removed from certain users by the bank after investigation. □

The example tells us that by referencing external graph G_2 with overlapping information, we can improve the accuracy of association analyses in graph G_1. The need for referencing external graphs has been recognized in medical knowledge discovery [91], which aims to find semantic patterns and improve the accuracy and efficiency of diagnoses and treatment. As reported in [91, 109], as high as 77% of the discovered patterns are across heterogeneous biological networks. Moreover, e-commerce needs to inspect multiple graphs, since a social media user engages an average of 6.6 different platforms [12], and analyzing a single platform cannot fully understand users' interests. In addition, recommendation [93], information diffusing prediction [115] and network dynamics analysis [114] have been deducing associations across multiple graphs.

To make effective use of external graphs in association analyses, we study discovery of association rules across G_1 and G_2. As a testbed, we consider Graph Association Rules (GARs) [40], which have been used in *e.g.,* recommendation, battery manufacturing and drug discovery [33]. This, however, introduces new challenges.

(1) Rule discovery across graphs. All discovery algorithms for GARs assume a single graph, and no algorithm is yet in place for discovering GARs across separate G_1 and G_2. We want to discover GARs for entities of G_1 enriched with data from G_2. To enrich an entity

*Corresponding author

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648162

(vertex) u in G_1, we have to identify what vertices v in G_2 refer to u, where u and v are often specified with subgraphs of different topological structures. How can we align the entities across G_1 and G_2?

(2) Data filtering. One way to incorporate the external knowledge of G_2 is by merging G_1 and G_2 into one graph. But this introduces noise, *i.e.,* data irrelevant to the entities in G_1. It makes the discovery of useful rules for G_1 harder (due to the noise) and more expensive (due to the large merged graph). As observed in [75], noise in the neighborhood of each entity incurs 8.26% loss of precision in entity alignment. Moreover, it is costly to discover rules from large graphs, *e.g.,* it does not terminate within 15 hours when mining GARs with 5 patterns nodes from a movie graph with 153 million vertices and edges, even when using 8 machines (see Section 6). How can we filter the noise and fetch only the data relevant to G_1?

(3) Incremental rule discovery. Worse still, no algorithm is in place for incrementally discovering GARs in response to updates, from a single graph or across two graphs. Real-life graphs constantly change. For example, 0.5+ million new users are added to Facebook per day [5], and Wikidata [3] publishes hundreds of live updates every minute [2]. As indicated in Example 1, association rules need to be dynamically adjusted in response to the updates.

Contributions & organization. This paper makes the first effort to answer these questions. We review GARs in Section 2.

(1) Graph enrichment (Section 3). We propose a notion of *graph joins* to align entities across G_1 and G_2, such that if a vertex u in G_1 and a vertex v in G_2 denote the same entity, we can fuse u and v and complement u with the relevant properties and links of v. Graph joins enrich graph G_1 with the information from external graph G_2. We also introduce a notion of GARs *pertaining to* G_1, to distinguish rules useful for the analysis of G_1 from the "noisy" ones.

(2) Rule discovery across graphs (Section 4). We introduce a framework to discover GARs pertaining to a graph G_1 by referencing an external graph G_2. We employ heterogeneous entity resolution (HER) to identify vertices in G_1 and G_2 that refer to the same entity [43]. We propose a graph filtering method and train an ML model to locate relevant data in G_2 that pertains to entities in G_1. After these, we adapt existing graph rule mining methods [35, 39] to the new discovery setting, to reduce irrelevant rules and search space.

(3) Incremental rule discovery (Section 5). We develop an incremental discovery algorithm for GARs in response to updates to G_1 and G_2. We incrementalize the batch discovery algorithm across G_1 and G_2 by inspecting only GAR candidates affected by updates. We show that the incrementalized algorithm is bounded relative [44] to the batch one, *i.e.,* it incurs only the necessary cost for incrementalization. Moreover, we show that the discovery algorithms (batch and incremental) can be parallelized with parallel scalability [67], *i.e.,* they guarantee to reduce runtime when given more processors.

(4) Experimental study (Section 6). Using real-life and synthetic data, we empirically find the following. (a) By referencing external graphs G_2, the accuracy of association deduction in graphs G_1 is improved by 30.4% on average with GARs, up to 38.6%. (b) Graph filtering speeds up GAR discovery by 17.4×, while retaining high accuracy in association analyses. (c) The incremental GAR discovery method consistently outperforms the batch algorithm even when the size of updates accounts for 30% of G_1 and G_2. (d) Our batch (resp. incremental) GAR discovery algorithm is parallelly scalable; it is 2.6× (resp. 2.4×) faster when 12 machines are used instead of 4.

We discuss related work in Section 7 and future work in Section 8.

Positioning in the state-of-the-art. (1) While methods have been developed for mining rules from graphs, *e.g.,* [35, 39, 41, 52, 69, 78], to the best of our knowledge, this work makes the first effort to discover rules across two separate graphs, enriching graph G_1 by referencing external G_2. (2) The proposed filtering method, together with graph joins, serve as an effective way to integrate G_1 and G_2 for *mining rules pertaining to* G_1. This allows us to adapt previous mining methods, *e.g.,* [35, 39, 40], to the new discovery setting, to avoid treating G_1 and G_2 indifferently, paying the cost of unnecessary search and returning excessive rules that are useless for G_1. (3) The work also proposes the first incremental rule discovery algorithm with effectiveness guarantees, not limited to incremental mining of frequent patterns [13, 81].

2 PRELIMINARIES

In this section, we first review basic notations of graphs and graph pattern matching. We then review Graph Association Rules (GARs) [40]. Assume three countably infinite sets of symbols, denoted by Γ, Υ and U, for labels, attributes and constants, respectively.

Graphs and patterns. We consider *graphs* $G = (V, E, L, F_A)$, where (a) V is a finite set of vertices, (b) $E \subseteq V \times \Gamma \times V$ is a finite set of edges, where each $e = (v, l, v')$ in E denotes an edge from vertex v to v' that is labeled $l \in \Gamma$, (c) L is a function such that for each vertex $v \in V$, $L(v)$ is a label from Γ, and (d) each vertex $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity, where $A_i \in \Upsilon$ and $a_i \in U$, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$, representing different properties.

Paths. An (undirected) path ρ of length m in graph G is a sequence $\rho = (u_0, l_0, u_1, \ldots, u_{m-1}, l_{m-1}, u_m)$, where (u_i, l_i, u_{i+1}) or (u_{i+1}, l_i, u_i) is an edge in G for each $i \in [0, m-1]$. We refer to u_m as the *terminal* vertex of ρ. We consider paths ρ without cycles.

Patterns. A *graph pattern* is $Q[\bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of *pattern nodes* (resp. *edges*); (2) L_Q assigns a label $L_Q(u) \in \Gamma$ to each pattern node $u \in V_Q$; (3) $\bar{x}$ is a list of distinct variables; and 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. For $x \in \bar{x}$, we use $\mu(x)$ and x interchangeably when it is clear in the context.

Pattern matching. Following [17, 42], a *match* of pattern $Q[\bar{x}]$ in graph G is defined as a homomorphism h from Q to G such that (a) for each pattern node $u \in V_Q$, $L_Q(u) = L(h(u))$; and (b) for each pattern edge $e = (u, l, u')$ in Q, $e' = (h(u), l, h(u'))$ is an edge in G.

We denote the match as a vector $h(\bar{x})$, consisting of $h(x)$ for all $x \in \bar{x}$ in the same order as $\bar{x}$, denoting entities identified by Q.

Graph association rules. We start with predicates of GARs.

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

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

where x, y are variables in $\bar{x}$; $x.A$ is an attribute A of pattern node x; $l(x, y)$ is an edge from x to y labeled l; c is a constant; and $\mathcal{M}$ is an

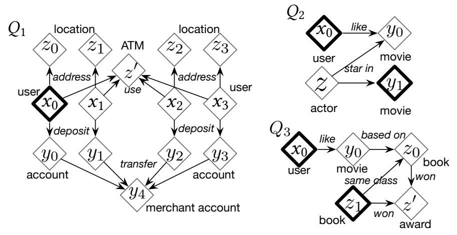


Figure 1: Patterns for graph association rules

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

One can "plug in" an existing ML model $\mathcal{M}$ that returns a Boolean value (*e.g.,* $\mathcal{M} \geq \theta$ for a predefined bound θ). We uniformly express various models we used as link prediction for $l(x, y)$, where the label l can indicate (1) a predicted link, (2) the match of x and y with '=', or (3) semantic similarity between nodes x and y with '$\approx$'.

GARs. A GAR (Graph Association Rule) φ is defined as [40]

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

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

Intuitively, pattern Q in a GAR identifies entities in a graph, and $X \rightarrow p$ is a dependency on the entities. Predicates $x.A = c$ and $x.A = y.B$ can be used to fill in missing properties. Edge predicates enforce the existence of edges, to deduce missing links.

Example 2: The money laundering mentioned in Example 1 can be detected by the following GAR with graph pattern shown in Figure 1, where nodes involved in the consequence are marked bold.

(1) GAR $\varphi_1 = Q_1[\bar{x}](\bigwedge_{i,j \in [0,3]} \mathcal{M}_1(z_i, z_j) \rightarrow p_1)$, where the ML model $\mathcal{M}_1$ checks whether locations z_i and z_j are close, and the consequence p_1 is x_0.risk = high. Intuitively, φ_1 says that if the four persons x_0 to x_3 make deposit into their accounts y_0 to y_3, respectively, which wire money to the same merchant account y_4, and if the four use the same ATM z' (specified in Q_1) and have close addresses (verified by $\mathcal{M}_1$), then x_0 (to x_3) should be labeled as high-risk.

GARs can also be used in knowledge graph-based recommendation [83, 104, 116] across multiple graphs. Below are two rules for a real case [29], whose patterns are also given in Figure 1.

(2) GAR $\varphi_2 = Q_2[\bar{x}](\emptyset \rightarrow p_2)$, where consequence p_2 is edge predicate like(x, y_1). It recommends a movie y_1 to user x_0 if x_0 likes to watch movie y_0 and the same actor z stars in both movies. Here the casts of movies are fetched from external graph G_2 (see Section 3).

(3) GAR $\varphi_3 = Q_3[\bar{x}](\mathcal{M}_3(z_0, z_1) \wedge z_0.\text{country} = z_1.\text{country} \rightarrow p_3)$, where p_3 is like(x_0, z_1). It states that if user x_0 likes a movie y_0 that is based on book z_0, then x_0 may also buy another book z_1 if z_0 and z_1 have the same class, country and similar topics (verified by $\mathcal{M}_3$), and if they won the same award z. It references an external knowledge graph G_2 to get information about movies and books. □

As shown in [40], graph functional dependencies (GFDs) [49], graph entity dependencies (GEDs) [42] and graph pattern association rules (GPARs) [47] are special cases of GARs. GARs extend these dependencies by supporting ML and edge predicates.

Table 1: Notations

Notations	Definitions
G_1, G_2, f	graph, external graph, HER function
$Q[\bar{x}]$, $h(\bar{x})$	graph pattern, pattern match
$h(\bar{x}) \models p_0$ (or X)	match $h(\bar{x})$ satisfies a predicate p_0 (or a conjunction X)
$G_\oplus(G_1, G_2, f)$ or $G_\oplus$	the join of G_1 and G_2 with HER function f
$\varphi \multimap G_\oplus$	GAR φ pertains to G_1 *w.r.t.* G_2 and f
sup$(\varphi, G_\oplus)$	support of GAR $\varphi \multimap G_\oplus(f)$

Semantics. To interpret $\varphi = Q[\bar{x}](X \rightarrow p)$, denote by $h(\bar{x})$ a match of Q in a graph G, and by p_0 a predicate of $Q[\bar{x}]$.

We say that $h(\bar{x})$ *satisfies* predicate p_0, denoted by $h(\bar{x}) \models p_0$, if the following condition is satisfied: (a) when p_0 is $l(x, y)$, there exists an edge with label l from $h(x)$ to $h(y)$; (b) when p_0 is $x.A = y.B$, attributes A and B exist at $h(x)$ and $h(y)$, respectively, and $h(x).A = h(y).B$; similarly for $x.A = c$; and (c) when p_0 is $\mathcal{M}(x, y)$, the ML model predicts true about a semantic relationship between x and y.

