THE UNIVERSITY
of EDINBURGH

Tackling the Veracity and Variety of Big Data

Ruochun Jin

Doctor of Philosophy

Laboratory for Foundations of Computer Science

School of Informatics

The University of Edinburgh

2022

Abstract

This thesis tackles the veracity and variety challenges of big data, especially focusing on graphs and relational data. We start with proposing a class of graph association rules (GARs) to specify regularities between entities in graphs, which capture both missing links and inconsistencies. A GAR is a combination of a graph pattern and a dependency; it may take as predicates machine learning classifiers for link prediction. We formalize association deduction with GARs in terms of the chase, and prove its Church-Rosser property. We show that the satisfiability, implication and association deduction problems for GARs are coNP-complete, NP-complete and NP-complete, respectively. The incremental deduction problem is DP-complete for GARs. In addition, we provide parallel algorithms for association deduction and incremental deduction.

We next develop a parallel algorithm to discover GARs, which applies an application-driven strategy to cut back rules and data that are irrelevant to users' interest, by training a machine learning model to identify data pertaining to a given application. Moreover, we introduce a sampling method to reduce a big graph G to a set H of small sample graphs. Given expected support and recall bounds, this method is able to deduce samples in H and mine rules from H to satisfy the bounds in the entire G.

Then we propose a class of temporal association rules (TACOs) for event prediction in temporal graphs. TACOs are defined on temporal graphs in terms of change patterns and (temporal) conditions, and may carry machine learning predicates for temporal event prediction. We settle the complexity of reasoning about TACOs, including their satisfiability, implication and prediction problems. We develop a system that discovers TACOs by iteratively training a rule creator based on generative models in a creator-critic framework, and predicts events by applying the discovered TACOs in parallel.

Finally, we propose an approach to querying relations $\mathcal{D}$ and graphs G taken together in SQL. The key idea is that if a tuple t in $\mathcal{D}$ and a vertex v in G are determined to refer to the same real-world entity, then we join t and v, correlate their information and complement tuple t with additional attributes of v from graphs. We show how to do this in SQL extended with only syntactic sugar, for both static joins when t is a tuple in $\mathcal{D}$ and dynamic joins when t comes from intermediate results of sub-queries on $\mathcal{D}$. To support the semantic joins, we propose an attribute extraction scheme based on K-means clustering, to identify and fetch graph properties that are linked to v via paths. Moreover, we develop a scheme to extract a relation schema for entities in graphs, and a heuristic join method based on the schema to strike a balance between the complexity and accuracy of dynamic joins.

Lay summary

The emergence and rapid growth of big graphs have brought new challenges of both "veracity" and "variety" in this big data era. Different from relational database, data quality (veracity) management in graphs, such as recovering missing friendships in social networks and identifying malicious webpages, is much more difficult due to their flexible topological structures, which calls for novel graph-oriented data quality managing algorithms. As for the challenge of "variety", the lack of schema dramatically hinders joint query across relational database and graphs, which further prohibits exploitation of data stored in different formats. Thus, in order to tackle these challenges, methods must be developed to address the following.

(1) How to effectively and efficiently capture associations (links) in graphs? Social networks and knowledge graphs usually have millions of vertices connected by edges with various labels. Without a predefined schema, these arbitrary edges construct flexible topologies in graphs that can never be harnessed by rules and constraints developed for relational data. In view of these, what rules should we use to catch associations in graphs? How can we efficiently discover and then apply these rules in big graphs?

(2) How to capture evolvement in temporal graphs? Each edge in a temporal graph carries a timestamp that records its time window, *e.g.,* the timestamp of a transaction between a customer and a product denotes the time of this purchase. These rich temporal conditions present extra challenges to graph analysis. Thus, rules that specify changes to graphs are desired for data quality management in temporal graphs.

(3) How to query relations and graphs taken together to make use of information from both? Unlike relational data, since there is no schema for graphs, it is nontrivial to write one query that accesses data in a relational database and a semistructured graph, and correlates their information. Moreover, the lack of linkages between these two sources makes it hard to decide which part of the data can be jointly queried.

Towards these questions, we develop a series of effective solutions, with both practical applications and theoretical analysis. More specifically, we propose a class of graph association rules to specify regularities between entities in graphs, which capture both missing links and inconsistencies. Moreover, we develop a parallel algorithm to discover these rules, where an application-driven strategy is applied to cut back rules and data that are irrelevant to users' interest. In addition, we propose a class of temporal association rules for event prediction in temporal graphs. We finally develop an approach to querying relations and graphs taken together in SQL, correlating information of the same real-world entity from different data sources.

Acknowledgements

I am highly grateful to my principal supervisor, Professor Wenfei Fan, for his strong support, valuable advice, continuous encouragement, and profound insights. During the arduous academic journey, Professor Fan's passion for science and extraordinary diligence have been always encouraging me to pursue for excellence. I am appreciated for every discussion we had during my PhD study, where his wisdom could always inspire and guide me to find the direction.

I am very grateful to Professor Yong Dou, who encouraged and supported me to enrol in and complete this PhD programme. I would also like to thank Professor Leonid Libkin, my second supervisor at the University of Edinburgh, from whose work I learned a lot about automata and complexity theory.

I especially thank the examiners of my thesis: Professor Andreas Pieris and Professor Floris Geerts, for providing precious and constructive feedback on this thesis.

I would also like to express my heartfelt gratitude to my collaborators including Dr. Yang Cao, Wenzhi Fu, Liang Geng, Muyang Liu, Dr. Ping Lu, Dr. Xiaojian Luo, Dr. Weijie Ou, Dr. Chao Tian, Resul Tugay, Dr. Jingbo Xu, Dr. Ruiqi Xu, Wenliang Yi, Dr. Qiang Yin, Dr. Wenyuan Yu and Dr. Jingren Zhou, for their helpful advice, effective support and pleasant cooperation.

I would like to thank Professor Peter Buneman for sharing not only his profound understanding about database research but also his latest progress in citation network study. I really enjoyed our discussions in his lovely office and I did learn more about the history in database research over the past decades. I would also like to thank all the labmates. Every seminar and lab lunch we had built my valuable memories.

Special thanks to Shangmin Guo, Siting Lu, Wenbin Hu, Yiyun Jin, Xiaoyu Zhang, Yan Yang, Yuanhao Li, Yanghao Wang, Tengfei Yuan, Weihong Li, Muhammad Ishaq, Daniel Mills, Rafael Karampatsis, Yun Lu, Lauren Watson, Giorgos Mousa, Markus Schneider, Bogdan Manghiuc, Peter Macgregor, Catriona Wedderburn, Nuiok Dicaire, Rudi Horn, Eric Munday, Patrick Chen, Asif Khan, Ibrahim Abu Farha and Nick McKenna, for their help and support on my life in Edinburgh.

Last but not least, I would express my deepest gratitude and love to my parents and my girlfriend Yaning. Their unconditional love and encouragement made this thesis possible.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ruochun Jin)*

Yaning, I love you.
Will you marry me?

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

It has been acknowledged and is predicted to continue that the quest for managing big graphs explodes exponentially [Bus20]. However, schemaless graphs make "veracity" and "variety" challenges of big data more staggering. Without a fixed schema, it is non-trivial to maintain data quality (veracity) in graphs, *e.g.,* recovering missing links and capturing entity associations. Meanwhile, flexible topological structures radically differentiate graphs from relational data, which hinders joint query across relations and graphs, introducing the "variety" challenge of big data.

We exemplify these challenges when managing real-life graph data as follows. We start with an example to show the challenge of "veracity" in graphs.

Example 1.1: As shown in G_1 of Figure 1.1, there exist data inconsistency in the population of France according to the English language Chapter and the French language Chapter of DBPedia knowledge graph [LIJ+15]. This data inconsistency is quite common due to asynchronous data updates. Worse still, since graphs have no schema, existing data quality constraints developed for relational data can hardly be adapted to managing data quality in graphs. One may apply graph functional dependencies (GFDs) [FWX16b, FL19] to capture the population inconsistency in G_1. However, GFDs stop short of catching missing links (G_2 in Figure 1.1), which have an existential

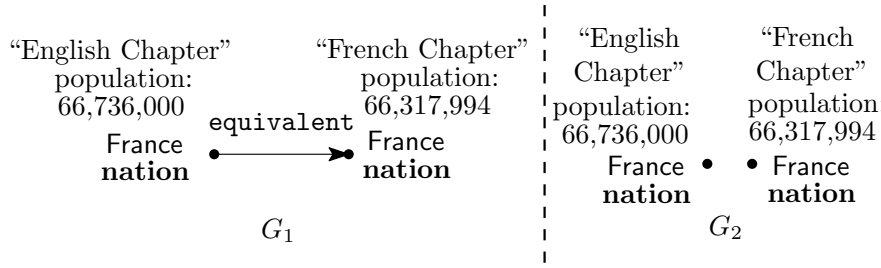
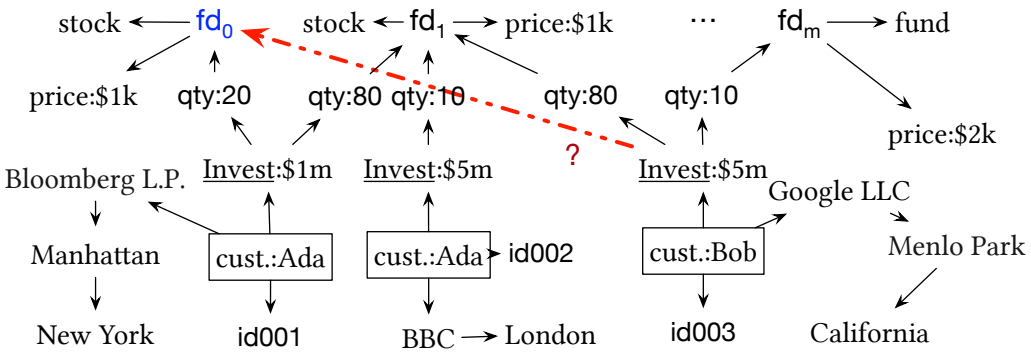


Figure 1.1: Data inconsistency and missing links in DBPedia knowledge graph [LIJ+15]

R

cid	name	credit	bal	city
cid01	Ada	fair	$500k	LDN
cid02	Bob	good	$110k	SF
cid03	Guy	good	$50k	NYC
cid04	Ada	fair	$100k	NYC

(a) Customer database $\mathcal{D}$



(b) Transaction knowledge graph G

Figure 1.2: Relational database $\mathcal{D}$ and graph G in a banking system

semantics. That is, if the link between the two vertices in G_1 is missing (see G_2), GFDs will fail to catch the inconsistency in population. Such problem of missing links is very common as well. This is because graphs are usually saved as edge files where each line records an edge of the graph. During data transmission, especially when transmitting large graphs, it is easy to lose lines (edges). This example shows that in order to effectively maintain good data quality (veracity) in graphs, rules that can jointly capture semantic inconsistencies and missing associations are desired. □

Another example is taken from a banking system, where information spreads across relations and graphs, demonstrating the challenge of "variety" introduced by graphs.

Example 1.2: Data is usually generated from various sources and is saved in radically different formats, *e.g.,* relations and graphs, which hinders users from fully exploring and leveraging the data value. This is the challenge of "variety" brought by the pervasively use of graphs. Consider a relational database $\mathcal{D}$ of customer accounts and a graph G of transactions in a banking system (Fig. 1.2). To decide whether to recommend a new financial product fd_0 to Bob (cid02 in $\mathcal{D}$), the bank needs to check the credit of Bob (in $\mathcal{D}$) and customers' investment history in the bank (in G). To check these, however, the bank has to synthesize relational data from $\mathcal{D}$ and transactions from graph G. Synthesizing or jointly querying graphs and relations is non-trivial since graphs do not have schema that may bridge these two data formats. □

In view of these challenges, this thesis studies various issues in graph analysis, and proposes practical techniques to tackle veracity and variety of big data in this era of emerging big graphs. Specifically, we propose a class of graph association rules (GARs) to specify regularities between entities in graphs, which capture both missing links and inconsistencies. This tackles the challenge of "veracity" introduced by big graph data as illustrated in Example 1.1. Then we develop a parallel algorithm to discover GARs efficiently, which applies an application-driven strategy to cut back rules and data that are irrelevant to users' interest. Furthermore, we propose a class of temporal association rules (TACOs) for event prediction, which extends association rules to temporal graphs. As for the challenge of "variety" brought by data spread across relations and graphs, we propose an approach to querying relations $\mathcal{D}$ and graphs G taken together in SQL. The key idea is that if a tuple t in $\mathcal{D}$ and a vertex v in G are determined to refer to the same real-world entity, then we join t and v, correlate their information and complement tuple t with additional attributes of v from graphs.

In this chapter, we firstly present research backgrounds and the review of related work. Then we outline the thesis with summary of our main contributions.

1.1 Background and State of the Art

Association in Graphs. Originating from capturing relationships of items in transactions, association rules prove to be effective on relational data [AIS93, ZZ02]. Similar rules have been applied on graphs [GTHS13] to analyze social networks, by extracting relations [EBBJ16, CF13]. Furthermore, association rules, *e.g.,* GPARs, have also been directly defined on graphs for graph data analysis and graph search [FWWX15, FWX16a, SWL$^+$18, NWS$^+$17]. These association rules in graphs are defined as changes of graph patterns, and apply graph pattern matching to capture relationships between vertices. Despite of wide application of association rules in graphs, there have been no formulation of association deduction and fundamental results for reasoning about association rules in graphs. Graph functional dependencies have recently been proposed for RDF [ACP10, HGPW16, FFTD15] and property graphs [FWX16b, FL19, FLLT20]. Expressed as universal logic sentences, these dependencies have been used to catch semantic inconsistencies in graphs. There has also been work on tuple-generating dependencies (TGDs [AHV95]) on graphs [FLTZ19, CP12], which are de-

fined with both universal and existential logic quantifiers.

Different from this rule-based approach, the machine learning (ML) community proposes to statistically learn representations for graphs and performs graph association analysis by link prediction models. Based on statistical learning, link prediction models learn vector embedding of each entity and relation [KP18]. They predict links over the embedding via additive functions [BUGD+13], product-based functions [TWR+16, ZTYL19], or deep neural networks [SRB+17]. Based on representation learning, state-of-the-art link prediction models have made significant progress in accuracy due to recent success of deep learning. However, predictions made by these models are unexplainable.

For practical use of graph rules, several parallel algorithms have been developed for graph pattern matching (*e.g.,* [FYX+18, AFU13, BLH19, RWHY19, SCC+14, RvRH+14, HAR11, SWW+12, QYC+14, RWHY19]) and GFD checking [FWX16b, FLTZ19, FLLT20], based on the following: (1) work unit distribution [FWX16b, FLTZ19, SCC+14]; (2) data replication [FLLT20, FYX+18, BLH19, RvRH+14]; (3) pattern decomposition and multiway join [AFU13, HAR11, SWW+12, QYC+14]; and (4) pattern match expansion by fetching data and verifying edges [RWHY19, WGH+19].

Rule Discovery in Graphs. Extensive efforts have been made on mining relational rules [HKPT99, CIP13, WGR01, SPK+19, SME+17, SPKN20], while there have also been several discovery methods for graph rules [FHLL20, FWWX15, NWS+17, KLL+19]. Rules on graphs are more complicated than relational rules. Thus, discovery of graph rules requires to mine both graph patterns and dependencies, and is more challenging than discovery of relational data quality rules. Applying classical level-wise search that is widely used in data mining, existing discovery methods enumerate and verify candidate rules in an exponentially large search space, with optimizations of pruning strategies. Specifically, GFDs [FHLL20], GPARs [FWWX15], graph temporal association rules [NWS+17] and graph differential dependencies [KLL+19] can be mined from graphs with different pruning strategies. GERM [BBBG09a] and LFR-Miner [LLLW10] revise pattern mining method gSpan [YH02] to mine graph evolution rules and link formation rules, respectively.

Similarly, some rule learners are in place to discover Horn rules of restricted forms from knowledge graphs modeled in RDF [MMM+04]. These rule learners typically enumerate paths as candidate rules and learn a weight for each candidate to quantify its

quality [MCRS19, GTHS13, OMP18]. The weight learning methods include Markov logic networks [KNKS11, RD06], path ranking [LMC11], relational dependency networks [NKK+10, NJ07], rule mining algorithms [MCRS19, GTHS13, GGM20, OMP18, CWG16], neural logic programming [YS20, YYC17, SADW19, RR17], neural theorem provers [RR17, MRS+20] and reinforcement learning [MCFS20, QCX+21, XHW17, DDZ+18, LSX18, SCH+18]. Specifically, AnyBURL [MCRS19] learns rules from paths of various lengths in a bottom-up manner. GPFL [GTHS13] is a probabilistic rule learner that optimizes AnyBURL by generalizing paths into templates, to reduce search space. Based on inductive logic programming, top-down rule learners have also been developed, such as AMIE [GTHS13] and ScaLeKB [CWG16], which repeatedly produce new rules at each level by specializing the ones derived in the upper level. RuDik [OMP18] discovers acyclic Horn rules by generating the universe of all possible rules, from which it selects rules according to a minimum weighted set cover of the given examples. RNNLogic [QCX+21] develops an EM-based method to train a rule generator, and Yang et al. [YYC17] have proposed a framework to learn rules through the differentiable model of Tensorlog [Coh16].

Existing levelwise discovery method presents two major challenges to tackle. First, rule discovery algorithms can hardly scale with large graphs due to exhaustive search. This prevents existing methods from discovering rules with large graph patterns. Second, a large number of rules typically hold on a given graph. It is hard for users to inspect the excessive number of discovered rules and identify useful ones to them.

Sampling has also long been studied to facilitate the discovery of association rules and frequent itemsets from relational data. For example, inspired by the VC-dimension theory [VC15] and Rademacher average [BBL05], bounds on the required sample size for finding approximate frequent itemsets have been established [RU14, RU15]. They ensure that for each itemset mined from the sample, the difference between its frequency in the entire dataset and the expected frequency threshold is within a user specified bound. Similarly, Chakaravarthy et al. have given a theoretical framework to analyse the impact of sample size on the quality of association rules mined from the samples [CPS09], *i.e.,* their support and confidence in the original relations. Sampling has also been adopted in the discovery of data cleaning rules from relational tables such as FDs [PN16] and (approximate) denial constraints (DCs) [BKN17, LHIK20]. In particular, Livshits et al. have shown how to estimate the number of violations of approximate DCs in sample data that is uniformly drawn from relations, by which it decides the right approximation threshold to use when discovering approximate DCs

from the sample data [LHIK20]. Their work guarantees that these rules also hold in the entire dataset with a high probability. When it comes to graphs, the majority of knowledge graph rule learners perform sampling to randomly extract paths from RDF and use the paths to generate restricted forms of Horn clauses, *e.g.,* [LMC15, MCRS19, GGM20, LMC11, MFW+18], with variants of random walks.

In order to accelerate discovery, parallel algorithms have been developed to mine rules from both relations, *e.g.,* [RK18, SGI19], and graphs, *e.g.,* [WdKdB+20, FHLL20, FWWX15, CGWJ16, CWG16]. However, none of these guarantees the parallel scalability except [FWWX15, FHLL20]. The parallelly scalable algorithms in [FWWX15, FHLL20] perform levelwise search on entire graphs. We extend the discovery algorithm of [FHLL20] in Chapter 3 to cope with sample graphs and ML predicates in mining GARs, without hampering its parallel scalability.

Event Prediction. Event prediction is to predict a real-world occurrence that relates to a particular topic and will take place at a specific time [Zha21]. Events range from large scale (*e.g.,* disease outbreaks and finance crisis), to medium-scale (*e.g.,* crime incidents and system failures), to small-scale (*e.g.,* authentication and fraud detection). Predicting events is important in a variety of domains such as disease control, transportation management, cybersecurity and business intelligence.

Most state-of-the-art event prediction methods work on temporal graphs (graphs with timestamps), since temporal graphs have been widely used to record facts and interactions between entities with temporal information. Temporal association rules have been studied for event prediction on temporal graphs [NWS+17, BBBG09a]. For example, GTARs (Graph Temporal Association Rules) [NWS+17] specify connections between events by means of two event patterns that share a common focus node and a single constant time interval. Similarly, GERs (Graph Evolution Rules) [BBBG09a] are defined with two connected (sub)-patterns that are decomposed from a singe pattern, to represent local changes such as edge insertions and deletions, and node and edge relabeling [SMD+16]. However, these rules have specific constraints for graph patterns and each rule can only express a constant interval between the occurrences of a pair of events, which are unsuitable for practical use.

A variety of ML models have also been explored for temporal event prediction, including (1) regression [MC17, NZV19, FJZL19, LDB18]; (2) point processes [QZX+18, WP13]; (3) survival analysis [DMS+17, VZT+19]; and (4) embedding via, *e.g.,* tensor decomposition [DKA11, YCA+17], recurrent neural network (RNN) [SDVB18,

TFBZ19, MGR+20, TDWS17, JQJR20], graph neural network (GNN) [LJL+21b, MRM20, CXWZ18, PDC+20, RCF+20], autoencoder [GKHL18, GCC20, RAH16] and diachronic encoder [GKBP20, XNA+19, DRT18, GDN18]. In particular, as a special type of event prediction, (temporal) recommendation has attracted a great deal of interest. For example, several dynamic recommender systems (DRS) have been developed to improve the recommendation quality. They can incorporate temporal factors of user preferences [RSDO20] using time-dependent neighborhood models [SD17, FL17, ML13], matrix factorization [Kor09, LLCL18, MVS+15, RN16] and tensor models [VJ12, WJW+18, HMB12]. Although they have demonstrated competitive accuracy, their predictions given by black-box models are all unexplainable.

Moreover, related to learning event prediction rules, mining temporal patterns and rules for Complex Event Processing (CEP) has also be extensively studied. CEP aims at processing large flows of events with timestamps to discover situations of interest, where patterns and rules are introduced to generalize experiences in historial sequential data and to predict the occurrence of future events [MCT14, EE11]. Specifically, a precise definition for the problem of automated CEP rules generation and a general approach for automatic CEP rule discovery have been proposed in [MCT14]. Representative CEP rule mining methods depend on Hidden Markov models [MP12, ZGPBM05], iterative learning with human experts [TGW09, SSS10], Markov logic networks [KD10], and temporal constraint networks [ÁFCO10].

Heterogeneous Data Management. Analysing heterogeneous data of different formats, *e.g.,* relations and graphs, from various sources has been a long-standing challenge, and multi-model systems have been proposed to address this issue, which support analyses on datasets in multiple data models. These multi-model systems can be categorized in the following three types: polyglot systems [DJL+16, JMCH15, Des18], multistores systems [KAB+12] and polystores systems [DES+15, KBV+16]. (1) To query graphs and relations, polyglot systems are either relation-based or graph-based. Relation-based ones shred graphs into edge relations/views and port graph computations to SQL-like systems (*e.g.,* GraphFrame [DJL+16], NScale [QD16]) or RDBMS (*e.g.,* Vertica [JMCH15], Vertexica [JRW+14], Cytosm [SAL+17], Grail [FRP15], and [ZY17]). Instead, graph-based ones (*e.g.,* GraphGen [Des18]) publish relations to graphs and use full-fledged graph engines. Polyglot systems aim to optimize queries on a homogeneous dataset (either graphs or relations), *i.e.,* each supports queries that involve only one type of the data. (2) Multistores are built on top of multiple data stores

such as RDBMS, Hadoop file system, cloud storage and NoSQL. They provide a single interface to query datasets in multiple native data stores [ABA+09, ZR11, KAB+12]. Query answering is based on a "mediate" data model such as relations [ABA+09, KAB+12] or an associative-array [ZR11]. (3) Polystores also interconnect storage engines of different data models; in addition, they employ native query languages of those models [DES+15, HdAC+14, KBV+16] instead of building a "mediator" as a multistore does. Inter-model queries are supported by composing native subqueries via, *e.g.,* function calls [KBV+16] or scope operations [DES+15].

However, these systems treat data of different formats as independent stores, and ignore the potential semantic connections between entities in the heterogeneous data. Without considering such semantic relatedness, systems such as multistores and polystores could only "syntactically" connect multiple datasets by *cross-product*. Moreover, this loss of semantic linkage may incur heavy cost when jointly querying relations and graphs. For example, relation-based polyglot systems require to traverse paths via costly SQL joins for attribute extraction.

Another filed of work related to this topic is Schema/data extraction (*a.k.a.* schema discovery and schema inference). These methods summarize semistructured data in terms of tree patterns and graph patterns with descriptions [CPF15, KMK15]. In addition, relational schemas have been extracted for navigating extracted data [CSE07], querying nested key-value data [DA16], developing schema [SAC+17] and integrating XML data with relations [JD14]. Related is also the work on attribute selection (*a.k.a.* feature selection in ML terms; see [HH03, SSGC17, CS14] for surveys). The idea is to filter out attributes from datasets that are irrelevant to the tasks over the data, mostly by ranking attributes using hand-crafted criteria [HH03, GFHK09] or by computing the closeness to a manually labeled training set [SSGC17, CS14]. The ML models target images, texts and tableaux data.

1.2 Contributions and Outline of Thesis

We summarize the contributions made in this thesis as follows:

- Chapter 2 proposes a class of graph association rules (GARs) that combine rules and machine learning models to capture missing links and inconsistencies in graphs.
  - We propose a class of graph association rules (GARs). GARs extend graph pattern association rules (GPARs) [FWWX15] with preconditions, and GFDs [FWX16b, FL19] with limited existential semantics. Moreover, GARs may

take embedding-based machine learning (ML) classifiers for link prediction as predicates, and thus provide a uniform framework to catch missing links and semantic errors in graphs, by *unifying rule-based and ML-based methods*.

- We study deducing associations from real-life graphs. We formalize the problem by extending the chase [SU80] with GARs, to uniformly apply rules and embedding-based ML classifiers. We show that the deduction has the Church-Rosser property, *i.e.,* the chase converges at the same answer no matter in which order the GARs are applied, even though the graphs may mutate.
- We study fundamental problems for graph associations, including (a) satisfiability to decide whether a set of GARs has no conflicts with each other; (b) implication to decide whether a GAR is entailed by a given set of GARs, to reduce redundant GARs; (c) association deduction to infer missing links and missing attributes in a real-life graph; and (d) incremental deduction to deduce changes to the associations in response to updates to graphs. We show that while GARs increase the expressive power of GFDs, the satisfiability, implication and association deduction problems retain *the same complexity* as their counterparts for GFDs. The incremental deduction problem for GARs is slightly harder than for GFDs, DP-complete vs. coNP-complete, unless P = NP. Thus, GARs strike a balance between the complexity and expressivity.
- For practical use of the association analysis, we parallelize the association deduction process by the fixpoint computation model of GRAPE [FYX+18]. We show that the parallelization guarantees to converge at the same set of associations deduced. Moreover, we provide a parallel incremental algorithm in response to updates, since it is costly to re-deduce associations starting from scratch when graphs change. We incrementally compute changes to associations, minimizing unnecessary recomputation.

◦ A parallelly scalable algorithm to discover GARs is developed in Chapter 3, applying an application-driven strategy to cut back rules and data that are irrelevant to users' interest, and a sampling method to reduce a big graph G for faster discovery.

- We formulate the discovery problem for GARs driven by applications and based on sampling, and present our three-step discovery scheme to discover useful rules from a big graph G for a given application $\mathcal{A}$: (a) reducing G to $\mathcal{A}$-relevant $G_{\mathcal{A}}$ that is related to an application of user's interest, (b) sampling a set H of graphs $H(\mathcal{A}, \rho\%)$ from $G_{\mathcal{A}}$, such that their sizes are at most $\rho\%$ of $G_{\mathcal{A}}$, (c) parallelizing discovery with the parallel scalability. We reduce irrele-

vant rules by proposing ML-based graph reduction (step (a)), and improve the scalability by combining steps (a), (b) and (c). Steps (a) and (b) reduce the problem of rule discovery from large G to much smaller $H(\mathcal{A}, \rho\%)$. We show that the scheme guarantees accuracy bounds. Indeed, all $\mathcal{A}$-relevant GARs that hold on G can be mined from $G_{\mathcal{A}}$ when the language model $\mathcal{M}_{\mathcal{A}}$ accurately labels the data. As opposed to prior methods [MCRS19, GGM20, CWG16] that only sample paths, step (b) samples general subgraphs by selecting representative data. These ensure accuracy bounds on the rules mined from $H(\mathcal{A}, \rho\%)$.

- A graph reduction method is developed to deduce $\mathcal{A}$-relevant graph $G_{\mathcal{A}}$ from graph G for a given application $\mathcal{A}$. It trains an ML model $\mathcal{M}_{\mathcal{A}}$ (long short-term memory (LSTM) networks [HS97]) to identify nodes, edges and properties in G that pertain to $\mathcal{A}$. We reduce G to a smaller graph $G_{\mathcal{A}}$ with only the data pertaining to $\mathcal{A}$, and discover *$\mathcal{A}$-relevant* rules from $G_{\mathcal{A}}$ instead of entire G.
- To reduce discovery cost, we propose a sampling method GSRD for reducing $G_{\mathcal{A}}$ to a set H of sample graphs $H(\mathcal{A}, \rho\%)$, such that their sizes are at most $\rho\%$ of $G_{\mathcal{A}}$. The samples consist of representative data cells in $G_{\mathcal{A}}$ along with their surrounding subgraphs. Denote by Σ_G and Σ_H the set of $\mathcal{A}$-relevant rules discovered from G and H, respectively. We show that given bounds σ and $\gamma\%$, we can deduce H such that (a) at least $\gamma\%$ of rules in Σ_G are covered by Σ_H, and (b) each of these rules can be applied at least σ times on the entire G, *i.e.,* the rules in Σ_G can be mined from H above recall $\gamma\%$ and support σ.
- We develop a parallel algorithm to discover GARs from the set H of samples $H(\mathcal{A}, \rho\%)$. We show that the algorithm guarantees the parallel scalability, *i.e.,* it guarantees to reduce runtime when more machines are used. In principle, it can scale with large graphs G by using more machines when needed.

◦ Chapter 4 introduces a class of temporal association rules, referred to as TACOs (TemporAl event prediCtiOn rules), that enrich event prediction ML models with temporal conditions and change patterns, establishes the complexity of reasoning about TACOs, and develops a system TASTE to discover TACOs and predict events in temporal graphs by employing TACOs.

- TACOs are defined on temporal graphs in terms of change patterns, temporal conditions and ML models (as predicates) for event prediction. In contrast to previous graph rules, TACOs are applied to *updates* to temporal graphs, which are typically much smaller than the entire graphs, to monitor changes and predict events.

- We study the classical problems for TACOs, including (a) satisfiability to check whether a set of TACOs has no conflicts, (b) implication to decide whether a set of TACOs entails another TACO, and (c) prediction to forecast whether an event will occur at a particular time. We show that these problems are Σ_2^p-complete, Π_2^p-complete and NP-complete, respectively. That is, despite the increased expressivity, TACOs do not make our lives much harder when applying and reasoning about TACOs compared with previous graph rules [FL19, FJL$^+$20].
- To make practical use of TACOs, we develop a system, referred to as TASTE (TemporAl SysTEm). TASTE (a) discovers high-quality TACOs with generative ML models, and (b) applies the discovered TACOs for temporal event prediction in parallel. We formalize the discovery problem for TACOs, in terms of support and confidence defined *w.r.t.* temporal pattern matching.
- We develop a *creator-critic* framework to discover TACOs in TASTE. The rule creator adopts GAN and LSTM language models to generate TACOs with candidate patterns and temporal conditions, while the critic evaluates such TACO rules on temporal graphs, collects high-quality rules, and provides feedback to the creator for improving the quality in the next round. Different from the traditional discriminator in GAN, the "critic" is a predefined scoring algorithm that requires no training. Thus, our "creator" is easier to train with weak supervision [Zho18] from the critic, while the dynamic training of GAN may converge to a poor generator or discriminator. This method avoids the exhaustive levelwise search in an exponentially large space [FHLL20], and is able to find TACOs with large patterns. We show that the method guarantees to find all TACOs after certain rounds with a probabilistic bound.
- We develop a parallel algorithm for temporal event prediction with TACOs. The algorithm is also used to support the critic module of rule discovery. In contrast to existing graph partitioning methods such as edge-cut or vertex-cut [RPGH14, AR06], we propose to partition a temporal graph based on temporal locality such that temporal pattern matching can be conducted locally at each fragment, and event prediction can be made communication-free. We show that the parallel event prediction algorithm guarantees to run faster when given more processors.

◦ Finally, based on entity matching across relations $\mathcal{D}$ and graphs G, Chapter 5 proposes an approach to querying $\mathcal{D}$ and G taken together in SQL, tackling the variety

challenge of big data.

– We propose a framework, referred to as RGAP (Relation GrAPh), to support SQL across a relational database $\mathcal{D}$ and a semistructured graph G. Assume an "oracle", referred to as HER for Heterogeneous Entity Resolution. Given a tuple t in $\mathcal{D}$ and a vertex v in G, HER checks whether t and v make a *match*, *i.e.,* they refer to the same real-world entity, despite their radically different topological structures. Then we can naturally "join" the same entities encoded by tuples in relations and by vertices in graphs, collect relevant attributes pertaining to the entities, and correlate their information. This is a simple semantic extension to the join operator in SQL.
– We extend relational algebra (RA) with syntactic sugar to support semantic joins across $\mathcal{D}$ and G, referred to as GRA (Graph Relational Algebra). Given a set S of tuples of a relation schema R, we deduce the following:
  - $f(S,G)$, the set of pairs (t,v) such that tuple $t \in S$ and v is vertex in G that matches tuple t, by invoking HER;
  - a relation schema R_G to augment R with additional attributes from G, *i.e.,* properties of the matching vertices in $f(S,G)$; and
  - an instance $h(S,G)$ of schema R_G, by value extraction from G.

  We join entities t and v as long as they make a match in $f(S,G)$, and complement tuples t with additional attributes of $h(S,G)$. More specifically, we support two different forms of join:
  - *static join*: when S is a set of input tuples in $\mathcal{D}$; and
  - *dynamic join*: when S is the intermediate result of a sub-query.

  We show how to efficiently implement the two forms of joins. While we can cache $f(S,G)$ and $h(S,G)$ in advance for static joins, for dynamic joins, naive implementation would require to compute S, $f(S,G)$ and $h(S,G)$ at run time on the fly, with a heavy cost. We show how to reduce dynamic joins to static joins for a large class of practical queries, making them accessible within RDBMS.
– To support semantic joins, one has to compute the match relation $f(S,G)$ and extract relations $h(S,G)$ for the matches from G. While $f(S,G)$ can be identified by HER [PMG+20, MLR+18, IJB10, FZMK20, FGJ+22], no method is available to deduce schema R_G and relation $h(S,G)$. This is nontrivial since we have to identify important properties by traversing paths from the match-

ing vertices in $f(S,G)$. We propose a clustering-based method to deduce R_G and $h(S,G)$. Taking a set Ω of keywords from users as input, we rank and pick attributes that cover most vertex-path pairs in G, meet users' query interests (represented by Ω), and diverse from existing attributes of R.

– For those queries in which dynamic joins cannot be reduced to static joins, we propose a *heuristic join* method to reduce the cost. To do this, for selected types τ of entities in G, we adjust the clustering method to deduce
  - a schema R_τ with attributes that cover important properties of τ-entities in G and match users' interests; and
  - an instance $g_\tau(G)$ of schema R_t extracted from G.

  With the extracted relations $g_\tau(G)$, given a dynamic join between intermediate query results S and graph G, we reduce it to relational ER between S and $g_\tau(G)$, which can be realized in RDBMS with a simple UDF. By caching $g_\tau(G)$ in advance, this strikes a balance between the efficiency and accuracy of dynamic joins, without calling external HER and extraction functions at run time. We also develop incremental algorithms to extract $h(S,G)$ and $g_\tau(G)$ in response to updates to relations $\mathcal{D}$ and graphs G.

Remark. It is worth mentioning that results in Chapter 2 have been published in [FJL$^+$20], the results in Chapters 3 and 4 have been accepted in VLDB 2022, and the results in Chapter 5 are taken from a submitted paper under review. My major contributions to each paper covered in this thesis are listed as follows. In Chapter 2, I was in charge of designing ML predicates for GARs, performing case studies and experimental evaluation. In Chapter 3, I developed the application-driven graph reduction and sampling strategy to discover rules that meet users' interests, and conducted the corresponding experiments. In Chapter 4, I proposed the critic-creator framework, which to the best of our knowledge is the first to introduce generative ML models and weak supervision into graph rule discovery. I also participated in the development of TACOs, implementation of the TASTE system, and experimental studies. In Chapter 5, inspired by clustering techniques, I developed the attribute and schema extraction methods to support RGAP, and carried out the experiments.

Chapter 2

Capturing Associations in Graphs

This Chapter proposes a class of graph association rules, denoted by GARs, to specify regularities between entities in graphs, which catch incomplete information in schemaless graphs, predict links in social graphs, identify potential customers in digital marketing, and extend graph functional dependencies (GFDs) to capture both missing links and inconsistencies. We start with examples to motivate the introduction of GARs.

Example 2.1: Consider the following real-life examples.

(1) Marketing. Unlike traditional marketing strategies such as TV advertising, e-commerce marketing promotes products by association analysis of purchases and user behaviors, which are often represented as graphs. It has proven important: "the visits where the shopper clicked a recommendation comprise just 7% of visits, but drive an astounding 24% of orders and 26% of revenue" [You17]. Moreover, associations play a vital role in recommendation systems [AT05, DLL$^+$10, LAR02].

For instance, graph G_1 of Fig. 2.1 depicts an e-commerce recommendation network [ZZY$^+$19]. A rule for marketing is as follows: if (a) a shopper Ada follows a store Uniqlo and clicks product Long-Sleeve Hoodie W sold by it, (b) Uniqlo also sells Denim Mini Skirt, which is combined with Long-Sleeve Hoodie W in some package deals, and (c) if the classes of the two products, Hoodie W and Mini Skirt, are correlated in women's shopping activities, then Ada may also be interested in Denim Mini Skirt. The link showing Ada's interest in Denim Mini Skirt is not in G_1.

(2) Link prediction. Association rules help us predict links in social networks. People visiting the same place, having common friends and similar interests tend to develop friendship [SNLM11, YLS$^+$11]. For example, graph G_2 in Fig. 2.1 is taken from social network Gowalla [LWSM14]. It suggests the following: if (a) two people Bob and

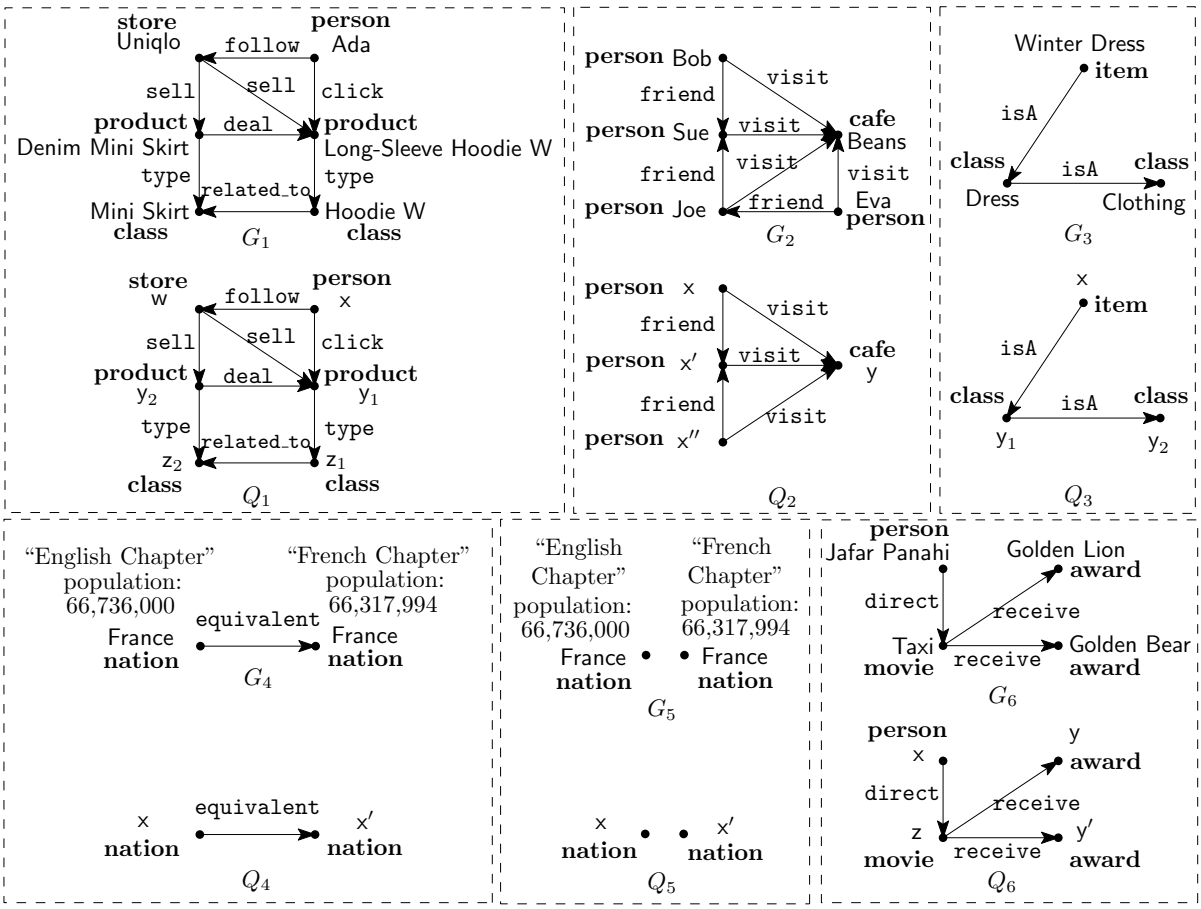


Figure 2.1: Example associations in real-life graphs

Joe have a common friend Sue, (b) all of them like to visit cafe Beans, and (c) if Bob and Joe share the same interest as Sue, then Bob and Joe are likely to become friends. The link between Bob and Joe is absent in G_2.

(3) Incomplete information. Unlike relational tables, real-life graphs typically do not come with a schema. As a result, it is more common to find information missing from graphs. As indicated by G_3 of Fig. 2.1, in the knowledge graph adopted by e-commerce platforms [LLB+20], there exist clothing items (*e.g.,* Winter Dress) without brand or material. To make them complete items, the missing data need to be added.