For $\varphi = Q[\bar{x}](X \rightarrow p)$, we write $h(\bar{x}) \models X$ if $h(\bar{x}) \models p_0$ for all p_0 in X. We write $h(\bar{x}) \models \varphi$ such that if $h(\bar{x}) \models X$, then $h(\bar{x}) \models p$. We write $G \models \varphi$ if for all matches $h(\bar{x})$ of Q in graph G, $h(\bar{x}) \models X \rightarrow p$. We write $G \models \Sigma$ for a set Σ of GARs if $G \models \varphi$ for all $\varphi \in \Sigma$.

The notations of the paper are summarized in Table 1.

3 SEMANTIC ENRICHMENT

In this section, we first present a notion of graph joins. We then propose a method to improve association analyses in a graph G_1 by incorporating relevant data from an external graph G_2.

Graph joins. We define graph joins with heterogeneous entity resolution (HER) across separate graphs. Consider a graph $G_1 = (V_1, E_1, L_1, F_1)$ and an external graph $G_2 = (V_2, E_2, L_2, F_2)$.

HER. We assume the availability of an HER *function* f that, given a graph G_1 and an external graph G_2, computes a set of pairs:

$$f(G_1, G_2) = \{(u, v) \mid u \in G_1, v \in G_2, u \Rightarrow v\}.$$

Here $u \Rightarrow v$ denotes that vertices u and v *match*, *i.e.,* they refer to the same real-world entity. We refer to $f(G_1, G_2)$ as *the set of matches across* G_1 *and* G_2 by f. To simplify the discussion, we assume *w.l.o.g.* that if $(u, v) \in f(G_1, G_2)$, then $L_1(u) = L_2(v)$; and $f(G_1, G_2)$ is "bijective", *i.e.,* each u has a unique match v and vice versa.

Several HER functions are already in place, *e.g.,* JedAI [84], parametric simulation [43], and ML models Silk [62], MAGNN [51] and EMBLOOKUP [15]. In this paper, we take parametric simulation [43] as HER; our method works with the other HER functions.

Graph joins. The *join of* G_1 *and* G_2 *with* HER f is a graph $G_\oplus(G_1, G_2, f) = (V_\oplus, E_\oplus, L_\oplus, F_\oplus)$, where $V_\oplus$ (resp. $E_\oplus$) is a revision of the union $V_1 \cup V_2$ (resp. $E \cup E_2$) such that u and v are represented as the same (merged) vertex in $V_\oplus$ if (u, v) is a match in $f(G_1, G_2)$; and $L_\oplus$ and $F_\oplus$ inherit the label and attribute assignments from G_1 and G_2. When u and v both carry attribute A, the merged vertex takes the value $v.A$ from G_2, assuming that the data in G_2 is more reliable.

We write $G_\oplus(G_1, G_2, f)$ as $G_\oplus$ when it is clear in the context.

Intuitively, if u and v are determined by HER to denote the same entity, the two are merged into the same vertex in $G_\oplus$, which inherits all the adjacent edges and attributes of u and v. This is analogous to a "semantic" extension of the natural join in SQL, where tuples t_1 and t_2 join if the two have the same common attribute values, and then t_1 is "enriched" with additional attributes of t_2.

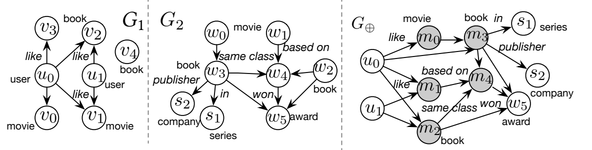


Figure 2: Graph join $G_\oplus$ of G_1 and G_2

Example 3: Figure 2 depicts the graph join $G_\oplus$ of two graphs G_1 and G_2, in which the merged vertices in $G_\oplus$ are colored in gray. That is, the HER function links each v_i (movie or book) in G_1 to w_i in G_2 for $i \in [0, 4]$, yielding the merged vertices m_i. Their adjacent edges labeled by *e.g.,* won, based on and like are inherited accordingly. □

GARs *across* G_1 *and* G_2. A GAR $\varphi = Q[\bar{x}](X \to p)$ across separate G_1 and G_2 is simply φ defined on $G_\oplus$; and it is interpreted in $G_\oplus$, *i.e.,* we consider matches $h(\bar{x})$ of Q in the join $G_\oplus$ of G_1 and G_2. This is how we interpret the GARs φ_1-φ_3 of Example 2.

Remark. One can adopt alternative definitions for graph joins $G_\oplus$, as long as there exist (a) a function f that returns pairs of vertices across G_1 and G_2 satisfying certain conditions θ (not limited to entity matching), just like θ-joins in relational queries; and (b) a scheme that determines which graph data related to the matched vertex pairs will be included in $G_\oplus$.

Association analyses with external graphs. On the one hand, the join $G_\oplus$ enriches graph G_1 with information from G_2. On the other hand, it is often not very practical to compute $G_\oplus$ since it incurs extra computational and maintenance costs, and moreover, the join $G_\oplus$ contains "noise" irrelevant to the entities in G_1.

To reduce noise and cost, we propose to make use of $G_\oplus$ for association analyses without physically computing the graph join.

GARs *pertaining to* G_1. We define the scope of GARs that can facilitate the association analyses in graph G_1. For a GAR $\varphi = Q[\bar{x}](X \to p)$, we assume *w.l.o.g.* that its consequence p involves two variables $x_p, x'_p \in \bar{x}$ ($x_p = x'_p$ if one is involved), which are referred to as the *pivots* of φ. We say that φ *pertains to* G_1 *w.r.t.* G_2 and f, denoted by $\varphi \multimap G_\oplus$, if there exists at least one match h of Q in $G_\oplus$ such that at least one of $h(x_p)$ and $h(x'_p)$ is in G_1, as mapping of pivots in φ.

Intuitively, such φ pertains to the entities of G_1 rather than to irrelevant entities in G_2. The reason for specifying this scope is twofold. (a) We want to reduce the search space and irrelevant rules. (b) We want to enrich G_1 for association analyses but not G_2, by taking the action specified in the consequence predicate p.

For instance, when p is $x_p.A = x'_p.B$ such that x_p and x'_p are pivoted at vertices $u \in G_1$ and $v \in G_2$, respectively, we enrich $u.A$ in G_1 with $v.B$. When p is $l(x_p, x'_p)$ and x_p and x'_p are pivoted at vertices $u, u' \in G_1$, we deduce a (missing) link from u to u' in G_1.

Association analyses pertaining to G_1. Based on this notion, we

- discover a set Σ of GARs pertaining to G_1 from $G_\oplus(G_1, G_2, f)$,
- incrementally mine GARs in response to updates to G_1 and G_2.

In the process, we inspect graph G_1 and only the data of G_2 that pertains to entities in G_1, not the entire G_2, to avoid noise and redundant computation. Along the same lines, we can deduce associations in G_1 by referencing G_2, with the discovered GARs.

We will develop discovery algorithms in Section 4 and incremental algorithms in Section 5, without physically computing the join.

4 RULE DISCOVERY ACROSS GRAPHS

This section formulates the problem of mining GARs across graphs G_1 and G_2, and proposes a discovery framework (Section 4.1). We also develop a strategy to identify relevant data of G_2 (Section 4.2).

4.1 The Discovery Problem

Following [35, 39], we measure the quality of GARs with *support.*

Support. We extend the support of GFDs [39] to GARs pertaining to G_1 *w.r.t.* G_2 and f. Given such a GAR $\varphi = Q[\bar{x}](X \to p)$ with pivots x_p and x'_p of p, we denote by $Q(G_\oplus, G_1)$ the set of matches h of Q in the join $G_\oplus$, such that at least one of $h(x_p)$ and $h(x'_p)$ is in G_1. Let $Q(G_\oplus, G_1, X, p)$ be the set of all such matches h in $Q(G_\oplus, G_1)$ with $h \models X$ and $h \models p$. The support of GAR $\varphi \multimap G_\oplus$ is

$$\text{sup}(\varphi, G_\oplus) = \|\{\langle h(x_p), h(x'_p)\rangle \mid h \in Q(G_\oplus, G_1, X, p)\}\|,$$

i.e., the number of valid matches with *distinct pivot mappings.* Intuitively, GARs with higher support can be applied more frequently.

Anti-monotonicity. One can verify that $\text{sup}(\varphi, G_\oplus)$ retains the anti-monotonicity *w.r.t.* the partial order $\preceq$ on GARs defined in [35]. For GARs $\varphi_1 = Q_1[\bar{x}_1](X_1 \to p)$ and $\varphi_2 = Q_2[\bar{x}_2](X_2 \to p)$, $\varphi_1 \preceq \varphi_2$ if the pattern Q_1 is a "subgraph" of Q_2 and X_1 is a subset of X_2 [35].

Lemma 1: *Given a graph join* $G_\oplus$ *and* GARs $\varphi_1 \preceq \varphi_2$, *if* $\varphi_1 \multimap G_\oplus$ *and* $\varphi_2 \multimap G_\oplus$, *then* $\text{sup}(\varphi_1, G_\oplus) \geq \text{sup}(\varphi_2, G_\oplus)$. □

Discovery problem. The new discovery problem is as follows.

- *Input*: A graph G_1, an external graph G_2, an HER function f, a positive integer k and a support threshold σ.
- *Output*: The set Σ of GARs that pertain to G_1 *w.r.t.* G_2 and f such that for each GAR $\varphi = Q[\bar{x}](X \to p) \in \Sigma$, $G_\oplus \models \varphi$ *w.r.t.* the matches $Q(G_\oplus, G_1)$, $\text{sup}(\varphi, G_\oplus) \geq \sigma$, and φ has at most k pattern nodes.

Parameter k is to control the cost of discovery following [35, 41].

A framework. A strawman approach is to treat $G_\oplus$ as the input graph and directly apply an existing mining algorithm [34, 35, 39, 41]. As remarked earlier, this may be prohibitively expensive. It may also end up with many GARs that reflect regularities in G_2 only.

Three-step discovery. In light of these, we propose a new 3-step framework to (1) access only the data in G_2 relevant to GARs pertaining to G_1; and (2) discover GARs that pertain to G_1 only.

- *Tentative join.* Given an HER function f, we compute the set $f(G_1, G_2)$ of matches across graph G_1 and external graph G_2.
- *Graph filtering* (Section 4.2). Applying a novel ML-based filtering strategy, we identify data in G_2 that is relevant to GARs pertaining to G_1, filtering out the rest to reduce noise and cost.
- *Mining.* We discover GARs over G_1 and the relevant data selected from G_2 in the second step, instead of the join $G_\oplus$.

The mining step follows [35, 39], except that it computes the support of a rule such that at least one pivot is mapped to a vertex in G_1. It generates candidate GARs in a levelwise manner with vertical spawning to construct patterns, and horizontal spawning to expand dependencies for GARs. All candidates *w.r.t.* each pattern Q are organized in a generation tree $T(Q)$. Every node w in $T(Q)$ encodes a candidate GAR $Q[\bar{x}](X \to p)$, and each of w's children in $T(Q)$ represents $Q[\bar{x}](X' \to p)$, where X' extends X with one more predicate. It iteratively conducts *grouped candidate validation,* which

Algorithm 1: Filter

```
Input: A graph $G_1$, external graph $G_2$, the set $f(G_1, G_2)$ of matches
       across $G_1$ and $G_2$ by HER function $f$, and threshold $\delta$.
Output: Filtered external graph $r_1(G_2)$.
1 initialize $r_1(G_2)$ as an empty graph;
2 foreach pair of matches $(u_0, v_0) \in f(G_1, G_2)$ do
3     generate paths $P_1(u_0)$:=$\mathcal{M}_\rho(G_1, u_0)$; $P_2(v_0)$:=$\mathcal{M}_\rho(G_2, v_0)$;
4 foreach path set $P_2(v_0, v_m)$ generated from $G_2$ do
5     calculate the ranking score $R(P_2(v_0, v_m))$ by using DPRA;
6     if $R(P_2(v_0, v_m))$ satisfies the threshold $\delta$ then
7         add all vertices and edges in $P_2(v_0, v_m)$ to $r_1(G_2)$;
8 return $r_1(G_2)$;
```

verifies "similar" candidates together with support and satisfaction checking, and prunes candidates by anti-monotonicity of support.

Putting the three steps together, we denote the discovery algorithm as JDisR. It returns those candidate GARs that are satisfied by the filtered graph join and have support values above the threshold.

Below we focus on graph filtering. As will be seen in Section 5, we also incrementally discover GARs in response to updates, and parallelize the discovery processes to scale with large graphs.

4.2 Graph Filtering

We introduce an ML-based method to select a subgraph $r_1(G_2)$ from G_2, consisting of data that is relevant to GARs pertaining to G_1.

Rationale. A naive approach to identifying $r_1(G_2)$ is to include all the data within $k-1$ hops of the matched vertices v by HER function f. Since we aim to find those GARs having at most k nodes in their patterns, all required GARs pertaining to G_1 can be found from G_1 and such $r_1(G_2)$. However, when real-life graphs follow, *e.g.,* the power-law node degree distribution [28], the size of $r_1(G_2)$ may grow exponentially with k, making GAR discovery inefficient.

To get a smaller $r_1(G_2)$, we propose to filter G_2 based on its *semantic relevance to* G_1. In practice, external (knowledge) graph G_2 often includes data that is extracted from multiple data sources, covering general and diverse use cases [60]. The graph G_1, on the contrary, targets limited, user-specific applications only.

Path-driven filtering. Algorithm 1 outlines Filter, our strategy to identify $r_1(G_2)$. As observed in [43], relevant entities are often linked via paths in a graph. Hence Filter consists of three major steps: (1) *path generation* (lines 2-3), which constructs a set of semantically meaningful paths in both G_1 and G_2 based on a language model $\mathcal{M}_\rho$, starting from the matches; (2) *path ranking* (lines 4-5) that quantitatively measures the semantic relevance between the generated paths; and (3) *data selection* (lines 6-7), which filters out the data in G_2 that is not endorsed by any high-ranking paths, *i.e.,* their semantic relationship to G_1 is weak; it is controlled by a score threshold δ, to strike a balance between effectiveness and efficiency.

This strategy is echoed by previous work on heterogeneous networks [61, 95], which finds that paths, as special structural features, can capture relevant semantics needed for recommendation and similarity search. Below we present the three steps in Filter.

(1) Path generation. Starting from the matches in $f(G_1, G_2)$ across G_1 and G_2 , Filter generates a collection of paths whose terminal vertices represent "important" properties, by using a language model $\mathcal{M}_\rho$. That is, for each pair $(u, v) \in f(G_1, G_2)$, it computes a set $P_2(v)$ of paths in G_2 via pre-trained $\mathcal{M}_\rho$ such that each path ρ in $P_2(v)$ originates from v and terminates at a semantically relevant property of v. Similarly the paths in $P_1(u)$ are selected from G_1 to get u's properties. Indeed, language models have been shown effective in "parsing" information within (knowledge) graphs [30, 72, 74]; they can explore the embedding sequence for a path, and generate a single representation to encode its holistic semantics [107].

More specifically, Filter initializes a path $\rho = (v, l, v_1)$ *w.r.t.* each edge (v, l, v_1) in G_2, where v is in $f(G_1, G_2)$ and matches a vertex of G_1. Then $\mathcal{M}_\rho$ is applied to vertex label $L_2(v_1)$ to get a set of edge labels with their possibility of following the "word" $L_2(v_1)$. The edge (v_1, l', v_2) (or (v_2, l', v_1)) is chosen from the set of edges incident to v_1, whose label l' has the highest possibility in this set. Filter appends l' and v_2 to ρ, and repeats the above steps, *i.e.,* feeding $L_2(v_2)$ to $\mathcal{M}_\rho$ to have another set of edge labels for subsequent edge selection. Such iteration for constructing ρ proceeds until (a) $\mathcal{M}_\rho$ returns the end of sentence tag, (b) there is no edge to choose, (c) ρ already has k vertices, or (d) ρ forms a cycle (abandoned). All the resulting paths ρ are included in $P_2(v)$. Along the same lines, it also builds the path set $P_1(u)$ for graph G_1 based on $\mathcal{M}_\rho$.

We implement $\mathcal{M}_\rho$ as a long short-term memory (LSTM) network, and collect sequences of vertex and edge labels on the random walk paths in G_1 and G_2 to build a training corpus. Taking the labels as sentences of words, we train $\mathcal{M}_\rho$ on the corpus driven by the "perplexity" loss [79]; the training is unsupervised.

We employ LSTM as an exemplary implementation, replaceable by *e.g.,* large language models. LSTM is adopted since it can effectively capture the semantic meaning of labeled paths in graphs [72–74], by "memoizing" long-term dependencies in a sequence and reasoning about paths. In particular, compared to large language models, LSTM is simple to train and is efficient in inference.

(2) Path ranking. Filter computes a ranking score $R(P_2(v_0, v_m))$ for each path set $P_2(v_0, v_m)$ generated from G_2, where all paths in $P_2(v_0, v_m)$ have the same starting vertex v_0 and terminal vertex v_m. It measures the "importance" of these paths in GARs pertaining to G_1. To achieve this, Filter applies a Dual-Path Ranking Algorithm (DPRA). In a nutshell, DPRA ranks $P_2(v_0, v_m)$ by considering both (a) $R_2(P_2(v_0, v_m))$, its semantic significance in G_2, and (b) $R_1(P_2(v_0, v_m))$, its semantic relevance to entities in G_1. Hence DPRA outputs $R(P_2(v_0, v_m)) = R_2(P_2(v_0, v_m)) + R_1(P_2(v_0, v_m))$.

Consider a generated path ρ=$(v_0, l_0, v_1, ..., v_m)$ in $P_2(v_0, v_m)$ from G_2, where v_0 has a match u_0 in G_1 by HER function.

(a) DPRA first computes the ranking score

$$R_2(\rho) = \prod_{i=0}^{m-1} \frac{1}{D(v_i)},$$

where $D(v_i)$ denotes the degree of v_i in G_2.

Intuitively, it simulates the process that a resource unit flows from the starting vertex v_0 of path ρ, and equally divides to each branch in the middle, representing the evolution of significance along ρ. The score estimates the reliability of ρ as a semantically meaningful connection between v_0 and property v_m [74].

(b) It then computes the other score $R_1(\rho)$ based on the semantic similarity between ρ and selected paths originated from u_0 in G_1.

(i) If ρ has no matched vertex by HER except v_0, DPRA sets $R_1(\rho) = 0$, as ρ is a "dangling" path in G_2 and has little relevance to G_1.

(ii) If there exists a vertex v_i ($i \in [1, m]$) on path ρ that has a match u in G_1 by HER, it finds the subset $P_1(u_0, u) = \{\rho' \mid \rho' \in P_1(u_0), u \text{ is on } \rho'\}$ from $P_1(u_0)$. That is, each path ρ' in $P_1(u_0, u)$ and path ρ originate from the same entity $u_0 \Rightarrow v_0$, and *intersect at another vertex* $u \Rightarrow v_i$. This indicates that ρ of G_2 is semantically related to ρ' of G_1, and the "importance" of ρ' should contribute to that of ρ for GARs pertaining to G_1.

In light of this, DPRA computes the score

$$R_1(\rho, v_i) = \max_{\rho' \in P_1(u_0, u)} \prod_{i=1}^{\mathsf{len}(\rho')} \frac{1}{D(u_i)},$$

where $\mathsf{len}(\rho')$ denotes the total number of vertices in path ρ' and u_i is a vertex on ρ'; and $R_1(\rho)$ is obtained by summing up the values of $R_1(\rho, v_i)$ for all matched vertices v_i ($i \in [1, m]$) by HER on ρ.

(c) Having computed $R_2(\rho)$ and $R_1(\rho)$ for all individual paths ρ, DPRA finally aggregates the scores by letting $R_2(P_2(v_0, v_m)) = \sum_{\rho \in P_2(v_0, v_m)} R_2(\rho)$ and $R_1(P_2(v_0, v_m)) = \sum_{\rho \in P_2(v_0, v_m)} R_1(\rho)$.

Theoretically the ranking scores for individual paths depress the role of high-degree nodes, which may underestimate important vertices in graphs following the power law. For instance, a number of people are born in the same populous country; dropping high-degree country vertices makes it difficult to find a GAR that people with the same birthplace are also likely to have the same citizenship. The final aggregation phase in DPRA is to avoid such punishment.

(3) Data selection. Filter derives the subgraph $r_1(G_2)$ by including only the vertices and edges that appear in those path sets $P_2(v_0, v_m)$ having scores above threshold δ. Obviously, smaller δ leads to more discovered GARs but lower efficiency. As will be seen in Section 6, medium δ values, *e.g.,* 0.05, suffice to help us find a large percentage of required GARs while it notably reduces the size of $r_1(G_2)$.

Example 4: Recall graphs G_1, G_2 and the HER matches from Example 3. Algorithm Filter generates paths from each vertex in the HER matches to filter G_2. For instance, starting from vertex w_3, the language model $\mathcal{M}_\rho$ creates fours paths that link to the publisher, series, award and same class book of w_3. Filter drops the paths *w.r.t.* publisher and series for their low scores. Similarly it processes the paths from w_2 and w_4. Finally Filter returns the filtered $r_1(G_2)$ from G_2 by removing the paths to s_1 and s_2. Joining it with G_1, the filtered graph join is smaller than the $G_\oplus$ depicted in Figure 2. □

Analysis. Once the matches across G_1 and G_2 are determined by HER function, it takes $O(|G_1|^2 + |G_2|^2)$ time to generate all the meaningful paths and construct $r_1(G_2)$ with paths of high scores. As shown in [43], HER takes at most quadratic ($O(|G_1||G_2|)$) time.

5 INCREMENTAL RULE DISCOVERY

In this section, we first develop an incremental GAR discovery algorithm that is relatively bounded (Section 5.1). We then parallelize the discovery algorithms with parallel scalability (Section 5.2).

5.1 Incremental Discovery Algorithm

We consider *batch updates* ΔG_1 (resp. ΔG_2) to graph G_1 (resp. G_2). To simplify the discussion, ΔG_1 and ΔG_2 are assumed to be sequences of edge insertions and deletions. Vertex updates are a dual of edge updates [66], which can be handled similarly [44, 46].

Incremental discovery. Suppose that given pattern size bound k and support threshold σ, batch GAR discovery across G_1 and G_2 returns a set $\Sigma(G_1, G_2, k, \sigma)$ of GARs pertaining to G_1. Let $\Delta\Sigma^+ = \Sigma(G_1 \otimes \Delta G_1, G_2 \otimes \Delta G_2, k, \sigma) \setminus \Sigma(G_1, G_2, k, \sigma)$, $\Delta\Sigma^- = \Sigma(G_1, G_2, k, \sigma) \setminus \Sigma(G_1 \otimes \Delta G_1, G_2 \otimes \Delta G_2, k, \sigma)$, and $\Delta\Sigma(G_1, G_2, \Delta G_1, \Delta G_2, k, \sigma) = (\Delta\Sigma^+, \Delta\Sigma^-)$, denoting the new GARs introduced by $(\Delta G_1, \Delta G_2)$, removed by $(\Delta G_1, \Delta G_2)$, and their combination, respectively. Here $G_i \otimes \Delta G_i$ refers to the updated G_i by applying ΔG_i to G_i ($i \in [1, 2]$).

The *incremental* GAR *discovery problem across* G_1 *and* G_2 *is*:

- *Input*: G_1, G_2, k, σ as in the discovery problem, updates (ΔG_1, ΔG_2), and a set $\Sigma(G_1, G_2, k, \sigma)$ of GARs mined across G_1 and G_2.
- *Output*: Changes $\Delta\Sigma(G_1, G_2, \Delta G_1, \Delta G_2, k, \sigma)$ to $\Sigma(G_1, G_2, k, \sigma)$.

We simply write $\Sigma(G_1, G_2, k, \sigma)$ as $\Sigma(G_1, G_2)$ when k and σ are clear in the context; similarly for $\Delta\Sigma(G_1, G_2, \Delta G_1, \Delta G_2)$.

When $(\Delta G_1, \Delta G_2)$ is small as commonly found in practice, it is often more efficient to compute the above changes than re-mining GARs across $G_1 \otimes \Delta G_1$ and $G_2 \otimes \Delta G_2$ starting from scratch.