(4) Catching both absent links and semantic errors. Graph functional dependencies (GFDs) have been studied [FWX16b, FL19]. Like relational functional dependencies (FDs), GFDs are universal logic sentences to catch semantic inconsistencies. However, GFDs stop short of catching missing links, which have an existential semantics.

On the one hand, GFDs may fail to catch semantic errors when links are missing. Consider graph G_4 taken from DBPedia [LIJ+15], in which the English and French Chapters return different populations for France. One can use a GFD to catch the

inconsistency: if two records x and x' refer to the same nation, then they must have the same population. However, if the equivalent link between x and x' is missing, then this GFD cannot catch the error. On the other hand, with such inconsistencies, conventional logical rules fail to connect the nation records in G_5 of Fig. 2.1. The scale of the problem is far more staggering in schemaless graphs.

(5) Incorporating machine learning (ML). Association deduction requires both logic-based and ML-based methods. On the one hand, we can use ML classifiers to predict links above between x and x'. On the other hand, we can use logic to interpret ML predictions and help improve its accuracy. For instance, if an ML classifier "predicts" that movie Taxi receives Golden Bear and Golden Lion awards (see G_6 of Fig. 2.1), then we can conclude that the predication is wrong since the two film festivals require their participants to be "initial release" and no movie receives both awards. □

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

We make an effort to answer these questions, from foundation to practice. We propose a class of graph association rules in Section 2.1, denoted by GARs, extending graph pattern association rules (GPARs) [FWWX15] with preconditions, and GFDs [FWX16b, FL19] with limited existential semantics. Then we study deducing associations from real-life graphs in Section 2.2, formalize the problem by extending the chase [SU80] with GARs, to uniformly apply rules and embedding-based ML classifiers. Fundamental problems for graph associations are studied in Section 2.3, and parallel association deduction process is given in Section 2.4 for practical use. We experimentally verify our methods in Section 2.5, and conclude our novelty and contributions compared with previous work in Section 2.6.

2.1 Graph Association Rules

We firstly review basic notations before introducing GARs. We assume three countably infinite sets of symbols, denoted by Γ, Υ and U, for labels, attributes and constants,

respectively.

Graphs. We consider directed labeled graphs, specified as $G = (V, E, L, F_A)$, where (a) V is a finite set of nodes; (b) $E \subseteq V \times \Gamma \times V$ is the set of edges, where $e = (v, \iota, v')$ denotes an edge from node v to v' that is labeled with $\iota \in \Gamma$; (c) each node $v \in V$ has label $L(v)$ from Γ; and (d) each node $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity, where $A_i \in \Upsilon$ and $a_i \in U$, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$, representing properties.

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

Example 2.2: Six patterns are given in Fig. 2.1. For example, pattern Q_1 shows that shop w sells products y_1 and y_2 of classes z_1 and z_2, respectively, y_1 and y_2 are linked in a special offer, z_1 and z_2 are related in order activities, and customer x follows shop w and clicks product z_1. Patterns Q_2-Q_6 in Fig. 2.1 can be interpreted similarly. □

Pattern matching. We adopt the homomorphism semantics following [FL19, ACP10, CP12]. A *match* of pattern $Q[\bar{x}]$ in graph G is a mapping h from Q to G such that (a) for each node $u \in V_Q$, $L_Q(u) = L(h(u))$; and (b) for each $e = (u, \iota, u')$ in Q, $e' = (h(u), \iota, h(u'))$ is an edge in G. Here $L_Q(u) = L(h(u))$ if $L_Q(u)$ is '_', *i.e.,* wildcard matches an arbitrary label. We denote the match as a vector $h(\bar{x})$, consisting of $h(x)$ for all $x \in \bar{x}$ in the same order as $\bar{x}$. Intuitively, $\bar{x}$ is a list of entities to be identified, and $h(\bar{x})$ is an instantiation for it.

We now define graph association rules (GARs).

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

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

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

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

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

Intuitively, the pattern Q in a GAR identifies entities in a graph, and the dependency $X \to Y$ is applied to the entities. Constant and variable literals $x.A = c$ and $x.A = y.B$ specify *value associations to* attributes, and attribute and edge literals $x.A$ and $\iota(x,y)$ enforce the existence of attributes and edges, *i.e., attribute and edge associations*, respectively. Moreover, one can "plug in" an existing well-trained ML classifier $\mathcal{M}$ for link prediction, and treat it as a Boolean predicate, *i.e.,* $\mathcal{M}(x,y,\iota)$ is true if $\mathcal{M}$ predicts the existence of a link labeled ι from x to y, and false otherwise. As will be seen shortly, it allows us to employ embedding-based ML classifiers in logic rules, and interpret such classifiers in logic.

Example 2.3: One can use the GARs below to deduce associations described in Example 2.1, using patterns Q_1-Q_6 of Fig. 2.1.

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

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

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

(4) $\varphi_4 = Q_4[x,x'](\emptyset \to Y_4)$, where Y_4 is $x.\mathsf{population} = x'.\mathsf{population}$, and population is an attribute of a nation entity. It says that records about the same nation should have the same population. It is a GFD [FWX16b] to catch inconsistencies.

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

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

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

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

For a set X of literals, we write $h(\bar{x}) \models X$ if match $h(\bar{x})$ satisfies *all* the literals in X. If X (resp. Y) is $\emptyset$ (*i.e.,* true), then $h(\bar{x}) \models X$ (resp. $h(\bar{x}) \models Y$) for any match $h(\bar{x})$ of Q in G. We write $h(\bar{x}) \models X \rightarrow Y$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models Y$. A graph G *satisfies* GAR φ, denoted by $G \models \varphi$, if for all matches $h(\bar{x})$ of Q in G, $h(\bar{x}) \models X \rightarrow Y$. Graph G *satisfies* a set Σ of GARs, denoted by $G \models \Sigma$, if $G \models \varphi$ for all $\varphi \in \Sigma$.

Example 2.4: Consider G_2 in Fig. 2.1 and GAR φ_2 in Example 2.3. Then $G_2 \not\models \varphi_2$, since there exists a match h_1: $x \mapsto$"Bob", $x' \mapsto$"Sue", $x'' \mapsto$ "Joe", $y \mapsto$"Beans", such that $h_1(\bar{x}) \models X_2$, but there exists no edge ("Bob", friend, "Joe") in G_2. Hence $h_1(\bar{x}) \not\models X_2 \rightarrow Y_2$, *i.e.,* $h_1(\bar{x})$ witnesses $G_2 \not\models \varphi_2$. Similarly, $G_i \not\models \varphi_i$ for other $i \in [1,6]$. □

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

(1) GFDs and graph entity dependencies (GEDs) [FWX16b, FL19] are GARs defined with constant and variable literals only, assuming that node id is a special attribute. That is, GARs extend GFDs and GEDs with the existential semantics for attributes and

edges, and by allowing ML classifiers as predicates. For instance, φ_4 of Example 2.3 is a GFD but all the other GARs there cannot be expressed as GFDs or GEDs. GARs can catch both missing links and semantic errors, as opposed to GFDs and GEDs that detect inconsistencies only.

(2) GPARs [FWWX15] are GARs $Q[\bar{x}](\emptyset \rightarrow \iota(x,y))$, in which $X \rightarrow Y$ specifies no precondition X and Y is a single edge literal $\iota(x,y)$. In contrast to GARs, GPARs do not allow ML classifiers. No GAR in Example 2.3 is expressible as GPARs.

(3) GARs unify logic and ML methods. On the one hand, $Q[\bar{x}](\mathcal{M}(x,y,\iota) \rightarrow \iota(x,y))$ plugs in an ML link predictor $\mathcal{M}(x,y,\iota)$, *e.g.,* GAR φ_5 of Example 2.3. On the other hand, GARs $Q[\bar{x}](\psi \rightarrow \mathcal{M}(x,y,\iota))$ help us interpret why $\mathcal{M}(x,y,\iota)$ predicts true with condition ψ. For instance, the ML classifier $\mathcal{M}$ in φ_6 may be interpreted as a rule like $Q_6[\bar{x}](z.\mathsf{name} = y'.\mathsf{movie\_name} \wedge z.\mathsf{director} = y'.\mathsf{movie\_director} \rightarrow \mathcal{M}(z,y',\mathsf{receive}))$, by extracting the attributes from the textual description of movies and awards.

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

Once the training completes, such an ML classifier $\mathcal{M}$ behaves just like a Boolean function. Given two entities h', t' and a relation r', $\mathcal{M}(h',r',t')$ returns a Boolean value. That is, $\mathcal{M}$ maps h', t' and r' to precomputed vectors $v_{h'}$, $v_{t'}$ and $v_{r'}$ as their embeddings. Then it feeds these vectors to the similarity function, and returns true (resp. false) if the score is above (resp. below) the threshold. The hypothesis of such ML link predictor is that all entities and relations have been covered by the training data and learned by $\mathcal{M}$ [BUGD$^+$13]; thus $\mathcal{M}$ can find embeddings of h', t' and r', and predict whether h' is linked to t' with an r'-edge. That is how the state-of-the-art embedding-based link prediction models SimpIE [KP18] and CompIEx [TWR$^+$16] work.

2.2 Deducing Associations

One of the central issues of the study is to deduce associations. There are two types of associations: (a) associations between entities (edge literals) and associations of attributes to entities (attribute literals); and (b) associations of values to attributes (variable and constant literals). We model association deduction by chasing graphs with GARs. Below we first extend the chase [SU80] from relations to graphs (Section 2.2.1) and then prove its Church-Rosser property (Section 2.2.2). Based on these, we will formulate the association deduction problem in Section 2.3.

2.2.1 Chasing with GARs

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

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

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

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

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

- If l is $x.A$, then G_c' extends G_c by adding attribute A to $\Delta F_{A_c}(h(x))$ with a special value "#" if $A \notin F_A(h(x))$. Here, "#" acts as a placeholder for the value of A, since the rule only enforce the existence of an attribute A without its specific value. This placeholder guarantees the data consistency as each attribute must have a "value". "#" can be implemented as "null" value in practice. If attribute $A = \#$ is later assinged a constant a by a GAR, then $A = a$, which incurs no conflicts. If A is already in $F_A(h(x))$, its value remains unchanged.
- If l is $\iota(x, y)$, then G_c' extends G_c with edge $(h(x), \iota, h(y))$.
- If l is $\mathcal{M}(x, y, \iota)$, then G_c' extends G_c by adding edge $(h(x), \iota, h(y))$. As a byproduct, it suggests to set $\mathcal{M}(x, y, \iota)$ true, *i.e.,* it provides feedback to ML predictor $\mathcal{M}$.
- If l is $x.A = y.B$, then G_c' extends G_c by (a) adding attributes A to $\Delta F_{A_c}(h(x))$ and B to $\Delta F_{A_c}(h(y))$ if the attributes are not there, and (b) letting $h(x).A = h(y).B$.

- If l is $x.A = c$, then G_c' extends G_c by adding attribute A to $\Delta F_{A_c}(h(x))$ if $A \notin F_A(h(x))$, and letting $h(x).A = c$.

Consistency. Conflicts may emerge in a chase step. We say that chase step $G_c \Rightarrow_{(\varphi,h)} G_c'$ is *invalid* if when it enforces literal l, either (a) l is $x.A = y.B$, but $h(x).A = c$ and $h(y).B = d$ are in G_c for distinct c and d, or (b) l is $x.A = c$, $h(x).A = d$ is in G_c and $c \neq d$. Otherwise the step is *valid*. We say that G_c' is *inconsistent* if either (a) or (b) happens. Note that edge and ML literals do not incur inconsistencies as multiple edges can co-exist between a pair of nodes.

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

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

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

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

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

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

Prior work on chasing graphs [FL19, FLTZ19] mainly changes attribute values. In contrast, *the topological structure* of G_c may be changed by new edges and attributes added when chasing GARs. Hence when G_c is extended to G_c', we have to check new possible matches of graph patterns in GARs.

Example 2.5: Consider the graph G_2 shown in Fig. 2.1. Assume that Σ consists of only one GAR φ_2 in Example 2.3. From $G_{c_0} = G_2$, we have the following chase steps:

(1) $G_{c_0} \Rightarrow_{(\varphi_2,h_1)} G_{c_1}$, where match h_1 is given in Example 2.4; and G_{c_1} extends G_{c_0} with edge ("Bob", friend, "Joe");

(2) $G_{c_1} \Rightarrow_{(\varphi_2,h_2)} G_{c_2}$, where h_2 is defined as follows: $x \mapsto$ "Bob", $x' \mapsto$ "Joe", $x'' \mapsto$ "Eva", $y \mapsto$ "Beans", and G_{c_2} extends G_{c_1} with edge ("Bob", friend, "Eva") using

φ_2. Note that match h_2 exists in the mutated chase graph G_{c_1} *only after* the edge ("Bob",friend,"Joe") is added in step (1). □

2.2.2 The Church Rosser Property

A major concern is whether the chase always terminates with the same result. Following [AHV95], we say that chasing with GARs is *Church-Rosser* if for all graphs G and all sets Σ of GARs, all chasing sequences of G by Σ are terminal and converge at the same result, regardless of what GARs in Σ are used and in what order they are applied.

Theorem 2.1: *Chasing with* GARs *is Church-Rosser.* □

Proof: We show the following: (1) any chasing sequence is finite and consists of at most $4|G|^2|\Sigma|$ steps, and (2) all chasing sequences terminate at the same result. A similar proof was given in [FL19] for GEDs, an extension of GFDs with vertex id equality.

(1) Any chasing sequence is finite. Given any terminal chasing sequence $\rho = (G_{c_0}, \ldots, G_{c_k})$ of graph G by a set Σ of GARs, we can verify that $k \leq 4|G|^2|\Sigma|$ as follows. Observe that a chase step does one of the following: (a) at most one attribute $x.A$ or one edge (x, ι, y) is added to G; (b) one attribute is assigned a constant; or (c) two attributes are set equal. Note that although there may exist multiple edges between any pair of nodes, the labels of new edges are constrained by the GARs in Σ, and hence at most $|\Sigma|$ edges can be added to G, *i.e.,* G_{c_0}. Then, we have that $k \leq 4|G|^2|\Sigma|$ and hence ρ is finite.

(2) All chasing sequences terminate at the same result. We show this by contradiction. Assume that there exist two terminal chasing sequences $\rho_1 = (G_{c_0}, G_{c_1}, \ldots, G_{c_k})$ and $\rho_2 = (G'_{c_0}, G'_{c_1}, \ldots, G'_{c_l})$ of G by Σ with different results, where $G_{c_0} = G'_{c_0}$. Because ρ_1 and ρ_2 have different results, we know that G_{c_0} is consistent and at least one of ρ_1 and ρ_2 is valid. Assume *w.l.o.g.* that ρ_1 is valid and the chase graph G_{c_k} is consistent. By analyzing the difference between ρ_1 and ρ_2, we show that ρ_1 is not terminal, a contradiction.

More specifically, since ρ_1 and ρ_2 have different results, there exist a literal l' of GAR φ_j and a chase step $G'_{c_j} \Rightarrow_{(\varphi_j,h)} G'_{c_{j+1}}$ in ρ_2 such that $G'_{c_{j+1}}$ extends G'_{c_j} *w.r.t.* the instantiation $h(l')$; and $h(l')$ does not hold in G_{c_k} of ρ_1. Here the instantiation $h(l')$ replaces each variable x in l' by $h(x)$. However, one can verify that $G_{c_k} \Rightarrow_{(\varphi_j,h)} G_{c_{k+1}}$ is a valid chase step expanding G_{c_k} *w.r.t.* $h(l')$, which contradicts the assumption that

sequence ρ_1 is terminal. Note that $G_{c_{k+1}}$ is the chase graph obtained from G_{c_k} and $h(l')$.

To show that $G_{c_k} \Rightarrow_{(\varphi_j,h)} G_{c_{k+1}}$ is a chase step, we prove the following properties by induction on the length of ρ_2: (a) all attributes and edges in G'_{c_i} $(i \in [0,l])$ are also in G_{c_k}; (b) if the prediction of the ML model $\mathcal{M}$ in G'_{c_i} is true, then the prediction of $\mathcal{M}$ is also true in G_{c_k}. If these hold, as $h(l')$ does not hold in G_{c_k}, and h is a match of the pattern of φ_j in G_{c_k}, we know that $G_{c_k} \Rightarrow_{(\varphi_j,h)} G_{c_{k+1}}$ is a chase step. Note that as opposed to the chase with GEDs [FL19], here we have to show that the prediction of ML classifier remains stable during the chase; to this end, we need to exploit the property of ML classifiers given in Section 2.1 (*i.e.,* all entities and relations in the graph are covered by the training data, and the embedding vectors of entities and relations will not change after the graph is extended), which justifies our choice of ML classifiers.

Basic case. At first, we consider the case when $i = 0$. Since ρ_2 starts with $G'_{c_0} = G_{c_0}$, and the prediction of $\mathcal{M}$ remains unchanged after training, properties (a) and (b) follow.

Inductive step. Assume that the properties hold for G'_{c_i} $(i \leq j)$. We next show that the properties also hold for $G'_{c_{j+1}}$. Suppose that the $(j+1)$-th step of ρ_2 is $G'_{c_j} \Rightarrow_{(\varphi,h)} G'_{c_{j+1}}$, where $\varphi = Q[\bar{x}](X \rightarrow Y)$ is a GAR in Σ, h is a match of Q in G'_{c_j} such that $h(\bar{x}) \models X$, l is a literal in Y, and $h(l)$ does not hold in G'_{c_j}. By the inductive hypothesis, we know that h is also a match of Q in G_{c_k} such that $h(\bar{x}) \models X$. Then (a) the attributes and edges in $G'_{c_{j+1}}$ must also be in G_{c_k}, since otherwise from the fact that $h(\bar{x}) \models X$ in G_{c_k} we can apply φ to further extend G_{c_k}, which contradicts the assumption that ρ_1 is terminal. Moreover, (b) the values returned by the ML model $\mathcal{M}$ in G_{c_k} are the same as those obtained in $G'_{c_{j+1}}$, since $\mathcal{M}$ behaves like a Boolean function after training and all embedding vectors are stable. □

By Theorem 2.1, we define *the result of chasing G by* Σ as the result of any terminal chasing sequence of G by Σ, denoted by $\mathsf{Chase}(G,\Sigma)$. If the sequence is valid, $\mathsf{Chase}(G,\Sigma)$ has the form of G_c. We refer to edges and attributes that are in G_c but not in G as *deduced associations* of G by Σ. Intuitively, they are missing links and attributes. We denote by $\mathsf{deduced}(G,\Sigma)$ the set of all such deduced associations.

As shown in Section 2.1, we can use deduced associations to retrain $\mathcal{M}$, improve its accuracy and explain its prediction.

2.3 Fundamental Problems

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

Satisfiability. The *satisfiability* problem is as follows.

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

Intuitively, this is to ensure that Σ is sensible and all GARs can be simultaneously applied without conflicts. For GFDs, the satisfiability problem is coNP-complete [FL19]. We next show that this problem is no harder for GARs.

Theorem 2.2: *The satisfiability problem is* coNP*-complete.* □

Proof: We only need to show that the satisfiability problem is in coNP, since GFDs are a special case of GARs, and the satisfiability problem for GFDs is already coNP-hard [FL19]. To this end, we first establish a characterization for the satisfiability problem. We then develop an NP algorithm to check whether a set Σ of GARs is not satisfiable.

Characterization. Based on the chase, we establish the following characterization for the satisfiability problem.

Lemma 2.1: *A set* Σ *of* GARs *is satisfiable if and only if* $\mathsf{Chase}(G_\Sigma, \Sigma)$ *is consistent, where* G_Σ *is defined as the disjoint union of patterns in* Σ *without any attributes, referred to as the* canonical graph *of* Σ*.* □

We verify Lemma 2.1 as follows.

($\Leftarrow$) Assume that $G_{c_k} = \mathsf{Chase}(G_\Sigma, \Sigma)$ and the chase graph G_{c_k} is consistent. In the following, we construct a graph G satisfying Σ from G_{c_k}. Note that G_{c_k} may not be a well-defined graph yet, since its nodes and edges may carry wildcards as labels. To construct graph G, we need to instantiate such wildcards with some labels in Γ.

However, we cannot simply replace wildcards by distinct labels that do not appear in G_{c_k} as in [FL19], since it may trigger more chase steps, and lead to conflict. As

a simple example, consider $\Sigma = \{\varphi_1, \varphi_2\}$, where $\varphi_1 = Q[x,y](\emptyset \to x.A = 1)$, $\varphi_2 = Q[x,y](\mathcal{M}(x,y,\iota) \to x.A = 2)$, and $Q[x,y]$ is a pattern with two isolated nodes x and y labeled wildcards. The proof of [FL19] transforms G_Σ to a graph by instantiating the labels of x and y with, say, a and b, respectively. However, the chances are that after training, $\mathcal{M}(v,v',\iota) = \mathsf{true}$ for any two nodes labeled a and b, respectively. Then we end up with a graph $G \not\models \Sigma$. That is, the proof of [FL19] fails to build a small model of Σ due to the presence of ML model $\mathcal{M}$. Hence for GARs we have to take special care to avoid conflicts introduced by the prediction of $\mathcal{M}$. Moreover, the additions of new edges introduced during the chase also complicates the consistency analysis of $\mathsf{Chase}(G_\Sigma, \Sigma)$. Note that none of these problems was encountered when dealing with GEDs [FL19].

To resolve possible conflicts introduced by ML prediction, we can construct G from G_{c_k} by replacing wildcards with distinct new labels that are outside the training set and thus are not seen by $\mathcal{M}$ during training. As the embedding-based ML model $\mathcal{M}$ is fixed after training and only labels in the training set can be embedded as vectors, $\mathcal{M}$ actually cannot give predictions when facing those new labels outside the training set. Hence, we enforce $\mathcal{M}$ to predict false when it takes unseen labels as inputs. Given this, it is easy to verify that every pattern in Σ has a match in G by the definition of G_Σ. It remains to show that $G \models \Sigma$. To this end, we show that if $G \not\models \Sigma$, then $G_{c_k} \not\models \Sigma$, which contradicts Theorem 2.1 and the semantics of GARs. It suffices to show the followings: ($\dagger$) for any GAR $\varphi = Q[\bar{x}](X \to Y)$ in Σ, any literal l in X or Y, and any match h of Q in G, we have that (1) h is also a match of Q in G_{c_k}, (2) if $h \models l$ in G, then $h \models l$ also holds in G_{c_k} and (3) if $h \not\models l$ in G, then $h \not\models l$ in G_{c_k}. If these hold, when $G \not\models \Sigma$ we can find a GAR $\varphi = Q[\bar{x}](X \to Y)$ and a match h of Q in G_{c_k} such that $G_{c_k} \not\models \Sigma$, a contradiction.

We next show the properties above. For (1), since only wildcards in Q can match wildcards in G_{c_k} and distinct new labels in G, it is easy to verify that h is a match of Q in G_{c_k}. For (2) and (3), if l is $x.A$, $\iota(x,y)$, $x.A = c$, $x.A = y.B$, or ML literal $\mathcal{M}(x,y,\iota)$ when neither x nor y is labeled with wildcard, then the statement holds since G and G_{c_k} only differ in those nodes or edges that are labeled with wildcard. When l is an ML literal $\mathcal{M}(x,y,\iota)$ and one of x and y is labeled wildcard, then $\mathcal{M}(x,y,\iota) = \mathsf{false}$ in G_{c_k} since $\mathcal{M}$ is not trained with nodes labeled wildcards; meanwhile since we replace wildcards with distinct new labels that are not used in the training of $\mathcal{M}$, we also have that $\mathcal{M}(x,y,\iota) = \mathsf{false}$ in G. Hence properties (2) and (3) follow.

($\Rightarrow$) Conversely, assume that Σ is satisfiable. We next show that for any terminal chasing sequence $\rho = (G_{c_0}, \ldots, G_{c_k})$ of G_Σ by Σ, the result G_{c_k} is consistent.

(1) To prove this, we first identify a property of ρ. When Σ is satisfiable, there exists a graph $G = (V, E, L, F_A)$ as shown in the proof above such that each pattern Q of GAR φ in Σ has a match h_φ in G. Then we define a mapping h from graph G_Σ to G by combining all such h_φ. We can show that for each chase step $G_{c_i} \Rightarrow_{(\varphi_i, h_i)} G_{c_{i+1}}$ ($i \in [0, k-1]$) in ρ that extends G_{c_i} *w.r.t.* the instantiation of a literal l, $h(\bar{x}) \models l$ holds in G. This can be proved by induction on chase steps. Since $G \models \Sigma$, we can inductively include all instantiations in G_{c_k} using h [FL19].

(2) Using the property above, we can show that G_{c_k} is consistent. Assume by contradiction that G_{c_k} is not consistent. Then there exist a GAR $\varphi = Q[\bar{x}](X \to Y)$ in Σ and a match h' of Q in G_Σ such that $G_{c_{k-1}} \Rightarrow_{(\varphi, h')} G_{c_k}$ and G_{c_k} is inconsistent. However, based on the property proven in (1), one can verify that all attribute values of G_{c_k} are also in G. Then G is also inconsistent, a contradiction.

Upper bound. We now develop an NP algorithm to decide, given a set Σ of GARs, whether Σ is not satisfiable, as follows.

(1) Construct the canonical graph G_Σ, and guess a sequence of steps $G_{c_0} \Rightarrow_{(\varphi_1, h_1)} G_{c_1} \Rightarrow \cdots \Rightarrow G_{c_{k-1}} \Rightarrow_{(\varphi_k, h_k)} G_{c_k}$ of $G_\Sigma = G_{c_0}$ by Σ such that $k \leq 4|G|^2|\Sigma|$.
(2) For each $i \in [1, k]$, check whether $G_{c_{i-1}} \Rightarrow_{(\varphi_i, h_i)} G_{c_i}$ is a chase step; if not, reject the guess; otherwise, continue.
(3) For each $i \in [1, k]$, check whether $G_{c_{i-1}} \Rightarrow_{(\varphi_i, h_i)} G_{c_i}$ is invalid; if any of these is invalid, return true.

The correctness of the algorithm follows from Lemma 2.1. For the complexity, step (1) is in PTIME by the definition of canonical graphs; steps (2) and (3) are in PTIME by the fact that $|G_{c_{i-1}}| \leq 5|G|^2|\Sigma|$ and $|G_{c_i}| \leq 5|G|^2|\Sigma|$. Thus the algorithm is in NP, and the satisfiability problem is in coNP. □

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

The *implication problem* is stated as follows.

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

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

The good news is that the implication analysis of GARs has the same complexity as its counterpart for GFDs [FL19], as opposed to TGDs [AHV95]. This is because (1) chasing with GARs does not generate new nodes; (2) while GARs enforce the existence of edges and attributes, the new additions are confined to those specified in GARs only. Taken together, these ensure that chasing with GARs will end up with a finite graph. In contrast, chasing with TGDs [AHV95, FLTZ19, CP12] may lead to infinite graphs and hence may not terminate.

Theorem 2.3: *The implication problem is* NP*-complete.* □

Proof: Similar to the proof of the satisfiability problem, we only need to show that the implication problem for GARs is in NP, since the implication problem for GFDs is NP-hard [FL19]. To this end, we first establish a characterization for the implication problem for GARs, and then provide an NP algorithm for it.

Lemma 2.2: *For a set* Σ *of* GARs *and a* GAR $\varphi{=}Q[\bar{x}](X{\rightarrow}Y)$, $\Sigma{\models}\varphi$ *if and only if either* X *is inconsistent, or all literals in* Y *can be inferred from* $\mathsf{Chase}(G_Q,\Sigma)$, i.e., *all instantiations of literals from* Y w.r.t. *the one-to-one mapping between* Q *and* G_Q *hold in* $\mathsf{Chase}(G_Q,\Sigma)$. *Here* G_Q *denotes the canonical graph of the* GAR φ *extended with literals in* X. □

Proof of Lemma 2.2. We show the correctness of Lemma 2.2.

($\Leftarrow$) Assume that X is inconsistent or all literals in Y can be inferred from the result $\mathsf{Chase}(G_Q,\Sigma)$. Consider the following two cases: (a) $\mathsf{Chase}(G_Q,\Sigma)$ is inconsistent; and (b) $\mathsf{Chase}(G_Q,\Sigma)$ is consistent.

Case (a). We have that X is not consistent, or for any graph G such that Q has a match h in G satisfying $h(\bar{x}) \models X$, $G \not\models \Sigma$ holds. This can be verified along the same lines as the proof of Lemma 2.1 given above. Then $\Sigma \models \varphi$ follows.

Case (b). Let $G_{c_k} = \mathsf{Chase}(G_Q,\Sigma)$. Then we know that for any graph G such that $G \models \Sigma$ and for any match h of Q in G with $h(\bar{x}) \models X$, $h(\bar{x}) \models Y$, *i.e.,* the attribute values in $h(\bar{x})$ satisfy all literals in Y; this is because all literals of Y can be inferred from G_{c_k}. Thus $\Sigma \models \varphi$.

($\Rightarrow$) Suppose that $\Sigma \models \varphi$. Let $G_{c_k} = \mathsf{Chase}(G_Q,\Sigma)$. Consider the two cases above. (a) When $\mathsf{Chase}(G_Q,\Sigma)$ is inconsistent, we can show that X is inconsistent or all literals in Y can be inferred from G_{c_k} since G_{c_k} is inconsistent. (b) When $\mathsf{Chase}(G_Q,\Sigma)$ is

consistent, assume by contradiction that there exists a literal l in Y such that l cannot be inferred from G_{c_k}. Using l and G_{c_k} we can construct a graph G such that $G \models \Sigma$ but $G \not\models \varphi$. That is, $\Sigma \not\models \varphi$, a contradiction. The construction of G is similar to the one given in the proof of Theorem 2.2, *i.e.,* the wildcards are replaced by distinct new labels that are outside the training set of $\mathcal{M}$.

Algorithm. Based on Lemma 2.2, we give an NP algorithm for the implication problem. Given a set Σ of GARs and GAR φ, it checks whether $\Sigma \models \varphi$ as follows.

(1) Construct the canonical graph G_Q and guess a sequence of steps $G_Q{=}G_{c_0} \Rightarrow_{(\varphi_1,h_1)} G_{c_1} \Rightarrow \cdots G_{c_{k-1}} \Rightarrow_{(\varphi_k,h_k)} G_{c_k}$ of G_Q by Σ such that $k \leq 4|\Sigma||\varphi|^2$.

(2) For each $i{\in}[1,k]$, check whether $G_{c_{i-1}} \Rightarrow_{(\varphi_i,h_i)} G_{c_i}$ is a chase step; if not, reject the guess; otherwise, continue.

(3) For each $i \in [1,k]$, check whether $G_{c_{i-1}} \Rightarrow_{(\varphi_i,h_i)} G_{c_i}$ is invalid; if any of these chase steps is invalid, return true; otherwise, continue.

(4) Check whether all literals of Y can be inferred from G_{c_k}; if so, return true; otherwise, reject the guess.

The correctness of the algorithm follows from Lemma 2.2. For its complexity, step (1) is in PTIME by the definition of G_Q; steps (2)-(4) are all in PTIME by the fact that $|G_{c_{i-1}}| \leq 5|\varphi|^2|\Sigma|$ and $|G_{c_i}| \leq 5|\varphi|^2|\Sigma|$. Thus, the algorithm is in NP, and so is the implication problem for GARs. □

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

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

The *association deduction* problem is stated as follows.

- Input: Graph G, GARs Σ, and a candidate association α.
- Question: Is α a deduced association of G by Σ, *i.e.,* whether $\alpha \in \mathsf{deduced}(G,\Sigma)$ (see Theorem 2.1 for the definition of $\mathsf{deduced}(G,\Sigma)$)?

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

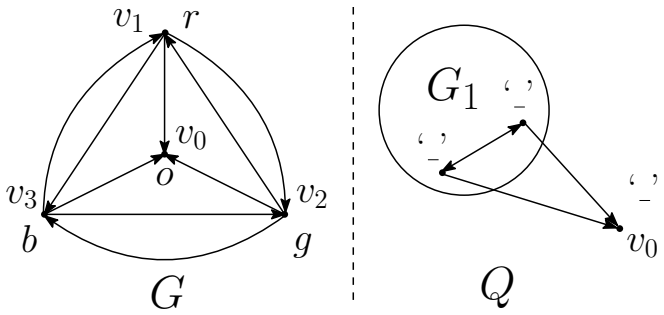


Figure 2.2: Graphs and patterns in Theorem 2.4

GARs.

A similar problem is studied in [FLTZ19], to deduce value associations $v.A = c$ or $v.A = v'.B$ using GFDs [FL19]. That problem is known NP-complete [FLTZ19]. Below we show that deducing associations with GARs is no harder.

Theorem 2.4: *The association deduction problem for* GARs *is* NP*-complete.* □

Proof: We show that the association deduction problem is NP-complete.

Upper bound. Given a graph G, a set Σ of GARs, and a candidate association α of G, we design the following NP algorithm to verify whether $\alpha \in \mathsf{deduced}(G,\Sigma)$.

(1) Guess a chasing sequence $G_{c_0}, \ldots, G_{c_k}$ such that $G_{c_0} = G$ and $k \leq 4|G|^2|\Sigma|$.

(2) Check whether α exists in G_{c_k}; if so, return true.

The correctness of the algorithm follows from Theorem 2.1. For the complexity, step (2) is in PTIME, since $|G_{c_k}| \leq 4|G|^2|\Sigma|$ (see the proof of Theorem 2.1). Therefore, the algorithm is in NP, and so is the association deduction problem. Although the chase is Church-Rosser, we cannot just compute one chase branch for the proof. This is because each chase step is in NP since each step requires graph homomorphism to apply a GAR. Thus, obtaining a sequence of chase steps with limited length is in P^{NP}, and we cannot use this one chase branch to show that the association deduction problem is in NP.

Lower bound. We show that the association deduction problem is NP-hard by reduction from the 3-coloring problem, which is known to be NP-complete [GJ79]. The 3-coloring problem is to decide, given an undirected graph $G_1 = (V_1, E_1)$, whether there exists a proper 3-coloring μ of nodes in V_1 such that for each edge $(v_1, v_2) \in E_1$, $\mu(v_1) \neq \mu(v_2)$.

Given undirected G_1, we construct a graph G, a set Σ of GARs, and a candidate association α such that $\alpha \in \mathsf{deduced}(G,\Sigma)$ if and only if G_1 has a property 3-coloring. Intuitively, we will use G to encode proper 3-coloring, Σ to encode G_1, and α to check

whether G_1 has a proper 3-coloring.

(1) Graph G=(V,E,L,F_A) is shown in Fig. 2.2, in which

(a) $V = \{v_0, v_1, v_2, v_3\}$, where v_1, v_2 and v_3 represent three different colors, and v_0 is a specific node to represent the candidate association, which will be clear soon;

(b) $E = \{(v_i, 0, v_j), (v_j, 0, v_i) \mid i, j \in [1,3] \wedge i \neq j\} \cup \{(v_i, 0, v_0) \mid i \in [1,3]\}$, *i.e.,* v_1, v_2 and v_3 form a clique, each node has an edge leading to v_0 except itself, and all edges are labeled the unique '0';

(c) the labeling function is defined as $L(v_0) = o$, $L(v_1) = r$, $L(v_2) = g$, and $L(v_3) = b$; and

(d) F_A is empty, *i.e., G* does not have any attribute.

(2) The set Σ consists of only one GAR $\varphi = Q[\bar{x}](X \to Y)$, which is defined as follows.

(a) Pattern $Q[\bar{x}]$=(V_Q, E_Q, L_Q, μ) is shown in Fig. 2.2, where

- $V_Q = V_1 \cup \{v_0\}$, *i.e., Q* consists of all nodes in G_1 and an extra node v_0;
- $E_Q = \{(u,0,v), (v,0,u) \mid (u,v) \in E_1\} \cup \{(v, v_0) \mid v \in V_1\}$, *i.e.,* each undirected edge (u,v) in G_1 is represented by two directed edges labeled 0, and each node in G_1 has an edge directing to v_0;
- all pattern nodes are labeled wildcard, *i.e.,* $L_Q(v) =$ '_' for all $v \in V_Q$; and
- for each pattern node $v_i \in V_Q$, $\mu(x_i) = v_i$.

(b) The literals in X and Y are such defined that X is empty-set, and $Y = (x_0.A)$, *i.e.,* the GAR φ deduces the existence of an attribute $x_0.A$.

(3) The candidate association α is defined as the A-attribute of node v_0 in G, *i.e.,* α is $v_0.A$.

It is easy to verify that $\alpha \in \mathsf{deduced}(G, \Sigma)$ if and only if G_1 has a proper 3-coloring, by checking the existence of the matches of pattern Q in graph G. □

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

- edge insertion (insert e), possibly with new nodes, and
- edge deletion (delete e), along with endpoints of degree 0.

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

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

We use $\mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$ to denote the set of *changes* to the set $\mathsf{deduced}(G, \Sigma)$ of associations in response to updates ΔG, *i.e.,* associations that are either in $\mathsf{deduced}(G, \Sigma)$

but not in deduced$(G \oplus \Delta G, \Sigma)$, or vice versa.

The *incremental deduction* problem is stated as follows.

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

The need for studying this problem is evident. It is costly to compute deduced$(G \oplus \Delta G, \Sigma)$ starting from scratch, by Theorem 2.4. Hence we want to incrementally compute the changes to deduced(G, Σ) such that deduced$(G \oplus \Delta G, \Sigma)$ = deduced$(G, \Sigma) \oplus$ deduced$_{\Delta}(G, \Delta G, \Sigma)$ by making maximum reuse of deduced(G, Σ). When ΔG is small, often so is deduced$_{\Delta}(G, \Delta G, \Sigma)$, which is less costly to compute.

A related problem was studied for GFDs, to decide whether a match $h(\bar{x})$ is a violation of GFDs in $G \oplus \Delta G$ but not in G, or vice versa [FLLT20]. It is shown coNP-complete. But the incremental deduction for GARs is slightly harder.

Theorem 2.5: *The incremental deduction problem is* DP*-complete for* GARs*, and remains* DP*-hard when either graph G or updates ΔG to G has a constant size.* □

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

Proof: We first provide a DP algorithm for the incremental deduction problem, and then show that the problem is DP-hard.

Upper bound. Given a graph G, a set Σ of GARs, a batch update ΔG, and a candidate association α of G or $G \oplus \Delta G$, we check whether $\alpha \in \mathsf{deduced}_{\Delta}(G, \Delta G, \Sigma)$ as follows.

(1) Check whether $\alpha \in$ deduced(G, Σ) or $\alpha \in$ deduced$(G \oplus \Delta G, \Sigma)$; if not, return false; otherwise, continue.

(2) Check whether $\alpha \notin$ deduced(G, Σ) or $\alpha \notin$ deduced$(G \oplus \Delta G, \Sigma)$; if not, return false; otherwise, return true.

The correctness is guaranteed by the following.

$$\begin{aligned}
&\alpha \in \mathsf{deduced}_{\Delta}(G, \Delta G, \Sigma) \\
\Leftrightarrow &\alpha \in \big(\mathsf{deduced}(G,\Sigma) \setminus \mathsf{deduced}(G \oplus \Delta G, \Sigma)\big) \\
&\quad \vee \alpha \in \big(\mathsf{deduced}(G \oplus \Delta G, \Sigma) \setminus \mathsf{deduced}(G,\Sigma)\big) \qquad (2.1) \\
\Leftrightarrow &\alpha \in \big(\mathsf{deduced}(G,\Sigma) \cup \mathsf{deduced}(G \oplus \Delta G, \Sigma)\big) \setminus \\
&\quad \big(\mathsf{deduced}(G,\Sigma) \cap \mathsf{deduced}(G \oplus \Delta G, \Sigma)\big) \qquad (2.2)
\end{aligned}$$

Equation (2.1) is from the definition of $\mathsf{deduced}_{\Delta}(G, \Delta G, \Sigma)$, and Equation (2.2) follows from a basic property of set theory, *i.e.,* $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B) = (A \cup B) \cap (\overline{A} \cup \overline{B})$, where A and B are two sets.

For the complexity, we can verify that step (1) is in NP and step (2) is in coNP by Theorem 2.4, and the fact that NP is closed under union and intersection. Therefore, the algorithm is in DP; so is the incremental deduction problem.

Lower bound. We show that the problem is DP-hard by reduction from the critical 3-colorability problem, which is DP-complete [Pap03]. The critical 3-colorability problem is to decide, given an undirected graph $G_1 = (V_1, E_1)$, whether G_1 is not 3-colorable, but deleting any vertex makes G_1 3-colorable (see proof of Theorem 2.4 for 3-colorability).

Given undirected G_1, we construct a (directed) graph G, a set Σ of GARs, a batch update ΔG, and a candidate association α such that $\alpha \in \mathsf{deduced}_{\Delta}(G, \Delta G, \Sigma)$ if and only if G_1 is not 3-colorable, but deleting any vertex makes G_1 3-colorable. Intuitively, we will use G to encode all proper 3-coloring as in the proof of Theorem 2.4, GARs in Σ to encode G_1 and its node deletions, ΔG to trigger the verification of 3-coloring, and $\alpha \in \mathsf{deduced}_{\Delta}(G, \Delta G, \Sigma)$ to encode the fact that G_1 changes from non-3-colorable to 3-colorable.

(1) The graph G is identical to its counterpart graph that is constructed in the lower bound proof of Theorem 2.4, which represents the proper 3-coloring.

(2) The set Σ consists of two groups of GARs. The first group is to encode the topological structure of G_1 and subgraphs of G_1 after node deletions, while the second group is to ensure that deleting any vertex makes G_1 become 3-colorable.

(a) The first group consists of $|V_1|+1$ GARs, each of which is in the form of $\varphi_i = Q_i[\bar{x}]$ $(\emptyset \rightarrow x_0.A_i)$ $(i \in [0, |V_1|])$. Here $Q_0[\bar{x}]$ is built from G_1 along the same lines as in the lower bound proof of Theorem 2.4 (assuming $V_1 = \{v_1, \ldots, v_{|V_1|}\}$), which

represents the structure of G_1. Each other pattern $Q_i[\bar{x}]$ ($i \in [1, |V_1|]$) is derived from Q_0 by (i) removing one pattern node v_i from Q_0, and (ii) adding a new edge (v_0, τ, v_0'), where v_0 is the extra pattern node as shown in Fig. 2.2 and v_0' is another newly added node carrying label 'τ'. Observe that there exists no node labeled 'τ' in G. Therefore, only φ_0 may be applied on G, while none of the patterns in GARs φ_i with $i \in [1, |V_1|]$ has a match in G. This can be used to deduce the candidate association.