Relative boundedness. Denote by $(G_1, G_2)_\mathcal{A}$ the data inspected by a batch GAR discovery algorithm $\mathcal{A}$ for computing $\Sigma(G_1, G_2)$, including the auxiliary structures used by $\mathcal{A}$. Given updates $(\Delta G_1, \Delta G_2)$, denote by AFF the difference between $(G_1 \otimes \Delta G_1, G_2 \otimes \Delta G_2)_\mathcal{A}$ and $(G_1, G_2)_\mathcal{A}$. An incremental GAR discovery algorithm $\mathcal{A}_\Delta$ is *bounded relative to* $\mathcal{A}$ [44, 46] if the size of data checked by $\mathcal{A}_\Delta$ can be expressed as a function of $|\Delta G_1|$, $|\Delta G_2|$ and $|\mathsf{AFF}|$.

Intuitively, $|\mathsf{AFF}|$ is the size of the affected area by ΔG_1 and ΔG_2 relative to $\mathcal{A}$. It is the minimum cost for any possible incrementalization of $\mathcal{A}$. A relatively bounded incremental $\mathcal{A}_\Delta$ is measured by this necessary cost without inspecting irrelevant parts.

Theorem 2: *There exist incremental graph filtering and* GAR *mining algorithms that are relatively bounded.* □

We next provide a constructive proof of Theorem 2 by incrementalizing Filter first, and then the mining step. A relatively bounded incremental algorithm for graph joins with HER (parametric simulation) has been developed in [43] with the time complexity $O(|\mathsf{AFF}|)$.

Incremental graph filtering. To incrementalize algorithm Filter (Section 4.2), we maintain all paths generated by the language model $\mathcal{M}_\rho$ as *auxiliary structures* along with their scores. Then it suffices to identify paths that interact with updates, revise the affected paths, and update the ranking scores accordingly.

The incremental filtering algorithm, referred to as IncFilter, works as follows. (1) For each updated edge (v, l, v') and the generated path ρ, it first checks whether v (resp. v') is on ρ; it marks ρ as *stale* if so, and v (resp. v') as an *interaction vertex.* (2) For each stale path ρ, it resumes the generation for ρ at interaction vertices in the updated graphs, starting from the first such vertex on ρ. Note that the re-generation between two consecutive interaction vertices is not induced if the edge selected with the former is unchanged. (3) Using the updated paths, IncFilter adjusts the ranking scores and selects paths with scores above threshold σ from each matched vertex in $G_2 \otimes \Delta G_2$, along the same lines as the last two steps in Filter.

IncFilter is bounded relative to Filter since Filter creates paths iteratively by checking the adjacent edges of their current terminal

Algorithm 2: IncJDisR

```
Input: Updated graphs $G_1 \otimes \Delta G_1$ and $r_1(G_2 \otimes \Delta G_2)$, GARs
       $\Sigma(G_1, G_2)$ mined w.r.t. $k$ and $\sigma$, and auxiliary structures $\mathcal{T}$.
Output: Changes $\Delta\Sigma(G_1, G_2, \Delta G_1, \Delta G_2)$ to $\Sigma(G_1, G_2)$.
1 $\Delta\Sigma^+ := \emptyset$; $\Delta\Sigma^- := \emptyset$; fuse $G_1 \otimes \Delta G_1$ and $r_1(G_2 \otimes \Delta G_2)$ as $G''_\oplus$;
2 retrieve valid (resp. invalid) boundary GARs $\Sigma_T$ (resp. $\Sigma_F$) from $\mathcal{T}$;
3 foreach GAR $\varphi \in \Sigma_T$ do
4     if ReCheck($\varphi$, $\sigma$, $G''_\oplus$, $\mathcal{T}$) = false or $G''_\oplus \not\models \varphi$ then
          $\Delta\Sigma^- := \Delta\Sigma^- \cup \{\varphi\}$;
5 foreach GAR $\varphi \in \Sigma_F$ s.t. $G''_\oplus \models \varphi$ do
6     if ReCheck($\varphi$, $\sigma$, $G''_\oplus$, $\mathcal{T}$) = true then $\Delta\Sigma^+ := \Delta\Sigma^+ \cup \{\varphi\}$;
7 update $\Delta\Sigma^+$ with BacktrackGT($\Sigma_T \cup \Sigma_F$, $G''_\oplus$, $\sigma$, $\mathcal{T}$);
8 update $\Delta\Sigma^+$ with ExpandGT($\Sigma_T \cup \Sigma_F$, $G''_\oplus$, $\sigma$, $\mathcal{T}$);
9 return ($\Delta\Sigma^+$, $\Delta\Sigma^-$);
```

vertices; hence all interaction vertices and the data inspected by the re-generation in IncFilter are covered by AFF. Similarly, score adjustment only inspects paths having the same source vertices as the re-generated ones, which are also included in AFF.

IncFilter takes $O(|\mathsf{AFF}|(|\Delta G_1| + |\Delta G_2|))$ time in the worst case since every unit update can trigger path re-generation. The space cost is $O((m_t + m'_t)(|G_1| + |G_2| + |\Delta G_1| + |\Delta G_2|))$, where m_t (resp. m'_t) denotes the number of HER matches (resp. newly added HER matches) across G_1 and G_2 (resp. $G_1 \otimes \Delta G_1$ and $G_2 \otimes \Delta G_2$); this includes the cost for maintaining intermediate results, *i.e.,* $O(m_t)$ old paths and another $O(m'_t)$ paths that start with new matches.

Incremental mining. We now deduce IncJDisR, an incremental GAR discovery algorithm. It takes as input the updated graph $G_1 \otimes \Delta G_1$, the new subgraph $r_1(G_2 \otimes \Delta G_2)$ returned by IncFilter, the set $\Sigma(G_1, G_2)$ of GARs mined by JDisR, and some auxiliary structures $\mathcal{T}$. It treats $G_1 \otimes \Delta G_1$ and $r_1(G_2 \otimes \Delta G_2)$ as vertex-cut partitions, "fuses" them into $G''_\oplus$ and incrementally computes $\Delta\Sigma(G_1, G_2, \Delta G_1, \Delta G_2)$.

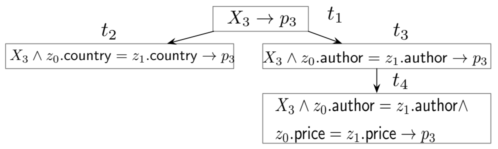
A brute-force incremental version of the mining step of JDisR would re-examine the support and satisfaction of candidate GARs φ. To simplify the discussion, below we focus on support checking; similarly for checking whether the updated graphs satisfy φ.

Support re-examination incurs redundant validation since many GARs in $G''_\oplus$ retain support above threshold. Instead, based on the anti-monotonicity of support and the generation trees $T(Q)$ in JDisR (Section 4.1), IncJDisR identifies a set of *boundary* GARs from $T(Q)$ as the "triggers" of the changes to $\Sigma(G_1, G_2)$, which may get in or out of $\Sigma(G_1, G_2)$ because of support changes. It re-checks these rules, and propagates possible updates over $T(Q)$.

Boundary GARs. A GAR is called *qualified* if it meets the support threshold σ. There are two types of *boundary* GARs*:* (1) a *valid boundary* GAR is a qualified GAR in $T(Q)$ without any children; (2) an *invalid boundary* GAR is an unqualified GAR in $T(Q)$.

Auxiliary Structures. To speed up re-checking of the support for previous GARs, we maintain the following auxiliary structures. (1) The generation tree $T(Q)$ *w.r.t.* each pattern Q; (2) previous support value of each candidate GAR; and (3) all pivot mappings for every candidate GAR. These are obtained during the execution of batch JDisR.

Algorithm. We present IncJDisR in Algorithm 2. After identifying the valid and invalid boundary GARs from the generation trees,



$X_3 \rightarrow p_3$
t_1
t_2
t_3
$X_3 \wedge z_0$.country $= z_1$.country $\rightarrow p_3$
$X_3 \wedge z_0$.author $= z_1$.author $\rightarrow p_3$
t_4
$X_3 \wedge z_0$.author $= z_1$.author$\wedge$
z_0.price $= z_1$.price $\rightarrow p_3$

Figure 3: Fraction of generation tree $T(Q_3)$

IncJDisR checks whether each valid (resp. invalid) one has become an unqualified (resp. qualified) GAR via procedure ReCheck; if so, such rules are included in the sets $\Delta\Sigma^-$ and $\Delta\Sigma^+$ regarding their satisfaction with the updated $G''_\oplus$, respectively (lines 3–6). Then it invokes BacktrackGT to start the backtracking process over the generation trees from every boundary GAR (line 7). In each step, procedure ReCheck is called to check the support of the GAR currently under inspection in $G''_\oplus$. It proceeds until reaching a qualified GAR that is satisfied by $G''_\oplus$ with minimum predicates, which will be included in $\Delta\Sigma^+$. Analogously, another procedure ExpandGT expands the generation trees from boundary GARs whose ancestors have not been added to $\Delta\Sigma^+$ in a levelwise manner, conducts verification and updates the set $\Delta\Sigma^+$ accordingly with qualified GARs that are satisfied by $G''_\oplus$ (line 8). IncJDisR finally returns $(\Delta\Sigma^+, \Delta\Sigma^-)$ (line 9).

ReCheck ensures that when validating an existing GAR φ, only new (resp. old) matches involving inserted (resp. deleted) edges are computed. This is achieved by firstly mapping edges to the graph updates in pattern matching. It then compares the pivot mappings embedded in these matches against the maintained ones, and increases (resp. decreases) the support value of φ if new pivot mappings appear (resp. all matches that cover certain old pivot mappings also involve deleted edges). For new candidate GARs, ReCheck applies the same validation method as that in JDisR.

Example 5: Recall G_1 and G_2 from Example 3 and $r_1(G_2)$ from Example 4; let $k = 5$ and $\sigma = 2$. Assume that the ML model $\mathcal{M}_3$ returns true for similarity checking among topics of all books in G_2, which are written by the same author; each book has a unique price (attribute, not shown); w_2 and w_4 have the same country (attribute), but different from that at w_3. Given these, batch JDisR spawns vertically to generate pattern Q_3 of Figure 1; a fraction of the generation tree $T(Q_3)$ is shown in Figure 3, where $X_3 = \mathcal{M}_3(z_0, z_1)$. The dependency encoded by node $t_i (i \in [1, 4])$ in $T(Q_3)$ is coupled with Q_3 to form a candidate GAR φ'_i, *e.g.,* φ'_2 corresponds to φ_3 of Example 2. Based on the support and satisfaction, JDisR only returns φ'_2 with support 2; note that φ'_4 has support 0 below threshold.

Suppose that updates $(\Delta G_1, \Delta G_2)$ insert $(u_1, \mathsf{like}, v_0)$ into G_1 and delete $(w_2, \mathsf{same\_class}, w_4)$ from G_2. For threshold σ=2, φ'_2 is a valid boundary GARs, while φ'_4 is invalid. IncJDisR checks their support and satisfaction changes. Since the support of φ'_2 decreases to 0, IncJDisR adds it to $\Delta\Sigma^-$ and backtracks over $T(Q_3)$. It then finds that the root φ'_1 also becomes unqualified with a new support 1 and stops the backtracking. Note that no expansion is initiated because now both φ'_2 and φ'_4 have support below σ. Hence IncJDisR just returns the GAR in $\Delta\Sigma^-$, which is to be removed. □

Analysis. IncJDisR is bounded relative to the mining step of JDisR, since (a) procedures BacktrackGT and ExpandGT just inspect those candidate GARs whose support and satisfaction changes may affect their memberships in $\Sigma(G_1, G_2)$. All such GARs in the generation trees are covered by affected area AFF. (b) The re-checking of previous GARs only finds matches involving updated edges, which are contained in AFF. (c) Each newly generated candidate GAR in expansion is also checked by JDisR when given the updated graph.

IncJDisR is in $O(|\mathsf{AFF}|^2)$ time because it generates and checks at most $O(|\mathsf{AFF}|)$ many candidate GARs, and every validation step needs $O(|\mathsf{AFF}|)$ time. Note that here AFF takes the difference of traces in data access for GAR validation, *i.e.,* pattern matching.