(b) The second group consists of only one GAR $\varphi_{|V_1|+1} = Q'[x](x.A_1 \wedge \ldots \wedge x.A_{|V_1|} \rightarrow x.A_0)$, where Q' includes a single pattern node x labeled 'o'. Intuitively, it states that if attributes $A_1, \ldots, A_{|V_1|}$ exist at node x, then x also has attribute A_0. Observe that G does not contain any attribute, and thus $\varphi_{|V_1|+1}$ cannot be applied on G.

(3) The update ΔG only has an insertion of edge (v_0, τ, v_2), where v_2 is a new node labeled 'τ'. Note that after this update, all GARs in Σ may be applied on $G \oplus \Delta G$.

(4) The candidate association α is defined as an attribute literal $v_0.A$, where v_0 is the only node without outgoing edges in G. Observe the following with regard to α.

(a) Since φ_0 does not have edges labeled τ, either it can be applied on both G and updated $G \oplus \Delta G$, or φ_0 cannot be applied on any of the two graphs. In addition, if G_1 is 3-colorable, then attribute $v_0.A$ exists in both $\mathsf{deduced}(G, \Sigma)$ and $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$ by φ_0, and hence $\alpha \notin \mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$. On the contrary, G_1 is not 3-colorable if $\alpha \in \mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$.

(b) When G_1 is not 3-colorable, the only way to ensure that $\alpha \in \mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$ is to enforce $\varphi_{|V_1|+1}$. However, all attributes $x_0.A_1, \ldots, x_0.A_{|V_1|}$ must exist in $\mathsf{deduced}(G \oplus \Delta G, \Sigma)$ to make $\varphi_{|V_1|+1}$ applicable. It means that all GARs φ_i with $i \in [1, |V_1|]$ must be applied on $G \oplus \Delta G$, *i.e.,* each corresponding to that undirected graph of Q_i ($i \in [1, |V_1|]$) is 3-colorable. That is, deleting one node makes G_1 become 3-colorable.

Based on the construction and observation above, we can verify that $\alpha \in \mathsf{deduced}_\Delta (G, \Delta G, \Sigma)$ if and only if G is not 3-colorable, but deleting any vertex makes G 3-colorable. □

2.4 Parallel Deduction Algorithm

In this section we show how to deduce associations with GARs in parallel by using the computation model of GRAPE [FYX+18]. We first review the GRAPE model (Section 2.4.1), and then provide algorithms for parallel association deduction (Section 2.4.2) and incremental deduction (Section 2.4.3).

2.4.1 Graph Centric Parallelization

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

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

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

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

Parallel computation. Given a query $Q \in \mathcal{Q}$, GRAPE posts the same Q to all workers. Then a PIE program is executed in supersteps under the BSP model [Val90], as follows.

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

*(2) Incremental computation (*IncEval*).* In the following supersteps, the partial answers $Q(F_i)$'s are iteratively updated by IncEval. More specifically, (a) master P_0 applies f_{aggr}

to messages from the last superstep, which resolves conflicts. Then these aggregated values are routed to relevant workers. (b) Upon receiving the message M_i (contents in M_i depend on the graph computation algorithm [FYX+18]), worker P_i *incrementally* computes $Q(F_i \oplus M_i)$ with IncEval in parallel for $i \in [1,n]$, by *treating M_i as updates*. At the end of each superstep, worker P_i sends a message to P_0 that consists of *changes* to the update parameters $\bar{x}_i$ of F_i just like in PEval.

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

PIE programs guarantee to converge at correct answers under a monotone condition as long as the sequential PEval, IncEval and Assemble are correct [FYX+18].

2.4.2 Parallel Association Deduction

We next provide a PIE program, denoted by PDeduce. Given a fragmented graph G and a set Σ of GARs, it computes deduced(G,Σ). We give its PEval, IncEval and Assemble, which are parallelized as described in Section 2.4.1.

Algorithm 2.1: PEval for program PDeduce

Input: Fragment $F_i = (V_i, E_i, L_i, F_{A_i})$ and a set Σ of GARs.

Output: Set $Q(F_i)$ of missing links and attributes of F_i by Σ

Declaration: **for each** node $v \in V_i$, two variables v.link and v.attr;
and an additional variable $F_i.H$;

1 $\Psi \leftarrow \Sigma$; $\mathcal{C}_V \leftarrow V_i$; $F_i.H \leftarrow \emptyset$;
2 **repeat**
3 $\Delta F_c \leftarrow \emptyset$;
foreach GAR $\varphi = Q[\bar{x}](X \rightarrow Y) \in \Psi$ **do**
4 extract a set $\mathcal{T}$ of partial matches $h(\bar{x}_p)$ for Q
s.t. X (resp. Y) can be (resp. cannot be) satisfied;
5 $(A_c, H_p) \leftarrow$ ExpandAssoc$(\varphi, \mathcal{T}, \mathcal{C}_V, F_i)$;
6 $\Delta F_c \leftarrow \Delta F_c \cup A_c$; $F_i.H \leftarrow F_i.H \cup H_p$;
7 update F_i with ΔF_c;
8 adjust $\mathcal{C}_V$ using nodes of ΔF_c; $\Psi \leftarrow$ SuccGAR$(\Sigma, \Delta F_c)$;
until $\Delta F_c = \emptyset$;
9 $Q(F_i)$ stores the deduced associations;

Challenges. As indicated in Section 2.2, a major task for deducing associations is

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

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

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

PEval. PEval of PDeduce is given in Algorithm 2.1. It takes a set Σ of GARs and a fragment F_i of graph G as input, and deduces a set $Q(F_i)$ of associations pertaining to F_i with Σ. It employs two status variables $v.\mathsf{link}$ and $v.\mathsf{attr}$ for each node v in F_i, recording v's adjacent edges and attribute values, respectively. It also uses a "global" status variable $F_i.H$ to store *partial matches* of the patterns in Σ that involve nodes residing at other workers, where a partial match maps only a subset of pattern nodes. The update parameters of F_i include (a) $F_i.H$ to pass partial matches to other workers, and (b) $v.\mathsf{link}$ and $v.\mathsf{attr}$ of border nodes v to reconcile values, where border nodes are those that are within $\mathsf{max}_{Q\in\Sigma}|Q|$ hops of the nodes on edges crossing different fragments.

Algorithm PEval iteratively applies *active* GARs in Σ, guided by *active* nodes in fragment F_i (lines 2-8). Here a GAR (resp. node) is *active* if it can be enforced (resp. involves in the mapping) in a chase step for deducing *new* associations in the current iteration. The active GARs and nodes are collected in sets Ψ and $\mathcal{C}_V$, initially Σ and V_i, respectively (line 1). For each active GAR, it first extracts a set $\mathcal{T}$ of partial matches under certain conditions (line 4), and then completes them and deduces associations A_c via procedure ExpandAssoc (line 5). At the end of each iteration, it updates F_i with the new associations ΔF_C that are accumulated during this iteration (line 7), and adjusts Ψ and $\mathcal{C}_V$ for the next iteration (line 8). The iterations proceed until no new associations

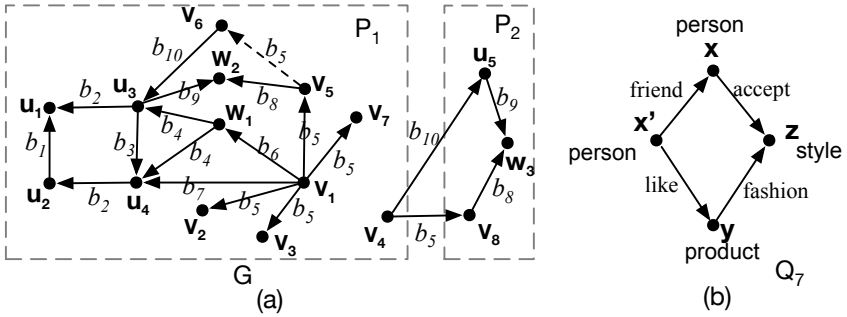


Figure 2.3: Example graph and pattern

can be deduced. The associations deduced in the process are stored in $Q(F_i)$.

PEval employs the following new techniques.

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

Match extraction. For an active GAR $Q[\bar{x}](X \to Y)$, PEval maps pattern nodes $\bar{x}_p$ ($\bar{x}_p \subseteq \bar{x}$) in literals X and Y to nodes in F_i, to extract partial matches $h(\bar{x}_p)$ (line 4), so that $h(\bar{x}_p)$ can (resp. cannot) satisfy X (resp. Y). This is conducted by using the index on pattern node labels and choosing nodes within $|Q|$ hops of the active nodes $\mathcal{C}_V$, by the locality of pattern matching. One can verify that only partial matches of this form can contribute to new associations.

Match completion. Procedure ExpandAssoc completes partial match $h(\bar{x}_p)$ by iteratively mapping the remaining $\bar{x} \setminus \bar{x}_p$ to nodes in F_i, following a *dynamic* candidate-size order [HKG+19] (line 5). That is, each time it maps a pattern node u that is connected to one of the matched pattern nodes, and currently has the *minimum* number of candidates. The candidates are inspected using the index on relevant triples, and each extended partial match should not satisfy $X \to Y$.

Once the partial $h(\bar{x}_p)$ is extended to a complete match $h(\bar{x})$ and $h(\bar{x})$ includes active nodes of $\mathcal{C}_V$, it deduces relevant associations directly and prunes all subsequent extensions of $h(\bar{x}_p)$ when pattern nodes in Y are already mapped in $h(\bar{x}_p)$ (*i.e.,association-guided* pruning). ExpandAssoc also returns a set H_p of partial matches that involve border nodes and hence need to be expanded at other workers. The status variable $F_i.H$ is extended with partial matches H_p (line 6).

Active GARs *and nodes*. After each iteration, we revise $\mathcal{C}_V$ with those nodes involved in the newly deduced associations ΔF_c and derive active GARs by procedure SuccGAR

for the next iteration (line 8). Extending templates that generalize nodes to their labels [FLTZ19], SuccGAR picks such active GARs that have the same templates in their preconditions (or pattern edges) as that of the associations in ΔF_c. For instance, a GAR becomes active if it has a literal $x.A = y.B$ in its X and there is a new association $v.A = 1$ with $L(v) = L_Q(x)$.

Example 2.6: A fragmented graph G is shown in Fig. 2.3(a) (excluding dotted edge), where v_1 to v_8 denote persons, u_1 to u_2 are classes, u_3 to u_5 are products, w_1 denotes a shop and w_2 to w_3 are styles; labels b_1 to b_{10} are related_to, type, deal, sell, friend, follow, click, accept, fashion and like, respectively.

Consider a set Σ of GARs including φ_1 of Example 2.3 and $\varphi_7 = Q_7[\bar{x}](\emptyset \rightarrow \mathsf{like}(x,y))$, where Q_7 is depicted in Fig. 2.3(b).

Given G and Σ, a partial match h'_1 of Q_1 from φ_1 is extracted by PEval at worker P_1 in the first iteration, where $x \mapsto v_1$, $w \mapsto w_1$, $y_1 \mapsto u_4$, $y_2 \mapsto u_3$, $z_1 \mapsto u_2$ and $z_2 \mapsto u_1$. Since h'_1 is already a complete match and $h'_1 \not\models Y_1$, it adds association $(v_1, \mathsf{like}, u_3)$ at P_1. The other partial matches with $x \mapsto v_1$ and $y_2 \mapsto u_3$ are dropped by association-guided pruning.

Then φ_7 is treated as an active GAR for the second iteration since Q_7 has a pattern edge (x', like, y) sharing the same template with the newly deduced association. PEval next extracts a partial match h'_2 for Q_7 that maps x' (resp. y) to active node v_1 (resp. u_3). When completing h'_2 by procedure ExpandAssoc, z of Q_7 is mapped ahead of x since z has only one candidate w_2 whereas x has four (v_2, v_3, v_5, and v_7). In fact, only a singe complete match of Q_7 is finally expanded from h'_2 and it yields a new association $(v_5, \mathsf{like}, u_3)$.

PEval also finds a partial match h'_3 of Q_7 in the first iteration at worker P_1. It maps x to v_8, x' to v_4 and y to u_5. Since h'_3 involves v_8 and u_5 that reside at worker P_2 (crossing-edges are maintained by both workers), the partial match h'_3 will be sent to P_2 for further completion. □

At the end of PEval, master P_0 collects the status variables of border nodes v from all fragments. It applies aggregate function f_{aggr} to reconcile $v.\mathsf{link}$ and $v.\mathsf{attr}$, and routes the aggregation and partial matches to relevant workers as messages. The aggregate function f_{aggr} completes two tasks after receiving the messages. First, f_{aggr} merges repetitive predicates so that only one of the repetitive predicates is enforced. This saves computation cost by skipping redundant operations. Second, f_{aggr} parses the partial matches encoded in the messages and sends the match information to PEval. Then PEval can complete the match in the next round. If conflicts emerge in attributes

v.attr of any node, P_0 terminates the process immediately with $\perp$.

Algorithm 2.2: IncEval for program PDeduce

Input: Fragment F_i=(V_i, E_i, L_i, F_{A_i}), GARs Σ, message M_i.

Output: Missing associations $Q(F_i \oplus M_i)$ deduced.

Declaration: Message $M_i = \{v.A, (v, \iota, v') \mid v, v' \in V_i,\ v.A$ and (v, ι, v') changed$\} \cup \{h(\bar{x}_p) \mid h(\bar{x}_p)$ is a partial match involving nodes in $V_i\}$

1 collect nodes (resp. changes) of M_i into $\mathcal{C}_V$ (resp. ΔF_c);

2 $\Psi \leftarrow \mathsf{SuccGAR}(\Sigma, \Delta F_c) \cup \{\varphi \mid \exists h(\bar{x}_p) \in M_i, h(\bar{x}_p)$ is a partial match of the pattern of $\varphi\}$; $F_i.H \leftarrow \emptyset$;

3 update F_i with ΔF_c;

4 apply active Ψ on F_i iteratively to deduce new associations;

5 $Q(F_i \oplus M_i)$ stores the deduced associations that are accumulated over supersteps;

IncEval. As shown in Algorithm 2.2, IncEval of PDeduce also deduces new associations incrementally. At worker P_i, it is triggered by message M_i that includes all the changes to the status variables of the border nodes in fragment F_i, and a set of partial matches to be further expanded at F_i.

Unlike PEval that initially makes the set Σ of GARs and the set V_i of nodes in F_i active, IncEval determines initial active nodes and GARs according to the changes and partial matches passed over in message M_i (lines 1-2). It treats the received changes directly as ΔF_c and updates fragment F_i with ΔF_c (line 3). After that, IncEval applies active GARs iteratively to deduce new associations pertaining to F_i (line 4), along the same lines as that in PEval, *i.e.,* lines 2-8 of Algorithm 2.1. The difference is that it also considers the partial matches in M_i, which are expanded just like extracted partial matches. IncEval stores the deduced associations that are accumulated over iterations as partial result $Q(F_i \oplus M_i)$.

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

Example 2.7: Continuing with Example 2.6, upon receiving partial match h'_3 at worker P_2, IncEval completes it by mapping the only remaining pattern node z of Q_7 to w_3. It then yields a missing link $(v_8, \mathsf{like}, u_5)$ deduced as a new association. This is a local edge for worker P_2 and it will be used to update both the fragments and indices at P_2. □

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

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

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

□

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

2.4.3 Incremental Deduction of Associations

As remarked in Section 2.3, real-life graphs frequently change and association deduction is costly over large-scale graphs. These highlight the need for incremental association deduction, *e.g.,* in updating the the recommendation of products in e-commerce. We next develop a parallel algorithm for incremental deduction, denoted by IncDeduce.

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

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

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

Auxiliary structures. In addition to the indices of PDeduce, for each edge e (resp. deduced association α), IncDeduce maintains a set $\mathsf{d}(e)$ (resp. $\mathsf{d}(\alpha)$) to store associations α' if the last chase steps for deducing α' include e (resp. α) in their mappings or precondition checking. Here e (resp. α) is also in $\mathsf{d}(e)$ (resp. $\mathsf{d}(\alpha)$). Note that these structures can be readily obtained when running PDeduce; their sizes are polynomial in $|G|$ and $|\Sigma|$ (the proof of Theorem 2.1).

Algorithm 2.3: Algorithm IncDeduce

Input: Fragmented chase graph G_c with auxiliary information, a set Σ of GARs and batch update $\Delta G = (\Delta G^+, \Delta G^-)$.

Output: The changes $\mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$.

1 update G_c with ΔG; $\mathsf{deduced}^+_\Delta := \emptyset$;
2 $\mathsf{deduced}^-_\Delta \leftarrow \mathsf{DisAssoc}^-(G_c, \Sigma, \Delta G^-)$;
3 update G_c with $\mathsf{deduced}^-_\Delta$;
4 $A_c \leftarrow \mathsf{RefineAssoc}(G_c, \Sigma, \Delta G)$;
5 refine $\mathsf{deduced}^+_\Delta$ and $\mathsf{deduced}^-_\Delta$ by A_c;
6 update G_c;
7 **return** $\mathsf{deduced}_\Delta(G, \Delta G, \Sigma) = (\mathsf{deduced}^+_\Delta, \mathsf{deduced}^-_\Delta)$;

Algorithm. As shown in Algorithm. 2.3, IncDeduce takes as input Σ, ΔG and moreover, the chase graph G_c and the corresponding auxiliary structures that are cached after the batch execution of PDeduce and are distributed across workers. Denote by ΔG^+ and ΔG^- the inserted and deleted edges in ΔG, respectively. IncDeduce computes the changes $\mathsf{deduced}_\Delta(G, \Delta G, \Sigma)$ to the old associations deduced.

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

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

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

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

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

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

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

Example 2.8: Recall graph G and GARs Σ from Example 2.6. Consider ΔG that inserts $(v_5, \mathsf{friend}, v_6)$ and deletes $(v_1, \mathsf{friend}, v_5)$. IncDeduce first checks association affected by the deletion, which is $(v_5, \mathsf{like}, u_3)$. Since this link can be deduced with the insertion in a way similar to Example 2.6, it remains valid and IncDeduce stops further checking of associations depending on it. Besides, no new association is deduced during the refinement phase in this case. Hence the result of batch PDeduce (Example 2.6) remains stable. □

We have the following about algorithm IncDeduce.

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

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

2.5 Experimental Study

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

Experimental setting. We used six real-life graphs as summarized in Table 2.1. In particular, Orkut is a large social network without informative attributes that can be used by GARs. We evaluated the efficiency of enforcing various GARs on it, and randomly included 20 attributes in Orkut.

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

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

ML classifiers. We adopted SimplE [KP18] and ComplEx [TWR$^+$16] to implement the

Table 2.1: Real-life graphs

Dataset	Type	Vertices	Edges
DBpedia [dbp21a]	knowledge base	6.2M	33.4M
YAGO [SKW07]	knowledge base	2M	5.7M
Pokec [Pok]	social network	1.6M	30.6M
Patent [LKF05]	citation network	3.7M	16.5M
IMDB [IMD21]	knowledge graph on movies	16.7M	43.2M
Orkut [YL15]	social network	3M	117M

ML classifier $\mathcal{M}$ for link prediction. We followed the protocol of [TWR$^+$16, KP18] to prepare training data; we obtained positive triples from original graphs and negative ones by combining entities and relations randomly. We created on average two negative samples per positive one for training, using 55% edges of each graph. We adopted the PyTorch framework, the hyper-parameter search strategy and training settings of [TWR$^+$16, KP18] to train classifier $\mathcal{M}$.

GAR *generator*. For each graph, we generated GARs using the training data in three steps. (1) We first added all missing links predicted by the ML classifier between the nodes covered by training data. (2) We next applied an extension of the discovery algorithm for GFDs [FHLL20] on the subgraph pertaining to updated training data to derive GARs. Starting from frequent single-node patterns, the algorithm in [FHLL20] interleaves vertical spawning to extend the patterns and horizontal spawning to find attribute dependencies. Apart from constant and variable literals considered in [FHLL20], we removed some edges from the discovered patterns and included them as edge literals in GARs. Attribute literals were added with attributes that appear in the matches. (3) After these, we replaced certain edge literals $\iota(x,y)$ with ML literals $\mathcal{M}(x,y,\iota)$ in the GARs mined, such that $\mathcal{M}$ predicts the existence of missing edges (v,ι,v') in the training data.

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

Evaluation. The accuracy is evaluated over the test set of each real-life graph, *i.e.,* the graph excluding the training data. It is to evaluate the quality of associations deduced. Following [FLTZ19, FWX16b], we treated the original graphs as "correct" and introduced noises by randomly removing 3% edges and 3% attributes of

each test set, since the quality of real-life graphs is unknown [ZKS+13]. We measured the accuracy by precision, recall and F-measure, which are defined as (1) the ratio of removed associations deduced to all associations deduced by the methods, (2) the ratio of associations correctly deduced to all the associations removed, and (3) $2 \cdot (\mathsf{precision} \cdot \mathsf{recall})/(\mathsf{precision} + \mathsf{recall})$, respectively. As remarked earlier, we used Orkut only to test efficiency.

Baselines. Apart from implementing PDeduce (Section 2.4.2) and IncDeduce (Section 2.4.3) in C++, we also compared with the following baselines. (1) A variant $\mathsf{PDeduce}_\mathsf{N}$ of PDeduce, without enforcing association-guided pruning; and a variant $\mathsf{IncDeduce}_\mathsf{N}$ of IncDeduce without early checking of affected associations. (2) The sequential repairing method of [FLTZ19], which deduces missing links and attributes, denoted as GRb. (3) ML link predictors SimplE [KP18] and ComplEx [TWR+16]; they are trained and tested with same data as above. (4) The link deduction algorithm in [FWWX15] with GPARs, denoted as mGPAR; and GMend of [FLTZ19] with an extension of GFDs, which deduces certain fixes to graphs, on deduction of missing attributes. (5) A sequential algorithm LinkH that finds missing links with the Horn rules discovered by AMIE [GTHS13].

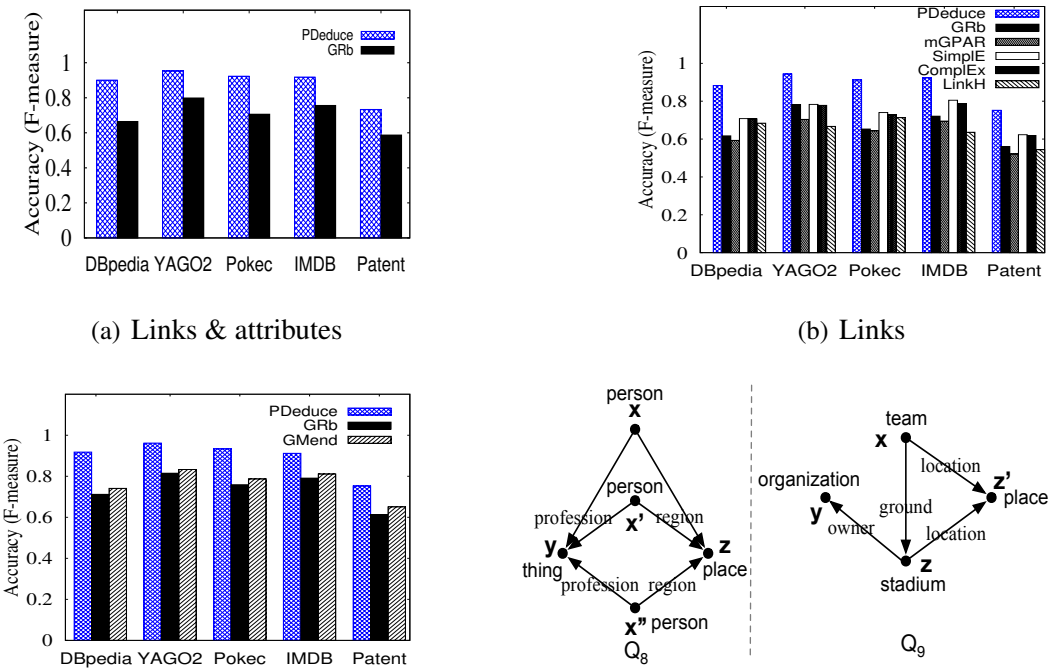
Among these, GRb, mGPAR, GMend and LinkH are also rule-based methods. To get a fair comparison, besides the subclasses of GARs they support, we mined additional rules using their corresponding discovery methods to make all rule-based ones employ the same amount of rules.

The experiments were conducted on GRAPE [FYX+18], deployed on an HPC cluster of up to 10 machines connected by 10-Gbps links. From each machine we used 2 processors powered by Intel Xeon 2.2GHz and 64G memory. Each experiment was run 5 times and the average is reported here.

Experimental results. We next report our findings.

Exp-1: Accuracy. We first tested the accuracy of PDeduce with all GARs mined. Figures 2.4(a) to 2.4(c) report the F-measure for deducing both missing links and attributes, missing links only and missing attributes only, respectively, over five real-life graphs on average. As shown there, PDeduce consistently outperforms other methods.

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



(a) Links & attributes

(b) Links

(c) Attributes

(d) Real-life GARs

Figure 2.4: Effectiveness

outperforms GRb and GMend by supporting ML literals. It beats LinkH for all reasons above.

(2) On average PDeduce is 20.8% and 22.1% more accurate than SimplE and ComplEx in deducing missing links, respectively. Since there is little gap between the accuracy of SimplE and that of ComplEx, and they both are embedding-based ML models, the impact of plugging which of the two ML classifiers into PDeduce is not substantial.

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

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

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

Varying $\|\Sigma\|$. Varying $\|\Sigma\|$ from 40 to 200 and 30 to 150, Figures 2.5(a)-2.5(b) report the results on DBpedia and YAGO, respectively. We can see that (1) the more rules are used, the longer all methods take, as expected. (2) PDeduce is on average 2.2 (resp. 14.3) times faster than $\mathsf{PDeduce}_\mathsf{N}$ (resp. GRb), validating the effectiveness of association-guided pruning.

Varying $|\Sigma_Q|$. We varied $|\Sigma_Q|$ from 3 to 7 over DBpedia and YAGO. As shown in Figures 2.5(c)-2.5(d), (a) all algorithms take longer on larger $|\Sigma_Q|$. (b) PDeduce and IncDeduce are feasible with real-life GARs, *e.g.,* they take 17.7s and 4.2s over DBpedia when $|\Sigma_Q| = 5$, as opposed to 304.5s by GRb and 33.9s by $\mathsf{PDeduce}_\mathsf{N}$. (c) PDeduce outperforms other batch algorithms, consistent with Figures 2.5(a) and 2.5(b).

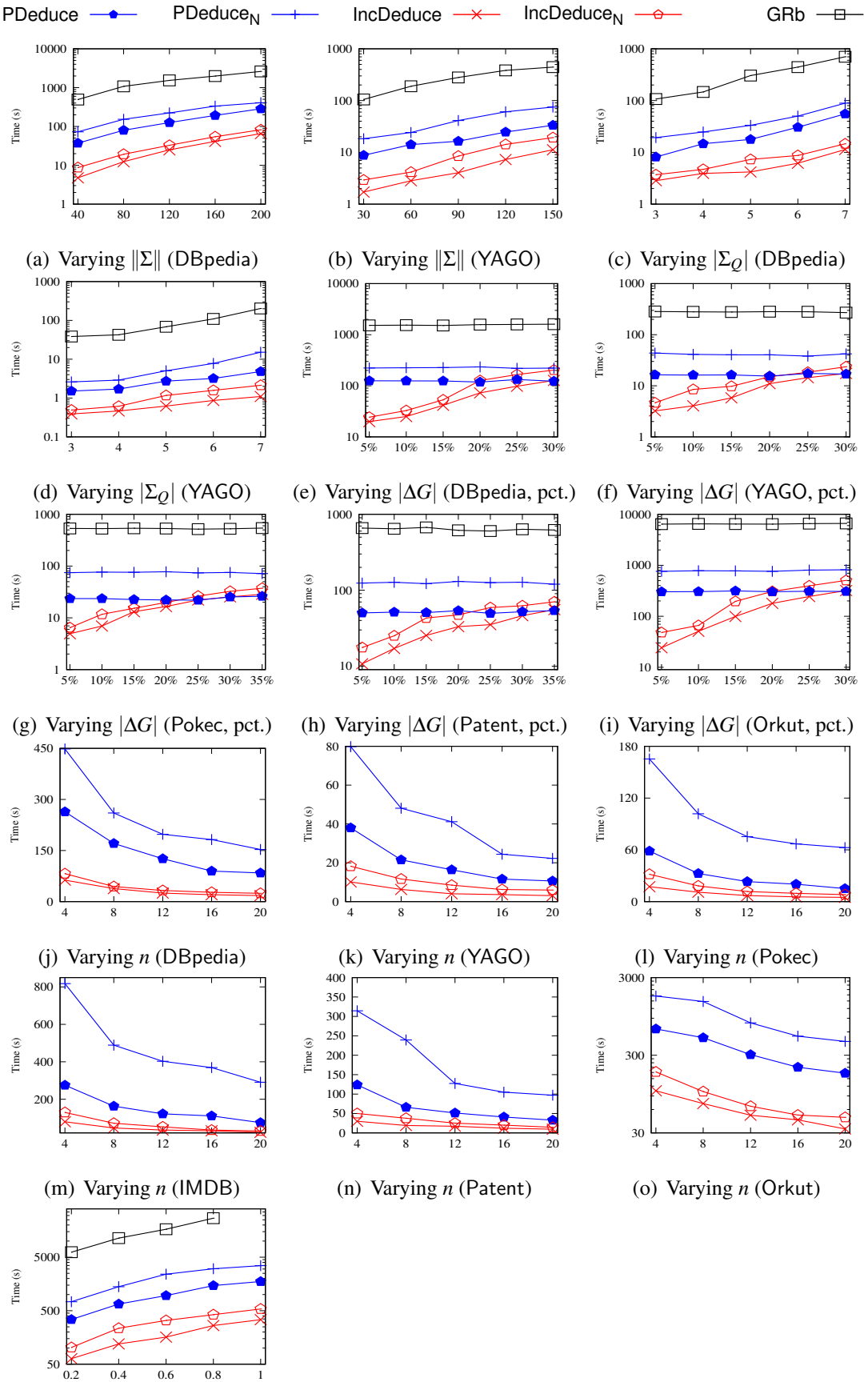
Incremental deduction. Varying $|\Delta G|$ from 5% up to 35% of $|G|$, Figures 2.5(e)-2.5(i) report the following over DBpedia, YAGO, Pokec, Patent and Orkut, respectively. (1) IncDeduce is 6.3 to 1.6 (resp. 5.1 to 1.4, 4.8 to 1.3, 4.7 to 1.6 and 9.5 to 1.7) times faster than PDeduce over the five real-life graphs, respectively, when $|\Delta G|$ varies from 5% to 20%. (2) IncDeduce beats PDeduce even when $|\Delta G|$ is up to 25% of $|G|$. This justifies the need for incremental deduction. (3) All incremental methods take longer for larger $|\Delta G|$, while the batch ones are indifferent to $|\Delta G|$.

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

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

Varying n. We varied the number n of processors from 4 to 20. As shown in Figures 2.5(j) to 2.5(o), (a) PDeduce scales well: the improvement is 3.1 (resp. 3.6, 3.9, 3.7, 3.6, 3.8) times over DBpedia (resp. YAGO, Pokec, IMDB, Patent, Orkut) when n varies from 4 to 20. (b) IncDeduce works well on real-life graphs: it takes only 3.1s to process 10% updates on YAGO using 20 processors; the results on other graphs are consistent. (c) On average, PDeduce beats $\mathsf{PDeduce}_\mathsf{N}$ by 2.7 times, up to 4.1 times. (d) Early checking of affected associations is effective for incremental association deduction: IncDeduce beats $\mathsf{IncDeduce}_\mathsf{N}$ by 1.5 times on average.

Synthetic graphs. Varying the scale factor from 0.2 to 1.0, we tested (incremental) association deduction on synthetic graphs. As shown in Fig. 2.5(p), (a) all the batch and incremental algorithms take longer over larger G, as expected. (b) PDeduce is feasible on large graphs, taking 1756.5s using 100 GARs on graphs with 300 million nodes and a billion edges; in contrast, GRb ran out-of-memory.



(a) Varying $\|\Sigma\|$ (DBpedia) (b) Varying $\|\Sigma\|$ (YAGO) (c) Varying $|\Sigma_Q|$ (DBpedia)

(d) Varying $|\Sigma_Q|$ (YAGO) (e) Varying $|\Delta G|$ (DBpedia, pct.) (f) Varying $|\Delta G|$ (YAGO, pct.)

(g) Varying $|\Delta G|$ (Pokec, pct.) (h) Varying $|\Delta G|$ (Patent, pct.) (i) Varying $|\Delta G|$ (Orkut, pct.)

(j) Varying n (DBpedia) (k) Varying n (YAGO) (l) Varying n (Pokec)

(m) Varying n (IMDB) (n) Varying n (Patent) (o) Varying n (Orkut)

(p) Synthetic (scale factor)

Figure 2.5: Efficiency and scalability

Exp-4: Case study. Figure 2.4(d) shows the patterns of two GARs discovered in the real-life datasets we used.

(1) In Pokec, GAR $\varphi_8 = Q_8[\bar{x}](\mathcal{M}(x,x',\mathsf{friend}) \wedge x.\mathsf{hobbies} = x'.\mathsf{hobbies} \wedge x'.\mathsf{hobbies} = x''.\mathsf{hobbies} \rightarrow \mathsf{friend}(x,x''))$ suggests that if three people have the same profession, region and hobbies, and two of them are predicted as friends by ML classifier, then another friend relationship should also be established. It identifies a link between two people (IDs: 361348, 361341) because of another one (ID: 361273), where all three like football and live in Kolarovo.

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

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

2.6 Novelty and Contributions

Different from previous graph rules, the novelty of GARs proposed in this Chapter consists of the following. (1) We make a first effort to incorporate ML classifiers into logic rules for association deduction. On the one hand, such rules plug in existing ML classifiers and improve the accuracy of association deduction. On the other hand, they can interpret links predicted by ML classifiers in logic. Moreover, in or-

der to support the GARs deduction with chase, we carefully select embedding-based ML models to guarantee the Church-Rosser property. (2) We propose a first framework to catch semantic inconsistencies and missing associations in the same process. Indeed, inconsistencies can also be modeled as erroneous associations (Section 2.2), and should be treated in a uniform framework for associations. That is why we opt to extend GFDs [FWX16b, FL19] to catch missing links and attributes, rather than to define a class of new rules starting from scratch. (3) GARs strike a balance between the expressivity and complexity, with necessary yet minimum extensions to GPARs and GFDs. It is well known that when universal logic rules and existential rules are put together, their static analyses are often undecidable, *e.g.,* the implication problem for TGDs (cf. [AHV95]), and for functional dependencies and inclusion dependencies put together [CV85]. GARs enrich GFDs (of universal semantics) with limited existential semantics, while their satisfiability and implication problems are decidable in coNP and NP, respectively, the same as for GFDs. While the complexity bounds for GARs are similar to GFD counterparts, the proofs are quite different (see Section 2.3). (4) GARs extend GPARs [FWWX15] with preconditions. This work provides the first formulation of association deduction with chase, and the first fundamental results for reasoning about graph association rules, which were not studied in [FWWX15, FWX16a]. Hence, the development of GARs is not easy adaptations of what was done for GFDs and GPARs.

We also adopt a different approach to optimize deduction in this Chapter. (1) We first develop two *sequential* algorithms for deduction and incremental deduction of associations. We then parallelize the algorithms following the fixpoint model of GRAPE, with convergence guarantees [FYX$^+$18]. These depart from the prior algorithms on GFDs [FWX16b, FLTZ19, FLLT20]. (2) We process a set of GARs at the same time, not a single pattern. Moreover, enforcing GARs may *mutate the topological structure of graphs*. In contrast, prior algorithms assume static graphs; they do not work for association deduction. (3) We propose a strategy to reduce redundant mutual effects between different types of updates in incremental deduction. (4) To the best of our knowledge, the incremental deduction algorithm also yields the first incremental graph repairing algorithm.

Chapter 3

Discovering Association Rules from Big Graphs

This Chapter tackles two challenges to discovery of graph rules. Existing discovery methods often (a) return an excessive number of rules, and (b) do not scale with large graphs given the intractability of the discovery problem. We propose an application-driven strategy to cut back rules and data that are irrelevant to users' interest, by training a machine learning (ML) model to identify data pertaining to a given application. Moreover, we introduce a sampling method to reduce a big graph G to a set H of small sample graphs. Given expected support and recall bounds, the method is able to deduce samples in H and mine rules from H to satisfy the bounds in the entire G. As proof of concept, we develop an algorithm to discover Graph Association Rules (GARs).

We firstly introduce the long-standing challenges to discovery of graph rules (Section 3.1), followed by a review of GARs (Section 3.2). Then we present the discovery problem statement in Section 3.3, and a 3-step discovery scheme including application-driven reduction (Section 3.4), graph sampling (Section 3.5), and parallel rule mining (Section 3.6). We experimentally verify this 3-step discovery scheme in Section 3.7, and conclude our novelty and contributions compared with previous work in Section 3.8.

3.1 Challenges to Discovery of Graph Rules

A variety of rules have been studied for graphs, to detect inconsistencies [FWX16b, FLLT20], resolve entities [FFTD15, FL19], reason about knowledge graphs [GTHS13, MCRS19], catch network evolution [BBBG09a, LLLW10], and recommend items to

users [FWWX15]. There have also been recent graph rules that embed machine learning (ML) classifiers as predicates [FJL+20], to deduce associations by unifying rule-based and ML-based methods.

To make practical use of the rules, effective methods have to be in place to discover useful rules from real-life data. Rules on graphs are more complicated than relational rules. For instance, graph functional dependencies (GFDs) [FWX16b] are defined as $Q(X \to Y)$, with a graph pattern Q to identify entities and an attribute dependency $X \to Y$ to apply to those entities. Given a graph G, rule discovery is to find a set Σ_G of non-redundant rules that can be frequently applied to G (measured by support). Discovery of such rules requires to mine both graph patterns and dependencies, and is more challenging than discovery of relational data quality rules.

Challenges. There are two major challenges to graph rule discovery. (1) *Scalability*. Rule discovery algorithms can hardly scale with large graphs. As shown in [FHLL20], for instance, the discovery problem for GFDs subsumes subgraph isomorphism, which is intractable (cf. [GJ79]). As a consequence, it is prohibitively costly to discover GFDs with graph patterns of 7 edges or more, which cannot finish in 1.66 hours on graphs with 32 million nodes and edges, even when using 8 machines [FHLL20]. It is also reported in [FHLL20] that mining GFDs with patterns of at most 5 edges in the same setting already takes 76 minutes. (2) *Excessive rules*. A large number of rules typically hold on a given graph. It is hard for users to inspect the excessive number of discovered rules and identify useful ones to them.

In fact, most existing discovery algorithms for graph rules follow the levelwise search paradigm, *e.g.,* [FHLL20, FWWX15, NWS+17, BBBG09a, KLL+19]. These methods enumerate candidate rules in an exponential search space, and evaluate each candidate by subgraph matching to check whether it meets the discovery requirement, *e.g.,* support threshold. The latter also incurs exponential cost as mentioned above. Therefore, such methods suffer from poor scalability in mining graph rules with large patterns, and often output excessive rules created from the large search space, while users' interests are not considered in pruning useless candidates. The challenges are already present when discovering relational functional dependencies (FDs) [PEM+15], but the problems become more staggering for graph rules.

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

Strategies. We explore new approaches to tackling the challenges.

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

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

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

(4) Proof of concept with GARs. As a proof of concept, we test the strategies with discovering Graph Association Rules (GARs) [FJL$^+$20]. GARs subsume GFDs [FWX16b], graph entity rules (GEDs) [FL19] and graph pattern association rules (GPARs) [FWWX15] as special cases, and hence are able to identify entities, catch conflicts, detect missing links and deduce associations. Moreover, GARs may plug in existing ML classifiers as predicates, to leverage well-trained ML models for entity resolution, link prediction and similarity checking, among other things.

Putting these together, we propose a 3-step scheme to discover useful rules from a big graph G for a given application $\mathcal{A}$: (a) reducing G to $\mathcal{A}$-relevant $G_{\mathcal{A}}$, (b) sampling a set H of $H(\mathcal{A}, \rho\%)$ from $G_{\mathcal{A}}$, and (c) parallelizing discovery with the parallel scalability. We reduce irrelevant rules by proposing ML-based graph reduction (step (a)), and improve the scalability by combining steps (a), (b) and (c). Steps (a) and (b) reduce the problem of rule discovery from large G to much smaller $H(\mathcal{A}, \rho\%)$. We show that the scheme guarantees accuracy bounds. As opposed to prior meth-

ods [MCRS19, GGM20, CWG16] that only sample paths, step (b) samples general subgraphs by selecting representative data. These ensure accuracy bounds on the rules mined from $H(\mathcal{A}, \rho\%)$.

3.2 Graph Association Rules

In this section we briefly review GARs (graph association rules) of [FJL$^+$20].

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

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

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

Pattern matching. A *match* of pattern $Q[\bar{x}]$ in a graph G is a homomorphic mapping h from Q to G such that (a) for each node $u \in V_Q$, $L_Q(u) = L(h(u))$; and (b) for each pattern edge $e = (u, L_Q(e), u') \in E_Q$, $e' = (h(u), L(e'), h(u'))$ is in G and $L_Q(e) = L(e')$. Here $L_Q(u) = L(h(u))$ if $L_Q(u)$ is '_', *i.e.,* wildcard can match an arbitrary label. We denote the match as a vector $h(\bar{x})$, consisting of $h(\mu(x))$ for all $x \in \bar{x}$ in the same order as $\bar{x}$. Intuitively, $\bar{x}$ is a list of entities to be identified, and $h(\bar{x})$ is an instantiation for it.