The space cost of IncJDisR is $O(C_t \cdot (|G_1|(|G_1| + |r_1(G_2)|) + \max(|G_1|+|r_1(G_2)|, |G_1 \otimes \Delta G_1|+|r_1(G_2 \otimes \Delta G_2)|)))$ in the worst case, where C_t denotes the total number of candidate GARs in the generation trees. It covers the cost for maintaining generation trees (resp. previous pivot mappings of each candidate and index for matching each GAR [40]), which is in $O(C_t)$ (resp. $O(|G_1|(|G_1| + |r_1(G_2)|))$ and $O(\max(|G_1| + |r_1(G_2)|, |G_1 \otimes \Delta G_1| + |r_1(G_2 \otimes \Delta G_2)|)))$.

Since the space cost for maintaining intermediate results is obviously larger that of the graphs, we persist auxiliary structures on disks to speed up the incremental discovery. In contrast, the data graphs are accommodated in main memory by default.

5.2 Parallel Discovery Algorithm

We parallelize the discovery framework for GARs across graphs and show that each step has performance guarantee (parallel scalability).

Parallel scalability. We adapt the notion of [67] to characterize the effectiveness of parallel algorithms. Consider a sequential algorithm $\mathcal{A}$ for a problem Y. Let $t(|I_Y|, |G_1|, |G_2|)$ be the worst-case runtime of $\mathcal{A}$ when solving the instance I_Y of Y on graphs G_1 and G_2. A parallel algorithm $\mathcal{A}_p$ for Y is *parallelly scalable relative to* $\mathcal{A}$ if for any instance I_Y and graphs G_1 and G_2, the runtime of $\mathcal{A}_p$ for handling I_Y using n processors can be expressed as:

$$T(|I_Y|, |G_1|, |G_2|, n) = O\big(\frac{t(|I_Y|, |G_1|, |G_2|)}{n}\big).$$

Intuitively, the parallel scalability guarantees speedup of parallel algorithm $\mathcal{A}_p$ relative to a "yardstick" sequential algorithm $\mathcal{A}$. Such $\mathcal{A}_p$ can reduce the cost of $\mathcal{A}$ when more processors are used.

Below we outline the parallelization for each step of the discovery framework. Given n machines, we partition the graphs into n fragments such that each fragment is small enough to fit into the memory of a machine; discovery is then conducted on all the fragments by multi-machine parallelism. We show that the parallelized batch and incremental algorithms are parallelly scalable and thus can scale with large graphs in principle by adding machines.

Parallelization. (1) A parallel algorithm for graph join (the first step) is in place with parametric simulation [43]. It evenly partitions G_1 and G_2 into n fragments each, and iteratively filters *invalid matches* in synchronized parallel steps. Along the same lines, the incremental graph join algorithm can also be readily parallelized.

(2) Using hash-based task assignment, a parallel filtering algorithm PFilter can be developed to deduce $r_1(G_2)$. It decomposes the operations of Filter into three stages, *i.e.,* path generation, score computation for individual paths, and score aggregation and data selection. Then each one is conducted by independent and parallel tasks, which are allocated to processors according to the hash values of different kinds of task identifiers. Similarly, we can parallelize the incremental filtering algorithm IncFilter into PIncFilter, by hash-based partitioning of the path re-generation and score adjustment.

(3) Denoted as PJDisR, the parallel discovery algorithm works on graphs that are partitioned across n processors. The overall procedure in PJDisR follows [35, 39], where candidate GARs are generated in a levelwise manner and verified in parallel synchronously. Each parallel round is responsible for validating rules that reside at the same level of the generation trees. It also maintains partial matches as workloads and eliminates the skewed ones by evenly partitioning the partial matches across consecutive rounds. Based on the same workload balancing strategy, the incremental discovery algorithm IncJDisR can be parallelized into PIncJDisR.

Analysis. The parallel batch graph join algorithm is in $O((|V_1| + |V_2|)(|E_1| + |E_2|)/n)$ time; due to even partition of workloads achieved by hash-based distribution, PFilter needs $O((|G_1|^2 + |G_2|^2)/n)$ time; and PJDisR takes $O(\sum_{i=1}^{k} C_i \cdot (|G_1| + |r_1(G_2)|)^i/n)$ time, where C_i denotes the number of candidate GARs generated with i pattern nodes; this cost can be verified along the same lines as that in [35, 39]. Putting these together, the three-step discovery framework is parallelly scalable. Similarly, the parallel incremental algorithm for graph join (resp. graph filtering, rule mining) is in $O(|\mathsf{AFF}|/n)$ (resp. $O(|\mathsf{AFF}|(|\Delta G_1| + |\Delta G_2|)/n)$, $O(|\mathsf{AFF}|^2/n)$) time, retaining the parallel scalability as the batch ones.

The total space costs of the parallel algorithms are the same as that of their sequential counterparts given above, except for the ones for mining GARs. Specifically, PJDisR and PIncJDisR need more space to maintain partial matches *w.r.t.* GARs at the same level of generation trees, yielding space costs of $O(C_t \cdot (|G_1| + |r_1(G_2)|)^k)$ and $O(C_t \cdot (|G_1|(|G_1|+|r_1(G_2)|)+(|G_1 \otimes \Delta G_1|+|r_1(G_2 \otimes \Delta G_2)|)^k))$, respectively. Due to the workload balancing strategies, the space costs are evenly partitioned across n processors for all parallel algorithms.

6 EXPERIMENTAL STUDY

Using real-life and synthetic graphs, we experimentally evaluated (1) the effectiveness of graph filtering, (2) the efficiency and (parallel) scalability of the (incremental) GAR discovery algorithms across graphs, and (3) the accuracy of association deduction with the GARs mined by referencing external graphs. Moreover, (4) we conducted a case study with the rules mined from real-life graphs.

Experimental setting. We start with the experimental setting.

Datasets. We used five real-life datasets; each is a pair of a graph G_1 and a knowledge graph G_2. (1) Graph movieLens (ml) [9], a movie rating network with 224K entities (users and movies) and 25M ratings between users and movies; it was paired with IMDB [7], a movie knowledge graph with 14.2M vertices and 114M edges. (2) MGP [8], an academic genealogy database with 288K mathematicians as entities and 317K edges; it was paired with DBLP [1] for publications and authors, having 8.7M vertices and 161M edges. (3) OSM [10], an open geographic database with 278K entities and 325K edges; it was paired with DBpedia [4], a knowledge graph of 5.2M vertices and 17.5M edges, for general facts. The three pairs are denoted as ml-IMDB, MGP-DBLP and OSM-DBP, respectively.

The other two are widely-used benchmark datasets for recommendation [105], a type of association deduction: (4) Amazon–FBS, in which graph G_1 has 95K entities and 847K links drawn from the product recommendation network Amazon-review (Amazon) [58], and knowledge graph G_2 includes 88K entities and 2.5M links from

Freebase (FBS) [23]. (5) Last–FBS, in which G_1 has 71K entities and 3M links collected from the online music platform Last.fm (Last), and G_2 includes 58K entities and 464K links from Freebase. Both retain the 10-core setting, *i.e.,* each user or item has at least 10 links.

Following [82], we also generated synthetic graphs to test the scalability of (incremental) discovery. Graph G_1 has up to 20M vertices and 150M edges, and G_2 has up to 50M vertices and 1.1B edges.

Accuracy measure. We evaluated the accuracy of association deduction with the mined GARs, including recommendation, in terms of F-measure. It is defined as $2 \cdot \frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$, where precision is the ratio of true associations to all associations deduced, while recall is the ratio of associations correctly derived to all the true associations.

Following [40, 54], to test the accuracy of association deduction over ml-IMDB, MGP-DBLP and OSM-DBP, we manually injected noises into their graphs G_1 by changing attribute values and removing edges. This was controlled by a noise ratio α%, *i.e.,* the ratio of changed values and removed edges to all attribute values and edges in each original graph G_1. By default, α% was set as 3%. Therefore, a deduced association is classified as a true association over these graphs if it restores a value change or edge removal. For benchmark datasets Amazon–FBS and Last–FBS, the accuracy of recommendation was evaluated directly over the real test sets provided in [105], which also include all true associations.

Updates. We randomly generated updates $(\Delta G_1, \Delta G_2)$ from G_1 and G_2 that include both edge and vertex updates, without heavily affecting the underlying degree distribution. They were controlled by the size $|(\Delta G_1, \Delta G_2)| = |\Delta G_1| + |\Delta G_2|$ and a ratio of edge and vertex insertions to deletions. We kept this ratio as 1 unless stated otherwise, *i.e.,* the size of each pair of graphs remains stable.

Algorithms. We implemented the following algorithms in C++. (1) PFilter and PIncFilter for graph filtering. (2) Batch GAR discovery algorithm PJDisR, and variant PJDisR_1 (resp. PJDisR_F), which mines GARs from the join of G_1 and entire 1-hop neighbors of HER matches in G_2 (resp. the entire join of G_1 and G_2). (3) Incremental GAR discovery algorithm PIncJDisR. (4) GARJDet, GARJDet_1 and GARJDet_F, which apply the GARs discovered by PJDisR, PJDisR_1 and PJDisR_F to deduce associations following [40], respectively.

We also compared with the following baselines. (5) AMIE+ [52], an algorithm for discovering Horn rules. (6) GFDDet [49] and GARDet [40], which enforce the GFDs and GARs that are mined from G_1 only, respectively, *without referencing* G_2. (7) HornDet, which uses Horn rules mined by AMIE+ from G_1 as in [40, 49]. (8) LiteralE [65], an ML-based association deduction method that leverages literals from knowledge graphs. (9) KGAT [105] and KGIN [106], two ML recommendation models; both incorporate data from knowledge graphs via graph attention networks [101].

We adopted SimplE [64] as the ML predicates for general GARs, due to its high accuracy and efficiency. We used KGAT [105] as the embedded ML model in GARs for recommendation.

Configuration. We used a hyper computing cluster with up to 12 machines; each is powered by 12 Intel Xeon 2.2GHz cores and 64GB DDR4 RAM. We adopted Solid-State Drives (SSDs) to store the auxiliary structures for incremental PIncJDisR. Unless stated otherwise, we set $\delta = 0.05$ in graph filtering; $\sigma = 2000$, $k = 5$, and $|(\Delta G_1, \Delta G_2)| = 10\%(|G_1| + |G_2|)$ in (incremental) discovery. We implemented parametric simulation [43] as the HER function f and used $n = 8$ machines for parallel algorithms by default. We terminated any test when it ran for more than 15 hours. All experiments were repeated five times, and the average is reported here.

Table 2: Percentage of the filtered data from G_2 ($\delta = 0.05$)

Datasets \ Methods	PFilter	1-hop nbr	2-hop nbr	full join (#matches by f)
MGP-DBLP	3.9%	10.8%	63.8%	100% (75K)
ml-IMDB	7.8%	11.5%	22.9%	100% (62K)
OSM-DBP	1.5%	5.1%	19.0%	100% (19K)
Amazon–FBS	26.2%	27.7%	100.0%	100% (25K)
Last–FBS	87.3%	81.8%	99.8%	100% (48K)

Experimental results. We next report our findings.

Exp-1: Graph filtering. We first tested the effectiveness of our graph filtering strategy. We compared the runtime of discovery algorithms PJDisR and PIncJDisR with PJDisR_1 and PJDisR_F.