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

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

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

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

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

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

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

Intuitively, a GAR is a combination of topological constraint Q and logical constraint $X \to p_0$. The pattern Q identifies entities in a graph, and the dependency $X \to p_0$ is applied to the entities. Constant and variable predicates $x.A = c$ and $x.A = y.B$ specify *value associations to* attributes, which can catch inconsistencies and moreover, identify entities (with $x.\mathsf{id} = y.\mathsf{id}$ when A and B are node ids). Attribute and edge predicates $x.A$ and $l(x,y)$ enforce the existence of attributes and edges, *i.e., attribute and edge associations*, respectively, which can deduce associations and missing links.

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

As shown in [FJL+20], GFDs [FWX16b], GEDs [FL19], and GPARs [FWWX15] are special cases of GARs without ML, edge and attribute predicates.

Example 3.1: Embedding ML predicates, GARs are able to predict relationships in professional networks, *e.g.,* colleagues in DingTalk [Din21], and fraudulent behaviors in e-commerce platforms, as follows.

(a) To establish domestic "colleague" connections, we use a GAR $\varphi_a = Q_a[\bar{x}_a](X_a \to \mathsf{colleague}(x_0, x_0'))$, where (i) the pattern Q_a is depicted in Fig. 3.1; and (ii) X_a is $\bigwedge_{i\in[1,k_0]} (x_0.\mathsf{city} = x_i.\mathsf{city} \wedge x_0'.\mathsf{city} = x_i.\mathsf{city}) \wedge \bigwedge_{i,j\in[1,k_0]} \mathcal{M}_a(x_i, x_j, \mathsf{similar\_profile})$. It indicates

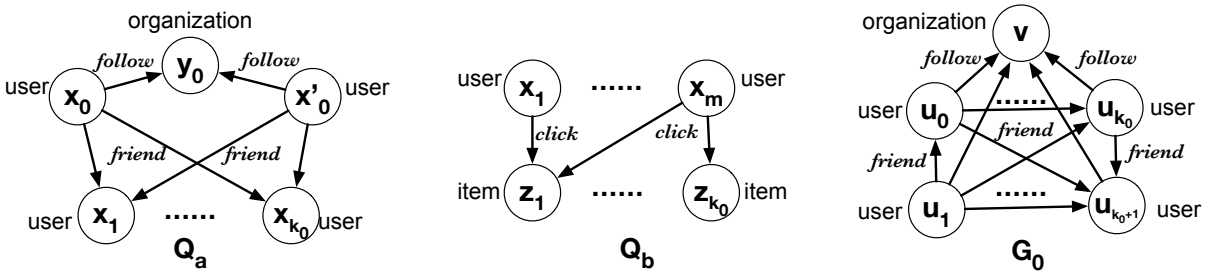


Figure 3.1: Patterns and graphs

that users x_0 and x_0' are likely to be colleagues when they follow the same organization y_0 and have k_0 common friends (*i.e.,* x_1 to x_{k_0}) with the same city attribute, and each pair of common friends have similar profiles, determined by the ML classifier $\mathcal{M}_a$.

(b) Consider GAR $\varphi_b = Q_b[\bar{x}_b](\bigwedge_{i,j\in[1,m]} \mathcal{M}_b(x_i, x_j, \mathsf{one\_group}) \rightarrow \mathsf{click}(x_1, z_{k_0}))$, where pattern Q_b is also shown in Fig. 3.1. It says that if a set of m users $x_1, \ldots, x_m$ are identified to be within the same community by ML classifier $\mathcal{M}_b$, and if all users in this community except x_1 conduct fake clicks on a set of k_0 items in the e-commerce platform, then x_1 might also perform fake click on item z_{k_0}.

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

Semantics. To interpret GAR $\varphi = Q[\bar{x}](X \rightarrow p_0)$, denote by $h(\bar{x})$ a match of Q in a graph G, and by p a predicate of $Q[\bar{x}]$. We write $h(\mu(x))$ as $h(x)$, where μ is the mapping in Q from $\bar{x}$ to nodes in G.

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

For a conjunction X of predicates, we write $h(\bar{x}) \models X$ if match $h(\bar{x})$ satisfies *all* the predicates in X. Note that if X is $\emptyset$ (*i.e.,* true), then $h(\bar{x}) \models X$ for any match $h(\bar{x})$ of Q in G. We write $h(\bar{x}) \models X \rightarrow p_0$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models p_0$. We say that a graph G *satisfies* GAR φ, denoted by $G \models \varphi$, if for all matches $h(\bar{x})$ of Q in G, $h(\bar{x}) \models X \rightarrow p_0$. Graph G *satisfies* a set Σ of GARs, denoted by $G \models \Sigma$, if $G \models \varphi$ for all $\varphi \in \Sigma$.

Example 3.2: Consider the graph G_0 depicted in Fig. 3.1, in which the users u_0 to u_{k_0+1} form a clique with all edges labeled friend, and all the users have the same city attribute value and similar profiles (not shown). Then $G_0 \not\models \varphi_a$ for GAR φ_a of Exam-

Notations	Descriptions
G, $Q[\bar{x}], \varphi$	graph, graph pattern, and GAR φ=$Q[\bar{x}](X \to p_0)$, resp.
$\mathcal{A}$	an application that consists of a set of predicates
$G_{\mathcal{A}}$	an $\mathcal{A}$-graph of G
$\mathcal{M}_{\mathcal{A}}$	an ML model for deducing $G_{\mathcal{A}}$ from graph G
$G_{\mathcal{M}}$	graph G expanded with edges predicted by ML model $\mathcal{M}$
$H(\mathcal{A}, \rho\%)$	a sample graph
Σ_G, Σ_H	GARs mined from G and a set H of sample graphs, resp.
$\mathsf{supp}(\varphi, G)$	the support of GAR φ in graph G

Table 3.1: Notations

ple 3.1. This is due to the match h of Q_a in G_0: $x_0 \mapsto u_0, x_0' \mapsto u_1, x_i \mapsto u_{i+1} (i \in [1, k_0]), y_0 \mapsto v$, which satisfies precondition X_a, but $h \not\models \mathsf{colleague}(x_0, x_0')$ because of the nonexistence of a colleague edge from u_0 to u_1 in G_0. □

Notations of this Chapter are summarized in Table 3.1.

3.3 A Discovery Scheme

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

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

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

In order to find out suitable predicates that model the application $\mathcal{A}$ to the user's interest, the rule discovery algorithm presents the user with diversified edges consisting of various labels in G and asks the user to select edges that are closely related to their desired application. Then the algorithm extracts predicates from each of these selected edges by replacing the vertices with variables while preserving edge labels. Finally, the most frequent predicates are added to the set $\mathcal{A}$ to represent the user's target application.

Example 3.3: Suppose a user is interested in athlete-related rules on the DBpedia [dbp21a] knowledge graph G. The algorithm presents diversified edges consisting of various labels in G, *e.g.,* $(\text{Usain Bolt}, \mathsf{occupation}, \text{sprinter})$, $(\text{Lionel Messi}, \mathsf{club}, \text{Barcelona})$, $(\text{Hillary Clinton}, \mathsf{party}, \text{Democratic Party})$, $(\text{Big Ben}, \mathsf{locate\_in}, \text{London})$ to the user, and requires the user to annotate "interesting" ones. The user marks $(\text{Usain Bolt}, \mathsf{wins}, \text{Beijing Olympic})$ and $(\text{Lionel Messi}, \mathsf{club}, \text{Barcelona})$ as interesting since they are closely related to athletes. Then, replacing these vertices/edges with their labels, the algorithm adds $\mathsf{wins}(x, y)$ and $\mathsf{club}(x, y)$ to set $\mathcal{A}$ to represent the user's target application. □

This simple model originated from the observation that an application benefits from a specific class of association rules, whose consequences include entities of some particular "types", *i.e.,* the labels in p_0. For instance, when using association rules in marketing [WZYY05] (resp. intrusion detection [TT06], disease diagnosis [NITC13]), the consequences only need to indicate that a customer buy a product (resp. a signature is generated by an attack attempt, a person is healthy or sick). A similar model has been adopted and shown effective in similarity search in heterogeneous network [SHY+11]. It abstracts nodes and edges to labels to define similarity measure.

Support. The support of a GAR $\varphi = Q[\bar{x}](X \to p_0)$ in a graph G indicates how often φ can be applied to G. As shown in [FHLL20], the conventional notion of support for relational rules is not anti-monotonic for rules such as GFDs on graphs. Below we further revise the notion of support of GFDs [FHLL20] for $\mathcal{A}$-relevant GARs. We consider connected pattern Q as commonly found in graph rules.

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

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

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

Extending the support of [FHLL20, FWWX15] that counts the number of matches at a single designated variable from $Q[\bar{x}]$, we treat both x_{p_0} and x'_{p_0} in p_0 as "pivots"

to better estimate the effectiveness of $\mathcal{A}$-relevant φ, *e.g.,* whether application $\mathcal{A}$ benefits more from those φ having larger support. Here pivots refer to designated focus nodes representing users' interest. Below we show that this measure has the anti-monotonicity under a well-defined ordering of GARs.

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

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

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

Lemma 3.1: *Given two* GARs φ_1 *and* φ_2*, if* $\varphi_1 \preceq \varphi_2$*, then for any graph* G*,* $\mathsf{supp}(\varphi_1, G) \geq \mathsf{supp}(\varphi_2, G)$. □

Proof: Let $\varphi_1 = Q_1[\bar{x}_1](X_1 \to p_0)$ and $\varphi_2 = Q_2[\bar{x}_2](X_2 \to p_0)$. Since $\varphi_1 \preceq \varphi_2$, we have that $Q_1 \sqsubseteq Q_2$. Therefore, for every graph G, each match of Q_2 in G at consequence p_0 is also a match of Q_1 in G at p_0. Then $Q_2(G, X_2 \wedge p_0, p_0) \subseteq Q_1(G, X_1 \wedge p_0, p_0)$, as precondition X_2 covers all predicates in X_1. Hence $\mathsf{supp}(\varphi_1, G) \geq \mathsf{supp}(\varphi_2, G)$. □

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

Example 3.4: Recall GAR φ_a from Example 3.1. Consider GAR φ'_a revised from φ_a by removing pattern edge $(x_0, \mathsf{friend}, x_{k_0})$ from Q_a and predicate $\mathcal{M}_a(x_1, x_2, \mathsf{similar\_profile})$ from X_a. $\varphi'_a \preceq \varphi_a$ and φ'_a induces more matches projected at the consequence. □

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

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

subset Σ^c is minimal without any redundant GARs.

Discovery problem. It is intractable to find a cover of $\mathcal{A}$-relevant GARs from a graph [FHLL20], since the problem is already intractable for GFDs and GFDs are a special case of GARs. To speed up this process, we discover GARs from a set H of sample graphs $H(\mathcal{A}, \rho\%)$ extracted from G such that (a) the size $|H(\mathcal{A}, \rho\%)|$ accounts for $\rho\%$ of $|G_{\mathcal{A}}|$ and (b) $H(\mathcal{A}, \rho\%)$ has representative data pertaining to $\mathcal{A}$.

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

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

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

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

Intuitively, the set Σ_H is accurate, since at least $\gamma\%$ of the $\mathcal{A}$-relevant GARs with support no smaller than σ in the entire graph G are covered by Σ_H. Following [FHLL20], we adopt parameter k to balance the complexity of discovery and the interpretability of GARs. It is an input parameter decided by user's demand in practice.

Remark. (1) As a hard constraint, a GAR $Q[\bar{x}](X \to p_0)$ is satisfied by a graph G only if all matches of Q in G satisfy the dependency $X \to p_0$ (see Section 3.2). However, when we discover GARs from a possibly dirty graph G, we cannot expect that the GARs mined are satisfied by all matches. Hence we only discover frequent GARs with support above a given threshold, regardless of whether G satisfies them, following the common practice of data mining. Domain experts will then examine the discovered GARs before the rules can be applied to association deduction. (2) Discovering GARs

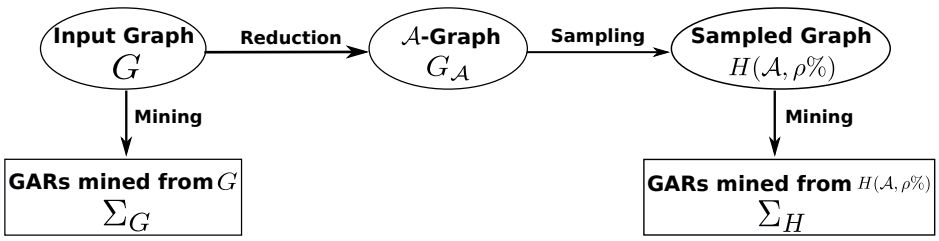


Figure 3.2: Workflow of GAR discovery

involves mining of both their patterns and dependencies. The former is similar to frequent subgraph mining [ABH14], which defines support by counting distinct matches of certain "pivots", conducts subgraph enumeration during the discovery, and may terminate early based on the anti-monotonicity of support. Besides, data dependencies $X \to p_0$ are discovered on entities identified by the matches of Q, by incorporating predicates such as value bindings and ML classifiers.

Discovery scheme. We propose a 3-step scheme to discover GARs, whose workflow is depicted in Figure 3.2.

Application-driven reduction (Section 3.4). The first step of the scheme deduces a graph $G_{\mathcal{A}}$ from G, referred to as the *$\mathcal{A}$-graph* of G, by retaining only the data pertaining to the given application $\mathcal{A}$.

Graph sampling (Section 3.5). Since $G_{\mathcal{A}}$ may still be large, the scheme deduces samples $H(\mathcal{A}, \rho\%)$ from $G_{\mathcal{A}}$ to further reduce the cost. We propose a sampling method to ensure support and recall bounds.

Mining (Section 3.6). The final step is to discover $\mathcal{A}$-relevant GARs Σ_H from small samples by using a parallelly scalable algorithm, where Σ_H is the union of the GARs mined from sample graphs $H(\mathcal{A}, \rho\%)$. It computes and returns a cover Σ_H^c of Σ_H.

3.4 Application Driven Discovery

In this section, we show how to reduce graphs G to smaller $\mathcal{A}$-graphs $G_{\mathcal{A}}$ for a given application $\mathcal{A}$, via an ML-based method.

We start with some notations, and then present the method.

Label triplets. A *label triplet* is defined as $\langle l_v, l_e, l'_v \rangle$, where l_v and l'_v are two node labels and l_e is an edge label in between. We say that an edge $e = (v, l, v')$ in graph G *conforms to* a label triplet $t = \langle l_v, l_e, l'_v \rangle$ if $L(v) = l_v$, $l = l_e$ and $L(v') = l'_v$. Here the special wildcard '_' also "equals" any arbitrary label. We refer to $\langle L(v), l, L(v') \rangle$ as the

label triplet $\mathcal{T}(e)$ *of edge* e. For a set T of label triplets, an edge e *conforms to* T if there exists a triplet $t \in T$ such that e conforms to it. A graph G *conforms to* the set T of label triplets if each edge e in G conforms to T.

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

Intuitively, for an application $\mathcal{A}$ modeled as a set of predicates (Section 3.3), the label triplets of predicates form a simple abstraction of the application $\mathcal{A}$. We opt to use language (ML) models to learn and analyse the distribution of label triplets created from application $\mathcal{A}$, which indicates the characteristics of the data pertaining to $\mathcal{A}$. We introduce label triplets to bridge the language models, applications and graphs. As will be seen shortly, it also strikes a balance between the efficiency and effectiveness in selecting data relevant to $\mathcal{A}$.

Example 3.5: Continuing with Example 3.1, the label triplets of predicates $x_0.\mathsf{city} = x_i.\mathsf{city}$ and $\mathcal{M}_a(x_i, x_j, \mathsf{similar\_profile})$ in φ_a are $\{\langle \mathsf{user}, \_, \mathsf{user}\rangle\}$ and $\{\langle \mathsf{user}, \mathsf{similar\_profile}, \mathsf{user}\rangle\}$, respectively. □

ML models and graph reduction. Given an application $\mathcal{A}$, *i.e.,* a set of predicates, a well-trained ML classifier $\mathcal{M}(x,y,l)$ for *e.g.,* link prediction, and a graph G, we employ a language model $\mathcal{M}_{\mathcal{A}}$, implemented as long short-term memory (LSTM) networks [HS97], to deduce the $\mathcal{A}$-graph $G_{\mathcal{A}}$ in the following four stages.

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

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

quences are *semantically* related to $\mathcal{T}(p)$.

(3) We select the top-m frequent triplets from $\Theta_{\mathcal{A}}$ to construct a set $T_{\mathcal{A}}$ of label triplets, referred to as $\mathcal{A}$-*triplets*. Here m is a predefined positive integer (see Section 3.7 for the impact of m on application-driven discovery). That is, we focus on triplets that are *most closely* related to application $\mathcal{A}$. Such $\mathcal{A}$-triplets and the triplets of the predicates in $\mathcal{A}$ co-occur with high probability. Thus it is very likely that the $\mathcal{A}$-relevant GARs include predicates related to these label triplets, and the (pattern) edges in such GARs also conform to them.

(4) We finally deduce $\mathcal{A}$-graph $G_{\mathcal{A}}$ from graph $G_{\mathcal{M}}$ by preserving only those edges that conform to $T_{\mathcal{A}}$. In particular, all the attributes of a node v are kept if one of v's adjacent edges in $G_{\mathcal{M}}$ conforms to $T_{\mathcal{A}}$. Filtered by label triplets in $T_{\mathcal{A}}$, graph $G_{\mathcal{A}}$ conforms to $T_{\mathcal{A}}$ and contains only vertices, edges and attributes pertaining to the application $\mathcal{A}$. Thus it suffices to discover $\mathcal{A}$-relevant GARs from $G_{\mathcal{A}}$.

The reduction takes a time linear to the number of generated triplets to run LSTM model and $O(|E_{\mathcal{M}}|)$ time to filter out irrelevant edges. Here we choose LSTM network since it can effectively model the semantics of labels on paths in knowledge graphs [LZL$^+$20, LSX18, LLL$^+$15]. That is, given an edge label l, LSTM generates a path following l with reasonable semantic meaning [MDB17]. Note that $\mathcal{M}_{\mathcal{A}}$ can also be implemented by other language models for sequence modeling [OMK20].

Example 3.6: Using label triplet $\{\langle \mathsf{user}, \mathsf{colleague}, \mathsf{user}\rangle\}$ of the consequence predicate $\mathsf{colleague}(x_0, x_0')$ in φ_a of Example 3.1 as seed input, $\mathcal{M}_{\mathcal{A}}$ outputs sequences $\Theta_{\mathcal{A}}$ of label triplets, in which the top-4 frequent triplets are $\{\langle \mathsf{user}, \_, \mathsf{user}\rangle\}$, $\{\langle \mathsf{user}, \mathsf{similar\_profile}, \mathsf{user}\rangle\}$, $\{\langle \mathsf{user}, \mathsf{friend}, \mathsf{user}\rangle\}$ and $\{\langle \mathsf{user}, \mathsf{follow}, \mathsf{organization}\rangle\}$. These make the set $T_{\mathcal{A}}$. Hence we only preserve edges in $G_{\mathcal{M}}$ that conform to $T_{\mathcal{A}}$ for the $\mathcal{A}$-graph $G_{\mathcal{A}}$, where data irrelevant to $\mathcal{A}$, *e.g.,* commute methods and dietary preferences in $G_{\mathcal{M}}$, is dropped. □

Model training. To train the language model $\mathcal{M}_{\mathcal{A}}$, *i.e.,* LSTM, we prepare a training corpus $\mathcal{D}_T$, which consists of sequences of label triplets that are collected by random walks in graph $G_{\mathcal{M}}$. More specifically, for each path $(e_1, e_2, \ldots, e_n)$ generated by a random walk in $G_{\mathcal{M}}$, we derive the label triplet of each edge and add sequence $(\mathcal{T}(e_1), \mathcal{T}(e_2), \ldots, \mathcal{T}(e_n))$ to training corpus $\mathcal{D}_T$ (here a path is a list of consecutive edges $e_i = (v_i, l_i, v_{i+1})$ for $i \in [1, n]$). The label triplets in each such sequence are semantically related, whose distribution can be learned by model $\mathcal{M}_{\mathcal{A}}$. Here we ap-

ply non-backtracking random walks (NBTRW) [LXE12] to sample the paths since NBTRW restrains the bias towards visiting high-degree nodes and closely knit communities around the seed nodes. This property helps us capture more representative localized structures in graph $G_{\mathcal{M}}$.

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

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

The training of model $\mathcal{M}_{\mathcal{A}}$ benefits from the usage of label triplets. Without them, $\mathcal{M}_{\mathcal{A}}$ has to inspect the triplets of attribute values, *i.e.,* $F_A(v)$, which are more diversified compared to labels $L(v)$. For example, different player nodes may have various names but the same label "Player". This larger vocabulary in the training corpus results in harder and slower training of $\mathcal{M}_{\mathcal{A}}$. In fact, abstracting edges and predicates as label triplets suffices to characterize the objective of various applications. For instance, a recommendation application would only concern about recommending "Sports Shoes" to "Players" without considering their specific names.

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

3.5 Sampling Big Graphs

In this section, we develop a sampling method to deduce a set H of sample graphs $H(\mathcal{A}, \rho\%)$ from $\mathcal{A}$-graph $G_{\mathcal{A}}$, such that the GARs mined from H satisfy the expected bounds on support and recall in graph G. We start with an overview of the sampling framework (Section 3.5.1) and introduce underlying techniques (Section 3.5.2). We then show the accuracy guarantees offered by it (Section 3.5.3).

3.5.1 Overview

For an application $\mathcal{A}$, an $\mathcal{A}$-relevant GAR $\varphi = Q[\bar{x}](X \rightarrow p_0)$ aims to "promote" the action specified by its consequence p_0. Thus when sampling big graphs G, we take distinct matches of variables of p_0 in G as the "pivots". This is justified by the support of φ, which is measured by the number of the pivots (see Section 3.3). Intuitively, sampling the pivots helps us discover GARs with a high support.

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

A sampling framework. Based on pivot sets, we propose a Graph Sampling framework for Rule Discovery of GARs, denoted as GSRD. Algorithm 3.1 shows the main steps of GSRD. The input of GSRD includes an $\mathcal{A}$-graph $G_{\mathcal{A}}$ deduced for application $\mathcal{A}$ (Section 3.4), the number N of sample graphs, two strategies M_v and M_s for sampling pivots and their surrounding subgraphs, respectively, as well as two sample ratios $\rho_v\%$ and $\rho\%$ for limiting the number of nodes and the sizes of sample graphs, respectively. It computes a set H of N sample graphs $H(\mathcal{A}, \rho\%)$ such that $|H(\mathcal{A}, \rho\%)| \leq \rho\% \times |G_{\mathcal{A}}|$, in N rounds.

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

(1) The first phase targets the pivot sets. More specifically, GSRD calls procedure PSample to sample pivots from $\mathcal{C}$, stored in a set $S_{\mathcal{A}}$ (line 6). PSample applies an input sampling strategy M_v (to be given in Section 3.5.2) to compute $S_{\mathcal{A}}$, and ensures that at most $\rho_v\%$ of the nodes from set $\mathcal{C}$ appear in the sampled pivots.

(2) In the second phase, GSRD samples substructures of selected pivots from $G_{\mathcal{A}}$. It picks nodes and edges within k hops from the nodes in $S_{\mathcal{A}}$ to build sample graph $H(\mathcal{A}, \rho\%)$, via procedure LSample (line 7). Such sampled data cells constitute the "substructures" surrounding the pivots; and $H(\mathcal{A}, \rho\%)$ includes all pivots and their substructures. Procedure LSample adopts another input strategy M_s to extract substructures (see Section 3.5.2), which is a linear time operation in the worst case. In

Algorithm 3.1: GSRD

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

Output: A set H of N sample graphs $H(\mathcal{A}, \rho\%)$.

```
1  i ← 0;   H ← nil;
2  repeat
3      C ← nil;
4      foreach predicate p_0 involved in A do
5          C ← C ∪ PS(p_0, G_A);   /* computing pivot sets */
6      S_A ← PSample(C, M_v, G_A, ρ_v%);
7      H(A, ρ%) ← LSample(S_A, M_s, G_A, ρ%);
8      H ← H ∪ {H(A, ρ%)};
9      i ← i + 1;
   until i = N;
10 return H;
```

addition, LSample guarantees that the size of $H(\mathcal{A}, \rho\%)$ is at most $\rho\% \times |G_{\mathcal{A}}|$.

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

Cost analysis. The cost for computing pivot sets is bounded by $O(|G_{\mathcal{A}}|)$ because it only needs label checking at constant times. Observe that extracting substructures is confined within the small localized areas around the pivots only in GSRD; thus the two phases for sampling pivots and substructures take at most $O(|H(\mathcal{A}, \rho\%)|^2 \log(|H(\mathcal{A}, \rho\%)|))$ time, including the sorting cost for applying locality-aware sampling (Section 3.5.2).

3.5.2 Representative Strategies

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

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

Clustering-assisted sampling. We propose to first cluster all pivots in the set $\mathcal{C}$ into multiple groups, such that each group consists of semantically similar pivots. We then construct a set of representative pivots by picking elements from *every* group guided

by the ratio $\rho_v\%$, by employing one of two sampling strategies. Below we first show how to cluster pivots and then present the sampling strategies.

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

Example 3.7: Figure 3.3(a) depicts a graph conversion process, in which the three directed edges are encoded as nodes r_0, r_1 and r_2 in the undirected graph. Here r_2 is connected to both r_0 and r_1 because of the nodes u_0 and p_1 that are shared by multiple edges. □

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

For efficient clustering, we adopt Lloyd's k-means algorithm [Llo82] with k-means++ seeding [AV06]. We also use two approaches to extracting node features for clustering. One takes mean word embeddings [PSM14] of the node attributes as the feature, since an application usually involves nodes with similar semantic meanings. The other learns node features with Deep Graph Informax (DGI) [VFH$^+$19], where topological structure and node attributes are considered.

Uniform sampling. We may select pivots from each group in *a uniform manner*, by randomly selecting each pivot independent of the others. Note that when sampling the pivots of edge predicates, it only considers those node pairs that are connected by edges in the $\mathcal{A}$-graph $G_{\mathcal{A}}$. By the semantics of GARs, only such pivots help us discover GARs that have edge predicates as the consequences. After uniform sampling, it is assured that the selected pivots cover all the semantics pertaining to the given application $\mathcal{A}$ and more pivots are picked out from larger groups.

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

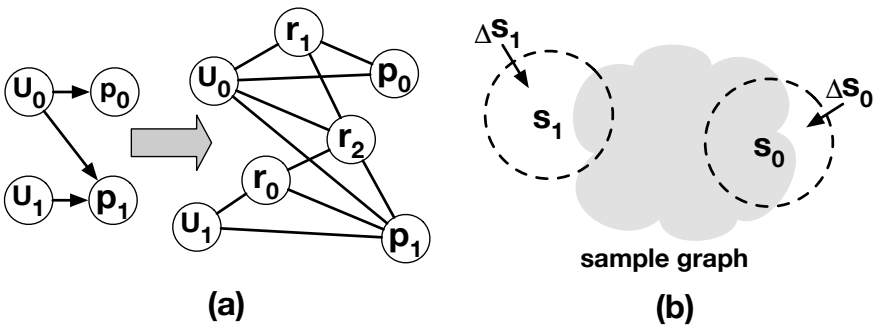


Figure 3.3: Demonstration of different strategies in GSRD

extraction. As another consequence, more pivots can be included in sample graphs of a fixed size, from which we can mine more GARs.

Example 3.8: The locality-aware sampling prefers the pivot s_0 to s_1 shown in Fig. 3.3(b). This is because the changes Δs_0 induced by s_0 to the sample graph is smaller than that induced by s_1, *i.e.,* Δs_1. □

Substructure extraction. We can use breadth-first search (BFS) as strategy M_s for extracting substructures of the pivots selected by M_v above. Starting from a sampled pivot v, the BFS proceeds up to the fixed depth k, and fetches the entire k-hop neighborhood of v as its substructure. We also remark that there exist other optimized extraction strategies for M_s (see details in Section 3.7).

3.5.3 Accuracy Guarantee

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

Characterization. We start with an observation. As observed in [New05], real-life graphs often have a power-law degree distribution, *i.e.,* a small number of nodes have much higher degree than the others. The application of GARs is analogous, *i.e.,* a small number of nodes are involved in the matches of patterns in most GARs. In practice, nodes with larger degrees are more likely to be matched by the patterns of GARs. Thus to have a high recall when mining GARs from sample graphs, we need to sample such critical nodes as many as possible under a bound on the total number of sampled nodes.

We next formalize this observation to estimate the recall.

Formalization. Given a GAR $\varphi = Q[\bar{x}](X \to p_0)$ and a node v in graph G, we say that v is a *pivot* of φ and *contributes to* the support of φ in G if v appears in the pivot set $\mathsf{PS}(\varphi, G)$. Here the *pivot set* $\mathsf{PS}(\varphi, G)$ *of* φ *in* $G_{\mathcal{A}}$ is defined as $Q(G, X \wedge p_0, p_0)$ (see Section 3.3), such that the support of φ in G equals the cardinality $\|\mathsf{PS}(\varphi, G)\|$. It helps analyse the accuracy bound below and is different from the previous notion of pivot set defined *w.r.t.* a single predicate p.

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

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

Here Δ can be estimated by using parameter estimation methods for power-law distribution, *e.g.,* [CSN09]. Intuitively, the larger Δ is, the fewer critical nodes can contribute to the support of most GARs.

Moreover, for each node v sampled by GSRD in the first phase and any $\mathcal{A}$-relevant GAR φ with $\mathsf{supp}(\varphi, G_{\mathcal{A}}) \geq \sigma$, $v \in \mathsf{PS}(\varphi, G_{\mathcal{A}})$ if and only if $v \in \mathsf{PS}(\varphi, H_v)$. Here H_v denotes the substructure of v extracted via BFS, *i.e.,* the entire k-hop neighborhood of v.

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

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

Theorem 3.1: *For an $\mathcal{A}$-graph $G_{\mathcal{A}}$, an expected recall value $\gamma\%$, support threshold σ for required $\mathcal{A}$-relevant* GARs $\Sigma_{G_{\mathcal{A}}}$ *in $G_{\mathcal{A}}$, and a constant $\varepsilon \in (0,1)$, if the support threshold for $\mathcal{A}$-relevant* GARs *is $\sigma' = \lceil \frac{\|V_H\|}{\|V_{\mathcal{A}}\|}\sigma(\gamma\%)^{1/\Delta} + 1 \rceil$ in sample graphs, then after creating a set H of $N = \lceil \ln\varepsilon \,/ \left(1 - \exp(-\frac{(\gamma\%)^{1-\frac{1}{\Delta}}\left(\|V_H\|\sigma(\gamma\%)^{\frac{1}{\Delta}} - \|V_{\mathcal{A}}\|\sigma'\right)^2}{3\|V_H\|\|V_{\mathcal{A}}\|\sigma}\right)\rceil$ sample graphs by* GSRD *with BFS as substructure extraction strategy,* $\mathsf{recall}(\Sigma_H, \Sigma_{G_{\mathcal{A}}}, \sigma) \geq \gamma\%$ *with probability* $1 - \varepsilon$. □

Proof: We prove this theorem in two steps: (1) when discovering GARs with the com-

puted support threshold σ' in a sample graph deduced via GSRD, $\mathsf{recall}(\Sigma_H, \Sigma_{G_\mathcal{A}}, \sigma)$ is no smaller than $\gamma\%$ with probability p_x; and (2) after mining GARs from $N = \lceil \frac{\ln(1-\varepsilon)}{\ln(1-p_x)} \rceil$ sample graphs deduced via GSRD in the same setting, the probability reaches $1-\varepsilon$.

(1) We first verify probability p_x. Let X_i be the number of $\mathcal{A}$-relevant GARs φ in $\Sigma_{G_\mathcal{A}}$ such that (a) φ can be found from the sample graph, and (b) the i-th node v_i sampled from $G_\mathcal{A}$ is in $\mathsf{PS}(\varphi, G_\mathcal{A})$. As the substructure H_{v_i} extracted by GSRD via BFS includes all the information for GAR discovery, the support of φ in the sample graph could increase when v_i is sampled from $G_\mathcal{A}$. Then after randomly sampling $\|V_H\|$ nodes from $G_\mathcal{A}$ in the first phase of GSRD, the percentage of required GARs in $\Sigma_{G_\mathcal{A}}$ that can be discovered from the sample graph is $X = X_1/\|\Sigma_{G_\mathcal{A}}\| + \cdots + X_{\|V_H\|}/\|\Sigma_{G_\mathcal{A}}\|$. Here we normalize the number X_i of discovered GARs in the range 0 to 1, since we will bound the probability p_x using the Chernoff bound [MU05], which can only be applied to variables distributed in $[0,1]$ (see below). In addition, we can verify that the expected value $\mathcal{E}(X)$ of X is bounded by $\frac{\|V_H\|}{\|\Sigma_{G_\mathcal{A}}\|} \times \frac{(\gamma\%)\|\Sigma_{G_\mathcal{A}}\|\sigma}{(\rho_{\max}\%)\|V_\mathcal{A}\|} \times \rho_{\min}\% = \frac{\|V_H\|}{\|V_\mathcal{A}\|}\sigma(\gamma\%)^{1+1/\Delta}$, where (a) the second term is the average number of GARs to which a sampled node v contributes; note that $(\gamma\%)\|\Sigma_{G_\mathcal{A}}\|$ is the minimum number of GARs needed to ensure that $\mathsf{recall}(\Sigma_H, \Sigma_{G_\mathcal{A}}, \sigma)$ is at least $\gamma\%$, $(\gamma\%)\|\Sigma_{G_\mathcal{A}}\|\sigma$ is the minimum number of nodes needed to ensure that $(\gamma\%)\|\Sigma_{G_\mathcal{A}}\|$ many GARs can be discovered from the sampled graph, and $(\rho_{\max}\%)\|V_\mathcal{A}\|$ is the maximum number of nodes that contribute to the support of an $\mathcal{A}$-relevant GARs; (b) the third term $\rho_{\min}\%$ approximates the probability that a sampled node is in $\mathsf{PS}(\varphi', G_\mathcal{A})$ for some GAR φ' in $\Sigma_{G_\mathcal{A}}$; and (c) the numerator of the first term is the number of sampled nodes which derives the average number of required GARs that can be discovered from the sampled data after multiplying with the following two terms; the denominator of the first term makes the entire expression as the percentage of required GARs in $\Sigma_{G_\mathcal{A}}$ that can be discovered from the sample graph. The expectation of X is no less than the above expression, since the numerator of the second term is picked as its minimum possible value, the denominator of the second term is its maximum possible value, and the third term approximates its minimum probability value. That is, the lower bound of the expectation of variable X is $\frac{\|V_H\|}{\|V_\mathcal{A}\|}\sigma(\gamma\%)^{1+1/\Delta}$.

Then using the Chernoff bound [MU05], we have the following:

$$\mathbf{Pr}[X \geq (1+\varepsilon_x)\mathcal{E}(X)] \leq \exp\left(-\frac{\varepsilon_x^2 \mathcal{E}(X)}{3}\right) \leq \exp\left(-\frac{\varepsilon_x^2 \|V_H\|\sigma(\gamma\%)^{1+1/\Delta}}{3\|V_\mathcal{A}\|}\right).$$

When $X < (1+\varepsilon_x)\mathcal{E}(X)$ with probability $1-p_x$, we have that $p_x = \exp\left(-\frac{\varepsilon_x^2\|V_H\|\sigma(\gamma\%)^{1+1/\Delta}}{3\|V_\mathcal{A}\|}\right)$; then $\varepsilon_x = \sqrt{-\frac{3\|V_\mathcal{A}\|\ln p_x}{\|V_H\|\sigma(\gamma\%)^{1+1/\Delta}}}$. On the other hand, X

should be greater than $\gamma\% \times \sigma'$ (*i.e.,* the minimum total support of discovered GARs) with high probability; since we have previously obtained the lower bound of X as $\frac{\|V_H\|}{\|V_{\mathcal{A}}\|}\sigma(\gamma\%)^{1+1/\Delta}$, thus we set $\gamma\% \times \sigma' = (1+\varepsilon_x)\frac{\|V_H\|}{\|V_{\mathcal{A}}\|}\sigma(\gamma\%)^{1+1/\Delta}$ and let the support threshold $\sigma' = \lceil \frac{\|V_H\|}{\|V_{\mathcal{A}}\|}\sigma(\gamma\%)^{1/\Delta} + 1 \rceil$. Putting these together, we conclude that $p_x = \exp\left(- \frac{(\gamma\%)^{1-\frac{1}{\Delta}} \left(\|V_H\|\sigma(\gamma\%)^{\frac{1}{\Delta}} - \|V_{\mathcal{A}}\|\sigma' \right)^2}{3\|V_H\|\|V_{\mathcal{A}}\|\sigma} \right)$.

(2) After discovering from a single sample graph in H, the recall reaches $\gamma\%$ with probability p_x. Then if we discover from N sample graphs, the corresponding probability becomes $1-(1-p_x)^N = 1-\varepsilon$. Thus $N = \lceil \log_{1-p_x} \varepsilon \rceil = \lceil \frac{\ln \varepsilon}{\ln 1-p_x} \rceil$, and N is an integer. □

Remark. (1) Note that the percentage of $\|V_H\|$ in $\|V_{\mathcal{A}}\|$ provides a guideline to determine the sample ratio $\rho_v\%$, guided by Theorem 3.1. (2) One can verify that for each GAR φ in Σ_H that has support σ' in sample graph $H(\mathcal{A}, \rho\%)$ of H, φ has support of at least σ' in the entire graph G, since each $H(\mathcal{A}, \rho\%)$ is essentially a subgraph of G.

Example 3.9: Consider an $\mathcal{A}$-graph $G_{\mathcal{A}}$ deduced from the citation network DBLP [Sch], which includes $\|V_{\mathcal{A}}\| = 16M$ nodes. Suppose that the expected support threshold σ is 50 on $G_{\mathcal{A}}$, we need a recall of 90% *w.r.t.* support 50, and $\frac{\rho_{\min}\%}{\rho_{\max}\%} = 0.09$. By Theorem 3.1, we can see that to achieve this expected recall value, it suffices to create $N = 9$ sample graphs by GSRD such that $\|V_H\| = 4.8M$, and set the support threshold $\sigma' = 3$ for sample graphs. □

3.6 Parallel Discovery

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

Sequential mining. To see the challenges inherent to GAR discovery, we start with a sequential algorithm for mining GARs, denoted as GARMine, by extending the GFD discovery algorithm of [FHLL20]. GARMine processes the N sample graphs in H one by one to mine GARs and returns their union. On each sample graph, it interleaves levelwise *pattern expansion* and *dependency expansion* to generate patterns Q and dependencies $X \to p_0$ for candidate GARs, respectively, following [FHLL20]. Apart from the constant and variable predicates of GFDs, here dependency expansion also includes new ML, attribute and edge predicates in GARs (see details shortly). GARMine returns

those candidate GARs having support above the threshold σ', which is determined by Theorem 3.1.

GARMine takes exponential time in the worst case due to graph homomorphism needed in GAR validation (cf. [GJ79]). To speed it up, we parallelize the discovery process. To measure the effectiveness of the parallelization, we review the notion of parallel scalability.

Parallel scalability. We adapt the parallel scalability of [KRS90] to characterize the effectiveness of parallel algorithms for GAR discovery. Denote by $T_{\mathsf{seq}}(|H|,k,\sigma')$ the worst-case cost of a sequential GAR discovery algorithm $\mathbb{A}$. Given a set H of sample graphs, support threshold σ' and integer k, $\mathbb{A}$ mines minimum GARs from H such that each GAR has at most k pattern nodes and a support at least σ' in the samples. We say that a parallel algorithm $\mathbb{A}_p$ for GAR discovery is *parallelly scalable relative to* $\mathbb{A}$ if with n processors,

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

where $T_{\mathsf{par}}(|H|,k,\sigma',n)$ denotes the worst-case parallel runtime of $\mathbb{A}_p$. Intuitively, parallel scalability measures the speedup over a yardstick sequential algorithm $\mathbb{A}$ by parallelization. A parallelly scalable $\mathbb{A}_p$ "linearly" reduces the runtime of $\mathbb{A}$ when n increases.

The main result of this section is the following.

Theorem 3.2: *There exists an algorithm* ParGARMine *for* GAR *discovery that is parallelly scalable relative to* GARMine. □

Proof: Observe that due to the pattern matching for validating candidate GARs, sequential GARMine takes $O(NF_{\ell_q}|H(\mathcal{A},\rho\%)|^{\ell_q})$ time to discover GARs having patterns with ℓ_q nodes from samples. Here F_{ℓ_q} denotes the number of candidates GARs with ℓ_q pattern nodes. Hence it suffices to prove that the parallel cost incurred by ParGARMine is in $O(NF_{\ell_q}|H(\mathcal{A},\rho\%)|^{\ell_q}/n)$.

By adopting the workload balancing strategy of [FHLL20], parallel matching of candidate patterns with ℓ_q nodes in ParGARMine takes $O(F_{\ell_q}|H(\mathcal{A},\rho\%)|^{\ell_q}/(n/N)) = O(NF_{\ell_q}|H(\mathcal{A},\rho\%)|^{\ell_q}/n)$ time. Since the resulting matches are also evenly distributed across processors, transmitting edges for validating edge and ML predicates can be done in $O(NF_{\ell_q}|H(\mathcal{A},\rho\%)|^{\ell_q}/n)$ time. From these the parallel scalability of ParGARMine relative to GARMine follows.

Although ParGARMine inspects N sample graphs $H(\mathcal{A},\rho\%)$, it is still faster than parallel GAR discovery on the entire $\mathcal{A}$-graph $G_{\mathcal{A}}$, especially when mining GARs with large patterns. Indeed, when $N \leq (\frac{|G_{\mathcal{A}}|}{|H(\mathcal{A},\rho\%)|})^{\ell_q}$, the parallel cost for validating a candidate GAR with ℓ_q pattern nodes is $O(N|H(\mathcal{A},\rho\%)|^{\ell_q}/n)$, which is smaller than $O(|G_{\mathcal{A}}|^{\ell_q}/n)$, the cost in processing the entire $G_{\mathcal{A}}$. □

Algorithm 3.2: ParGARMine

Input: A set H of N sample graphs $H(\mathcal{A},\rho\%)$, processors $P_1,\ldots,P_n$, positive integer k and support threshold σ'.

Output: A set Σ_H of all minimum GARs in H such that each has at most k pattern nodes and a support at least σ' in H.

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

Parallel mining. Algorithm ParGARMine conducts levelwise expansion of patterns and dependencies simultaneously at a designated coordinator. It validates candidate patterns and GARs in parallel with n workers, since the expensive subgraph matching in validation dominates the cost of the discovery process. It extends parallel GFD discovery [FHLL20] by (a) partitioning multiple sample graphs, (b) supporting attribute, edge and ML predicates, and (c) applying new pruning strategies during the expansions.

The details of ParGARMine are shown in Algorithm 3.2. In contrast to discovering rules from a single graph, it employs n workers to mine a set Σ_H of GARs from the set H of N sample graphs, such that each GAR in Σ_H has support at least σ' and at most k pattern nodes. It starts by distributing the N sample graphs to n workers

(line 1). It evenly allocates computing resources to sample graphs, such that each sample is assigned a distinct set of $\lfloor \frac{n}{N} \rfloor$ processors except one that takes all the rest. We fragment and distribute each sample graph across $\lfloor \frac{n}{N} \rfloor$ workers via vertex-cut partitioning [ZWL+17].

Following the BSP model [Val90], ParGARMine next works in k^2 rounds to generate and validate GARs (lines 3-13). As k pattern nodes result in at most k^2 edges, each one of the k^2 rounds intends to find GARs with a specific number of pattern edges in $[1, k^2]$.

Pattern expansion. In each round ℓ_q, ParGARMine expands patterns at level ℓ_q by creating a set Q^{ℓ_q} of patterns with ℓ_q edges via procedure QExpand at coordinator P_c (line 4). QExpand generates Q^{ℓ_q} by expanding each pattern in Q^{ℓ_q-1} with a single new edge; initially the edges in Q^1 should conform to the triplets of predicates in application $\mathcal{A}$ if they are available (Section 3.4). ParGARMine then computes the matches of such patterns in sample graphs using the parallel pattern matching strategy of [FHLL20]; it prunes from Q^{ℓ_q} all patterns that have less than σ' matches in each sample, since they cannot appear in GARs that have support at least σ' (line 5).

Dependency expansion. Given patterns Q^{ℓ_q}, ParGARMine expands dependencies $X \rightarrow p_0$ at level ℓ_p; it combines them with Q^{ℓ_q} to produce candidate GARs, in ℓ_p^m iterations (lines 8-12). Here ℓ_p^m denotes the maximum number of predicates in X, estimated by combinations of possible predicates *w.r.t.* the pattern nodes in Q^{ℓ_q} (line 5). In each iteration ℓ_p, ParGARMine calls procedure PExpand at the coordinator to compute a set Σ^{ℓ_p} of GARs, such that each GAR in Σ^{ℓ_p} has a pattern from Q^{ℓ_q} and ℓ_p predicates in its precondition X ($X = \emptyset$ when $\ell_p = 0$) (line 9), where X is expanded from a counterpart in GARs Σ^{ℓ_p-1} with one new predicate. To speed up the process, PExpand associates each pair of nodes with a set of predicates that they satisfy, similar to the evidence sets for discovering DCs [LHIK20].

ParGARMine validates GARs in Σ^{ℓ_p} by extending the parallel method of [FHLL20]. It adds to the set Σ_H those GARs that meet support bound σ' (lines 10-11).

Handling edge and ML predicates. In dependency expansion, procedure PExpand also generates possible attribute, edge and ML predicates that are unique to GARs. More specifically, for each candidate pattern Q and attribute dependency $X \rightarrow p_0$, PExpand expands X with $x.A$, $l(x,y)$ or $\mathcal{M}(x,y,l)$ for all x and y in Q, l in the label set Γ, A in the attribute set Υ, and ML classifiers $\mathcal{M}$ if applicable. Recall that $\mathcal{A}$-graph $G_{\mathcal{A}}$ already incorporates edges predicted by ML model (Section 3.4); hence when discovering $\mathcal{A}$-

relevant GARs, we can treat ML predicates and edge predicates in the samples of the $\mathcal{A}$-graph. All such expanded dependencies will be validated.

For GFDs, the parallel validation is only conducted on matches computed for patterns Q^{ℓ_q} [FHLL20]. In contrast, due to the edge and ML predicates introduced by GARs, we have to inspect the existence of additional edges, which may reside at different workers. Thus, if a match h at processor P_i involves two nodes v and v' and if we need to check the existence of an edge e from v to v' together with h, ParGARMine transmits e to P_i from other workers if it exists before the local checking, using an additional superstep of BSP.

Cover. We revise the parallel implication checking algorithm for GFDs [FHLL20] to compute the cover Σ_H^c of GARs Σ_H returned by ParGARMine, based on a characterization of GAR implication [FJL$^+$20].

Example 3.10: Consider GAR φ_a of Example 3.1 and suppose that its consequence predicate is covered by application $\mathcal{A}$. By the number of pattern edges, ParGARMine can find φ_a in round $2k_0+2$. More specifically, after generating the pattern Q_a in this round, it validates combinations of the predicates of Q_a to mine dependencies, including the one of φ_a that consists of $2k_0$ variable predicates, k_0^2 ML predicates and an edge predicate $\mathsf{colleague}(x_0, x_0')$. Note that the ML predicates predict similar_profile links, which have been added to the $\mathcal{A}$-graph in graph reduction. ParGARMine performs similarly in checking the ML and edge predicates of φ_a, *i.e.,* inspecting similar_profile and colleague links, with necessary communication when the links and matches of Q_a are not at the same worker. □

Pruning strategies. To reduce unnecessary expansion of patterns and dependencies that cannot yield minimum GARs *w.r.t.* support threshold σ', procedures QExpand and PExpand employ new pruning strategies in addition to those adopted in GFD discovery [FHLL20].

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

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

Both strategies leverage the anti-monotonicity of the support of GARs (Lemma 3.1). Without it, the algorithm easily expands a GAR φ' from φ such that $\varphi \preceq \varphi'$ but the support of φ cannot reach σ', which has already been verified in prior levels, *i.e.,* φ' is useless.

Analyses. To see that ParGARMine is correct, observe the following. (a) The parallel matching method of [FHLL20] ensures that the matches of candidate patterns computed at n processors are the same as that deduced sequentially. (b) All the data related to validating edge and ML predicates in a GAR is sent to the same processor in advance.

3.7 Experimental Study

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

Experimental setting. We start with the experimental setting.

Datasets. We used four real-life graphs: (1) DBLP [dbl21a], a real-life citation network with 0.2M nodes and 0.3M edges, where the attributes constitute bibliographic records of research papers in computer science; (2) YAGO [SKW07], a knowledge graph with 3.5M nodes and 7.4M edges; (3) DBpedia [dbp21a], a larger knowledge graph with 5.2M nodes and 17.5M edges; its attribute values indicate various types of facts related to the entities (nodes); and (4) IMDB [IMD21], a graph database that includes attributes for the information of movies, directors and actors, having 5.1M nodes and 5.2M edges.

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

Algorithms. We implemented the following, all in C++. (1) The graph reduction

method (Section 3.4). (2) Various graph sampling approaches $\mathsf{GSRD}_{\mathsf{y+z}}$ in framework GSRD (Section 3.5), where y denotes the strategy of GSRD for sampling pivots, including CA (cluster-assisted, where pre-trained Glove word embeddings [PSM14] or DGI representations [VFH+19] are used as node features) and LC (locality-aware); and z denotes the strategy for extracting substructures, including OB (BFS), WB (BFS with bounded width) and RW (random walk). For example, $\mathsf{GSRD}_{\mathsf{CA+RW}}$ samples pivots via clustering with word embeddings (by default) and extracts substructures via random walk. Here WB is a variant of OB, which takes an additional bound on the number of neighbors that each BFS step explores. It strikes a balance between the size and diameter of the substructure S, *i.e.,* the longest shortest distance between any two nodes in S, and helps us mine GARs with patterns of a large diameter. RW also takes two parameters: the depth k of random walk, and the size of substructure. It performs random walk from a sampled pivot with at most k steps, to find substructure subject to the specified size bound. RW is able to extract irregular substructures for pivots. (3) The parallel GAR discovery algorithm ParGARMine (Section 3.6). (4) A variant $\mathsf{ParGARMine}_{\mathsf{w}}$ of ParGARMine that discovers GARs from the entire graphs in parallel. (5) The parallel GFD discovery algorithm DisGFD [FHLL20], for efficiency comparison.

We also implemented three other baseline graph sampling strategies, also in C++. (6) UniNode uniformly samples nodes, controlled by a given sample ratio of the nodes. It returns the subgraph induced by all the selected nodes as a sample graph. (7) UniEdge, which extracts edges from graphs in a uniform manner to build samples, guided by a sample ratio. (8) PRA, which samples paths with a linear path ranking model to select nodes that are mostly related to the query nodes [LMC11]; we picked the query nodes by uniform node sampling, preserving all edges connected to the sampled nodes. In addition, we compared with (9) AnyBURL [MCRS19] for efficiency in mining Horn rules.

ML models. We used the SimplE link prediction model [KP18] as the ML classifier in GARs due to its high accuracy and efficiency. To train SimplE, we adopted the default configurations in [KP18], and took 85% and 15% of each graph as the training set and validation set, respectively. The trained model is used to recover missing links.

The LSTM model used for graph reduction was implemented as [MKS18] with its default training configuration and two 650 wide layers. We targeted the discovery of $\mathcal{A}$-relevant GARs for a specific application $\mathcal{A}$. By default we considered 7 predicates in a single application $\mathcal{A}$.

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

Table 3.2: Effectiveness of ML-based graph reduction

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

Experimental results. We next report our findings.

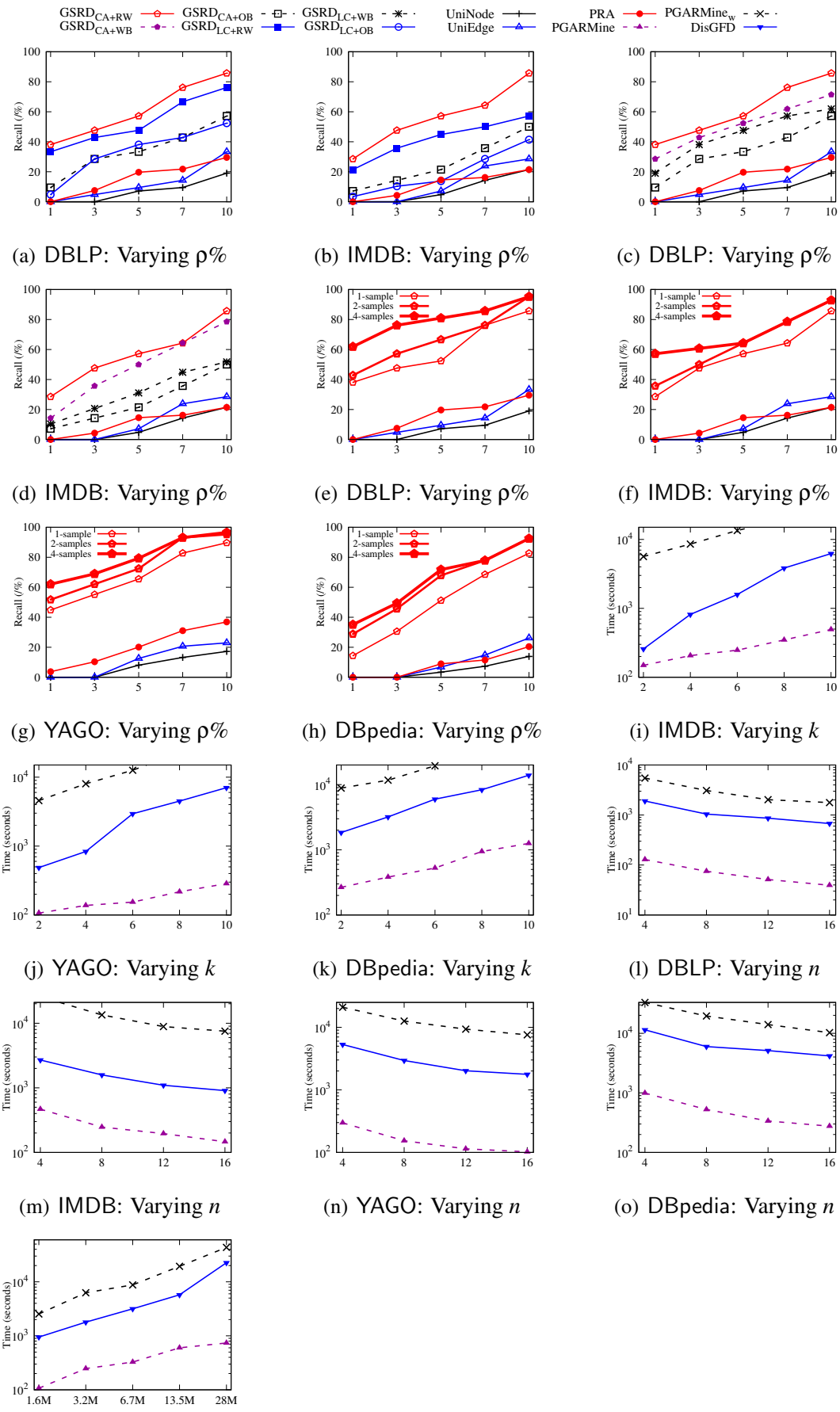
Exp-1: Effectiveness of application-driven reduction. We first evaluated the performance of our graph reduction strategy. The ML-based method selects the top-m frequent label triplets from a candidate set generated by $\mathcal{M}_{\mathcal{A}}$, which are most closely relevant to the application $\mathcal{A}$ (Section 3.4). We studied the impact of m on the effectiveness of graph reduction by varying m from 3 to 10. We fixed the upper bound $k = 6$ for pattern nodes, and set the same support threshold as 1000 when mining GARs from both the graphs G and $\mathcal{A}$-graphs $G_{\mathcal{A}}$ deduced by the reduction. The results on the four real-life graphs are reported in Table 3.2. We find the following.

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

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

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

These verify the effectiveness of the ML-based graph reduction. For all the other



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

(d) IMDB: Varying ρ% (e) DBLP: Varying ρ% (f) IMDB: Varying ρ%

(g) YAGO: Varying ρ% (h) DBpedia: Varying ρ% (i) IMDB: Varying k

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

(m) IMDB: Varying n (n) YAGO: Varying n (o) DBpedia: Varying n

(p) Synthetic G: Varying $|G|$

Figure 3.4: Performance evaluation

experiments, m was set to be 7 by default.

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

(1) Impact of pivot sampling. Fixing $k = 8$ and $N = 1$, we varied $\rho\%$ from 1% to 10% on DBLP and IMDB. Here the support thresholds σ' for discovery in sample graphs were determined by following the formula in Theorem 3.1, which is unaffected by the specific substructure extraction strategy and hence is applicable to both WB and RW as well. As shown in Figures 3.4(a) and 3.4(b), (a) $\mathsf{GSRD}_{\mathsf{CA+RW}}$ consistently performs the best among all the methods. (b) It outperforms $\mathsf{GSRD}_{\mathsf{LC+RW}}$ (resp. $\mathsf{GSRD}_{\mathsf{LC+OB}}$) by 24% (resp. 35%) on average in the recall of the discovered GARs, validating the need of node clustering in sampling pivots. We also find that clustering-assisted sampling exhibits little difference with different types of node features. It means that word embeddings, *i.e.,* node attributes, suffice to distinguish application-related pivots.

(2) Impact of substructure extraction. In the same setting, Figures 3.4(c) and 3.4(d) report the recall of GARs mined from the samples deduced by different strategies from DBLP and IMDB, respectively. We can see that (a) when combing clustering with various approaches for substructure extraction, $\mathsf{GSRD}_{\mathsf{CA+RW}}$ still offers the highest recall and $\mathsf{GSRD}_{\mathsf{CA+WB}}$ performs better than $\mathsf{GSRD}_{\mathsf{CA+OB}}$. (b) On DBLP, $\mathsf{GSRD}_{\mathsf{CA+RW}}$ is at least 8 (resp. 6 and 2.9) times more accurate than UniNode (resp. UniEdge and PRA), when $\rho\% = 5\%$. We examine the substructures extracted by RW and find that the semantics related to them, *e.g.,* node labels, is more diversified than those extracted by OB and WB, making it possible to find more GARs. That is, while the theoretical bounds (Theorem 3.1) are proved for $\mathsf{GSRD}_{\mathsf{CA+OB}}$, $\mathsf{GSRD}_{\mathsf{CA+RW}}$ achieves better bounds in practice since OB and WB introduce a bias towards high-degree nodes [KMT11].

The results on YAGO and DBpedia are consistent (not shown).

(3) Impact of N. Using the same k, σ and range of $\rho\%$ as in Exp-2(1), Figures 3.4(e) to 3.4(h) report the results with different number N of samples deduced from the real-life graphs. Here we only tested $\mathsf{GSRD}_{\mathsf{CA+RW}}$ for framework GSRD, which has

been verified to be the best combination in Exps (2)(a) and (2)(b) and this also holds when $N>1$. We find that (a) all sampling methods perform better with more sample graphs, as expected. (b) $\mathsf{GSRD}_{\mathsf{CA+RW}}$ on average beats UniNode (resp. UniEdge and PRA) by 8.0 (resp. 5.3 and 4.3) times when $\rho\%$ varies from 7% to 10% and using 2 samples, which is consistent with Figures 3.4(a) to 3.4(d). (c) The recall offered by $\mathsf{GSRD}_{\mathsf{CA+RW}}$ "converges" fast as N increases. For instance, on DBpedia, the recall already reaches 92% when N=2 and $\rho\%$=10%. These validate the effectiveness of GSRD for GAR discovery.

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

Exp-3: Efficiency. Using $n = 8$ machines, we tested the efficiency of ParGARMine in finding GARs from sample graphs, and compared it with $\mathsf{ParGARMine}_{\mathsf{w}}$ and DisGFD that operate on the entire graphs. We took the sample graphs deduced by $\mathsf{GSRD}_{\mathsf{CA+RW}}$ from IMDB, YAGO and DBpedia with sample ratio 10% and $N = 2$; the support thresholds were decided along the same lines as that in Exp-2(1). Figures 3.4(i) to 3.4(k) report the runtime for mining GARs with different upper bounds k for patterns. The corresponding time for building samples is on average 198 seconds. We find the following.

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

(2) It is feasible for ParGARMine to discover GARs with large patterns. From $N = 2$ sample graphs of YAGO (resp. IMDB), it takes ParGARMine 285 (resp. 492) seconds to find GARs with $k = 10$, while $\mathsf{ParGARMine}_{\mathsf{w}}$ cannot terminate in 7 (resp. 16) hours. Note that the deduced $\mathcal{A}$-graph of IMDB is large.

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

The results on DBLP are consistent and hence not shown.

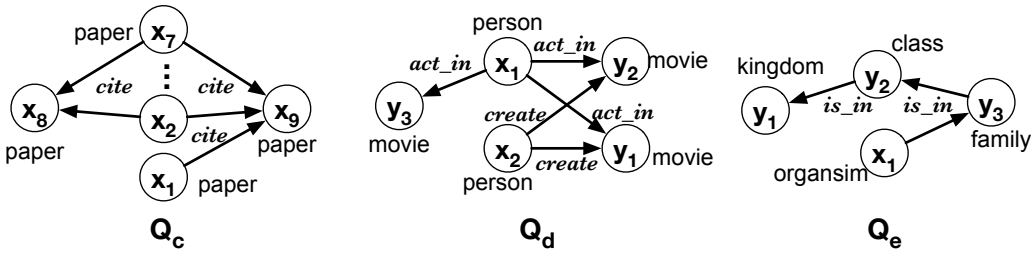


Figure 3.5: Discovered GARs

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

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

(1) Parallel scalability. Fixing $k = 6$ and employing the same $\rho\%$, σ and N as in Exp-3, we varied the number n of machines used from 4 to 16. As reported in Figures 3.4(l) to 3.4(o) on the four real-life graphs, respectively, (a) ParGARMine and $\text{ParGARMine}_{\text{w}}$ are on overage 3.2 and 3 times faster, respectively, *i.e.,* they scale well with n. (b) ParGARMine outperforms $\text{ParGARMine}_{\text{w}}$ by 58.3 times on average. These empirically verify Theorem 3.2 and further show the effectiveness of sampling-based GAR discovery.

(2) Larger graphs. Fixing $k = 6$, $n = 8$, $\sigma = 1000$, $\rho\% = 10\%$ and $N = 2$, we varied the size $|G| = |V| + |E|$ of synthetic graphs G from $1.6M$ to $28M$. The results in Fig. 3.4(p) show that sampling-based GAR discovery can scale with large graphs, *e.g.,* when $|G| = 28M$, it takes 735 seconds to mine GARs from two sample graphs.

Exp-5: Effectiveness of GARs. We manually checked the GARs discovered from real-life graphs. Below are some example GARs with support above 1000. Their graph patterns are shown in Fig. 3.5.

(1) In the samples of DBLP, we find a GAR $\varphi_c = Q_c[\bar{x}_c](\mathcal{M}(x_8, x_9, \text{similar\_topic}) \rightarrow \text{cite}(x_1, x_8))$. The GAR states that if all of the papers x_2, ..., x_7 cite both x_8 and x_9, x_1 cites x_9 (pattern Q_c), and if x_8 and x_9 are about similar topics (determined by ML predicate $\mathcal{M}(x_8, x_9, \text{similar\_topic})$), then x_1 also cites x_8.

(2) In IMDB, a GAR φ_d is $Q_d[\bar{x}_d](y_1.\text{language} = y_2.\text{language} \wedge y_1.\text{language} = y_3.\text{language} \wedge \mathcal{M}(y_1, y_2, \text{same\_series}) \wedge \mathcal{M}(y_2, y_3, \text{same\_series}) \rightarrow \mathcal{M}(x_2, y_3, \text{create}))$. It says that

the ML classifier $\mathcal{M}$ should predict that person x_2 is the creator of movie y_3 if person x_1 (resp. x_2) is the actor (resp. creator) of both movies y_1 and y_2, x_1 acts in y_3, and if all movies have the same language and are from the same series (determined by ML classifier). The GAR indicates that it is possible to interpret ML models "recursively" with both logic conditions and those ML models that we know how to interpret.

(3) In the knowledge graph DBpedia, a simple discovered GAR is $\varphi_e = Q_e[\bar{x}_e](\emptyset \rightarrow \mathsf{belong\_to}(x_1, y_1))$. It indicates the traditional biological classification. That is, if a directed chain is formed from x_1 to y_1 through class y_2 and family y_3 in the taxonomic hierarchy, then organism x_1 must belong to the kingdom y_1.

(1) Knowledge graph completion. We first applied GARs to restore missing information, including edges and attributes, in knowledge graphs YAGO and DBpedia. We picked edge and attribute predicates for application $\mathcal{A}$, *e.g.,* $\mathsf{member\_of}(x,y)$, $\mathsf{birthPlace}(x,y)$, $\mathsf{award}(x,y)$, $\mathsf{citizenship}(x,y)$, $x.\mathsf{education}$, $x.\mathsf{club}$ and $x.\mathsf{party}$. To evaluate the performance, for each graph, we constructed a test set by randomly selecting $10K$ application-related edges and attributes that conform to the predicates of $\mathcal{A}$ in the original graph as positive samples, and picking $10K$ nonexistent links and attributes as negative ones. Then we applied the GARs discovered with $\mathsf{GSRD}_{\mathsf{CA+RW}}$ and $N = 2$, $\rho\% = 10\%$, $\sigma = 1000$, $k = 8$ and $m = 7$, referred to as the default setting, to classify the information in test set, *i.e.,* whether they should be restored. F-measure was employed as the accuracy metric, and link prediction models SimplE [KP18] and ComplEx [TWR$^+$16] were treated as baselines. We find that enforcing the $\mathcal{A}$-relevant GARs consistently achieves high accuracy above 0.87, and on average it beats SimplE and ComplEx by 9.2% and 11.5%, respectively.

(2) Inconsistency detection. We also studied error detection. This experiment was conducted over DBpedia, with an application $\mathcal{A}$ consisting of predicate $\mathsf{same\_kingdom}(x,y)$. That is, the "kingdom" of two species should be the same under certain conditions, *e.g.,* two species are in the same "class". We randomly drew 7% of species entities from DBpedia and changed their kingdom values, to form the test set. The $\mathcal{A}$-relevant GARs discovered with default setting were used to detect the erroneous kingdom values. We find that such GARs perform comparably to the entire set of GFDs mined from the same graph [FHLL20], with F-measure above 0.96.

Exp-6: Ablation study. Observe that Exp-1 and Exp-2 already show graph reduction is more important for achieving high recall in GAR discovery. We next preformed

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

Table 3.3: Ablation study on the efficiency of GAR discovery

an ablation study using DBpedia and YAGO to investigate how each stage influences the efficiency. Fixing $k = 3$, $n = 8$, $\sigma = 1000$, $\rho\% = 10\%$ and $N = 1$, we omitted one of graph reduction and sampling stages. As shown in Table 3.3, the discovery time significantly increases when sampling is left out, indicating that sampling is more critical for improving efficiency.

Summary. We find the following. (1) The application-driven graph reduction method is effective. It reduces the graphs by 76% on average, while achieving recall of 85% for discovered $\mathcal{A}$-relevant GARs. (2) The sample graphs deduced by framework GSRD are of high quality. On 4 such samples with sample ratio 10%, more than 94% of the GARs in the $\mathcal{A}$-graphs $G_{\mathcal{A}}$ can be mined, with support at least 1000 in $G_{\mathcal{A}}$. (3) The sampling-based discovery scheme is efficient. It speeds up mining from the entire graphs by 60.6 times on 2 sample graphs with sample ratio 10%, while retaining recall above 91%. It speeds up algorithm DisGFD of [FHLL20] by 52.7 times for GFD discovery. (4) Algorithm ParGARMine is parallelly scalable: on average it is 3.2 times faster when n varies from 4 to 16.

3.8 Novelty and Contributions

The rule discovery methods proposed in this Chapter differ from the prior work on graph rule discovery as follows. (1) We propose application-driven reduction and graph sampling strategies with accuracy guarantee to reduce excessive rules and improve efficiency, as opposed to mining rules from the entire graphs. In light of these, the problem and even the notion of support are different. (2) We study the discovery of GARs from general property graphs, without requiring to encode graph data in RDF as knowledge graph rule learners [MCRS19, GTHS13, CWG16, OMP18, Coh16]. These rule learners may exhibit poor scalability on RDFs that are transformed from property graphs, since their node attributes often yield a large number of RDF triples. (3) We discover GARs with ML predicates and graph patterns of generic subgraphs. In contrast, no prior methods consider ML predicates, and most of them study path patterns only.

To the best of our knowledge, in this Chapter, we provide the first accuracy guarantee on recall and support for sampling-based discovery of graph rules, *i.e.,* the percentage of useful and frequent rules *w.r.t.* some support threshold that can be mined from samples of bounded sizes. Different from mining relational rules, GAR discovery has to inspect not only dependencies on attributes, but also graph patterns. Thus, neither the relational sampling techniques nor their quality analyses can be applied to GARs mining.

Chapter 4

Towards Event Prediction in Temporal Graphs

This Chapter proposes a class of temporal association rules, denoted by TACOs, for event prediction. Event prediction is to predict a real-world occurrence that relates to a particular topic and will take place at a specific time [Zha21]. Events range from large scale (*e.g.,* disease outbreaks and finance crisis), to medium-scale (*e.g.,* crime incidents and system failures), to small-scale (*e.g.,* authentication and fraud detection). Event prediction is important in a variety of domains such as disease control, transportation management, cybersecurity and business intelligence.

To illustrate, consider our familiar online recommendation, a special case of event prediction ("sale events"), to recommend items to users. Conventional recommendation models are mainly categorized into collaborative filtering (CF) by learning from user-item interaction history, content-based (CB) by assessing the similarity of content features of users and items, and hybrid by integrating the two [ZYST19]. In the real world, however, e-commerce companies often monitor changes to transaction graphs, and employ rules to catch users' temporal interests and detect fraudulent behaviors.

Example 4.1: Below are real-life example rules taken from daily practice of an e-commerce platform that serves billions of people.

(1) If a movie is nominated for a film award and if a user watches the movie within two weeks after its nomination, then recommend the movie to the friends of the user between the nomination date and the date of the award ceremony. Here the nomination indicates a "change" that triggers the recommendation in a time interval. Such cases are beyond conventional CF and CB models.

(2) If a user searched "barbecue" at least m times in June, then recommend meat to the user in July and August.

Apart from sales promotion and item recommendation, changes to temporal graphs are also important for predicting general events.

(3) If at least m cases of an infectious disease z are reported in an area within the past 2 weeks, then offer vaccines for z to the people there.

(4) If device M is used to access k accounts only once within a short period of one hour, and if each of these accounts has been regularly accessed by other devices at least m times in the past month, then M is likely committing account-takeover attacks. □

Challenges. No matter how important, event prediction is challenging. While rules have been studied for graphs, *e.g.,* graph functional dependencies (GFDs) [CP12], graph association rules (GARs) [FJL$^+$20] and Horn rules [GTHS13], they are defined on *static* graphs and cannot express temporal changes. There are several open questions. How can we specify patterns of changes to graphs and temporal interests of users in logic rules? Can we improve the accuracy of event-prediction ML models with temporal conditions, and interpret their predictions? How expensive is it to reason about rules with temporal conditions and embedded ML models? How can we efficiently discover such rules from real-life graphs? Is it possible to scale with large graphs when we apply the rules to predict events? Take rule discovery from these challenges as an example. Graph rules are often defined with a graph pattern [FJL$^+$20, CP12]. We find that conventional levelwise algorithms take days to mine temporal rules with patterns of more than 5 nodes. Even for simpler GFDs with patterns of 6 edges, it takes 1.5 hours on graphs with 32M nodes and edges when using 8 machines [FHLL20].

In order to tackle these issues, we propose a class of logic rules, referred to as TACOs (TemporAl event prediCtiOn rules), that enrich event prediction ML models with temporal conditions and change patterns (Section 4.1), establish the complexity of reasoning about TACOs (Section 4.2), and develop a system TASTE (Section 4.3) to (1) discover TACOs with large patterns by generative machine models in a creator-critic framework (Section 4.4) and (2) predict events in parallel by employing the rules (Section 4.5). Experiments on real-life and synthetic temporal graphs verify the efficiency, effectiveness and scalability of our system (Section 4.6). We finally conclude our novelty and contributions compared with previous work in Section 4.7.

4.1 Temporal Event Prediction Rules

We firstly introduce TACOs in this section. We start with basic notations (Section 4.1.1) and then define temporal prediction rules (Section 4.1.2).

4.1.1 Preliminaries

Assume three countably infinite sets of symbols, denoted by Γ, Υ and Ω, for labels, constants and timestamps, respectively. We denote a *time window* by τ, which is $[t_1, t_2]$ for timestamps t_1 and t_2 ($t_1 \leq t_2$).

Temporal graphs. A *temporal graph* is specified as $G=(V, E, L, T, F_A)$, where (a) V is a finite set of nodes; (b) $E \subseteq V \times \Gamma \times \Omega \times V$ is the set of edges, where $e=(v, l, t, v')$ denotes an edge from v to v' that is labeled with $l \in \Gamma$ and carries timestamp $t=T(e) \in \Omega$; (c) each node $v \in V$ has label $L(v)$ from Γ; and (d) node 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 denotes a property and a_i is a constant, written as $v.A_i=a_i$, and $A_i \neq A_j$ if $i \neq j$.

We refer to G as a *graph* when it is clear from the context.

Intuitively, G is a directed labeled graph in which each edge has a timestamp recording when it is added to G or when it is last updated, as commonly found in transaction graphs of e-commerce companies. Note that from a node v_1 to v_2, there may exist multiple edges, possibly with the same label but different timestamps. To simplify the presentation, we specify node timestamps by attaching timestamped self-loop edges to nodes to mark the times of node updates.

Δ-*patterns*. A Δ-*pattern* is $\Delta Q[\bar{x}] = (V_Q, E_Q, L_Q, T_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of pattern nodes (resp. edges); (2) L_Q assigns a label in Γ to each pattern node $u \in V_Q$ (resp. edge $e \in E_Q$); (3) T_Q assigns a time window τ to each pattern edge; in particular, one designated edge e_x is given $\tau = [0, 0]$ that fixes the current time; each pattern edge carries time window $[-\infty, +\infty]$ by default, unless a limited time window is assigned by T_Q; (4) $\bar{x}$ is a list of distinct variables; and (5) μ is a bijective mapping from $\bar{x}$ to V_Q, *i.e.,* it assigns a distinct variable to each node in V_Q. For $x \in \bar{x}$, we use $\mu(x)$ and x interchangeably when it is clear from the context.

Example 4.2: Figure 4.1 depicts Δ-patterns ΔQ_1–ΔQ_4 for the cases of Example 4.1: (1) ΔQ_1 specifies the change that a movie is nominated for an award, together with potential users, in which the designated edge e_x is labeled nomination; (2) ΔQ_2 depicts that a user poses m queries about barbecue in June, where edge e_x indicates the last

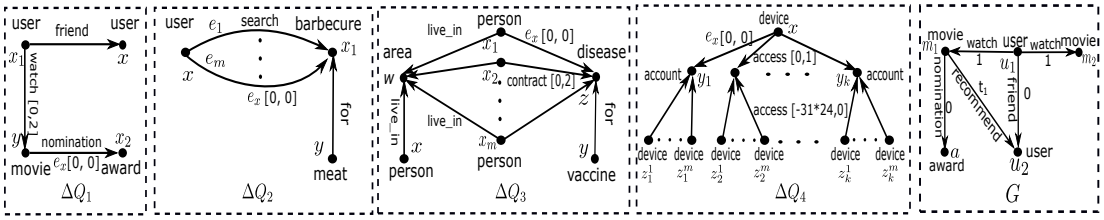


Figure 4.1: Δ-patterns and temporal graph

query; (3) ΔQ_3 shows the change of m cases of disease z found in an area w, along with a vaccine y for z; here e_x indicates the first case reported; and (4) ΔQ_4 is an abnormal pattern of account accesses, where e_x is the first access edge from device x to account y_1. □

Temporal pattern matching. A *match* of Δ-pattern $\Delta Q[\bar{x}]$ in graph G is a homomorphism h from $\Delta Q[\bar{x}]$ 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 $e=(u,l,\tau,u')$ in $\Delta Q[\bar{x}]$, $e'=(h(u),l,t,h(u'))$ is an edge in G and $t - t^* \in \tau$. Here t^* denotes the *current time*, *i.e.,* the timestamp of the edge in G to which the designated edge of $\Delta Q[\bar{x}]$ is mapped via h.

We denote by $\Delta Q(G)$ the set of all matches of $\Delta Q[\bar{x}]$ in graph G.

Intuitively, $\Delta Q[\bar{x}]$ is a *change pattern*. It monitors updates to temporal graph G. The matches of $\Delta Q[\bar{x}]$ in G can be computed in ΔG_Q, which consists of nodes and edges with timestamps in the range $[t_{\mathsf{min}}, t_{\mathsf{max}}]$ relative to the current time t^*, and is referred to as *updates to G relative to* ΔQ. Here t_{min} and t_{max} denote the earliest and latest timestamps specified in $\Delta Q[\bar{x}]$, respectively. That is, ΔG_Q is composed of updates (nodes and edges added to G) in time window $[t^* + t_{\mathsf{min}}, t^* + t_{\mathsf{max}}]$. The updates are typically much smaller than G. As a consequence, it is often *more efficient* to compute matches of a Δ-pattern than matches of a conventional graph pattern.

4.1.2 Definition and Semantics of Rules

TACOs are defined in terms of Δ-patterns and predicates.

Predicates. *Predicates* p of a Δ-pattern $\Delta Q[\bar{x}]$ have the form:

$$x.A \mid l(x,y) \mid \mathcal{M}(x,y,l,t) \mid x.A \oplus y.B \mid x.A \oplus c \mid e_1.t \oplus e_2.t \mid e.t \oplus c,$$

where $x, y \in \bar{x}$, e_1, e_2 and e are pattern edges of $\Delta Q[\bar{x}]$, $x.A$ denotes an attribute A of pattern node x, c is a constant, $l(x,y)$ is an edge from x to y labeled l, $e.t$ is the timestamp of edge e, and $\oplus$ is one of $=, \neq, <, \leq, >, \geq$. We write $e.t \in [t_1, t_2]$ if $e.t \geq t_1$ and $e.t \leq t_2$. We refer to $e_1.t \oplus e_2.t$ and $e.t \oplus c$ as *temporal predicates*.

In particular, $\mathcal{M}$ is an ML classifier for event prediction on temporal graphs; it returns true if it predicts that the event indicated by edge $l(x,y)$ will take place at time t^*+t. It can be a dynamic recommendation model *e.g.,* SASRec [KM18], and returns true with l = recommend to recommend item x to user y at time t^*+t. It may be a temporal graph completion model, *e.g.,* RE-GCN [LJL+21b], to predict a link of $l(x,y)$ at time t^*+t. We refer to $\mathcal{M}$ as an *ML predicate*.

ML models. We require the model $\mathcal{M}$ to work in a "transductive setting" [CAEHP+20], in which all node and edge labels are observed in both model training and TACO discovery (see Section 4.3). In this setting, $\mathcal{M}$ infers information between nodes with observed labels, *e.g.,* suggesting new links between users and items. Once the training of $\mathcal{M}$ and TACO discovery complete, the embeddings of the observed node and edge labels are fixed and no new embeddings that may violate the discovered TACOs will be introduced. Hence it is efficient to apply $\mathcal{M}$ for checking. This also enables interpretation of $\mathcal{M}$ via the logic predicates in the discovered TACOs (see below).

Rules. A *temporal event prediction rule* (TACO) φ is defined as

$$\Delta Q[\bar{x}](X \to (p_0, \tau)),$$

where $\Delta Q[\bar{x}]$ is a Δ-pattern, X is a conjunction of predicates of $\Delta Q[\bar{x}]$, p_0 is a predicate of $\Delta Q[\bar{x}]$, and τ is a time window. When $p_0 = l(x,y)$, the edge represented by p_0 is predicted and is not necessarily already in $\Delta Q[\bar{x}]$. We refer to $\Delta Q[\bar{x}]$, X, p_0 and $X \to (p_0,\tau)$ as the *pattern*, *precondition*, *event* and *dependency* of φ, respectively.

Intuitively, the pattern $\Delta Q[\bar{x}]$ of φ monitors changes to graph G. The TACO says that if the precondition X holds on the entities matched by $\Delta Q[\bar{x}]$, then the event specified by p_0 will take place within the time window τ. Here p_0 may indicate the purchase of an item y by a user x or outbreak of disease x in area y, etc.

Example 4.3: The rules of Example 4.1 can be expressed as TACOs.

(1) $\varphi_1 = \Delta Q_1[\bar{x}](X_1 \to (\mathsf{recommend}(y,x), [0,t_1]))$, where $X_1 = \emptyset$ and t_1 is a timestamp in weeks. The TACO says that if user x_1 watches movie y in 2 weeks after the nomination of y for award z and if x is a friend of x_1, then recommend y to x by the date t^*+t_1 of the award ceremony. Note that the condition that user x_1 watches the movie in 2 weeks is specified by function T_Q in ΔQ_1 (see Fig. 4.1).

(2) $\varphi_2 = \Delta Q_2[\bar{x}](X_2 \to (\mathsf{recommend}(y,x), [0,60]))$, where X_2 is $(\bigwedge_{i\in[1,m-1]} e_i.t < e_{i+1}.t) \wedge (\bigwedge_{i\in[1,m]} e_i.t \in [t_{\mathsf{june}}, t'_{\mathsf{june}}])$. Here $e_i.t < e_{i+1}.t$ ensures that searches e_i and e_{i+1} are dis-

tinct and t_{june} (resp. t'_{june}) indicates the date of June 1 (resp. June 30). It says that if x searches barbecue at least m times in June (specified in X_2), then recommend meat to x in the next two months (60 days).

(3) $\varphi_3 = \Delta Q_3[\bar{x}](X_3 \rightarrow (\mathsf{offer}(y,x),[2,6]))$. Precondition X_3 is $\bigwedge_{i,j\in[1,m],i\neq j}(x_i.\mathsf{id} \neq x_j.\mathsf{id})$, to enforce that the cases x_i and x_j are distinct. Observe that in the Δ-pattern ΔQ_3, the time window $[0,2]$ indicates that person x_i contracts disease z in 2 weeks after the current time t^*, *i.e.,* the earliest case confimed. It says that if at least m cases of disease z are confirmed in area w within 2 weeks and if person x lives in w, then offer vaccine y for disease z to x in a month.

(4) $\varphi_4 = \Delta Q_4[\bar{x}](X_4 \rightarrow (x.\mathsf{status} = \mathsf{fraud},[0,0]))$, where X_4 is $\bigwedge_{i\in[1,k]}(\mathcal{M}(x,y_i,\mathsf{attack}, e_i.t)\wedge\ (\bigwedge_{j,l\in[1,m],j\neq l} z_i^j.\mathsf{id}\neq z_i^l.\mathsf{id}))$. Here $\mathcal{M}$ is an ML model, which suspects that x attacked account y_i at time $e_i.t$ and e_i is the edge from device x to y_i. The time window $[0,1]$ in ΔQ_4 is in hours; hence $[-31{*}24,0]$ is bounded by 31 days (*i.e.,* $31{*}24$ hours). The TACO says that if device x accesses $y_1,\ldots,y_k$ within one hour, ML model $\mathcal{M}$ suspects them as attack behaviours, and if each y_i ($i\in[1,k]$) has been accessed by other m devices z_i^1, …, z_i^m in the past month, then x is likely a fraud. □

Remark. (1) GARs [FJL$^+$20], GTARs [NWS$^+$17] and GERs [BBBG09a] are special cases of TACOs. (a) GARs are TACOs when all time windows are $[-\infty,\infty]$ and preconditions X carry no temporal predicates; similarly for graph entity dependencies (GEDs) [FL19] and graph pattern association rules (GPARs) [FWWX15]. We adopt a single predicate p_0 for the event in each TACO to simplify the discussion; this does not lose expressive power compared to GARs and GEDs; this can be verified along the same lines as [FHLL20]. (b) A GTAR can be expressed as TACOs, where each TACO (i) has a pattern that extends the antecedent pattern P_1 of the GTAR with the nodes in its consequent pattern P_2, and (ii) encodes a single edge from P_2 as the event p_0 with $\tau = [t_\alpha, t_\alpha]$, where t_α is the constant time interval specified in the TACO. (c) Similarly, a GER can be encoded as a set of TACOs, one for each update indicated by its patterns with the maximum timestamp [BBBG09a].