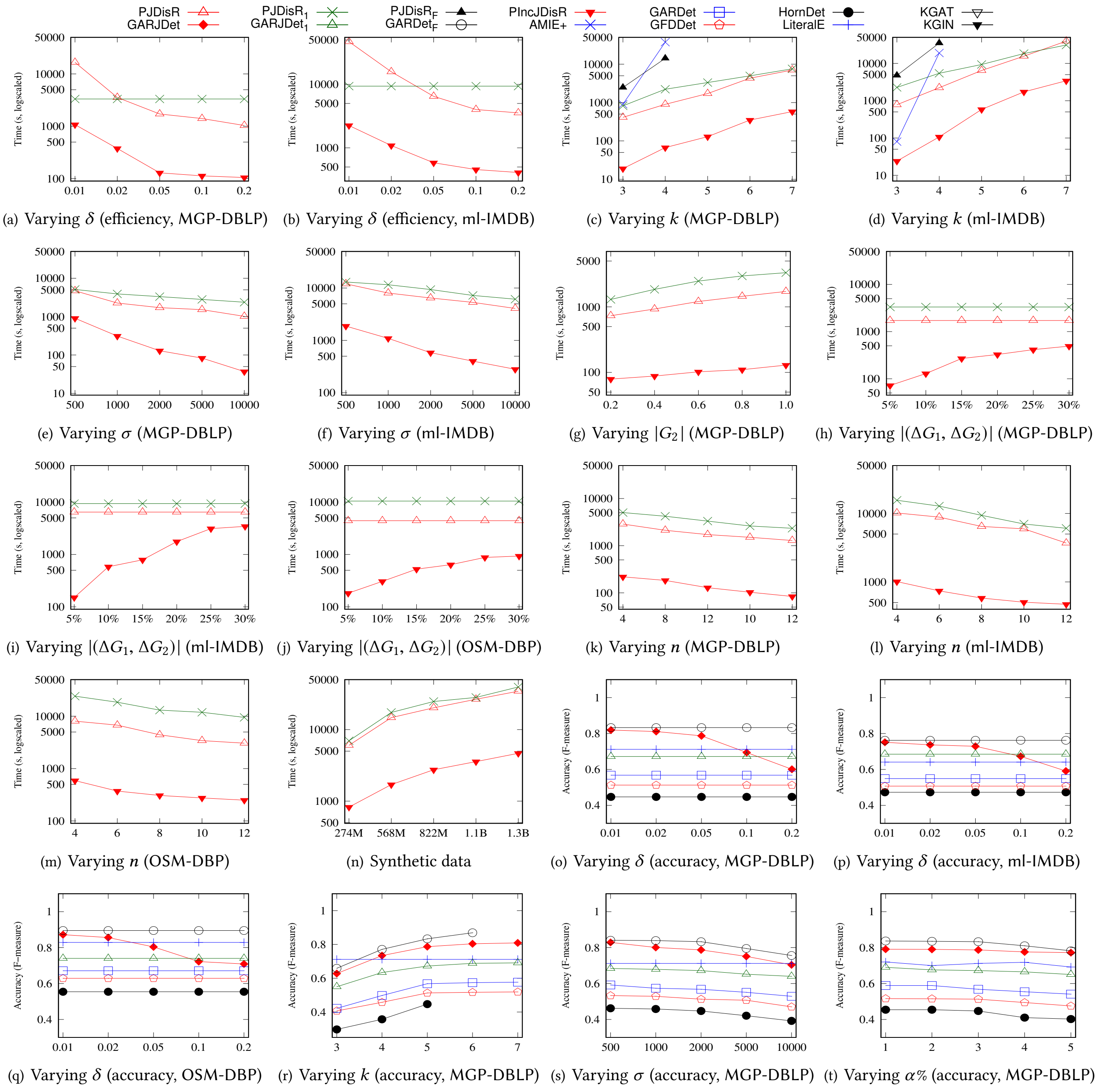
Varying the score threshold δ from 0.01 to 0.2, Figures 4(a)–4(b) report the runtime of these algorithms on MGP-DBLP and ml-IMDB, respectively. We find the following. (1) Since fewer paths are selected from G_2 by PFilter and PIncFilter when δ gets larger, *i.e.,* the filtered subgraph $r_1(G_2)$ is smaller, PJDisR and PIncJDisR take less time with the increase of δ, as expected, *e.g.,* when $\delta = 0.05$, the size of $r_1(G_2)$ is on average only 25.3%$|G_2|$. (2) On average, PJDisR beats PJDisR_F by 17.4× on average, and the gap gets larger as δ grows. In fact, the runs of PJDisR_F cannot terminate in 15 hours with default $k = 5$ (hence not shown). When $\delta = 0.2$, PJDisR is 36.8× faster, while the mined GARs still achieve high accuracy in association deduction (see Exp-3). (3) PJDisR is on average 2.1× faster than PJDisR_1 when $\delta = 0.05$, since mostly 1-hop neighbors of matched vertices in G_2 are larger than $r_1(G_2)$ extracted by PFilter. (4) PFilter and IncFilter on average take 92.1s and 33.8s on the two datasets (not shown), respectively, while PJDisR and PIncJDisR take 9320s and 1209s on average. Thus (incremental) GAR discovery cost is dominated by the mining time, not by filtering.

In addition, Table 2 reports the percentage of the data in external G_2 that is extracted by different graph filtering approaches. Here δ is set 0.05 for PFilter, and 1-hop nbr (resp. 2-hop nbr) represents the extraction of entire 1-hop neighbors (resp. 2-hop neighbors) of all HER matched vertices in G_2. As shown there, on average PFilter extracts 25.3% data from G_2 in the five datasets, which is often much smaller than filtering the entire 1-hop neighbors. Extracting 2-hop neighbors can take as high as the entire G_2. GAR discovery needs more than 15 hours with this strategy except on recommendation benchmarks. Joins of G_1 with the entire G_2 increase the size $|G_1|$ by 9.97× on average, and make PJDisR_F prohibitively costly.

Exp-2: Rule discovery. We next evaluated the efficiency and (parallel) scalability of the (incremental) discovery algorithms.

Varying k. We first tested the impact of pattern node numbers. From the results in Figures 4(c)–4(d), we find the following.

(1) When the bound k on pattern node numbers is varied from 3 to 7, PJDisR and PIncJDisR consistently beats the baselines except AMIE+. AMIE+ is fast for $k = 3$ since it mines Horn rules that are much simpler than GARs. However, when $k > 3$, AMIE+ gets much slower because it is a single-machine algorithm; and it uses



(a) Varying δ (efficiency, MGP-DBLP) (b) Varying δ (efficiency, ml-IMDB) (c) Varying k (MGP-DBLP) (d) Varying k (ml-IMDB)

(e) Varying σ (MGP-DBLP) (f) Varying σ (ml-IMDB) (g) Varying $|G_2|$ (MGP-DBLP) (h) Varying $|(\Delta G_1, \Delta G_2)|$ (MGP-DBLP)

(i) Varying $|(\Delta G_1, \Delta G_2)|$ (ml-IMDB) (j) Varying $|(\Delta G_1, \Delta G_2)|$ (OSM-DBP) (k) Varying n (MGP-DBLP) (l) Varying n (ml-IMDB)

(m) Varying n (OSM-DBP) (n) Synthetic data (o) Varying δ (accuracy, MGP-DBLP) (p) Varying δ (accuracy, ml-IMDB)

(q) Varying δ (accuracy, OSM-DBP) (r) Varying k (accuracy, MGP-DBLP) (s) Varying σ (accuracy, MGP-DBLP) (t) Varying $\alpha\%$ (accuracy, MGP-DBLP)

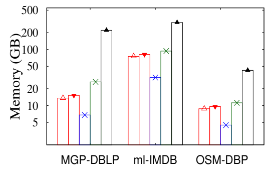
Figure 4: Performance evaluation

SQL to validate rules, which does not explore the locality of graph pattern matching and is costly for large rules.

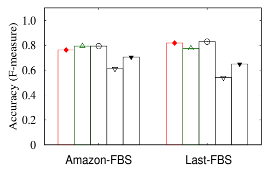
(2) PJDisR and PIncJDisR are able to discover GARs with fairly large patterns, *e.g.*, the two take 7056s and 582s to mine GARs with 7 patterns nodes and 7 edges on MGP-DBLP, respectively, when $k = 7$, while PJDisR$_F$ cannot terminate in 15 hours when $k = 5$.

Varying σ. We varied σ from 500 to 10000 to check the impact of support threshold on the discovery algorithms. As shown in Figures 4(e)–4(f) over MGP-DBLP and ml-IMDB, respectively, (1) the runtime of all levelwise methods decreases as σ grows. This is because the anti-monotonicity of support can prune more candidate rules when σ gets larger, reducing unnecessary validation. (2) PJDisR is at least 14.4× faster than PJDisR$_F$, again verifying the effectiveness of the graph filtering strategy.

Varying $|G_2|$. We also tested the impact of the size $|G_2|$ of external G_2 by randomly selecting 20% to 100% entities and relations in



(a) Memory consumption



(b) Recommendation accuracy

Figure 5: Memory usage and recommendation accuracy

external DBLP. As shown in Figure 4(g), when G_2 grows, PJDisR and PIncJDisR take longer since the size of $r_1(G_2)$ also increases. This said, PJDisR still outperforms PJDisR$_1$ by at least 1.7×. We also find that larger G_2 can help improve the accuracy of association deduction with GARJDet, as expected. For instance, the accuracy grows from 0.65 to 0.72 over MGP-DBLP when 100% of entities and relations from DBLP are selected as G_2, instead of 20%.

Incremental discovery. We next tested the efficiency of incremental PIncJDisR, by varying the size $|(\Delta G_1, \Delta G_2)|$ of updates from 5% to 30% of the size of original (G_1, G_2). As reported in Figures 4(h)–4(j) over the three real-life datasets, (1) all incremental methods get slower when updates get larger, as expected. (2) PIncJDisR is on average 10.6× faster than batch PJDisR, up to 42.9× when the size $|(\Delta G_1, \Delta G_2)|$ accounts for 5% of $|(G_1, G_2)|$. (3) PIncJDisR outperforms PJDisR even when $|(\Delta G_1, \Delta G_2)|$ reaches 30% of $|G_1| + |G_2|$.

Parallel scalability. Varying the number n of machines used, we checked the parallel scalability of the discovery methods. As shown in Figures 4(k)–4(m), PJDisR (resp. PIncJDisR) is on average 2.6× (resp. 2.4×) faster when n is varied from 4 to 12. It takes only 1287s on MGP-DBLP when $k = 5$ by using $n = 12$ machines.

Synthetic data. We also tested the scalability of our methods by varying the size of synthetic (G_1, G_2) from 274M to 1.3B. As shown in Figure 4(n), (1) all algorithms take longer on larger graphs, as expected. (2) PJDisR and PIncJDisR scale with large graphs. They take 6029s and 825s on the graphs of size 274M, respectively.

The results on other datasets are consistent and hence not shown.

Storage cost. Figure 5(a) reports the memory consumption of PJDisR, PIncJDisR and baselines. As shown there, PJDisR (resp. PIncJDisR) consumes on average 4.9× (resp. 4.4×) less memory than PJDisR$_1$ and PJDisR$_F$, by using smaller filtered subgraph. AMIE+ is a single-machine algorithm for simpler Horn rules and takes less memory when $n = 8$. However, it runs out of memory when the size of graph G_1 reaches 136M. PIncJDisR takes on average 417.7 GB disk to maintain auxiliary structures. AMIE+ does not use disk (not shown).

Exp-3: Association deduction. We next evaluated the accuracy of association deduction with the GARs mined across graphs. For rule-based methods, we applied the entire set of rules mined from either graphs G_1 alone or (filtered) graph joins of G_1 and G_2.

Varying δ. Figures 4(o)–4(q) report the accuracy of GARJDet with the GARs mined by PJDisR over MGP-DBLP, ml-IMDB and OSM-DBP, respectively, when δ is varied from 0.01 to 0.2. We can see that (1) GARJDet gets less accurate when δ gets larger. This is because more paths in external graphs G_2 are omitted during graph filtering in response to a larger δ. (2) The accuracy of GARJDet increases steadily when δ drops from 0.2 to 0.05; then it grows much slower,

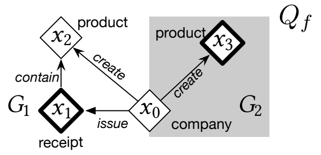


Figure 6: GAR across G_1 and G_2 for fraud detection

while the discovery spends much longer at this point (Figures 4(a)–4(b)). (3) When $\delta = 0.05$, GARJDet on average beats GARDet and GFDDet that enforce rules discovered from G_1 only, by 30.4% and 41.7% in accuracy, respectively, verifying the effectiveness of referencing G_2 with paths generated by LSTM. Moreover, it is 10.7% and 6.5% more accurate than GARJDet$_1$ and the ML-based LiteralE, respectively. (4) GARJDet also outperforms HornDet, since (a) GARs are more powerful than Horn rules and hence can deduce more types of associations, and (b) GARJDet references G_2. (5) While GARJDet$_F$ is a bit more accurate than GARJDet since all GARs mined from the entire graph joins by PJDisR$_F$ are applied, the time for GAR discovery is more than 15 hours. Thus our graph filtering strategy with LSTM strikes a balance between the accuracy of association deduction and the cost of rule discovery.