(2) As demonstrated by φ_4 of Example 4.3, one can plug a well-trained ML model $\mathcal{M}$ for temporal event prediction in precondition X, and enrich $\mathcal{M}$ with additional temporal and logic conditions.

(3) A TACO of the form $\Delta Q[\bar{x}](X \rightarrow (\mathcal{M}(x,y,l,t),\tau))$ interprets ML predictions with

logic predicates (see Section 4.6 for an example).

(4) One can associate TACOs with a probability for the predicted events. However, this would incur higher complexity (#P-hard) as indicated by probabilistic logic programming [KDR+11].

Semantics. Denote by $h(\bar{x})$ a match of $\Delta Q[\bar{x}]$ in a graph G, and p a predicate of $\Delta Q[\bar{x}]$. We write $h(\mu(x))$ as $h(x)$ for pattern node $x \in \bar{x}$, where μ is the mapping in ΔQ from $\bar{x}$ to nodes in V_Q.

We say that match $h(\bar{x})$ *satisfies* a predicate p, denoted by $h(\bar{x}) \models p$, if the following condition is satisfied: (a) if p is $x.A$, then node $h(x)$ carries attribute A; (b) if p is $l(x,y)$, then there is an edge from $h(x)$ to $h(y)$ labeled l; (c) if p is $\mathcal{M}(x,y,l,t)$, then the ML classifier $\mathcal{M}$ predicts an association between $h(x)$ and $h(y)$ with label l at time $t^* + t$; (d) if p is $x.A \oplus y.B$, then attributes A and B exist at $h(x)$ and $h(y)$, respectively, and $h(x).A \oplus h(y).B$; similarly for $x.A \oplus c$; and (e) if p is $e_1.t \oplus e_2.t$, then the timestamps of edges matching e_1 and e_2 have the $\oplus$ relationship; similarly for $e.t \oplus c$.

For a conjunction X of predicates, we write $h(\bar{x}) \models X$ if $h(\bar{x}) \models p$ for all p in X. We write $h(\bar{x}) \models \varphi$ for TACO $\varphi = \Delta Q[\bar{x}](X \rightarrow (p_0, \tau))$ if $h(\bar{x}) \models X$ implies $h(\bar{x}) \models p_0$ and p_0 occurs within time window τ.

We say that graph G *satisfies* TACO $\varphi = \Delta Q[\bar{x}](X \rightarrow (p_0, \tau))$, denoted by $G \models \varphi$, if for all matches $h(\bar{x})$ of $\Delta Q[\bar{x}]$ in G, $h(\bar{x}) \models \varphi$. We say that G *satisfies* a set Σ of TACOs, denoted by $G \models \Sigma$, if for all TACOs $\varphi \in \Sigma$, $G \models \varphi$, *i.e., G* satisfies every TACO in Σ.

Example 4.4: Consider the temporal graph G illustrated in Fig. 4.1, where each edge is annotated a timestamp. Then G satisfies the TACO φ_1 of Example 4.3. Observe that there only exists a singe match $h(\bar{x})$ of ΔQ_1 in G: $x \mapsto u_2$, $x_1 \mapsto u_1$, $x_2 \mapsto a$ and $y \mapsto m_1$. As u_2 is linked from m_1 via a recommend edge with timestamp t_1 in G, we know that $h(\bar{x}) \models (\text{recommend}(y,x), [0,t_1])$, and $G \models \varphi_1$ follows. □

The notations of this chapter are summarized in Table 4.1.

4.2 Reasoning about Temporal Rules

In this section we study fundamental problems for TACOs, including the satisfiability, implication and prediction problems. We show that although TACOs are more expressive than GARs [FJL+20], these problems for TACOs are not much harder than for GARs.

Notations	Definitions
G, $\Delta Q[\bar{x}]$	graph and Δ-pattern, respectively
ΔG_Q	updates to G relative to pattern ΔQ
φ	TACO $\Delta Q[\bar{x}](X \rightarrow (p_0, \tau))$
$\mathcal{M}(x, y, l, t)$	an ML predicate
$h(\bar{x})$	a match of pattern $\Delta Q[\bar{x}]$
$\mathsf{supp}(\varphi, G)$	the support of TACO φ in graph G
$\mathsf{conf}(\varphi, G)$	the confidence of TACO φ in graph G
$\alpha, \beta, \gamma, \delta$	thresholds (node, support, confidence, time window)

Table 4.1: Notations

Satisfiability. The *satisfiability* problem is as follows.

- Input: A set Σ of TACOs.
- Question: Does there exist a graph G such that $G \models \Sigma$ and for each TACO $\Delta Q[\bar{x}](X \rightarrow (p_0, \tau)) \in \Sigma$, ΔQ has a match in G?

Intuitively, this is to make sure that the discovered TACOs have no conflicts and can be applied to a graph at the same time.

The problem is coNP-complete for GARs [FJL$^+$20] and Σ_2^p-complete for graph denial constraints (GDCs) [FL19]. We show the following.

Theorem 4.1: *The satisfiability problem is Σ_2^p-complete for* TACOs. □

Here Σ_2^p denotes the class of decision problems that can be checked in NP using an NP oracle, *i.e.,* $\Sigma_2^p = \mathsf{NP}^{\mathsf{NP}}$. Similarly, the class $\Pi_2^p = \mathsf{coNP}^{\mathsf{NP}}$ is defined (see [Pap03] for more details). Both Σ_2^p and Π_2^p are in the polynomial hierarchy of complexity theory [Pap03]. We assume *w.l.o.g.* that ML prediction (*i.e.,* testing with pre-trained $\mathcal{M}$) takes polynomial time (PTIME), as commonly found in practice.

Proof: We first show a small model property for the satisfiability problem of TACOs, based on which we design an Σ_2^p algorithm for the problem. After that we show that the satisfiability problem is Σ_2^p-hard.

Small model property. We start with the following small model property: if a set Σ of TACOs is satisfiable, then there exists a temporal graph G_Σ such that its size $|G_\Sigma| \leq 4|\Sigma|^3$ and $G_\Sigma \models \Sigma$.

Given a set Σ of TACOs, if Σ is satisfiable, then by the definition of satisfiability, we know that there exists a temporal graph $G{=}(V, E, L, T, F_A)$ such that $G \models \Sigma$ and for each TACO $\varphi{=}\Delta Q[\bar{x}](X{\rightarrow}(p_0, \tau)){\in}\Sigma$, ΔQ has a match h_φ in G. From these we can

deduce a small graph G_Σ satisfying Σ as follows: (1) we first deduce a subgraph G_s of G by combining the match h_φ of each ΔQ from Σ; and then (2) we construct G_Σ by revising the attributes in G_s.

(1) We first build graph G_s by combining the matches h_φ. More specifically, the graph G_s=$(V_s, E_s, L_s, T_s, F_A^s)$ is defined as follows.

(a) V_s=$\bigcup_{\varphi\in\Sigma}\{h_\varphi(x)|x \in V_\varphi\}$, where V_φ is the set of nodes in the Δ-pattern ΔQ of TACO φ, and h_φ is the match of ΔQ in G.

(b) E_s=$\bigcup_{\varphi\in\Sigma}\{(h_\varphi(u), l, t, h_\varphi(v)) \mid (u, l, \tau, v) \in E_\varphi, t - t^* \in \tau\}$, where E_φ is the set of edges in the Δ-pattern ΔQ of TACO φ. Note that the timestamps of these matched edges (*i.e.,* the values t) are also copied from G.

(c) L_s copies the labels of the nodes from G, *i.e.,* $L_s(v) = L(v)$.

(d) For the node attributes, we only copy the ones appearing in Σ. That is, for each TACO $\varphi = \Delta Q[\bar{x}](X \to (p, \tau))$, match h of ΔQ in G_s, and predicate $x.A$, $x.A \oplus c$ or $x.A = y.B$ in φ, we define $F_A^s(v_1).A = F_A(v_1).A$ and $F_A^s(v_2).B = F_A(v_2).B$, where $v_1 = h(x)$ and $v_2 = h(y)$. Different from GDCs in [FL19], TACOs also contain ML predicates. Thus we also include the attribute values that are inspected by the ML models for each node in G_s.

We can verify that $G_s \models \Sigma$ since G_s is a subgraph of G and $G \models \Sigma$.

(2) We next normalize the attributes in G_s to obtain the small model G_Σ following the techniques in [FL19]. Note that although G_s only has $|\Sigma|$ many nodes at most, the size of G_s cannot be bounded since the attributes in G may carry unboundedly large attribute values and they are directly copied to G_s.

To obtain a small model G_Σ, we normalize G_s as follows. (a) We first separate the attribute values in G_s into numeric values (*i.e.,* the values on which we can define the built-in predicates $=, \neq, <, \leq, >, \geq$) and non-numeric values (*i.e.,* the values on which we can only enforce predicates $=$ and $\neq$). (b) Then we replace numeric attribute values in G_s with numbers chosen from the range $[N_{\min} - |\Sigma|, N_{\max} + |\Sigma|]$ and preserve the relative order of attribute values as in the original G_s. Here $N_{\min}$ and $N_{\max}$ are the minimum and maximum numeric values appearing in Σ. (c) For those non-numeric values that appear in G_s but not in Σ, we replace them with distinct new small values and preserve the relationships between attribute values *w.r.t.* $=$ and $\neq$. (d) Observe that TACOs may

contain ML predicates $\mathcal{M}(x,y,l,t)$ and the attribute values they check may be changed after the normalization above. However, we can assume *w.l.o.g.* that the ML model $\mathcal{M}$ can return infinitely many true and false results for different nodes pairs x and y when their attributes are taken from any dense domain, *i.e.,* infinite output assumption [FLT20]. In addition, $\mathcal{M}$ usually inspects a constant number of attributes only, *e.g.,* selecting a small number of relevant attributes before training the models [CS14]. These are guaranteed by most high dimensional ML models [Agg14]. Then it follows that the original predictions made by model $\mathcal{M}$ remain intact after the attribute values are replaced with the small ones that are carefully selected.

Denote by G_Σ the graph obtained from G_s after the attribute normalization. We can verify that there is no attribute value in G_Σ with size larger than $2|\Sigma|$ and hence $|G_\Sigma| \leq 4|\Sigma|^3$. Moreover, we can show that $G_\Sigma \models \Sigma$ along the same lines as that in [FL19]. Then the small model property is proved.

Upper bound. Given a set Σ of TACOs, we develop the following Σ_2^p algorithm to check whether Σ is satisfiable. (1) Guess a temporal graph G such that $|G| \leq 4|\Sigma|^3$. (2) Check whether $G \models \Sigma$; if so, return true; otherwise, reject the current guess.

The correctness follows from the small model property above. To see the complexity, since checking whether $G \models \Sigma$ is in NP (see Theorem 4.2), we know that the algorithm is in $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2^p$.

Lower bound. The lower bound follows from the Σ_2^p-completeness of the satisfiability problem for GDCs [FL19]. A GDC is defined as $Q[\bar{x}](X \rightarrow p)$, where $Q[\bar{x}]$ is a conventional graph pattern (*i.e.,* a Δ-pattern without time window on edges), X (resp. p) is a conjunction of predicates (resp. a predicate) of $Q[\bar{x}]$ in the form of $x.A \oplus c$ or $x.A \oplus y.B$. One can easily verify that GDCs are a special case of TACOs, where all the pattern edges in TACOs are assigned the time window $[-\infty, +\infty]$. Since the satisfiability problem for GDCs is Σ_2^p-complete, the satisfiability problem for TACOs is Σ_2^p-hard. □

Implication. A set Σ of TACOs *implies* a TACO φ, denoted by $\Sigma \models \varphi$, if for all graphs G, if $G \models \Sigma$ then $G \models \varphi$.

The *implication problem* is stated as follows.

- Input: A set Σ of TACOs and a TACO φ.
- Question: Does $\Sigma \models \varphi$?

The implication analysis helps us remove redundant rules.

The implication problem is NP-complete for GARs [FJL$^+$20] and Π_2^p-complete for

GDCs [FL19]. For TACOs it is no harder than for GDCs.

Proof: Similar to the proof of Theorem 4.1, we show a small model property for the implication problem of TACOs to prove its upper bound.

Small model property. We prove the following small model property: if for a set Σ of TACOs, $\Sigma \not\models \varphi$, then there exists a temporal graph G_φ such that $|G_\varphi| \leq 8|\varphi|(|\varphi|+|\Sigma|)^2$ and $G_\varphi \models \Sigma$ but $G_\varphi \not\models \varphi$.

Given a set Σ of TACOs and a TACO $\varphi = \Delta Q[\bar{x}](X \to (p_0, \tau))$, if $\Sigma \not\models \varphi$, then by the definition of implication, we know that there exists a temporal graph $G=(V,E,L,T,F_A)$ such that $G \models \Sigma$ but $G \not\models \varphi$. Since $G \not\models \varphi$, there exists a match h of ΔQ in G such that $h \models X$ but $h \not\models (p_0, \tau)$. Based on this, we can deduce a graph G_φ witnessing $\Sigma \not\models \varphi$ as follows: (1) we first deduce a subgraph G_s of G from the match h; and then (2) we construct the small model G_φ witnessing $\Sigma \not\models \varphi$ by normalizing the attributes in G_s.

(1) We first build graph G_s by using the match h. More specifically, G_s is defined as $(V_s, E_s, L_s, T_s, F_A^s)$ with the following.

(a) $V_s = \{h_\varphi(x) \mid x \in V_\varphi\}$, where V_φ is the node set of the Δ-pattern ΔQ of TACO φ.

(b) $E_s = \{(h(u), l, t, h(v)) \mid (u, l, \tau, v) \in E_\varphi, t - t^* \in \tau\}$, where E_φ is the set of edges in the Δ-pattern ΔQ of TACO φ. Again the timestamps of edges are copied from G.

(c) L copies the node labels from G, *i.e.,* $L_s(v) = L(v)$.

(d) For the attributes of the nodes, we only copy the ones appearing in Σ and φ. That is, for each TACO $\varphi' = \Delta Q'[\bar{x}'](X' \to (p', \tau'))$ in $\Sigma \cup \{\varphi\}$, match h' of $\Delta Q'$ in G_s, and predicate $x.A$, $x.A \oplus c$ or $x.A \oplus y.B$ in φ', we define $F_A^s(v_1).A = F_A(v_1).A$ and $F_A^s(v_2).B = F_A(v_2).B$, where $v_1 = h'(x)$ and $v_2 = h'(y)$. For each node in G_s, we also include the attribute values that are inspected by the ML models.

We next normalize the attribute values of G_s in the same way as in the the proof of Theorem 4.1. Denote by G_φ the graph obtained from G_s after such normalization. One can verify that the size of each attribute value in G_φ is bounded by $2(|\Sigma| + |\varphi|)$, and moreover, $|G_\varphi| \leq 8|\varphi|(|\varphi|+|\Sigma|)^2$. One can also show that $G_\varphi \models \Sigma$ but $G_\varphi \not\models \varphi$ along the same lines as the argument for the implication problem of GEDs [FL19]. These complete the proof of the small model property.

Upper bound. Based on the small model property, we develop the following Σ_2^p algorithm to check whether TACOs $\Sigma \not\models \varphi$: (1) guess a temporal graph G such that

$|G| \leq 8|\varphi|(|\varphi|+|\Sigma|)^2$; and then (2) check whether $G \models \Sigma$ but $G \not\models \varphi$; if so, return false; otherwise, reject the current guess.

It is easy to check the correctness of the algorithm using the small model property. The algorithm is in $\mathsf{coNP}^{\mathsf{NP}} = \Pi_2^p$, since $G \models \Sigma$ can be verified in NP (see Theorem 4.2).

Lower bound. The lower bound directly follows from the Π_2^p-complete implication problem for GDCs [FL19]. As mentioned in the proof of Theorem 4.1, GDCs are a special case of TACOs. Since the implication problem for GDCs [FL19] is Π_2^p-complete, we know that the implication problem for TACOs is also Π_2^p-hard. □

Prediction. We also study the *prediction problem*.

- Input: A temporal graph G, a set Σ of TACOs, a time window τ, a label l, and two nodes u and v in G.
- Question: Does there exist edge labeled l from u to v in τ by Σ?

This is to study the complexity of temporal prediction with TACOs.

A similar problem (deduction problem) was shown NP-complete for GARs [FJL+20]. It is no harder to predict events with TACOs.

Theorem 4.2: *The prediction problem is* NP*-complete for* TACOs. □

Proof: We start with a small model property for the prediction problem of TACOs, from which we can provide an NP algorithm for the problem. Then we prove that the prediction problem is NP-hard.

Small model property. We show that if an edge e labeled l from u to v is predicted in time window τ when applying a set Σ of TACOs on a temporal graph G, then we can construct a proof tree $\mathcal{T}$ with $|\mathcal{T}| \leq |\Sigma|^2|G|^4(|G|+|\Sigma|)^3$ witnessing the prediction. Here (1) the root of $\mathcal{T}$ is (p,τ), where p is the predicate $l(u,v)$ representing the edge e labeled l from u to v; and (2) nodes in $\mathcal{T}$ are in the form of (p_i,τ_i), indicating that (i) the predicate p_i is predicted in the time window τ_i by Σ, or (ii) p_i always holds in G with $t_i = [-\infty,\infty]$.

The proof tree $\mathcal{T}$ is constructed from the root (p,τ) in a top-down manner: for a node (p^t,τ^t) in $\mathcal{T}$, we define its children based on how (p^t,τ^t) is predicted in G by Σ. More specifically, assume that (p^t,τ^t) is predicted using the TACO $\varphi = \Delta Q[\bar{x}](X \to (p_0,\tau_0))$ from Σ and a match h of ΔQ in G; let $p_1,\ldots,p_n$ be all predicates in the precondition X, and $\tau_1,\ldots,\tau_n$ be the time windows in which the instantiated predi-

cates $h(p_1),\ldots,h(p_n)$ are predicted, where $h(p_j)$ is obtained from p_j by replacing the variables x in ΔQ with the nodes $h(x)$ in G. Then we add $(p_1,\tau_1),\ldots,(p_n,\tau_n)$ as the children of (p^t,τ^t) to $\mathcal{T}$, and label the edges with $\langle\varphi,h\rangle$. The construction stops when each path of $\mathcal{T}$ reaches a node whose corresponding predicate always holds in G, *i.e.,* node (p_j,τ_j) with $\tau_j=[-\infty,\infty]$.

It is easy to verify that (1) when (p,τ) is predicted, we can construct a proof tree $\mathcal{T}$ witnessing the prediction; and (2) the size of $\mathcal{T}$ is bounded by $|\Sigma|^2|G|^4(|G|+|\Sigma|)^3$, since there exist at most $|\Sigma|^2|G|^4(|G|+|\Sigma|)^2$ predicates and the number of all possible time windows is bounded by $(|G|+|\Sigma|)$.

Upper bound. Using this small model property, we design an NP algorithm for the prediction problem of TACOs: guess a tree $\mathcal{T}$ with $|\mathcal{T}|\leq|\Sigma|^2|G|^4(|G|+|\Sigma|)^3$ and check whether it is a proof tree of prediction (p_0,τ). If so, return true; otherwise, reject the guess.

The correctness follows from the small model property, and the complexity follows from the following two facts: (1) the proof tree encodes all matches that are needed for the prediction; and (2) one can verify whether $\mathcal{T}$ is a valid proof tree in PTIME.

Lower bound. We show the lower bound by reduction from the graph homomorphism problem, which is NP-complete (cf. [GJ79]). The graph homomorphism problem is to decide, given two undirected graphs G_1 and G_2, whether there exists a homomorphism h from G_1 to G_2. Given $G_1=(V,E_1)$ and $G_2=(V_2,E_2)$, we use a set Σ consisting of only one TACO φ_1 to encode G_1, a temporal graph G to encode G_2, and (p_0,τ) to represent the existence of the homomorphism h. More specifically, we define the following.

(1) The TACO φ_1 is $\Delta Q_1[\bar{x}](\emptyset\rightarrow(\tau(u_p^1,u_p^2),[0,0]))$. Here the Δ-pattern ΔQ_1 is defined as $(V_1^p,E_1^p,L_Q^1,T_Q^1,\mu_1)$, where (a) V_1^p=$V_1\cup\{u_p^1,u_p^2\}$, V_1 is the set of nodes in G_1, and u_p^1 and u_p^2 are two distinct new nodes that are not in V_1; (b) E_1^p encodes each undirected edge in E_1 using two directed edges (*i.e.,* E_1^p=$\{(u,\tau,0,v),(v,\tau,0,u)\mid(u,v)\in E_1\}$; here the edges in E_1^p carry a specific label τ and time window $[0,0]$); (c) L_Q^1 assigns label τ to each node in V_1 and two distinct labels β_1 and β_2 to nodes u_p^1 and u_p^2, respectively; (d) T_Q^1 assigns the time window $[0,0]$ to each edge in E_1^p; and (e) μ_1 assigns a distinct variable for each node in V_1^p.

(2) The temporal graph G is defined as (V_2^g,E_2^g,L,T,F_A), where (a) $V_2^g=V_2\cup\{u_g^1,u_g^2\}$, V_2 is the node set of graph G_2, and u_g^1 and u_g^2 are two distinct new nodes that do not

appear in V_2; (b) E_2^g encodes each undirected edge in E_2 by using two directed edges (*i.e.,* $E_2^g = \{(u,\tau,0,v),(v,\tau,0,u) \mid (u,v) \in E_2\}$); (c) L assigns the label τ to each node in V_2 and two distinct labels β_1 and β_2 to nodes u_g^1 and u_g^2, respectively; (d) T assigns the timestamp 0 to each edge in E_2^g; and (e) each node v in G carries a tuple $F_A(v)=\emptyset$, *i.e.,* the node does not contain any attribute.

We show that there exists a homomorphism from G_1 to G_2 if and only if φ_1 predicts that there is an edge e labeled τ from u_g^1 to u_g^2 in temporal graph G during time window $[0,0]$. Note that u_g^1 and u_g^2 in G carry labels different from the ones in V_1; hence nodes from V_1 can only be mapped to those copied from V_2 in G.

($\Rightarrow$) Assume that an edge e labeled τ from u_g^1 to u_g^2 in G during time window $[0,0]$ is predicted by φ_1. By the construction we know that no such edge exists in the original G. Thus the TACO φ_1 can be applied on G to predict the edge e, and there must exist a match of ΔQ_1 in G, from which we can easily deduce a homomorphism from G_1 to G_2 by following the construction above.

($\Leftarrow$) Let h be a homomorphism from G_1 to G_2. Then we can construct a match h_1 of ΔQ_1 in G. From h_1, we enforce the existence of an edge e labeled τ from u_g^1 to u_g^2 in time window $[0,0]$ using φ. □

4.3 Temporal System

In this section we introduce TASTE (TemporAl SysTEm), a system for discovering TACOs and predicting events with TACOs. We first state the corresponding discovery and prediction problems (Section 4.3.1). We then present the architecture of TASTE (Section 4.3.2).

4.3.1 TACO Discovery and Event Prediction

We first define quality measures for TACOs.

Quality metrics. We use support and confidence to quantify the quality of TACOs, for frequency and reliability, respectively.

Support. We define $\mathsf{supp}(\Delta Q, X, G)$ as the support of Δ-pattern ΔQ in a graph G *w.r.t.* a conjunction X of predicates, to measure how often ΔQ can find matches in G satisfying X. More specifically,

$$\mathsf{supp}(\Delta Q, X, G) = |\Delta Q(e_x, X, G)|,$$

where $\Delta Q(e_x, X, G)$ refers to the set of edges $h(e_x)$ in all matches h of ΔQ in G that satisfy X. Hence it counts the number of distinct "satisfiable" matches in regards to the designated edge e_x.

Similarly, the *support* for a TACO $\varphi = \Delta Q[\bar{x}](X \to (p_0, \tau))$ is

$$\mathsf{supp}(\varphi, G) = |\Delta Q(e_x, \varphi, G)|,$$

where $\Delta Q(e_x, \varphi, G)$ is the set of edges $h(e_x)$ in those matches $h(\bar{x}) \in \Delta Q(G)$ such that $h(\bar{x}) \models X$, $h(\bar{x}) \models p_0$, and p_0 occurs in τ.

Intuitively, the higher support of a TACO φ is, the more "frequent" that φ finds positive evidence. Moreover, one can verify that this measure is anti-monotonic *w.r.t.* a partial order $\preceq$ over the domain of TACOs. That is, for two TACOs $\varphi = \Delta Q[\bar{x}](X \to (p_0, \tau))$ and $\varphi' = \Delta Q'[\bar{x}'](X' \to (p_0', \tau'))$, $\varphi \preceq \varphi'$ if (a) all nodes and edges in ΔQ also appear in $\Delta Q'$ with the same labels, and the two Δ-patterns share the same designated edge e_x; (b) the time window of each edge in ΔQ covers that in $\Delta Q'$; and (c) $p_0 = p_0'$ and τ' is a subinterval of τ. Then $\mathsf{supp}(\varphi, G) \geq \mathsf{supp}(\varphi', G)$ for any G if $\varphi \preceq \varphi'$.

Confidence. We define the *confidence* of a TACO $\varphi = \Delta Q[\bar{x}](X \to (p_0, \tau))$ in graph G, denoted by $\mathsf{conf}(\varphi, G)$, as follows:

$$\mathsf{conf}(\varphi, G) = \frac{\mathsf{supp}(\varphi, G)}{\mathsf{supp}(\Delta Q, X, G)}.$$

It uses the ratio to quantify the likelihood that event p_0 happens in τ with satisfiable matches of Δ-pattern ΔQ and precondition X. A TACO with high confidence tends to offer reliable event predictions.

In $\mathsf{supp}(\varphi, G)$ and $\mathsf{conf}(\varphi, G)$, one can replace G with smaller ΔG_Q. That is, support and confidence can be equivalently computed in updates ΔG_Q to G relative to ΔQ instead of in the entire G.

TACO discovery. The discovery problem for TACOs is as follows.

- *Input*: A temporal graph G, a positive integer α, a support threshold $\beta > 0$, a confidence threshold $\gamma \in [0, 1]$, and a bound $\delta > 0$ on the lengths of time windows.
- *Output*: A set Σ of TACOs such that for each $\varphi \in \Sigma$, $\mathsf{supp}(\varphi, G) \geq \beta$, $\mathsf{conf}(\varphi, G) \geq \gamma$, $t_2 - t_1 \leq \delta$ for each time window $[t_1, t_2]$ in φ, and there are at most α nodes in the pattern of φ.

We call the tuple $(\alpha, \beta, \gamma, \delta)$ a *discovery requirement* d. The problem aims to discover the set of all TACOs that conform to d, and have the expected number of pattern nodes bounded by α.

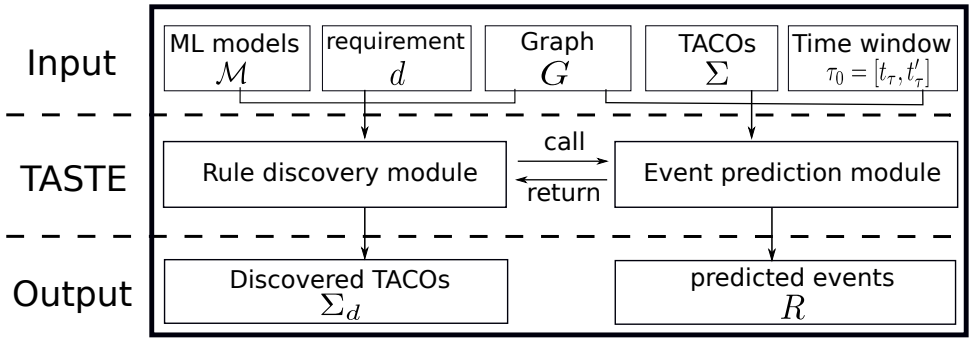


Figure 4.2: Architecture, inputs and outputs of TASTE

Temporal event prediction. The problem is stated as follows.

- *Input*: A graph G, a set Σ of TACOs and a time window τ_0.
- *Output*: A set R of edges for G predicted by Σ such that the occurrence of the event encoded by each edge in R is within τ_0.

To simplify the discussion, we consider events p_0 represented by edges $l(x,y)$, as commonly found in practice. We deduce the time windows of such edges in R from the time windows and designated edges specified in the TACOs of Σ (see Section 4.5 for more).

We can use each TACO's confidence as the probability of the predicted event. But the prediction problem would become harder, *i.e.,* #P-complete, as indicated by probabilistic logic programming [KDR+11]. We defer a full treatment of such extension to future work.

4.3.2 Overview of TASTE

The architecture of TASTE is depicted in Fig. 4.2. TASTE supports two modules for TACO discovery and event prediction, as follows.

(1) Rule discovery. Taking a temporal graph G, pre-trained ML models $\mathcal{M}$, and the discovery requirement d that is specified by parameters α, β, γ and δ as input, the rule discovery module outputs a set Σ_d of TACOs such that every TACO φ in Σ_d satisfies the requirement d as in the problem statement for TACO discovery. Here $\mathcal{M}$ can be any pre-trained ML model for temporal event prediction in the transductive setting (Section 4.1), and is embedded as ML predicates when mining TACOs. We refer to the TACOs that conform to the input discovery requirement d as *high-quality rules*.

(2) Event prediction. After a set Σ_d of high-quality TACOs is discovered, the event prediction module employs the TACOs to predict events by running a parallel algorithm. It may take a particular event p_0 from users and predict whether p_0 will happen and when

it will happen. It may also predict all possible events by applying Σ_d to temporal graph G. It should be mentioned that TASTE also accepts rules developed by domain experts, converts those rules to TACOs, and applies these rules and the discovered ones together.

We will present algorithms underlying the discovery and prediction modules in Sections 4.4 and 4.5, respectively.

4.4 Generation-based TACO discovery

In this section we propose a Creator-Critic Discovery approach, denoted by CCD, to discovering high-quality TACOs Σ_d. We develop such an algorithm for the rule discovery module of TASTE.

Below we first present the generation-based approach and justify the method from the perspective of probability distribution learning. We then present CCD and its functions and models. Finally we formally prove the performance guarantees of CCD.

Generation-based discovery. One might be tempted to extend the existing levelwise search-based methods, *e.g.,* [BBBG09a, FHLL20], to enumerate candidate TACOs and identify the required ones. However, these approaches need to search in a lattice and take exponential time, despite pruning strategies to reduce the cost [BBBG09a, FHLL20, NWS+17]. As will be seen in Section 4.6, it takes days for levelwise search to discover TACOs with patterns of 7 edges. Hence existing algorithms for discovering, *e.g.,* GFDs [FHLL20], are unable to find practical TACOs.

In view of this, we propose CCD for TACOs. CCD discovers high-quality TACOs by employing generative adversarial networks (GAN), and using interactive learning with weak supervision between a rule creator and a rule critic. This GAN-based framework is inspired by the latest progress in drug discovery and compound design [BX21, SCPR21], where a GAN is trained to generate promising candidates without enumerating all possible combinations in the large search space. Such generative approach significantly accelerates the drug discovery process and produces more promising compounds.

As shown in Fig. 4.3, in each iteration, the creator of CCD first employs ML models to generate a set Σ of candidate TACOs. Then the critic selects high-quality rules from Σ and adds them to Σ_d. Taking rules in Σ_d as feedback from the critic, the creator trains itself again to improve the probability of generating high-quality TACOs in the next iteration. CCD returns Σ_d when the number of iterations reaches a user-specified

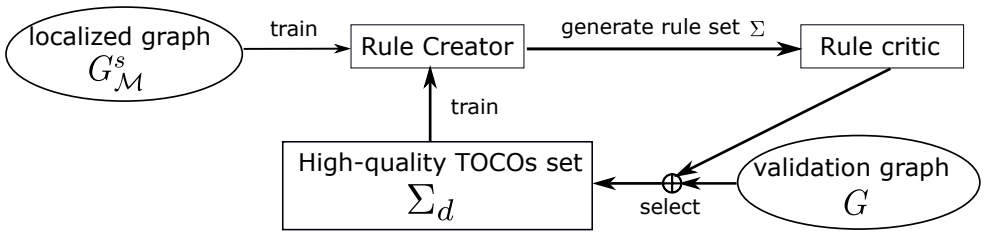


Figure 4.3: Dataflow of the iterative TACO discovery.

bound I.

The method is capable of discovering high-quality TACOs. From the perspective of probability theory, each TACO φ can be viewed as an event and the union of all TACOs constitutes a discrete countably infinite sample space $\mathcal{A}$. Given parameter d, the discovery process aims to find a probability distribution $\mathbb{P}$ described by a probability mass function $P\colon \mathcal{A} \to \mathbb{R}$ (real numbers), where

$$\begin{cases} P(\varphi) > 0, \text{if } \varphi \text{ is a high-quality TACO } \textit{w.r.t. } d, \\ P(\varphi) = 0, \text{if } \varphi \text{ is not a high-quality TACO } \textit{w.r.t. } d. \end{cases} \tag{4.1}$$

When such a distribution $\mathbb{P}$ is found, all desired rules can be obtained by sampling from $\mathbb{P}$. In order to find the distribution, conventional levelwise methods build a searching lattice to enumerate each rule φ in $\mathcal{A}$ and check whether it meets the requirement d. In contrast, CCD interactively trains deep generative models [ZZHH20a, MKS18] in the creator to approximate the target distribution $\mathbb{P}$, avoiding costly search in the exponentially large sample space.

This approach is feasible since deep generative models, *e.g.,* GAN, are able to approximate various distributions [CWD+18], such as images [JLO21], texts [dRP21] and even chemical compounds [SCPR21]. Moreover, empirical research verifies that it is practical to train such models to approximate a target distribution $\mathbb{P}$ within a limited number of iterations [GSW+20], where the training loss of the model converges. CCD also optimizes rule discovery in a data-driven manner, since generative models in the rule creator are able to generate rules that are topologically and semantically similar to rules from the training data (Σ_d in Fig. 4.3) [ZZHH20a, LZL+20, GSW+20]. Meanwhile, an increasing number of high-quality rules will be selected by the critic and added to Σ_d in each iteration. Thus, the creator can retrain its generative models using Σ_d as feedback to improve the quality of approximation. We will provide a probabilistic bound later in this section.

Algorithm. Implementing the creator-critic method, CCD discovers high-quality TACOs in three phases, as shown in Algorithm 4.1.

(1) It first calls function MLExp to prepare graph data $G_{\mathcal{M}}$ from G, for facilitating the discovery of TACOs with ML predicates $\mathcal{M}$.

(2) Then the rule creator is pretrained using TrainCreator with the localized graph $G^s_{\mathcal{M}}$ sampled by LocalizedSample (lines 2-3), such that it starts to generate rules that are likely to reach the thresholds of support and confidence in the input requirement d. The creator generates a set Σ of candidate TACOs by GenerateRule; the critic next evaluates Σ on temporal graph G and saves high-quality TACOs in a set Σ_d via function SelectRule (lines 4-5).

(3) In the third phase, algorithm CCD iteratively discovers high-quality TACOs and adds more such rules to Σ_d via an interactive training process of I iterations (lines 6-11). Each iteration is similar to the second phase, except that both $G^s_{\mathcal{M}}$ and Σ_d are used to train and improve the creator (line 8). Here the TACOs in Σ_d are feedback from the critic. Finally Σ_d is returned (line 12).

Parameters. In addition to the discovery requirement d of four thresholds and ML models $\mathcal{M}$, CCD takes another two input parameters: sample size N and iteration number I. Here N strikes a balance between the probability of generating high-quality rules and the workload of each iteration; and I determines the size of Σ_d and the cost of model training, where larger I usually helps generate more high-quality TACOs and train the model better.

We next present the functions and models adopted in CCD.

Assistance functions. We start with two functions MLExp and LocalizedSample, which are invoked by CCD to prepare graph data for model training. Initially, MLExp expands graph G to $G_{\mathcal{M}}$ by faithfully adding edges predicted by the input ML models $\mathcal{M}$. This allows the creator to incorporate $\mathcal{M}$ as ML predicates and accelerates TACO discovery since there is no need to repeatedly apply $\mathcal{M}$ during the discovery process. Function MLExp simply applies pre-trained ML models and is efficient.

Besides, LocalizedSample collects a set $G^s_{\mathcal{M}}$ of N Δ-patterns by sampling localized graph structures from $G_{\mathcal{M}}$, as training data for the creator in each iteration (lines 2 and 7). When deducing a pattern ΔQ_i, LocalizedSample applies temporal random walk [NLR$^+$18] with a randomly selected source node v to sample temporal paths, where the timestamps of all edges on a temporal path fall into a given time window. It finds top $\alpha - 1$ frequently sampled nodes around v; recall that α is the threshold on pattern node numbers. Then ΔQ_i is formed by these nodes and the edges connecting

Algorithm 4.1: Creator-Critic Discovery (CCD)

Input: A temporal graph G, discovery requirement d=$(\alpha, \beta, \gamma, \delta)$, ML models $\mathcal{M}$, sample size N, and iteration number I.

Output: A set Σ_d of TACOs mined from G such that every rule in Σ_d conforms to the requirement d.

1 $\varepsilon \leftarrow 0$; $\Sigma_d \leftarrow \emptyset$; $G_{\mathcal{M}} \leftarrow \mathsf{MLExp}(G, \mathcal{M})$;
2 $G^s_{\mathcal{M}} \leftarrow \mathsf{LocalizedSample}(G_{\mathcal{M}}, \alpha, N)$;
3 $\mathsf{TrainCreator}(G^s_{\mathcal{M}}, \alpha, \delta)$; /* *Pretrain the rule creator* */
4 $\Sigma \leftarrow \mathsf{GenerateRule}(\alpha, \delta, \Sigma_d)$;
5 $\Sigma_d \leftarrow \mathsf{SelectRule}(\Sigma, \beta, \gamma, G)$;
/* *Generate* TACOs *via interactive training* */
6 **while** $\varepsilon < I$ **do**
7 $G^s_{\mathcal{M}} \leftarrow \mathsf{LocalizedSample}(G_{\mathcal{M}}, \alpha, N)$;
8 $\mathsf{TrainCreator}(G^s_{\mathcal{M}} \cup \Sigma_d, \alpha, \delta)$;
9 $\Sigma \leftarrow \mathsf{GenerateRule}(\alpha, \delta, \Sigma_d)$;
10 $\Sigma' \leftarrow \mathsf{SelectRule}(\Sigma, \beta, \gamma, G)$;
11 $\Sigma_d \leftarrow \Sigma_d \cup \Sigma'$; $\varepsilon \leftarrow \varepsilon + 1$;
12 **return** Σ_d;

them. Guided by the Δ-patterns in $G^s_{\mathcal{M}}$ during pretraining (line 3), the rule creator can learn to generate patterns that are more likely to find matches in the graph. This is because each ΔQ_i must have matches as it is obtained by random walk in $G_{\mathcal{M}}$, and the generative models learn to generate "new" patterns that are semantically and structurally similar to ΔQ_i. Without pretraining, randomly initialized generative models in the creator may create meaningless TACOs.

The sample size N strikes a balance between the probability of generating "good" patterns that have plentiful matches and the cost in each iteration. A larger N allows more sampling trails with higher probability of generating good patterns and more efficient ML training with batch optimizations [ZZHH20a, MKS18]. If N is small, CCD needs to run more iterations to find adequate number of TACOs (see below).

Rule creator. When generating a TACO φ, the creator first generates its Δ-pattern, and then the dependency (lines 4 and 9).

Δ-pattern generation. The creator generates patterns in two steps: structure generation and semantic label generation, by employing temporal graph GAN and LSTM networks, respectively.

(1) In the first step, the creator takes each Δ-pattern ΔQ_i from $G^s_{\mathcal{M}}$ and Σ_d as input; it adopts TagGen [ZZHH20a], an end-to-end deep generative framework for temporal graphs, to deduce candidate Δ-patterns ΔQ^g_i. Due to the GAN module in TagGen [ZZHH20a], each generated ΔQ^g_i and the input ΔQ_i have the same number of nodes (at most α), and bear similar topology and time constraints. No labels were generated in this step as TagGen does not support labels. Hence we need the next step to attach labels to the patterns.

(2) In the second step, the creator employs an LSTM language model $\mathcal{M}_L$ to generate labels for each candidate Δ-pattern ΔQ^g_i. We adopt LSTM networks since it can model the rich semantics of labels on paths in knowledge graphs [LLL$^+$15, LZL$^+$20, LSX18]. More specifically, it first trains $\mathcal{M}_L$ on a corpus C driven by the perplexity [MKS18], where each word is a pair $\langle L(e), L(v)\rangle$ of edge label and node label, named "label pair", and v is the destination node of e. The corpus C is composed of label pair sequences of temporal paths, which are derived by applying temporal random walk for each ΔQ_i. Here the label pair sequence of a temporal path $(v_0, e_1, v_1, e_2, \ldots, e_s, v_s)$ is $(\langle \varnothing, L(v_0)\rangle, \langle L(e_1), L(v_1)\rangle, \ldots, \langle L(e_s), L(v_s)\rangle)$. After the training, for every two nodes u and v in ΔQ^g_i with the shortest temporal path ρ from u to v, $\mathcal{M}_L$ generates a label pair sequence with a random seed, and attaches this sequence of labels to ρ. Only shortest paths are considered since they hold stronger associations [AHAS03, KLAF17] compared with other longer paths. Finally the creator builds a Δ-pattern $\Delta Q'_i$ from ΔQ^g_i by keeping the most frequent label attached to each node and edge.

Note that the creator retrains itself using the latest Σ_d to increase the probability of generating high-quality Δ-patterns (line 8). The rationale behind this is that (1) in training data, the Δ-pattern ΔQ_i of a high-quality TACO in Σ_d has multiple matches; and (2) the GAN and LSTM networks enable the creator to generate new Δ-pattern $\Delta Q'_i$ that are similar to ΔQ_i in terms of topological structure, temporal constraints and label semantics [ZZHH20a, LZL$^+$20, LSX18]. Therefore, the generated $\Delta Q'_i$ is also likely to find sufficient matches in the graph and should be kept to build other candidate TACOs. The samples $G^s_{\mathcal{M}}$ returned by LocalizedSample are also used in retraining, to introduce disturbance to the creator, which prevents the ML generative models from converging at a local minimum where cliche patterns are repeatedly generated. This advocates the creator to generate "novel" patterns rather than reiterating existing ones.

Dependency generation. Given a generated Δ-pattern ΔQ, the creator adapts the lev-

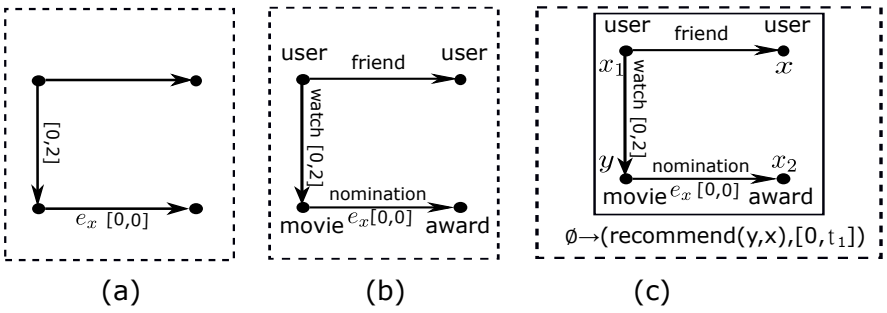


Figure 4.4: TACO generation process.

elwise expansion process of [FHLL20] to construct a set of valid dependencies $X \to (p_0, \tau)$ and form candidate TACOs for Σ. More specifically, for each possible event (p_0, τ), it starts with $X = \emptyset$ and iteratively extends the precondition X for ΔQ. The creator also verifies whether the candidate TACO $\varphi = \Delta Q[\bar{x}](X \to (p_0, \tau))$ is valid in G, *i.e.,* whether $G \models \varphi$. But unlike the algorithm in [FHLL20] that directly verifies the validity of φ after its generation, the creator first checks whether φ is redundant, *i.e.,* whether φ is implied by the set Σ_d of TACOs discovered in the previous iterations ($\Sigma_d \models \varphi$). If φ is not redundant, *i.e.,* $\Sigma_d \not\models \varphi$, it proceeds with the validation of φ by the critic. We find that checking implication in advance helps reduce the discovery cost, since implication is conducted on the discovered TACOs, which are much smaller than the temporal graph. In addition, we perform satisfiability checking of the TACOs in Σ and newly discovered candidate rules at this stage, to avoid further validation of inconsistent candidate TACOs in the large G. This process terminates when no more predicates can be added. The termination is guaranteed since predicates in X are defined on (a) nodes in ΔQ and (b) attributes and values (including timestamps) in ΔG_Q; hence the number of all possible predicates is bounded by the sizes of ΔQ and ΔG_Q.

Rule critic. For each generated candidate TACO φ in Σ that passes the implication and satisfiability checking, the rule critic computes $\mathsf{supp}(\varphi, G)$ and $\mathsf{conf}(\varphi, G)$, and selects high-quality TACOs whose support and confidence are above the thresholds (SelectRule in lines 5 and 10). These rules are added to Σ_d and are provided to the rule creator for improving generative models in the next iteration. Computing $\mathsf{supp}(\varphi, G)$ and $\mathsf{conf}(\varphi, G)$ is efficient since the procedures can be parallelized with optimizations that are unique to temporal pattern matching (see Section 4.5) and we use existing efficient subgraph matching method, *i.e.,* DAF [HKG+19] and its proposed CS structures to reduce redundant computation.

Example 4.5: Continuing with Example 4.4, Figure 4.4 shows the generation of TACO φ_1 from temporal graph G. First, the creator calls TagGen to create a can-

didate Δ-pattern (Fig. 4.4(a)) with time constraints. Then, as shown in Fig. 4.4(b), the LSTM model $\mathcal{M}_L$ is employed to add labels to nodes and edges with semantic meanings. The creator completes the generation by constructing dependencies with variables (Fig. 4.4(c)). Validated by the rule critic in G, this generated candidate rule is preserved in Σ_d, since its support and confidence are both 1. Note that if the edge labeled with nomination is dropped from φ_1, the confidence would reduce to 0.5. □

Performance guarantees. We next show that when the input iteration number I is sufficiently large, CCD can return all TACOs that satisfy the discovery requirement d with a high probability. Recall that when accumulating the training data in each round for the generative model, CCD conducts temporal random walk to randomly sample subgraphs from the entire graph, by function LocalizedSample. Based on this, we have the following.

Theorem 4.3: *Given graph G, requirement d and constant $\varepsilon \in (0,1)$, after* $\frac{|G|^\alpha}{N\beta}\left(1-\ln\frac{\beta\varepsilon}{|G|^\alpha} + \sqrt{\ln\frac{\beta\varepsilon}{|G|^\alpha}\left(\ln\frac{\beta\varepsilon}{|G|^\alpha}-2\right)}\right)$ *iterations,* CCD *can discover all* TACOs *satisfying d with probability at least* $1-\varepsilon$. □

Proof: Consider the discovery of a TACO $\Delta Q[\bar{x}](X \to (p_0,\tau))$ that satisfies the input requirement d. We assume that once a Δ-pattern $\Delta Q[\bar{x}]$ is sampled from G and used to train the generative models, that pattern $\Delta Q[\bar{x}]$ will be regenerated by the GAN and LSTM model in CCD. This assumption is reasonable as it is common to observe training samples in the outputs of deep generative models, *e.g.,* text generation models [dRP21, CWD+18]. Since we adapt the levelwise expansion process of [FHLL20] in CCD to construct dependencies $X \to (p_0,\tau)$, we can discover this TACO if the Δ-pattern $\Delta Q[\bar{x}]$ is sampled from G by CCD. Then it suffices to show that CCD can sample such Δ-pattern $\Delta Q[\bar{x}]$ with a high probability.

We prove this in the following four steps. (1) At first, we show that (a) the number of all possible Δ-patterns is bounded by $|G|^\alpha$, since these patterns are subgraphs of G (*i.e.,* $\mathsf{supp}(\Delta Q,\emptyset,G) \geq 1$), and (b) the total number of Δ-patterns that have support larger than β (*i.e.,* $\mathsf{supp}(\Delta Q,\emptyset,G) \geq \beta$) is bounded by $\frac{|G|^\alpha}{\beta}$. (2) We next bound the expected number of the appearances of $\Delta Q[\bar{x}]$ during the discovery in CCD. To this end, we define a random variable Y_M (resp. X_i) to denote the number of appearances of $\Delta Q[\bar{x}]$ that are generated after sampling M patterns in CCD (resp. when sampling the i-th pattern), *i.e.,* $Y_M = X_1 + X_2 + \cdots + X_M$. Then we can show that the expected

number of the appearances of $\Delta Q[\bar{x}]$ after sampling M patterns in G is larger than $M\frac{\beta}{|G|^\alpha}$, *i.e.,* $\mathbb{E}(Y_M) \geq M\frac{\beta}{|G|^\alpha}$. This is because that CCD samples patterns by temporal random walk. (3) Using the Chernoff–Hoeffding bound [DP09], we have that after sampling $M = \frac{|G|^\alpha}{\beta}\left(1 - \ln\frac{\beta\varepsilon}{|G|^\alpha} + \sqrt{\ln\frac{\beta\varepsilon}{|G|^\alpha}(\ln\frac{\beta\varepsilon}{|G|^\alpha} - 2)}\right)$ patterns, CCD can generate the Δ-pattern $\Delta Q[\bar{x}]$ with probability at least $1 - \frac{\beta\varepsilon}{|G|^\alpha}$. (4) Finally using the generalized bonferroni inequality [CB01], one can verify that for all Δ-patterns $\Delta Q[\bar{x}]$ with $\mathsf{supp}(\Delta Q, \emptyset, G) \geq \beta$ to be sampled after generating M patterns by CCD, the probability is at least $\frac{|G|^\alpha}{\beta}(1 - \frac{\beta\varepsilon}{|G|^\alpha}) - (\frac{|G|^\alpha}{\beta} - 1) = 1 - \varepsilon$. □

Cost analysis. To see the cost of CCD, observe the following. Function LocalizedSample takes $O(|G|)$ time for applying temporal random walk [NLR+18]. The cost for model training in TrainCreator is linear to the number of training samples, *i.e.,* $O(N + |\Sigma_d|)$. GenerateRule (creator) takes time polynomial in the size $|G|$ to generate Δ-patterns and dependencies. However, SelectRule (*i.e.,* critic) takes $O(|G|^\alpha)$ time to compute the support and confidence of each candidate TACO because of the graph homomorphism in temporal matching; nonetheless, we make use of parallelism (see Section 4.5) and the auxiliary structure in [HKG+19] to speed up the computation. As will be seen shortly, the parallelized process guarantees to reduce runtime when more processors are used, and is able to scale with large graphs.

Remark. (1) While the theoretical iteration number I may be large, in practice, when sample size N is set 250 by default, the generative models converge within 25 iterations (*i.e.,* I=25), where extra iterations add few novel TACOs to Σ_d (see Section 4.6). Therefore, we set I using a practical small value. This helps us reduce the overall cost and justifies the usage of deep generative models.

(2) The users may opt to inspect the generated TACOs in each round, select rules of their interest, and add to Σ_d. This incorporates user interests into discovery. They may also terminate the iterative process manually when they are satisfied with the TACOs in Σ_d so far.

4.5 Parallel Event Prediction

In this section we develop a parallel algorithm to support the event prediction module of TASTE. The algorithm is also used to compute support and confidence for the rule critic of algorithm CCD (Section 4.4). We propose a partitioning strategy for temporal

graphs, and show that the algorithm guarantees the parallel scalability.

We start with a sequential prediction approach with TACOs.

Sequential algorithm. Given a temporal graph G, a set Σ of TACOs and a time window τ_0, a sequential prediction algorithm, denoted as SeqEP, finds all edges (events) R predicted by Σ in G as follows. For each TACO $\varphi = \Delta Q[\bar{x}](X \to (p_0, \tau))$ in Σ with $\tau = [t_1, t_2]$, SeqEP (1) finds all matches of ΔQ in ΔG_Q via graph homomorphism; and (2) for each such match $h(\bar{x})$, it checks whether $h(\bar{x}) \models X$ and the time window $[t^* + t_1, t^* + t_2]$ deduced is a subinterval of τ_0; here t^* is the current time (Section 4.1.1); if so, SeqEP adds to R the edge that links the nodes matching the variables in p_0.

Note that SeqEP is applied to subgraphs ΔG_Q of G for ΔQ in Σ, which is typically much smaller than the entire graph G. In the sequel we refer to the union of such ΔG_Q's simply as G. Even so, the exponential cost of graph homomorphism [GJ79] in temporal pattern matching (step (1)) motivates us to parallelize SeqEP.

Parallel scalability. To measure the effectiveness of parallelization, we adapt the criterion introduced by [KRS90] to graph computation. Consider a problem $\mathcal{P}$ posed on a graph G. We denote by $T_s(|I_\mathcal{P}|, |G|)$ the worst-case complexity of a sequential algorithm $\mathcal{F}$ for handling an instance $I_\mathcal{P}$ of problem $\mathcal{P}$ over G. For a parallel algorithm $\mathcal{F}_p$, we denote by $T_p(|I_\mathcal{P}|, |G|, k)$ the time taken by it for processing problem instance $I_\mathcal{P}$ on G using k processors. We say that algorithm $\mathcal{F}_p$ is *parallelly scalable relative to* $\mathcal{F}$ if

$$T_p(|I_\mathcal{P}|, |G|, k) = O\big(\frac{T_s(|I_\mathcal{P}|, |G|)}{k}\big)$$

for any instance $I_\mathcal{P}$. That is, the parallel algorithm $\mathcal{F}_p$ achieves a "linear" reduction in sequential running time of a yardstick algorithm $\mathcal{F}$, allowing us to process large graphs by adding resources.

Parallelizing event prediction. A typical strategy for parallelizing sequential graph computation is to first partition a graph into k small fragments, and then conduct the computation over fragments at k processors in parallel with necessary message passing, *e.g.,* the deduction algorithm with GARs [FJL$^+$20]. Following this paradigm, we could partition a temporal graph via an existing graph partitioning method, *e.g.,* edge-cut or vertex-cut [RPGH14, AR06]. However, this easily incurs a large amount of communication in the subsequent parallel prediction. This is because most of the previous partitioning methods aim to minimize the number of cut edges or vertices but overlook the timestamps, which are crucial to temporal pattern matching; the edges in a match of a Δ-pattern are often partitioned into different fragments and demand

communication.

To rectify this problem, we partition a temporal graph G based on *temporal locality*, a unique property of temporal pattern matching with TACOs. That is, the timestamps of edges in each match $h(\bar{x})$ of Δ-pattern ΔQ are within the range of localized timespan $[t^* + t_{\mathsf{min}}, t^* + t_{\mathsf{max}}]$, where t_{min} (resp. t_{max}) is the minimum (resp. maximum) timestamp in ΔQ as stated in Section 4.1.1.

Temporal partitioning. We propose a temporal partitioning strategy with which parallel event prediction can be made communication-free. Intuitively, it divides a time interval into k subintervals and guarantees that every specific range of timestamps for finding match $h(\bar{x})$ is *entirely* covered by one subinterval. Guided by the resulting subintervals, the temporal graph G is partitioned into k fragments $F_1, \ldots, F_k$, such that each F_i consists of the edges whose timestamps are within one subinterval. By the temporal locality, temporal pattern matching and hence event prediction can be conducted in parallel over such F_i's with no communication.

The cost of parallel event prediction is determined by the maximum size $\max_{i\in[1,k]} |F_i|$ of fragments. Thus we want to find a good division of the time interval to minimize $\max_{i\in[1,k]} |F_i|$. To do this, we develop function BTPart, shown as part of Algorithm 4.2. It takes as input a candidate time interval $[t_0, t_k)$ that is deduced from the TACOs Σ and time window τ_0 for matching designated edges in event prediction (see below), a set $\{|G_t| \mid t \in [t_0, t_k)\}$ of sizes of t-graphs, the number k of fragments (processors), and the maximum (resp. minimum) timestamp t_{max} (resp. t_{min}) in Σ. Here G_t refers to a t-graph that is composed of all edges in G bearing timestamp t. BTPart computes a set $\{t_1, \ldots, t_{k-1}\}$ of $k-1$ partition points for the interval $[t_0, t_k)$ such that fragment F_i becomes $G_{[t_{i-1}+t_{\mathsf{min}}, t_i+t_{\mathsf{max}})}$ for $i \in [1,k]$, which includes the edges of G with timestamps in the range $[t_{i-1} + t_{\mathsf{min}}, t_i + t_{\mathsf{max}})$. Here t_{min} and t_{max} are to ensure the entire coverage as mentioned above.

Procedure BTPart adopts dynamic programming. It maintains a 2D array S, where $S[t][i]$ records the minimum size of the largest fragments that are obtained by partitioning $G_{[t_0+t_{\mathsf{min}}, t+t_{\mathsf{max}})}$ into i subintervals. Hence $S[t_k][k]$ is our objective value. BTPart first handles the base case i=1 (lines 1-2), where $S[t][1]$=$|G_{[t_0+t_{\mathsf{min}}, t+t_{\mathsf{max}})}|$ since there is only one fragment *w.r.t.* the single subinterval. Here the size is derived from that of the input t-graphs. For cases where i>1, $S[t'][i]$ is determined by checking the values regarding all possible ranges $[t_0, t)$ for t<t' and their $i-1$ subintervals (lines 3-5). After $S[t_k][k]$ is computed, the corresponding $k-1$ partition points can be identified and

are returned as the result (lines 6-8).

By induction on the iterations, one can verify that the returned partition points yield minimum $\max_{i\in[1,k]} |F_i|$ (*i.e.,* $S[t_k][k]$).

Algorithm 4.2: Parallel Event Prediction (ParEP)

Input: The number k of processors, a k-way randomly partitioned temporal graph G, a set Σ of TACOs, a time window τ_0.

Output: A set R of edges predicted by Σ with events occur within τ_0.

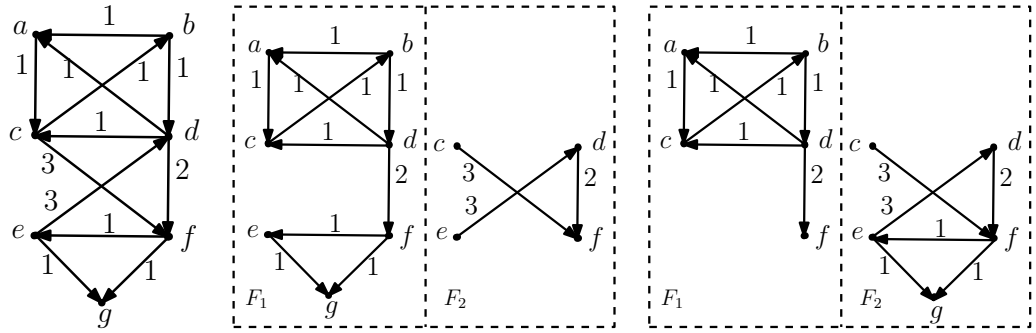
1 $[t_0,t_k) \leftarrow \mathsf{RefTime}(\Sigma,\tau_0)$; $(t_{\min},t_{\max}) \leftarrow \mathsf{ExtractTS}(\Sigma)$;
2 collect the size $|G_t|$ of t-graph G_t for $t \in [t_0,t_k)$;
3 $\{t_1,\ldots,t_{k-1}\} \leftarrow \mathsf{BTPart}([t_0,t_k),\{|G_t| \mid t\in[t_0,t_k)\},k,t_{\min},t_{\max})$;
4 $F_i \leftarrow G_{[t_{i-1}+t_{\min},t_i+t_{\max})}$ for each $i \in [1,k]$;
5 $\mathsf{RBalance}(\{F_i \mid i \in [1,k]\})$;
6 run $\mathsf{SeqEP}(F_i,\Sigma,\tau_0)$ at each fragment F_i to get R_i for $i \in [1,k]$;
7 **return** $\bigcup_{i\in[1,k]} R_i$;

Function $\mathsf{BTPart}([t_0,t_k),\{|G_t| \mid t \in [t_0,t_k)\},k,t_{\min},t_{\max})$**:**

1 **foreach** $t \in [t_0,t_k)$ **do**
2 $\quad S[t][1] \leftarrow |G_{[t_0+t_{\min},t+t_{\max})}|$;
3 **foreach** $i \in [2,k]$ **do**
4 $\quad$ **foreach** $t' \in [t_0,t_k)$ **do**
5 $\quad\quad S[t'][i] \leftarrow \min_{t\in[t_0,t')} \max(S[t][i-1], |G_{[t+t_{\min},t'+t_{\max})}|)$;
6 **foreach** $i \in [2,k]$ in descending order **do**
7 $\quad t_{i-1} \leftarrow \arg\min_{t\in[t_0,t_i)} \max(S[t][i-1], |G_{[t+t_{\min},t_i+t_{\max})}|)$;
8 $\quad$ **return** $\{t_1,\ldots,t_{k-1}\}$;

Parallel algorithm. Capitalizing on the temporal partitioning, we develop a parallel event prediction algorithm, denoted by ParEP. As shown in Algorithm 4.2, initially ParEP uses function RefTime to deduce a candidate time interval $[t_0,t_k)$ for those edges in G that can potentially match designated edges in TACOs Σ (line 1). Since we only predict edges in time window τ_0, a timestamp t is in $[t_0,t_k)$ if and only if the gap between t and τ_0 is smaller than that between t and τ for some (p_0,τ) in Σ. It also extracts maximum and minimum timestamps from Σ via ExtractTS (line 1). Then ParEP performs temporal partitioning to get k fragments F_1 to F_k and each F_i is assigned to a distinct processor (lines 2-4). ParEP designates one processor as the coordinator to collect the sizes of t-graphs from all processors and apply function BTPart.

To further balance the workload of parallel prediction, ParEP next redistributes the



(a) G (b) A temporal partition (c) A rebalanced partition

Figure 4.5: Temporal partitioning and rebalancing

data in large fragments F_i having $|F_i| > |G|/k$ by function RBalance (line 5). More specifically, for each large F_i, it evenly partitions those edges of F_i that can match the designated edges in Σ; while the set of candidate matches in F_i for other pattern edges in Σ are replicated at all processors. Here candidate matches are identified via label comparison. Then SeqEP is invoked at all processors in parallel with no communication to get the predicted edges, and their union is finally returned (lines 6-7).

Example 4.6: Consider partitioning the temporal graph G shown in Fig. 4.5(a) into 2 fragments, where only the timestamp of each edge is marked while labels are omitted to simplify the discussion. Suppose that $(t_{\min}, t_{\max}) = (0,1)$ for a given set Σ of TACOs and the candidate time interval $[t_0, t_k)$ deduced from Σ and the input time window τ_0 is $[1,4)$. We can see the following.

(1) The temporal partition generated by BTPart is shown in Fig. 4.5(b). Here the candidate interval is divided into $[1,2)$ and $[2,4)$; thus fragments F_1 and F_2 have edges whose timestamps are within $[1,2+1)$ and $[2,4+1)$, respectively. Note that the edge (d,f) with timestamp 2 is replicated at both fragments. This ensures that all patterns in Σ can be matched on F_1 or F_2 locally without communication.

(2) While Fig. 4.5(b) is an optimal temporal partition, it is skewed, *i.e.,* the edges with timestamp 1 in F_1 make a large part of G. By moving edges $(e,f),(e,g)$ and (f,g) with timestamp 1 to fragment F_2 via function ReBalance, the partition becomes balanced (Fig. 4.5(c)). □

The parallel scalability of ParEP is assured as follows, where the sequential SeqEP takes $O(|\Sigma||G|^{|\Sigma|})$ time in the worst case and the cost of ParEP is bounded by $O(|\Sigma|\frac{|G|^{|\Sigma|}}{k})$.

Theorem 4.4: ParEP *is parallelly scalable relative to* SeqEP. □

Proof: Sequential algorithm SeqEP takes $O(|\Sigma||G|^{|\Sigma|})$ time in the worst case due to

Dataset	$\|V\|$	$\|E\|$	Relation type	Timestamp interval
ICEWS18	23K	469K	256	1 day
GDELT	8K	2.2M	240	15 minutes
YAGO	11K	201K	10	1 year
WIKI	13K	670K	24	1 year
MovieLens	80K	10M	10	1 day
Amazon	12.2M	30.3M	5	1 day

Table 4.2: Datasets

graph homomorphism. For the complexity of ParEP, observe that (a) temporal partitioning needs $O(k|G|^2)$ time and (b) the cost of redistributing large fragments is bounded by $O(k|G|)$. (c) The computation workload of event prediction at each processor includes (1) handling the fragment that is not large, which takes $O(|\Sigma|(\frac{|G|}{k})^{|\Sigma|})$ time; and (2) the workload that is assigned via rebalancing, which is bounded by $O(k|\Sigma| \times \frac{|G|}{k}|G|^{|\Sigma|-2}) = O(|\Sigma||G|^{|\Sigma|-1})$. Note that since the candidate matches of designated edges in large fragments are evenly partitioned, we only check the matches of the other $|\Sigma|-2$ pattern nodes in the entire candidate set, with a cost of $O(|G|^{|\Sigma|-2})$. Putting these together, the complexity of ParEP is $O(|\Sigma|(\frac{|G|}{k})^{|\Sigma|} + k|G|^2 + |\Sigma||G|^{|\Sigma|-1})$ $\leq O(|\Sigma|\frac{|G|^{|\Sigma|}}{k})$ when $k \ll |G|$. Then from these the parallel scalability of ParEP follows.□

Remark. (1) ParEP can be readily adapted to predict whether a particular event p_0 will happen and when it will take place, by allowing users to set the range τ_0 and applying relevant TACOs in Σ. (2) We use ParEP to compute the support and confidence of TACOs in Σ, and support the functionality of the rule critic in algorithm CCD. To do these, the input time window τ_0 is defined as the smallest range that covers all the timestamps in G.

4.6 Experimental Study

Using real-life and synthetic graphs, we experimentally evaluated (1) the efficiency and (2) the quality of the creator-critic rule discovery method, (3) the accuracy of TASTE for event prediction and dynamic recommendation, and (4) the (parallel) scalability of ParEP.

Experimental setting. We start with the experimental setting.

Datasets. We used six real-life temporal graph benchmark datasets, shown in Table 4.2 and classified into three different classes: (1) event-based temporal knowledge graphs:

ICEWS18 [JQJR20] from the Integrated Crisis Early Warning System [BLO+15], and GDELT [JQJR20] from the Global Database of Events, Language, and Tone [LS13]; (2) knowledge graphs with temporally associated facts: YAGO [MBS15] and WIKI [LC18], *e.g.,* YAGO records the time when a football player plays for a club; and (3) dynamic recommendation datasets: MovieLens [HK16] of movie ratings and Amazon [MTSvdH15] of product ratings, where original timestamps were reorganized such that the time granularity between two adjacent timestamps is one day.

As designed in [JQJR20, LJL+21b], each dataset has been divided into training, validation and test sets, with proportion of 80%, 10% and 10%, respectively, by timestamps. The training and validation sets were used for model training and rule discovery, while the test set was for accuracy test. Each dataset includes ground truth of event (temporal edge) prediction results [JQJR20, LJL+21b], and the testing set actually poses the set of "queries" *w.r.t.* predicting temporal events.

We also designed a graph generator to create larger synthetic datasets, for evaluating the scalability. The synthetic graphs had up to $10M$ nodes and $1B$ edges in the range of 10000 timestamps, with labels, attributes and values drawn from 200 symbols.

Algorithms. The creator and critic in the rule discovery module were implemented in Pytorch and C++, respectively, while the temporal event prediction module was implemented in C++. We compared TASTE with five baseline methods for event prediction: (1) AGER, which apply GERs [BBBG09b] that capture local changes in temporal graphs for event prediction; (2) SACN [STH+19b], a convolution-based embedding approach for knowledge graph completion; (3) REGCN [LJL+21b], a knowledge graph reasoning method based on Graph Convolution Network; (4) Caser [TW18b], a sequential recommendation algorithm based on convolutional neural networks; and (5) SASRec [KM18], a transformer-based sequential recommender system. We adopted open-source codes of SACN [STH+19a], REGCN [LJL+21a], Caser [TW18a] and SASRec [KM19] with default configurations, and implemented AGER in C++. We have also implemented two levelwise search-based rule mining methods to discover GERs and TACOs in C++, denoted as GERMine and TACOMine, as discovery baselines.

The baselines SACN, REGCN, Caser and SASRec were parallelized with multithreads by PyTorch. For AGER, GERMine and TACOMine, we applied the same parallelization method of ParEP (Section 4.5) to compute the corresponding matches, which dominate their costs. The thread number in parallelization is equal to the number k of

I \ N	50	100	150	200	250	300
15	20.00%	23.81%	28.57%	39.05%	48.57%	65.71%
20	23.81%	25.71%	40.95%	55.24%	66.67%	73.33%
25	30.47%	41.90%	44.76%	71.43%	84.76%	96.19%
30	39.05%	44.28%	48.94%	76.15%	87.23%	97.14%

Table 4.3: Quality of the creator-critic discovery on ICEWS18

cores used by TASTE, for a fair comparison.

ML models. REGCN and SASRec were adopted as the ML predicates in TACOs for temporal graph completion and dynamic recommendation, respectively. For models in the creator of CCD, we used the code of TagGen provided by the authors with default configurations [ZZHH20b], and implemented the LSTM model as [MKS18] with its default training configuration and two 650-wide layers.

We conducted experiments on a cluster of up to 72 Intel Xeon 3.1 GHz processing cores on two machines connected by 10Gbps links, with 256GB memory. By default we set α=5, β=100, γ=0.8 and δ=20 for discovery requirement; the number of iterations I=25 and sample size N=250 for discovery module CCD; and the number of cores $k = 32$ for prediction method ParEP, unless stated otherwise. All the experiments were repeated 5 times. The average is reported here.

Experimental results. We next report our findings. In every experiment, the results on at least one graph from each of the three classes are shown. We defer the observations on other graphs to the Appendix A.

Exp-1: Efficiency. We first compared the efficiency of CCD, GERMine and TACOMine for rule discovery. Since it is very costly for levelwise methods to discover rules with large patterns, in order to compare efficiency within bearable running time, we set a target of discovering 100 high-quality rules as benchmark. That is, each discovery process terminated when 100 rules had been discovered.

(1) Varying α. We varied α from 3 to 11 to study the impact of pattern node number on discovery methods using ICEWS18, WIKI and MovieLens. As shown in Figures 4.6(a) to 4.6(c), CCD is on average 9.1 and 14.3 times faster than GERMine and TACOMine when $\alpha \leq 5$, respectively. The computation time of levelwise search-based methods grows exponentially as α gets larger, and both GERMine and TACOMine cannot terminate in 1.2 days when $\alpha > 5$, while the cost increase of CCD is mild. This

is because larger number of pattern nodes incurs exponentially larger search space for levelwise mining, but little extra cost for ML generative models. In particular, CCD finds TACOs with patterns of more than 20 edges in 1639s.

(2) Varying β. We varied β from 50 to 150, to study the impact of support threshold over ICEWS18, YAGO and Amazon. As reported in Figures 4.6(d) to 4.6(f), the runtime of CCD does not change much, as it always takes multiple iterations for the creator to generate high-quality rules regardless of the value of β. In contrast, GERMine and TACOMine take less time with larger β in most cases, since higher bound on support prunes more candidates and reduces search space.

(3) Varying γ. Varying confidence γ from 0.7 to 0.9, we report the results on GDELT, WIKI and Amazon in Figures 4.6(g) to 4.6(i), respectively. It is shown that the all discovery algorithms take longer time with the increase of γ. However, the generation-based CCD is less sensitive to γ. This is because the levelwise methods have to expand their search space at an exponential scale to get the requested number of high-quality rules with higher confidence.

(4) Varying δ. We varied δ from 10 to 30. Here δ counts the number of discrete timestamps in the required time window. As shown in Figures 4.6(j) to 4.6(l) on ICEWS18, YAGO and MovieLens, respectively, the runtime for all three increases as δ grows, since a longer time window bound gives more generation workload for TagGen in the creator and expands the search space for GERMine and TACOMine.

(5) Impact of N and I. We also tested the impact of the sample size N and iteration number I on the efficiency of CCD. It exhibits a moderately runtime increase with the increase of N or I (not shown). This is as expected, since a larger N (resp. I) causes more work for the generator (resp. more rounds of the entire computation).

Exp-2: Quality of discovery. Recall that TACOs obtained by the generation-based CCD are a subset of those returned by levelwise search algorithms. Thus we checked how many rules in the complete set found by levelwise method TACOMine can be discovered by CCD in all the datasets, *i.e.,* the coverage of complete TACOs. As reported in Table 4.3, the coverage becomes higher with larger size N of samples or more training iterations I. This is because enlarging either N or I could increase the possibility of approximating the target distribution and hence generating high-quality rules, as discussed in Section 4.4. We can also see that small N and I suffice to get a large portion of high-quality TACOs, *e.g.,* the coverage reaches 84.76% when $N = 250$ and $I = 25$,

while the time of TACOMine is reduced by 18.5 times simultaneously (see Exp-1).

In addition to the adopted GAN model, we tested the performance of using classic graph generation models, *i.e.,* Erdös-Rényi (ER) [PA59] and Barabási-Albert (BA) [AB02] models, in CCD. We find that when $N = 300$ and $I = 30$, the coverage vlaues of the TACOs found with ER and BA are merely 33.07% and 50.89%, respectively, much lower than that by GAN (see detailed results in the Appendix A). This is because these classical models cannot iteratively learn from the graph data and generate high-quality patterns in an adaptive manner.

We also manually checked the discovered TACOs. Besides the typical ones that include logic and temporal predicates only, some TACOs can also help enrich or interpret ML predictions (see the Appendix A).

Exp-3: Accuracy. As shown in Table 4.4, we evaluated the accuracy of TASTE with the TACOs and baselines on two tasks: temporal event prediction and dynamic recommendation [LJL$^+$21b, KM18]. Note that there was no result for SACN and REGCN (resp. Caser and SASRec) on dynamic recommendation (resp. temporal event prediction) datasets since they are not designed for the task. TASTE (ParEP) applied the discovered TACOs with confidence above 0.9, in which graph patterns have at most 9 nodes. We find that very few TACOs with more than 9 pattern nodes have high confidence and support, similar to the findings of frequent pattern mining [EASK14]. We adopted Hit Rate@10, the fraction of times that the ground-truth item is among the top 10 items [KM18, LJL$^+$21b], to evaluate the accuracy.

For event prediction on ICEWS18, GDELT, YAGO and WIKI, TASTE on average outperforms AGER, SACN and REGCN by 34.9%, 24.5% and 12.2%, respectively. As for the dynamic recommendation on MovieLens and Amazon, TASTE is 35.8%, 22.5% and 10.6% more accurate than AGER, Caser and SASRec, respectively. These show that by combining rules and ML models, TASTE beats the state-of-the-art deep-learning-based REGCN and SASRec in accuracy on both tasks, while none of REGCN and SASRec works on both.

Exp-4: Scalability. We finally evaluated (1) the parallel scalability of the prediction module ParEP in TASTE system by varying the number k of processors, (2) the impact of pattern size on its efficiency, and (3) the scalability of ParEP over larger synthetic graphs. Since GERs are a special case of TACOs, ParEP is also applicable to GERs and the runtime of AGER is not shown.

(1) Parallel scalability. Varying k from 4 to 64, Figures 4.6(m)-4.6(o) report the re-

Dataset	AGER	SACN	REGCN	Caser	SASRec	TASTE
ICEWS18	59.32%	63.28%	68.58%	-	-	75.24%
GDELT	54.60%	62.32%	66.31%	-	-	73.41%
YAGO	61.32%	64.03%	74.73%	-	-	84.14%
WIKI	58.56%	63.58%	71.32%	-	-	82.53%
MovieLens	65.80%	-	-	75.83%	80.45%	87.30%
Amazon	59.30%	-	-	63.50%	73.10%	73.10%

Table 4.4: Event prediction/recommendation accuracy

sults on GDELT and WIKI for event prediction, and on MovieLens for recommendation in the same setting as Exp-3, respectively. As shown there, (a) ParEP is parallelly scalable. When k increases from 4 to 32, it is on average 3.2 times faster on the three graphs. This verifies the effectiveness of ParEP under data-partitioned parallelism. (b) In addition to its higher accuracy, ParEP performs better in efficiency than SACN and REGCN (resp. Caser and SASRec), *e.g.,* when $k = 64$, it is on average 24.2, 45.4, 6.2 and 5.7 times faster than SACN, REGCN, Caser and SASRec, respectively.

(2) Impact of pattern size. Varying the size $|\Delta Q|$ of Δ-patterns in TACOs, which is measured as the sum of the pattern node and edge numbers in each ΔQ, we report the performance of different methods in Figures 4.6(p) to 4.6(r). The results show that ParEP becomes slower with the increase of $|\Delta Q|$, as expected. Nonetheless, it is still efficient when handling relatively large Δ-patterns. For instance, it needs 1645 seconds on GDELT when $|\Delta Q| = 15$, which is better than 16380 seconds by SACN (see Figure 4.6(m)).

(3) Scalability. Fixing $k = 32$, we varied the size $|G|=|V|+|E|$ of synthetic graphs G using a scale factor from 0.2 to 1.0, and evaluated all approaches, where ParEP applied 100 TACOs. As shown in Figures 4.6(s) and 4.6(t) for prediction and recommendation, respectively, ParEP outperforms the baselines in all cases. On average it takes 1403s when $|G|$=810M, while the others cannot finish in 1 day.

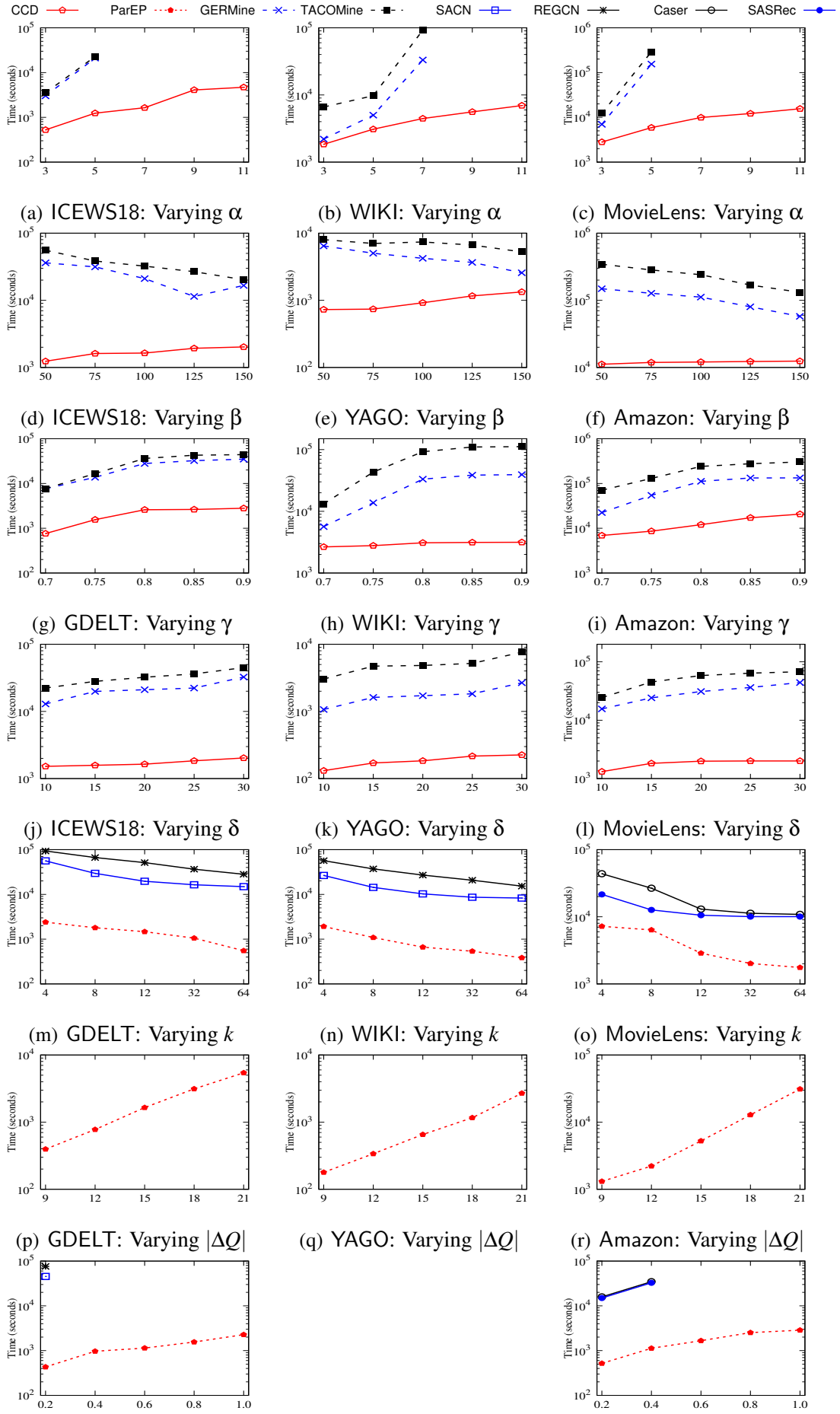
Summary. We find the following. (1) On average the generative ML method of CCD outperforms the levelwise algorithms in efficiency by more than 31 times. It is able to discover TACOs with patterns of 20 edges in 1639s from temporal graphs, while the levelwise methods could not finish in 1.2 days. (2) CCD is able to find as high as 84.76% of the complete rules derived by levelwise method, using 250 samples and 25 training attempts. (3) By combining rules and ML models, on average the discovered TACOs improve the existing approaches by 23.8% and 23.0% in accuracy, for event

prediction and dynamic recommendation, respectively. (4) Our algorithm ParEP is parallelly scalable and scales well with the datasets, it takes less than 1403s on graphs with $810M$ nodes and edges using 32 processors.

4.7 Novelty and Contributions

Our TACOs-based approach to event prediction proposed in this Chapter differs from the prior work in the following. (1) As an effort to unify rule-based and ML-based methods for event prediction, we propose TACOs by embedding event-prediction ML models as predicates. This allows us to not only leverage existing ML models, but also refine the ML models with logic conditions. (2) Unlike existing ML-based prediction strategies (*e.g.,* DRS) that adopt unexplainable blackbox models, our method with TACOs offers a logic interpretation of the ML predictions for temporal events. (3) TACOs are more expressive than GARs, GTARs and GERs. As will be seen in Section 4.1, these rules can all be expressed as TACOs. TACOs may embed prediction ML methods beyond GTARs and GERs, and support various temporal conditions on different events, while GTARs and GERs can only express constant time intervals. (4) We establish the complexity of classical problems for temporal graph rules. While these problems were studied for GARs on static graphs, no prior work has considered these for GTARs or GERs. (5) We propose a new approach to learning graph rules, which departs from prior mining methods for GARs, GTARs and GERs.

The proposed creator-critic rule learning framework is also different from existing rule learners as follows: (1) we study rule discovery in temporal graphs, while the prior learners focus on static graphs. (2) We propose to discover TACOs with general topological structures, beyond merely paths. (3) We discover TACOs that may carry ML predicates. (4) We propose a deep-generative-model-based approach to learning graph rules with large graph patterns. In contrast, conventional mining methods for graph rules [FWWX15, FHLL20] can hardly find rules with patterns of 7 edges or more.



(a) ICEWS18: Varying α (b) WIKI: Varying α (c) MovieLens: Varying α

(d) ICEWS18: Varying β (e) YAGO: Varying β (f) Amazon: Varying β

(g) GDELT: Varying γ (h) WIKI: Varying γ (i) Amazon: Varying γ

(j) ICEWS18: Varying δ (k) YAGO: Varying δ (l) MovieLens: Varying δ

(m) GDELT: Varying k (n) WIKI: Varying k (o) MovieLens: Varying k

(p) GDELT: Varying $|\Delta Q|$ (q) YAGO: Varying $|\Delta Q|$ (r) Amazon: Varying $|\Delta Q|$

(s) Synthetic: Varying $|G|$ (predict.)