Sensitivity to discovery parameters. Figures 4(r)–4(s) show the accuracy of GARJDet by applying GARs discovered with different bound k on pattern node numbers and support threshold σ, respectively, over MGP-DBLP. We find that (1) in most of the cases, GARJDet still outperforms the competitors except for GARJDet$_F$, which is consistent with Figures 4(o)–4(q). (2) GARJDet is more accurate when given a larger k or a smaller σ, since more useful GARs are returned by PJDisR and applied in association deduction in such settings.

Varying α%. As shown in Figure 4(t) on MGP-DBLP, the accuracy gap between GARJDet and LiteralE, GARDet and HornDet remains consistent when varying noise ratio α% from 1% to 5%. GARJDet is 38.0% (resp. 55.9%) more accurate than GARDet (resp. GFDDet). LiteralE is better than the two but worse than GARJDet. All rule-based methods get less accurate when α% increases, since their rules may yield more false associations from the noise.

Recommendation. We further compared GARJDet with ML-based KGAT and KGIN on benchmarks Amazon–FBS and Last–FBS, to evaluate the accuracy of recommendation on real test sets. As shown in Figure 5(b), (1) GARJDet is on average 36.1% and 17.0% more accurate than KGAT and KGIN, respectively, although the latter two also leverage information from the same knowledge graphs. (2) Since graphs G_1 are short of, *e.g.,* attributes and labels, HornDet, GFDDet and GARDet do not find many meaningful rules from G_1 alone; hence their accuracy is rather low, *e.g.,* 0.11 for GARDet (not shown). (3) In particular, GARJDet performs much better than KGAT and KGIN on Last–FBS. As observed in [105], this is because the preferences of users with many interactions are too general to be caught by these ML models in Last–FBS.

Exp-4: Case study. Besides the real case of anti-money laundering given in Example 1, below we show another exemplary GAR discovered across a pair of graphs G_1 and G_2 for fraud detection in receipts [19]; its pattern is depicted in Figure 6.

It is reported that dishonest people often commit document forgery by, *e.g.,* modifying the true prices of purchases or restaurants' addresses in receipts to get money from insurance or meet the requirements of reimbursement. We can use GAR φ_f to catch such fraudulent receipts. Here graph G_1 is constructed with information extracted from the receipt dataset for document forgery detection in [6], while external G_2 refers to the knowledge graph built from the Sirene database [11, 97], which includes statistical data about millions of French companies.

Specifically, GAR $\varphi_f = Q_f[\bar{x}](x_1.\text{\#articles}=1 \wedge x_1.\text{year}=x_3.\text{year} \wedge \mathcal{M}_f(x_2, x_3) \rightarrow x_1.\text{total} = x_3.\text{price})$. It says that if receipt x_1 is issued by company x_0 which also creates x_2, the only product contained in receipt x_1 (in G_1), then x_1's total to pay must be equal to the price of product x_3 when x_3's year of production is the same as that of x_1, and x_2 and x_3 refer to the same product (determined by the ML model $\mathcal{M}_f$). Note that the correct detailed information of products, *e.g.,* year of production and price, can only be fetched from the external graph G_2. This GAR helps detect higher prices declared by fraudsters for products that are damaged or stolen, the most common type of fraud in receipt data [19]. Similarly, one can extract the real addresses of restaurants from G_2 to detect erroneous ones presented in the receipts, by using GAR across G_1 and G_2.

Summary. We find the following. (1) By referencing external (knowledge) graphs, the accuracy of association deduction with GARs is improved by 30.4% on average, up to 38.6%. Moreover, it outperforms the state-of-the-art ML-based methods by more than 10% over recommendation benchmarks. (2) Graph filtering is effective: it accelerates GAR discovery by 17.4× on average, while retaining high accuracy in association deduction. (3) Mining GARs across graphs G_1 and external graphs G_2 is feasible: it takes 1287s to mine GARs having 10 pattern nodes and edges from graphs of size 170M. (4) The incremental GAR discovery method outperforms the batch counterpart by 3.4× even when the size of updates accounts for 30% of the original graphs; it beats PJDisR by 85.5× when $|(\Delta G_1, \Delta G_2)| = 1\% (|G_1| + |G_2|)$. (5) Our batch (resp. incremental) GAR discovery algorithm is parallelly scalable, improving the performance by 2.6× (resp. 2.4×) on average when using 12 machines instead of 4.

7 RELATED WORK

Data enrichment. Data enrichment aims to enhance a dataset by extracting additional information from external sources. It has been employed in recommender systems [24, 61, 103, 104, 108], link prediction [55, 65, 92, 111], and entity resolution (ER) [27, 76, 102]. Recommenders, *e.g.,* KPRN [107], KGAT [105] and KGIN [106], learn embeddings of user-item relationships from a knowledge base. Link prediction and ER algorithms leverage features that are extracted from social graphs [16], texts [18, 98], pre-computed rules [57] and knowledge graphs [63, 87, 99]. Cross-graph embedding was used to integrate link prediction and network alignment [31, 32].

In contrast to the prior work, (1) this work studies rule discovery by referencing external graphs G_2, to improve the accuracy of association deduction. (2) It proposes a graph filtering method and an ML model to identify properties from G_2 that are relevant to entities in G_1, an approach that has not been explored before.

Rule discovery. Discovery of association rules has been well studied for relational data, *e.g.,* [22, 36–38, 45, 77, 85, 86, 117]. Over graphs, the prior work has mostly focused on rules defined with graph patterns *without* logic conditions, and hence reduces to graph pattern mining. For example, graph evolution rules [21, 89], link formation rules [71], and predictive graph rules [100] are mined by employing pattern mining techniques [50] such as gSpan [112]. RNNLogic [88] and NeuralLP [113] adopt specialized machine learning models, again to learn rules defined with paths but *without* logic conditions. Horn rule learners, *e.g.,* AnyBURL [78], AMIE [53], AMIE+ [52], and GPFL [56], employ either top-down or bottom-up approaches for discovering rules with "path" patterns.

Closer to this work are discovery algorithms for graph functional dependencies (GFDs) [39], graph differential dependencies (GDDs) [69], graph association rules (GARs) [35, 41], and graph cleaning rules (GCRs) [34], which are defined with both graph patterns and logic conditions. The algorithms are either mining-based via levelwise traversal [39, 69], or learning-based [34, 41] by iteratively generating candidate rules and validating them. The algorithm of [35] adopts a strategy to select relevant predicates and a sampling method to strike a balance between accuracy and scalability.

As opposed to the prior work, (1) we develop the first algorithms for discovering rules by referencing an external graph G_2, rather than from a single graph G_1. (2) The algorithms align entities across G_1 and G_2 via HER, and train an ML model to identify relevant properties. None of these has been studied before. (3) We adapt the mining-based algorithm to GARs across G_1 and G_2. The objective is to reduce noise and the size of merged data. Moreover, we show that the algorithms remain parallelly scalable in the multi-graph setting.

Incremental rule discovery. Incremental rule discovery has been studied over relational data [14, 90, 96, 110, 118]. On graphs, existing work mostly focuses on frequent subgraph mining (FSM) [50]. For instance, IncGM+ [13] adopts existing method on incremental graph pattern matching [94]. TipTap [20, 81] computes a bounded approximation for the collection of frequent subgraphs.

To the best of our knowledge, no incremental algorithm is yet in place for mining rules with both general patterns and logic conditions in response to updates. We develop the first such algorithms, which discover rules across two separate graphs, and moreover, guarantee both the relative boundedness and the parallel scalability.

8 CONCLUSION

This work has made the first effort to study (1) rule discovery from a graph G_1 by referencing external graphs G_2, and proposing a notion of graph joins; (2) a graph filtering model to identify only information relevant to entities in G_1, reducing noise and cost; and (3) the first incremental rule discovery algorithms, in a single graph or across different graphs. These algorithms have performance guarantees, namely, parallel scalability and/or relative boundedness. Our experimental study has verified that the methods are promising in improving the accuracy of association deduction.

One topic for future work is to identify external graphs that help enrich our graphs on hand, possibly by using HER. Another topic is to study rule discovery across G_1 and G_2 under privacy constraints.

ACKNOWLEDGMENTS

This work was supported in part by Royal Society Wolfson Research Merit Award WRM/R1/180014 and NSFC 62302027.

REFERENCES

[1] 2022. DBLP collaboration network. *https://www.aminer.org/citation.*

[2] 2022. Wikidata – Recent changes. *https://www.amazon.science/blog/combining-knowledge-graphs-quickly-and-accurately.*

[3] 2022. Wikipedia. *https://www.wikipedia.org.*

[4] 2023. DBpedia. *http://www.dbpedia.org.*

[5] 2023. Facebook Demographic Statistics. *https://backlinko.com/facebook-users.*

[6] 2023. Fraud Detection Contest Dataset. *http://findit.univ-lr.fr.*

[7] 2023. IMDB dataset. *https://www.imdb.com/interfaces.*

[8] 2023. The Mathematics Genealogy Project. *https://mathgenealogy.org.*

[9] 2023. Movielens. *http://grouplens.org/datasets/movielens/.*

[10] 2023. OpenStreeMap. *http://www.openstreetmap.org.*

[11] 2023. Sirene Database. *https://www.sirene.fr/sirene/public/accueil.*

[12] 2023. Social Network Usage and Growth Statistics. *https://backlinko.com/social-media-users.*

[13] Ehab Abdelhamid, Mustafa Canim, Mohammad Sadoghi, Bishwaranjan Bhattacharjee, Yuan-Chi Chang, and Panos Kalnis. 2017. Incremental Frequent Subgraph Mining on Large Evolving Graphs. *IEEE Trans. Knowl. Data Eng.* 29, 12 (2017), 2710–2723.

[14] Ziawasch Abedjan, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. 2014. Detecting unique column combinations on dynamic data. In *ICDE*. 1036–1047.

[15] Ghadeer Abuoda, Saravanan Thirumuruganathan, and Ashraf Aboulnaga. 2022. Accelerating Entity Lookups in Knowledge Graphs Through Embeddings. In *ICDE*. 1111–1123.

[16] Muhammad Aurangzeb Ahmad, Zoheb Borbora, Jaideep Srivastava, and Noshir S. Contractor. 2010. Link Prediction Across Multiple Social Networks. In *ICDM Workshops*. 911–918.

[17] Waseem Akhtar, Alvaro Cortés-Calabuig, and Jan Paredaens. 2010. Constraints in RDF. In *SDKB*. 23–39.

[18] Bo An, Bo Chen, Xianpei Han, and Le Sun. 2018. Accurate Text-Enhanced Knowledge Graph Representation Learning. In *NAACL-HLT*. 745–755.

[19] Chloé Artaud, Nicolas Sidere, Antoine Doucet, Jean-Marc Ogier, and Vincent Poulain D'Andecy Yooz. 2018. Find it! Fraud Detection Contest Report. In *ICPR*. 13–18.

[20] Çigdem Aslay, Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, and Aristides Gionis. 2018. Mining Frequent Patterns in Evolving Graphs. In *CIKM*. 923–932.

[21] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. 2009. Mining Graph Evolution Rules. In *ECML/PKDD*. 115–130.

[22] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *Proc. VLDB Endow.* 11, 3 (2017), 311–323.

[23] Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*. 1247–1250.

[24] Sarah Bouraga, Ivan Jureta, Stéphane Faulkner, and Caroline Herssens. 2014. Knowledge-based recommendation systems: a Survey. *Int. J. Intell. Inf. Technol.* 10, 2 (2014), 1–19.

[25] K Buehler. 2019. Transforming approaches to aml and financial crime. *McKinsey* (2019).

[26] Business of Data. 2020. How Graph Databases are Transforming Advanced Analytics. *https://www.business-of-data.com/articles/graph-databases.*

[27] Muhao Chen, Yingtao Tian, Kai-Wei Chang, Steven Skiena, and Carlo Zaniolo. 2018. Co-training Embeddings of Knowledge Graphs and Entity Descriptions for Cross-lingual Entity Alignment. In *IJCAI*. 3998–4004.

[28] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[29] Brian Dean. 2020. Movie Recommendations Powered by Knowledge Graphs and Neo4j. *https://towardsdatascience.com/movie-recommendations-powered-by-knowledge-graphs-and-neo4j-33603a212ad0.*

[30] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*. 601–610.

[31] Xingbo Du, Junchi Yan, and Hongyuan Zha. 2019. Joint Link Prediction and Network Alignment via Cross-graph Embedding. In *IJCAI*. 2251–2257.

[32] Xingbo Du, Junchi Yan, Rui Zhang, and Hongyuan Zha. 2022. Cross-Network Skip-Gram Embedding for Joint Network Alignment and Link Prediction. *IEEE Trans. Knowl. Data Eng.* 34, 3 (2022), 1080–1095.

[33] Wenfei Fan. 2022. Big Graphs: Challenges and Opportunities. *Proc. VLDB Endow.* 15, 12 (2022), 3782–3797.

[34] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Muyang Liu, Ping Lu, and Chao Tian. 2023. Making It Tractable to Catch Duplicates and Conflicts in Graphs. *Proc. ACM Manag. Data* 1, 1 (2023), 86:1–86:28.

[35] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Ping Lu, and Chao Tian. 2022. Discovering Association Rules from Big Graphs. *Proc. VLDB Endow.* 15, 7 (2022), 1479–1492.

[36] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management.* Morgan & Claypool Publishers.

[37] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering Conditional Functional Dependencies. *IEEE Trans. Knowl. Data Eng.* 23, 5 (2011), 683–698.

[38] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*. 384–398.

[39] 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.

[40] 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.

[41] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards Event Prediction in Temporal Graphs. *Proc. VLDB Endow.* 15, 9 (2022), 1861–1874.

[42] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.

[43] Wenfei Fan, Ping Lu, Kehan Pang, Ruochun Jin, and Wenyuan Yu. 2024. Linking Entities across Relations and Graphs. *ACM Trans. Database Syst.* (2024).

[44] Wenfei Fan and Chao Tian. 2022. Incremental Graph Computations: Doable and Undoable. *ACM Trans. Database Syst.* 47, 2 (2022), 6:1–6:44.

[45] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *Proc. VLDB Endow.* 14, 8 (2021), 1351–1364.

[46] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD*. 459–471.

[47] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *Proc. VLDB Endow.* 8, 12 (2015), 1502–1513.

[48] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Adding counting quantifiers to graph patterns. In *SIGMOD*. 1215–1230.

[49] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. 1843–1857.

[50] Philippe Fournier-Viger, Ganghuan He, Chao Cheng, Jiaxuan Li, Min Zhou, Jerry Chun-Wei Lin, and Unil Yun. 2020. A survey of pattern mining in dynamic graphs. *WIREs Data Mining Knowl. Discov.* 10, 6 (2020).

[51] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *WWW*. 2331–2341.

[52] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.* 24, 6 (2015), 707–730.

[53] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*. 413–422.

[54] Congcong Ge, Yunjun Gao, Honghui Weng, Chong Zhang, Xiaoye Miao, and Baihua Zheng. 2020. KGClean: An Embedding Powered Knowledge Graph Cleaning Framework. *CoRR* abs/2004.14478 (2020).

[55] Genet Asefa Gesese, Russa Biswas, Mehwish Alam, and Harald Sack. 2021. A survey on knowledge graph embeddings with literals: Which model links better literal-ly? *Semantic Web* 12, 4 (2021), 617–647.

[56] Yulong Gu, Yu Guan, and Paolo Missier. 2020. Towards learning instantiated logical rules from knowledge graphs. *arXiv preprint arXiv:2003.06071* (2020).

[57] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2018. Knowledge Graph Embedding With Iterative Guidance From Soft Rules. In *AAAI*. 4816–4823.

[58] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. 507–517.

[59] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia: Extended Abstract. In *IJCAI*. 3161–3165.

[60] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *ACM Comput. Surv.* 54, 4 (2021), 71:1–71:37.

[61] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S. Yu. 2018. Leveraging Meta-path based Context for Top- N Recommendation with A Neural Co-Attention Model. In *KDD*. 1531–1540.

[62] Robert Isele, Anja Jentzsch, and Christian Bizer. 2010. Silk server-adding missing links while consuming linked data. In *COLD*. 85–96.

[63] Jun'ichi Kazama and Kentaro Torisawa. 2007. Exploiting Wikipedia as External Knowledge for Named Entity Recognition. In *EMNLP-CoNLL*. 698–707.

[64] Seyed Mehran Kazemi and David Poole. 2018. SimplE Embedding for Link Prediction in Knowledge Graphs. In *NeurIPS*. 4289–4300.

[65] Agustinus Kristiadi, Mohammad Asif Khan, Denis Lukovnikov, Jens Lehmann, and Asja Fischer. 2019. Incorporating Literals into Knowledge Graph Embeddings. In *ISWC*. 347–363.

[66] Walter G Kropatsch. 1995. Building irregular pyramids by dual-graph contraction. *IEE Proceedings-Vision, Image and Signal Processing* 142, 6 (1995), 366–374.

[67] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of

efficient parallel algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.

[68] Eren Kurshan, Hongda Shen, and Haojie Yu. 2020. Financial Crime & Fraud Detection Using Graph Computing: Application Considerations & Outlook. In *TransAI*. 125–130.

[69] 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.

[70] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[71] Cane Wing-ki Leung, Ee-Peng Lim, David Lo, and Jianshu Weng. 2010. Mining interesting link formation rules in social networks. In *CIKM*. 209–218.

[72] 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*. 684–695.

[73] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2018. Multi-Hop Knowledge Graph Reasoning with Reward Shaping. In *EMNLP*. 3243–3253.

[74] 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*. 705–714.

[75] Yuanna Liu, Jie Geng, Xinyang Deng, and Wen Jiang. 2021. Relation-Aware Neighborhood Aggregation for Cross-lingual Entity Alignment. In *FUSION*. 1–7.

[76] Yinan Liu, Wei Shen, Yuanfei Wang, Jianyong Wang, Zhenglu Yang, and Xiaojie Yuan. 2021. Joint Open Knowledge Base Canonicalization and Linking. In *SIGMOD*. 2253–2261.

[77] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *Proc. VLDB Endow.* 13, 10 (2020), 1682–1695.

[78] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *IJCAI*. 3137–3143.

[79] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and Optimizing LSTM Language Models. In *ICLR*.

[80] Mohammad Hossein Namaki, Yinghui Wu, Qi Song, Peng Lin, and Tingjian Ge. 2017. Discovering Graph Temporal Association Rules. In *CIKM*. 1697–1706.

[81] Muhammad Anis Uddin Nasir, Çigdem Aslay, Gianmarco De Francisci Morales, and Matteo Riondato. 2021. TipTap: Approximate Mining of Frequent k-Subgraph Patterns in Evolving Graphs. *ACM Trans. Knowl. Discov. Data* 15, 3 (2021), 48:1–48:35.

[82] Sadegh Nobari, Xuesong Lu, Panagiotis Karras, and Stéphane Bressan. 2011. Fast random graph generation. In *EDBT*. 331–342.

[83] Enrico Palumbo, Diego Monti, Giuseppe Rizzo, Raphaël Troncy, and Elena Baralis. 2020. entity2rec: Property-specific knowledge graph embeddings for item recommendation. *Expert Syst. Appl.* 151 (2020), 113235.

[84] George Papadakis, Georgios M. Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. 2020. Three-dimensional Entity Resolution with JedAI. *Inf. Syst.* 93 (2020), 101565.

[85] 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.

[86] Thorsten Papenbrock and Felix Naumann. 2016. A hybrid approach to functional dependency discovery. In *SIGMOD*. 821–833.

[87] Pouya Pezeshkpour, Liyan Chen, and Sameer Singh. 2018. Embedding Multimodal Relational Data for Knowledge Base Completion. In *EMNLP*. 3208–3218.

[88] 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*.

[89] Erik Scharwächter, Emmanuel Müller, Jonathan F. Donges, Marwan Hassani, and Thomas Seidl. 2016. Detecting Change Processes in Dynamic Networks by Frequent Graph Evolution Rule Mining. In *ICDM*. 1191–1196.

[90] 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*. 253–264.

[91] Feichen Shen and Yugyung Lee. 2016. Knowledge discovery from biomedical ontologies in cross domains. *PloS one* 11, 8 (2016), e0160005.

[92] Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Trans. Knowl. Data Eng.* 27, 2 (2015), 443–460.

[93] Kai Shu, Suhang Wang, Jiliang Tang, Reza Zafarani, and Huan Liu. 2016. User Identity Linkage across Online Social Networks: A Review. *SIGKDD Explor.* 18, 2 (2016), 5–17.

[94] Xibo Sun, Shixuan Sun, Qiong Luo, and Bingsheng He. 2022. An In-Depth Study of Continuous Subgraph Matching. *Proc. VLDB Endow.* 15, 7 (2022), 1403–1416.

[95] 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.

[96] Zijing Tan, Ai Ran, Shuai Ma, and Sheng Qin. 2020. Fast incremental discovery of pointwise order dependencies. *Proc. VLDB Endow.* 16, 2 (2020), 1669–1681.

[97] Beatriz Martínez Tornés, Emanuela Boros, Antoine Doucet, Petra Gomez-Krämer, Jean-Marc Ogier, and Vincent Poulain d'Andecy. 2019. Knowledge-Based Techniques for Document Fraud Detection: A Comprehensive Study. In *CICLing*. 17–33.

[98] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing Text for Joint Embedding of Text and Knowledge Bases. In *EMNLP*. 1499–1509.

[99] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. 2019. Entity alignment between knowledge graphs using attribute embeddings. In *AAAI*. 297–304.

[100] Karel Vaculík. 2015. A Versatile Algorithm for Predictive Graph Rule Mining. In *ITAT*. 51–58.

[101] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[102] Alina Vretinaris, Chuan Lei, Vasilis Efthymiou, Xiao Qin, and Fatma Özcan. 2021. Medical Entity Disambiguation Using Graph Neural Networks. In *SIGMOD*. 2310–2318.

[103] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems. In *KDD*. 968–977.

[104] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *WWW*. 3307–3313.

[105] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *KDD*. 950–958.

[106] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning Intents behind Interactions with Knowledge Graph for Recommendation. In *WWW*. 878–887.

[107] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable Reasoning over Knowledge Graphs for Recommendation. In *AAAI*. 5329–5336.

[108] Ze Wang, Guangyan Lin, Huobin Tan, Qinghong Chen, and Xiyang Liu. 2020. CKAN: Collaborative Knowledge-aware Attentive Network for Recommender Systems. In *SIGIR*. 219–228.

[109] Antony J Williams, Lee Harland, Paul Groth, Stephen Pettifer, Christine Chichester, Egon L Willighagen, Chris T Evelo, Niklas Blomberg, Gerhard Ecker, Carole Goble, et al. 2012. Open PHACTS: semantic interoperability for drug discovery. *Drug discovery today* 17, 21-22 (2012), 1188–1198.

[110] Renjie Xiao, Zijing Tan, Shuai Ma, Wei Wang, et al. 2022. Dynamic Functional Dependency Discovery with Dynamic Hitting Set Enumeration. In *ICDE*. 286–298.

[111] Ruobing Xie, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2017. Image-embodied Knowledge Representation Learning. In *IJCAI*. 3140–3146.

[112] Xifeng Yan and Jiawei Han. 2002. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*. 721–724.

[113] Fan Yang, Zhilin Yang, and William W. Cohen. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *NIPS*. 2319–2328.

[114] Reza Zafarani and Huan Liu. 2016. Users joining multiple sites: Friendship and popularity variations across sites. *Inf. Fusion* 28 (2016), 83–89.

[115] Qianyi Zhan, Jiawei Zhang, Senzhang Wang, Philip S. Yu, and Junyuan Xie. 2015. Influence Maximization Across Partially Aligned Heterogenous Social Networks. In *PAKDD*. 58–69.

[116] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *SIGKDD*. 353–362.

[117] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A statistical perspective on discovering functional dependencies in noisy data. In *SIGMOD*. 861–876.

[118] Lin Zhu, Xu Sun, Zijing Tan, Kejia Yang, Weidong Yang, Xiangdong Zhou, and Yingjie Tian. 2019. Incremental discovery of order dependencies on tuple insertions. In *DASFAA*. 157–174.