(t) Synthetic: Varying $|G|$ (recomm.)

Figure 4.6: Performance evaluation

Chapter 5

Joins across Relations and Graphs

This Chapter proposes an approach to querying relations $\mathcal{D}$ and graphs G taken together in SQL. The need for querying relations and graphs taken together has been increasingly evident. While most business data resides in relational database management systems (RDBMSs), graphs have found prevalent use in practice, *e.g.,* transaction graphs, knowledge bases, social networks and road networks. Gartner predicts that the practice of graph analytics will double annually [Bus20]. With this comes the need for writing queries across relations and graphs, to synthesize relevant information pertaining to the same real-life entities.

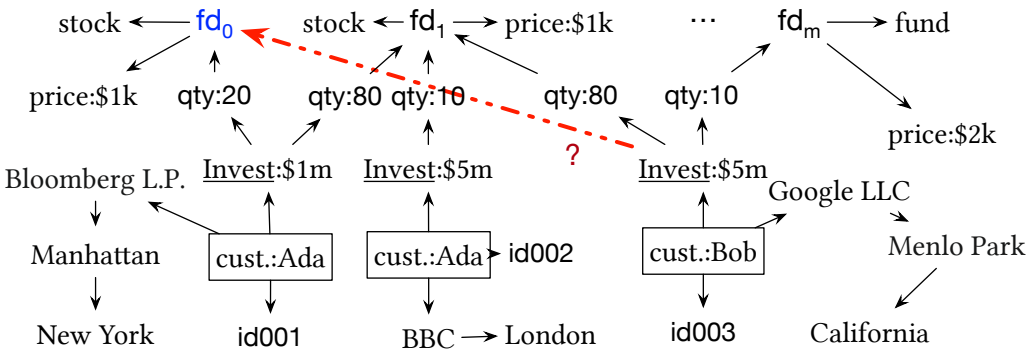
Example 5.1: Consider a relational database $\mathcal{D}$ of customer accounts and a graph G of transactions at a bank as shown in Fig. 5.1. To decide whether to recommend a new financial product fd_0 to Bob (cid02 in $\mathcal{D}$), the bank checks (a) whether Bob has good credit (in $\mathcal{D}$), and (b) whether there exists a customer who has invested in fd_0 (in G), has account balance close to Bob's (in $\mathcal{D}$), and has co-invested the same amount as Bob in a financial product at the same price as fd_0 (in G). For instance, Ada (cid04 in $\mathcal{D}$ and id001 in G) from NYC is such a customer. To check these, however, the bank has to synthesize relational data from $\mathcal{D}$ and transactions from graph G. □

No matter how desirable, it is nontrivial to write a query that accesses data in a relational database $\mathcal{D}$ and a graph G, aligns entities across $\mathcal{D}$ and G, and correlates their information, in particular when $\mathcal{D}$ and G are independent data sources as often found in practice. While relations in $\mathcal{D}$ have a regular structure, graph G is often schemaless. Entities in G are typically encoded as vertices v, as opposed to tuples t in $\mathcal{D}$. As a consequence, the same entity often has radically different "topological" structures in relations and graphs. Worse still, a property of entity v in graph G is

R

cid	name	credit	bal	city
cid01	Ada	fair	$500k	LDN
cid02	Bob	good	$110k	SF
cid03	Guy	good	$50k	NYC
cid04	Ada	fair	$100k	NYC

(a) Customer database $\mathcal{D}$



(b) Transaction knowledge graph G

Figure 5.1: Relational database $\mathcal{D}$ and graph G in Example 5.1

often linked via a path from v, in contrast to "local" attributes of a relational tuple. For instance, the price of Ada(id001)'s transaction for fd_1 is linked to Ada via a path (Invest, qty, fd_1, price) in Fig. 5.1. Moreover, such paths may have varying lengths, *e.g.,* New York is connected to Ada id001 via a path of 3 edges while London is connected to Ada id002 via a path of 2 edges. Here the location vertices are needed to decide which Ada in G (id001 or id002) matches Ada cid01 in $\mathcal{D}$.

Add to the complication that practitioners often want to write queries against $\mathcal{D}$ and G in SQL. Indeed, our FinTech collaborators are used to SQL and want to hold on to the query planers and optimizers of RDBMSs. One might want to first shred a graph into relations and then write SQL queries. However, it would require costly joins for traversing paths to link entities and find attributes [FGJ+22].

In light of these, we are aware of no systems that support SQL queries across relations and graphs, neither RDBMSs and graph (database) systems [neo, AG08, DJL+16, GXD+14, FYX+18], nor modern federated systems [QD16, JRW+14, SAL+17, FRP15, ZY17, Des18, ABA+09, ZR11, KAB+12, DES+15, HdAC+14, KBV+16]. To answer the question of Example 5.1, our FinTech colleagues have to write multiple queries on $\mathcal{D}$ and G separately, and go between RDBMS and graph systems back and forth. This often incurs a heavy cost.

A new approach. We propose a framework, referred to as RGAP (Relation GrAPh),

to support SQL across a relational database $\mathcal{D}$ and a semistructured graph G. RGAP advocates the following strategies.

(1) Semantic join. Assume an "oracle", referred to as HER for Heterogeneous Entity Resolution. Given a tuple t in $\mathcal{D}$ and a vertex v in G, HER checks whether t and v make a *match*, *i.e.,* they refer to the same real-world entity, despite their radically different topological structures. Then we can naturally "join" the same entities encoded by tuples in relations and by vertices in graphs, collect relevant attributes pertaining to the entities, and correlate their information. This is a simple semantic extension to the join operator in SQL.

Several accurate methods are in place for HER, either machine learning (ML) classifiers such as DeepMatcher [MLR+18], JedAI [PMG+20], MAGNN [FZMK20], or parametric simulation that embeds ML models in topological matching and inductively inspects "deep" features [FGJ+22].

(2) Static and dynamic joins. We extend relational algebra (RA) with syntactic sugar to support semantic joins across $\mathcal{D}$ and G, referred to as GRA (Graph Relational Algebra). Given a set S of tuples of a relation schema R, we deduce the following:

- $f(S,G)$, the set of pairs (t,v) such that $t \in S$ and v is vertex in G that matches t, by invoking HER;
- a relation schema R_G to augment R with additional attributes from G, *i.e.,* properties of the matching vertices in $f(S,G)$; and
- an instance $h(S,G)$ of schema R_G, by value extraction from G.

We join entities t and v as long as they make a match in $f(S,G)$, and complement tuples t with additional attributes of $h(S,G)$. More specifically, we support two different forms of join:

- *static join*: when S is a set of input tuples in $\mathcal{D}$; and
- *dynamic join*: when S is the intermediate result of a sub-query.

We show how to efficiently implement the two forms of joins. While we can cache $f(S,G)$ and $h(S,G)$ in advance for static joins, for dynamic joins, naive implementation would require to compute S, $f(S,G)$ and $h(S,G)$ at run time on the fly, with a heavy cost. We show how to reduce dynamic joins to static joins for a large class of practical queries, making them accessible within RDBMS.

(3) Attribute extraction. To support semantic joins, one has to compute the match relation $f(S,G)$ and extract relations $h(S,G)$ for the matches from G. While $f(S,G)$ can

be identified by HER [PMG+20, MLR+18, IJB10, FZMK20, FGJ+22], no method is available to deduce schema R_G and relation $h(S,G)$. This is nontrivial since we have to identify important properties by traversing paths from the matching vertices in $f(S,G)$. We propose a clustering-based method to deduce R_G and $h(S,G)$. Taking a set Ω of keywords from users as input, we rank and pick attributes that cover most vertex-path pairs in G, meet users' query interests (represented by Ω), and diverse from existing attributes of R.

(4) Heuristic dynamic joins. For those queries in which dynamic joins cannot be reduced to static joins, we propose a *heuristic join* method to reduce the cost. To do this, for selected types τ of entities in G, we adjust the clustering method for (3) to deduce

- a schema R_τ with attributes that cover important properties of τ-entities in G and match users' interests; and
- an instance $g_\tau(G)$ of schema R_t extracted from G.

With the extracted relations $g_\tau(G)$, given a dynamic join between intermediate query results S and graph G, we reduce it to relational ER between S and $g_\tau(G)$, which can be realized in RDBMS with a simple UDF. By caching $g_\tau(G)$ in advance, this strikes a balance between the efficiency and accuracy of dynamic joins, without calling external HER and extraction functions at run time. We also develop incremental algorithms to extract $h(S,G)$ and $g_\tau(G)$ in response to updates to database $\mathcal{D}$ and graphs G.

This Chapter aims to enrich RDBMSs with a capacity of semantically querying relations $\mathcal{D}$ and graphs G in SQL. Its novelty consists of the following: (1) a notion of semantic joins; (2) methods for implementing static joins and dynamic joins within and atop existing RDBMS; (3) clustering-based methods to extract (a) attributes from graphs to complement relations, and (b) a schema for entities of a specific type in a graph; and (4) a method for incrementally maintaining extracted relations in response to updates.

The need for synthesizing data in $\mathcal{D}$ and G also arises from data lakes [NZM+19]. Given a query Q on $\mathcal{D}$, open challenges in data lakes include: (a) *query-driven data discovery* to find relevant graphs with vertices matching tuples in $Q(\mathcal{D})$; (b) *on-demand data integration* to augment tuples in $Q(\mathcal{D})$ with relevant properties of matching vertices; and (c) *data extraction* to abstract a schema and relations from raw data in graphs. This work sheds lights on these aspects.

The rest of this Chapter is organized as follows. We propose semantic joins in Section 5.1, develop our attribute extraction scheme in Section 5.2, and present the

schema extraction method, heuristic joins and incremental algorithm in Section 5.3. The experimental study is reported in Section 5.4. We finally conclude our novelty and contributions compared with previous work in Section 5.5.

5.1 Semantic Join

In this section, we introduce RGAP, an approach to querying relations and graphs in SQL. We first extend relational algebra and SQL with semantic joins (static and dynamic, Section 5.1.1). We then show how to implement semantic joins in existing RDBMSs (Section 5.1.2).

We consider relational databases and graphs, reviewed as follows.

Relations. A database schema is $\mathcal{R} = (R_1, \ldots, R_n)$, in which each R_i is a relation schema of the form $R_i(A_1, \ldots, A_{k_i})$, and A_i is an attribute. A *database* $\mathcal{D}$ *of* $\mathcal{R}$ is $(D_1, \ldots, D_n)$, where D_i is a *relation* of R_i ($i \in [1,n]$). We assume *w.l.o.g.* that each tuple t carries a tuple id.

Graphs. We consider *directed labeled graphs* $G = (V, E, L)$, where (a) V is a finite set of vertices, (b) $E \subseteq V \times V$ is a set of edges, and (c) L is a function such that for each vertex $v \in V$ (resp. edge $e \in E$), $L(v)$ (resp. $L(e)$) is a vertex (resp. edge) label. While edge labels typify predicates, vertex labels may carry values. Graph G may be a transaction graph, a social network, a knowledge base, etc.

A *path* ρ *from a vertex* v_0 in graph G is a list $\rho = (v_0, v_1, \ldots, v_l)$ such that (v_{i-1}, v_i) is an edge in E for $i \in [1,l]$. The *length of* ρ, denoted by $\mathsf{len}(\rho)$, is l, *i.e.,* the number of edges on path ρ. A path is *simple* if $v_i \neq v_j$ for $i \neq j$, *i.e.,* a vertex appears on ρ at most once. We consider simple paths in the sequel, simply referred to as paths.

5.1.1 Extending Relational Algebra

We start with parameter functions taken by RGAP.

Parameters. RGAP assumes the availability of the following.

*Heterogeneous Entity Resolution (*HER*)*. HER provides a function f that given a graph G and a set S of tuples, computes a set $f(S,G)$:

$$f(S,G) = \{(t,v) \mid t \in S, v \in V \text{ in } G, t \Rightarrow v\}.$$

Here $t \Rightarrow v$ denotes that tuple t and vertex v make a *match*, *i.e., t* and v refer to the same

real-world entity. We refer to f as the HER *function* and $f(S,G)$ as the *match relation* of S and G.

We denote by $R_m(\mathsf{tid},\mathsf{vid})$ *the schema of the match relation*, such that a tuple $(t.\mathsf{id}, v.\mathsf{id})$ of schema R_m denotes that (t,v) is a match in $f(S,G)$ for tuple t with $t.\mathsf{id}$ and vertex v with $v.\mathsf{id}$.

We support rule-based JedAI [PMG+20], parametric simulation [FGJ+22], and ML models DeepMatcher [MLR+18], Silk [IJB10], MAGNN [FZMK20] as HER.

Attribute Extraction. Given a graph G, a relation schema R, and a set Ω of keywords that indicates users' interest, an *attribute extraction* scheme deduces a schema R_G and a population function h:

- $R_G = (\mathsf{vid}, B_1, \ldots, B_m)$, where vid denotes a vertex v that matches tuples of R by HER, and B_i's are features of v; and
- h is a function that given a set S of tuples of R, returns an instance $h(S,G)$ of schema R_G by extracting corresponding properties of the vertices in $f(S,G)$ that match tuples in S.

As will be seen in Section 5.2, R_G is composed of attributes $B_1,\ldots,B_m$ that (a) meet users' interests indicated in Ω, and (b) are absent from schema R, as additional and alternative attributes to complement R. Function h populates $h(S,G)$ by traversing paths in G from matching vertices in $f(S,G)$. We refer to R_G as the *extracted schema* for R from G, and to $h(S,G)$ as the *extracted relation* for S from G.

We refer to the extraction method as AExt (attribute extraction).

Algebra. Consider a graph G and a database $\mathcal{D} = (D_1,\ldots,D_n)$ of schema $\mathcal{R} = (R_1,\ldots,R_n)$. We extend relational algebra (RA) to Graph Relational Algebra (GRA) across $\mathcal{D}$ and G, as follows:

$$\begin{aligned} Q ::= {} & R \mid \pi_X Q \mid \sigma_C Q \mid Q_1 \times Q_2 \mid Q_1 \cup Q_2 \mid Q_1 \setminus Q_2 \\ & \mid R \bowtie_f G[\vec{B}] \mid Q_1 \bowtie_f G[\vec{B}/\Omega]. \end{aligned}$$

Here $R \in \mathcal{R}$ is a relation atom, and operators $\pi, \sigma, \times, \cup$ and $\setminus$ are projection, selection, Cartesian product, set union and set minus as in RA, respectively. Just like their counterparts in RA, they operate on relation(s) and return a relation as their result. In addition, the predicates of σ_C and π_X may be defined over attributes not only in schemas of $\mathcal{R}$ but also in the extracted schemas R_G (see below).

GRA extends RA with new operations $R \bowtie_f G$ and $Q_1 \bowtie_f G$.

(1) Static joins. We refer to $R \bowtie_f G[\vec{B}]$ as a *static join*, which operates on (D, G), where D is an *input relation* of schema $R \in \mathcal{R}$. It returns a relation of the schema of $R \bowtie R_m \bowtie R_G[\vec{B}]$, where R_m is the match relation, and R_G is the extracted schema of R, such that

$$R \bowtie_f G[\vec{B}] = \{(t, t'[\vec{B}]) \mid (t, v) \in f(D, G), (v.\mathsf{id}, t') \in h(D, G)\}.$$

Here f is the match function that returns matches (t, v), and $h(D, G)$ is the relation extracted from G. Moreover, $\vec{B} \subseteq (B_1, \ldots, B_m)$ in schema $R_G = (\mathsf{vid}, B_1, \ldots, B_m)$. As will be seen in Section 5.1.2, R_G is extracted offline in advance and $\vec{B}$ is known to the users.

Intuitively, for each match (t, v) in $f(D, G)$, we extract attributes pivoted at vertex v and append them to t. Since t and v denote the same entity e, the extracted attributes complement the information of e encoded by t. In contrast to conventional joins in RA, $R \bowtie_f G[\vec{B}]$ connects tuples t and vertices v if they *semantically* refer to the same entity, and expand t with correlated attributes from graph G.

(2) Dynamic joins. We refer to $Q_1 \bowtie_f G[\vec{B}/\Omega]$ as a *dynamic join*, where Q_1 is a sub-query that returns a set S of tuples of a relation schema R_{Q_1}. Here R_{Q_1} is deduced from Q_1 and $\mathcal{R}$, and is not necessarily an input schema in $\mathcal{R}$. Similarly, S is dynamically computed and is not included in $\mathcal{D}$. The dynamic join also returns a relation of the schema of $R_{Q_1} \bowtie R_m \bowtie R_G[\vec{B}]$, where R_m is the schema of the match relation, R_G is the extracted schema from G for R_{Q_1}, and

$$Q_1 \bowtie_f G[\vec{B}/\Omega] = \{(t, t'[\vec{B}]) \mid S = Q_1(\mathcal{D}), (t, v) \in f(S, G), (v.\mathsf{id}, t') \in h(S, G)\}.$$

As opposed to static join, the match relation $f(S, G)$ and extracted relation $h(S, G)$ can only be computed online after S is available, which cannot be determined statically and materialized in advance. Users may specify a set $\vec{B}$ of desired attributes from graph G; alternatively, they may provide a set Ω of keywords to hint at attributes and values they would like to have from G via examples (see below).

Intuitively, dynamic join computes match relation $f(S, G)$ by invoking HER and extracts relation $h(S, G)$ by calling AExt *after* sub-query Q_1 returns relation S. It also deduces schema R_{Q_1} of S and schema R_G. After $f(S, G)$ and $h(S, G)$ are in place, $Q_1 \bowtie_f G$ is computed via two conventional joins just like static join above.

In contrast to relational joins, semantic joins create new attributes extracted from graphs that are not in the input schema $\mathcal{R}$.

About $\vec{B}$ and Ω. RGAP works with semantic joins as follows. Consider a relational database $\mathcal{D}$ of schema $\mathcal{R}$ and a graph G.

(1) As offline initialization, RGAP computes the following: (a) for each schema R of $\mathcal{R}$ and the instance D of $\mathcal{D}$, matches $f(D,G)$, extracted schema R_G and relation $h(D,G)$ (see Section 5.2); and (b) for each type τ of entities, a relation schema R_τ (Section 5.3.1). Users may provide keywords of their interests to guide the extraction.

(2) When users write queries with static joins, it suffices to reference the extracted schema R_G, which is pre-computed. For dynamic joins, users may specify attributes $\vec{B}$ by referencing R_τ, or provide keywords Ω to exemplify attributes and values expected from G. In the latter case, RGAP either uses Ω to guide attribute extraction for dynamic evaluation (Section 5.1.2), or matches the keywords to those attributes extracted in advance for heuristic joins (Section 5.3).

Example 5.2: Continuing Example 5.1, the recommendation can be formulated as a query q using either static or dynamic join.

(1) Static join. A GRA with static join for q consists of T^s and q_s:

$$T^s = R \bowtie_f G[\mathsf{qty}, \mathsf{prod}, \mathsf{price}]$$

$q_s = \sigma_{C_1} T_1^s \bowtie_{C_2} T_2^s \bowtie_{C_3} T_3^s$, where

- $C_1 : T_1^s.\mathsf{cid} = \mathsf{cid02} \wedge T_1^s.\mathsf{credit} = \mathrm{good}$
- $C_2 : T_1^s.\mathsf{bal} \approx T_2^s.\mathsf{bal} \wedge T_1^s.\mathsf{prod} = T_2^s.\mathsf{prod} \wedge T_1^s.\mathsf{qty} = T_2^s.\mathsf{qty} \wedge T_2^s.\mathsf{price} = p_0$
- $C_3 : T_2^s.\mathsf{cid} = T_3^s.\mathsf{cid} \wedge T_3^s.\mathsf{prod} = \mathsf{fd}_0$

Here (a) R is the input customer relation (see Fig. 5.1), and attributes qty, prod, price are from the extracted schema R_G of R; (b) $x \approx y$ checks whether x is close to y, *e.g.,* whether $|x-y| \leq c$ for some pre-defined constant c; (c) p_0 is the price of the new product fd_0; (d) HER function f links $\mathcal{D}$ and G in relation $R_m(\mathsf{cid}, \mathsf{vid})$, where cid01 matches id002, cid02 matches id003, and cid04 matches id001; and (e) T_1^s, T_2^s and T_3^s rename intermediate relation T^s.

(2) Dynamic join. One can also write q by using dynamic join:

$$E = \pi_{R'.\mathsf{cid}}(\sigma_{R.\mathsf{cid}=\mathsf{cid02} \wedge R.\mathsf{credit}=\mathrm{good}} R \bowtie_{R.\mathsf{bal} \approx R'.\mathsf{bal}} R')$$

$$T^d = E \bowtie_f G[\mathsf{qty}, \mathsf{prod}, \mathsf{price}]$$

$q_d = T_1^d \bowtie_{C_4} T_2^d \bowtie_{C_5} T_3^d$, where

- $C_4 : T_1^d.\mathsf{cid} = \mathsf{cid02} \wedge T_1^d.\mathsf{qty} = T_2^d.\mathsf{qty} \wedge T_2^d.\mathsf{price} = p_0$
- $C_5 : T_2^d.\mathsf{cid} = T_3^d.\mathsf{cid} \wedge T_3^d.\mathsf{prod} = \mathsf{fd}_0$

Here R' renames R; and T_1^d, T_2^d and T_3^d rename T^d. Compared to T^s, T^d is smaller by filtering out users whose account balance is not large enough. Here E is known only at the query execution time, in contrast to R in the static join that is known in advance. □

Extending SQL. We also propose gSQL (graph SQL), SQL with syntactic sugar, to express GRA queries. A gSQL query over database schema $\mathcal{R} = (R_1, \ldots, R_n)$ and graph G is of the form:

select $A_1, \ldots, A_h$
from $R_1, \ldots, R_n, S_1$ join $G_1[\vec{B}_1/\Omega_1], \ldots, S_m$ join $G_m[\vec{B}_m/\Omega_m]$
where CONDITION-1 {and —**or** } ... {and —**or** } CONDITION-P

Here $G_1, \ldots, G_m$ are renamings of the graph G and $S_1, \ldots, S_m$ are either relations in $\mathcal{R}$ or gSQL sub-queries over $\mathcal{R}$ and G. Users may specify expected attributes $\vec{B}_i$ or provide keywords Ω_i for graph G_i just as in GRA described above. Each CONDITION in the **where** clause is an SQL condition over relations $R_1, \ldots, R_n$, and the result relations of semantic joins in the **from** clause. As indicated above, gSQL extends SQL just with syntactic sugar.

Note that gSQL can readily incorporate other SQL keywords, *e.g.,* (inner/outer) joins, not in and views, to express all GRA queries.

Example 5.3: One can write in gSQL the queries q_s and q_d of Example 5.2 almost identically to the standard SQL syntax. For instance, one can write q_d of Example 5.2 in gSQL as follows:

```
with q1 as (
  select R2.cid from R as R1, R as R2
  where R1.cid = cid02 and R1.credit = good and R1.bal ≈ R2.bal )
select T2d.user
from q1 join G[qty, prod, price] as T1d, q1 join G[qty, prod, price] as T2d,
     q1 join G[qty, prod, price] as T3d
where C
```

Here q_1 implements RA expression E of q_d in Example 5.2, and C in the where clause is the same as the join conditions in q_d. Note that writing gSQL queries is almost the same as writing SQL except that we can freely use semantic joins just like natural joins in SQL, which extract attributes from G that are new to the relational database. □

The notations of this chapter are summarized in Table 5.1.

symbols	notations
$\mathcal{D}$, G	relational database, semistructured graph
$R \bowtie_f G$, $Q_1 \bowtie_f G$	static join, dynamic join
$f(S,G)$	HER match relation of schema $R_m(\mathsf{tid},\mathsf{vid})$
$h(S,G)$	relation extracted from G, with schema $R_G(\mathsf{vid},B_1,\ldots,B_m)$
R_τ	schema extracted for τ entities in G
$g_\tau(G)$	instance of schema R_τ for properties of τ entities in G
GRA, gSQL	Graph Relational Algebra, graph SQL (SQL with syntactic sugar)
RGAP, HER, AExt	(Relation GrAPh), Heterogeneous Entity Resolution, Attribute Extraction

Table 5.1: Notations

5.1.2 Implementation

Below we show how existing RDBMSs can be used to implement semantic joins. We first show that static joins can be implemented without any change to RDBMS, by offloading HER and attribute extraction to an offline pre-processing. We then show that a large class of practical queries with dynamic joins can be similarly supported by RBDMS, by reducing to static joins; in Section 5.3 we will show how to process the rest of the queries with dynamic joins.

Note that static joins $R \bowtie_f G$ are a special case of dynamic joins $Q_1 \bowtie_f G$ when Q_1 is a relation atom R in the input schema $\mathcal{R}$. However, the implementations of the two are quite different. Static joins can be rewritten to equivalent SQL queries. In contrast, dynamic joins need to compute matches $f(S,G)$ and extract attributes $h(S,G)$ online, and are thus more expensive than static joins.

Static joins. For a GRA query Q with static joins posed on database $\mathcal{D}$ and graph G, Q can be rewritten into an equivalent conventional SQL query Q' over schemas $\mathcal{R}$, R_m and R_G. The deduced SQL query Q' is then executed by the RDBMS underlying RGAP.

More specifically, it converts each static join $R \bowtie_f G[\vec{B}]$ into a join $R \bowtie_{\mathsf{tid}} \pi_{\mathsf{tid},\vec{B}} R_G$, where R_G is the extracted relation from G for R. As remarked earlier, $\vec{B}$ comes from R_G; matches $f(D,G)$, schema R_G and $h(D,G)$ are independent of the query Q and are computed offline in advance, where D is the instance of R in database $\mathcal{D}$.

Example 5.4: Continuing with Example 5.2, assume that HER function f links R and G in match relation $R_m(\mathsf{cid},\mathsf{vid})$, and the extracted relation is $R_G(\mathsf{vid},\mathsf{qty},\mathsf{prod},\mathsf{price})$. Then one can implement the static join $R \bowtie_f G$ simply as relational joins $R \bowtie_{\mathsf{cid}} R_m \bowtie_{\mathsf{vid}} R_G$. □

Dynamic joins. Dynamic joins cannot materialize matches $f(S,G)$ and extracted relation $h(S,G)$ beforehand. A naive implementation would require to access foreign data (*i.e.,* G) and external functions (*i.e.,* f and h) during the join execution time. In principle, this can be realized by either (a) encapsulating functions HER f and relation extraction h as UDFs within RDBMS, or (b) implementing the logic of dynamic joins outside RDBMS after S is computed in RDBMS. However, in either way it would incur a heavy cost across $\mathcal{D}$ and G.

To reduce the cost, we propose an efficient implementation of dynamic joins for a large class of queries, referred to as *entitic queries*. Intuitively, a query Q is an entitic query if each tuple of its answers can be attributed to one or more entities (*i.e.,* tuples) in the database $\mathcal{D}$. For such queries, we can avoid computing HER matches and extracted relations at execution time, by converting dynamic joins between S and G to static joins between $\mathcal{D}_Q$ and G, where $\mathcal{D}_Q$ consists of tuples in $\mathcal{D}$ that encode exactly those entities to which answers to Q refer. As will be shown in Section 5.4, this approach is an order of magnitude faster than the two naive implementation methods, respectively, for dynamic joins of entitic queries.

More specifically, there are three cases where a query Q is identified as an entitic query: (a) Q refers to only one relation in $\mathcal{D}$ and the output schema R_Q of Q contains its primary key; (b) Q refers to multiple relations in $\mathcal{D}$ of which the primary keys are in R_Q; or (c) there exists an entitic query Q' such that $Q = \pi_{R_Q}(Q')$, where $\pi_{R_Q}(Q')$ projects the results to Q' on the attributes of R_Q.

We start with type (a) queries Q; we compute $Q \bowtie G$ as follows:

(1) Like offline pre-processing in static joins, we find HER matches $f(\mathcal{D},G) = D_m$ and extract relation $h(\mathcal{D},G) = D_G$ in advance.
(2) Given Q, we first compute its answers S in $\mathcal{D}$.
(3) We then compute $Q \bowtie G$ via $S \bowtie D_m \bowtie D_G$, just like how we compute static join $\mathcal{D} \bowtie G$ via $\mathcal{D} \bowtie D_m \bowtie D_G$ above.

Here we assume that the HER match relation D_m has a schema (tid, vid), where tid values are the primary key values in all relations of $\mathcal{D}$. Note that only steps (2) and (3) are carried out at run time after the intermediate results S to Q become available.

Intuitively, $f(S,G) \subseteq f(\mathcal{D},G)$ if query answers in S refer to entities in $\mathcal{D}$. With pre-computed D_m (*i.e.,* $f(\mathcal{D},G)$), this converts the computation of $f(S,G)$ into a join of S with D_m, yielding the match relation between entities in S and G. This can be further joined with pre-computed D_G (*i.e.,* $h(\mathcal{D},G)$). There is no need for invoking

HER and AExt during run time on the fly.

Example 5.5: Recall query q_d in Example 5.2. Observe that sub-query E in the dynamic join $E \bowtie G$ of q_d is an entitic query of type (a) since it contains exactly one primary key of R, *i.e.,* cid. Hence, $E \bowtie G$ can be rewritten into a static join with pre-computed relations R_m and R_G in Example 5.4, via $E \bowtie R_m \bowtie R_G$, avoiding the need to compute HER $f(S,G)$ and extract relation $h(S,G)$ for E online. □

For entitic queries Q of type (b), *i.e.,* when each answer tuple t of Q refers to multiple entities in $\mathcal{D}$ via primary keys, one can still reduce a dynamic join $Q \bowtie G$ to a static join with pre-computed D_m and D_G. The idea is to take into account all entities for the answer S to Q in $\mathcal{D}$ in step (3) above. Assume that S has two primary key attributes, say pk_1 and pk_2. Then we compute $f(S,G)$ via $D_m \bowtie_{\mathsf{pk}_1} S \bowtie_{\mathsf{pk}_2} D_m$. That is, for each answer tuple t in S, we find matches from G to both entities $t[\mathsf{pk}_1]$ and $t[\mathsf{pk}_2]$ in $\mathcal{D}$, extract tuples for them accordingly by further joining D_G, and compose them together by joining t over pk_1 and pk_2.

Example 5.6: Continuing Example 5.1, suppose we want to find and compare investments on product fd_0 from customers with the same account balance. This is formulated as a GRA query q_2 as follows:

$$E = \pi_{\rho:R_1.\mathsf{cid}\to\mathsf{cid}_1,\rho:R_2.\mathsf{cid}\to\mathsf{cid}_2} R_1 \bowtie_{\mathsf{bal}} R_2;$$
$$q_2 = E \bowtie_f G[\mathsf{qty},\mathsf{prod},\mathsf{price}].$$

Note that $E \bowtie_f G$ is an entitic query of type (b) since E contains two keys cid_1 and cid_2. With pre-computed match relation $D_m(\mathsf{cid},\mathsf{vid}) = f(R,G)$ and extracted relation $D_G(\mathsf{vid},\mathsf{qty},\mathsf{prod},\mathsf{price})$, q_2 is evaluated as follows (denote $D_m \bowtie_{\mathsf{vid}} D_G$ by T_G):

$$T_G^1 \bowtie_{\mathsf{cid}_1 = T_G^1.\mathsf{cid}} E \bowtie_{\mathsf{cid}_2 = T_G^2.\mathsf{cid}} T_G^2,$$

where T_G^1 and T_G^2 rename T_G. That is, with pre-computed relations $f(D,G)$ and $h(D,G)$, we can express a dynamic join $E \bowtie_f G$ as two conventional joins on cid_1 and cid_2 of E, respectively. □

Along the same lines, one can turn dynamic joins for queries Q of type (c) into static joins by computing the answers to type (a) or (b) queries Q' first and projecting on R_Q (recall that $Q = \pi_{R_Q}(Q')$).

Most of typical set-valued queries are entitic queries, *e.g.,* all non-aggregate TPC-H benchmark queries. Hence we can answer such queries within RDBMS by reducing dynamic joins to static joins.

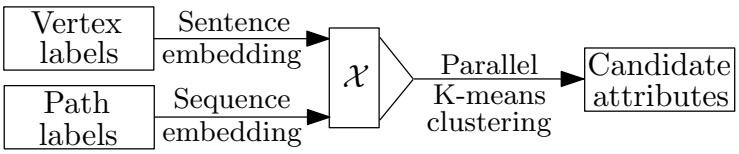


Figure 5.2: Clustering-based attribute extraction

Remark. When the graph G is extracted as relation D_G, existing query processing algorithms based on worst-case optimal (wco) join [TGR20, Ngo18, NRRL19, KEK16] can be directly applied to improve the performance. Slightly different from previous wco join algorithms, *e.g.,* [Vel12], when constructing the trie index structure for $S \bowtie D_m \bowtie D_G$, the algorithm should start from checking keys tid and vid, which reduces the time and space costs of constructing index structures.

5.2 Attribute Extraction

A key step in semantic joins is to extract relations from G. In this section we present AExt of RGAP, a method to deduce extracted schema R_G for a relation schema R from graph G, and populate extracted relation $h(S,G)$ with attributes pivoted at vertices in G, based on matches of HER and query interests (keywords) provided by the user.

Overview. For a relation schema $R(A_1,\ldots,A_{k_R})$, let S be a set of tuples of R (input tuples for static join or query results for dynamic join), and $f(S,G) = \{(t_i,v_i) \mid i \in [1,N]\}$ be the set of matches identified by HER, where t_i is a tuple in S, and v_i is a vertex in G. The users also provide a set $\Omega = \{w_1,w_2,\ldots,w_p\}$ of keywords, where each w_ρ exemplifies part of query results to the user's interests.

Given these, we deduce *extracted schema* $R_G = (\mathsf{id}:\mathsf{vid},\ B_1:\tau_1,\ldots,B_m:\tau_m)$ for R and build an *extracted relation* $h(S,G)$ of R_G, where for $j \in [1,m]$, B_j is an attribute in Υ and τ_j is its type. The attributes of R_G are determined by users' interests Ω.

Intuitively, from the set of vertices that match tuples in S, we extract tuples of schema R_G, where attributes $B_1,\ldots,B_m$ record values reached via paths from v_i for $(t_i,v_i) \in f(S,G)$. Since t_i and v_i denote the same real-life entity e_i, we can find additional information of e_i from G to complement $h(S,G)$. Such information is typically encoded as properties of v_i by vertices close to v_i in G.

AExt has two phases. (1) Extract schema R_G to reflect users' interests. (2) Populate relation $h(S,G)$ with values in graph G. Below we present the two phases. Since we (1) employ sentence embedding and sequence embedding to represent vertex labels and edge labels in a path as vectors, respectively, and (2) apply K-means clustering

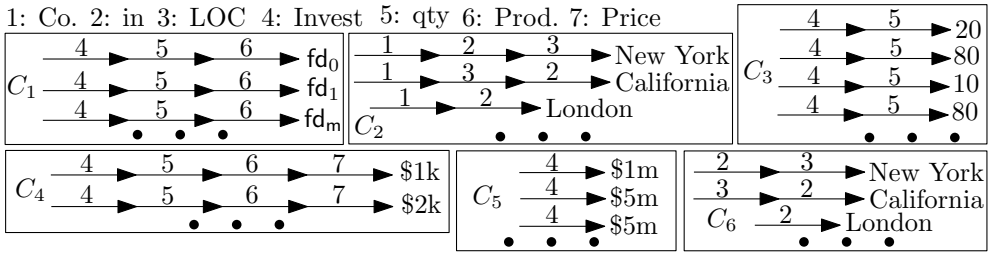


Figure 5.3: Part of clustering result of vertex-path pairs in G.

(KMC) to select attributes, we start with a review of these methods.

Sentence and Sequence Embedding. Sentence embedding aims at learning semantically meaningful vector representations of sentences, such that the semantic distance between sentences can be quantified by similarity metrics [LFS+17]. With the recent success of attention-based models [VSP+17], one can choose from pre-trained models [RG19] to obtain high-quality sentence embeddings without training a model starting from scratch. Sequence embedding deals with the problem of representing token sequences as vectors, such that the learned representations capture the sequential information of tokens in the sequence. Since long short-term memory (LSTM) networks have shown promising results in sequence modeling [GBC16, CGCB14], people commonly adopt the vector output of the LSTM network in the last time step as the embedding to represent the sequence [ETJ+18].

K-means Clustering. We adopt KMC for attribute extraction since it can be efficiently parallelized [LF89]. Popular for cluster analysis in data mining, KMC [M+67] aims to partition data points into clusters that minimize squared Euclidean distances within each cluster, so that data points in the same cluster are much closer than those in different ones. This problem is NP-hard [MNV09]. Nonetheless, heuristic algorithms have been developed, which converge quickly at a local optimum; the algorithms partition data points by iteratively assigning each to its closest cluster, where the center of each cluster is updated when new data points are added. We should remark that other clustering methods can also be used by AExt, not limited to KMC.

(1) Schema construction. For a given relation schema R, we build its extracted schema R_G from graph G in two steps:

(a) extract a set Γ of attributes based on matches $f(S,G)$; and

(b) design a ranking function that picks the best m attributes from Γ for schema R_G, based on users' interests Ω.

Step (a) Attribute extraction. As shown in Fig 5.2, we identify candidate attributes via KMC on the vector representations of both vertex labels and edge labels on the paths.

We assume three parameters: a bound k on linking path lengths, the number H of clusters, and the number m of attributes for R_G, which we will elaborate shortly.

More specifically, given a match $(t_i, v_i) \in f(S,G)$, for each vertex v_{ij} to which v_i is linked via the shortest path ρ_{ij} ($\mathsf{len}(\rho_{ij}) \leq k$), we take (v_{ij}, ρ_{ij}) as a vertex-path pair of v_i and include all such pairs in a set Ψ. Only the shortest paths are considered since a shorter path tends to represent stronger relatedness in semantic graphs [KLAF17]. We employ a Bert embedding model [RG19] to extract a vector representation $x_{v_{ij}}$ of vertex label for each v_{ij}. Since most vertex labels are "sentences" in natural languages, we can utilize a pre-trained Bert model instead of training a new one starting from scratch.

We then build a vector representation $x_{\rho_{ij}}$ for each path ρ_{ij} via a sequence embedding model, *e.g.,* LSTM networks, where ρ_{ij} is viewed as a sequence of edge labels. More specifically, we feed edge labels on path ρ_{ij} in sequence to the well-trained LSTM network as inputs and, similarly to [ETJ+18], and take the network output in the last time step as $x_{\rho_{ij}}$. Due to the sequence modeling capacity of LSTM, the embedding $x_{\rho_{ij}}$ can discern different orders of edge labels. For example, the path from Ada to New York in Fig 5.1 has edge label sequence: (Co., in, LOC), while that from Bob to California has the sequence: (Co., LOC, in). Both paths have same edge labels but in different orders, which are embedded as two different vectors. This will benefit the downstream clustering task by discriminating different orders of edge labels. To train such LSTM networks, we perform random walk in G, collect edge labels on walk paths as training corpus where each label is viewed as a word, and train the LSTM network on this corpus driven by perplexity [SSN12]. Concatenating $x_{v_{ij}}$ and $x_{\rho_{ij}}$ as x_{ij}, we represent each vertex-path pair (v_{ij}, ρ_{ij}) by one feature vector. We include all x_{ij}'s in a set $\mathcal{X}$.

We perform KMC on the set $\mathcal{X}$ with limited iterations to assign each vertex-path pair into one of H clusters. Each cluster C_ε represents a candidate attribute B_ε whose type τ_ε is the type of the majority vertex label in C_ε. Denote by $\Gamma = \{B_1, \ldots, B_H\}$ the set of all such candidate attributes. Note that the training and clustering are both unsupervised, saving the costly manual annotations.

Step (b) Attribute ranking. We rank attributes B_ε in Γ by

$$r_A(B_\varepsilon) = \frac{|C_\varepsilon|}{|\Psi|} - \max_{\varphi \in [1,k_R]} \frac{\sum_{v_{ij} \in C'_\varepsilon} \cos(x_{v_{ij}}, x_{t_i.B_\varphi})}{|C'_\varepsilon|} + \max_{\rho \in [1,p]} \frac{\sum_{v_{ij} \in C'_\varepsilon} \cos(x_{v_{ij}}, x_{w_\rho})}{|C'_\varepsilon|}.$$

Here C_ε is the cluster that represents B_ε, Ψ is the set of all vertex-path pairs, C'_ε is a randomly selected subset of C_ε as vertex-path pair representatives for ranking, $\cos(\cdot,\cdot)$

vid	qty	price
id001	20	$1k
id001	80	$1k
id002	10	$1k
id003	80	$1k
id003	10	$2k

Table 5.2: Extracted relation R_G

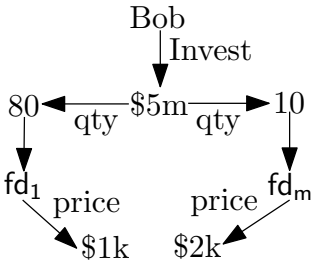


Figure 5.4: Directed out tree

returns the cosine similarity of two vectors, φ iterates over attributes of tuple t_i, and $x_{t_i.B_\varphi}$ (resp. x_{w_ρ}) is the sentence embedding of attribute $t_i.B_\varphi$ (resp. string $w_\rho \in \Omega$) in the same way as $x_{v_{ij}}$, k_R is the arity of relation S, p is the number of query-interest keywords. We pick m attributes with the highest scores from Γ to build the extracted schema R_G of R.

Intuitively, this function gives higher scores to candidate attributes that (1) match more vertex-path pairs in G (the first term), (2) are not semantically similar to existing attributes in R (the second term), and (3) are semantically related to one of users query interests in Ω (the third term). Thus, the extracted schema R_G is to reduce the number of null values, provide versatile information that is complementary to tuples of schema R, and meet the user's interests.

Example 5.7: Given matches $\{(\text{cid01}, \text{Ada\_id002}), (\text{cid02}, \text{Bob\_id003}), (\text{cid04}, \text{Ada\_id001})\}$ in Example 5.2, using well-trained LSTM networks [SSN12] and word embedding models [RG19], we embed vertex-path pairs as vectors and cluster them as shown in Fig. 5.3, where cluster C_ε corresponds to candidate attribute B_ε. For an interest set $\Omega = \{\text{quantity number 80, price \$2k}\}$ given by the user, we rank the candidates in decreasing order of the scores returned by r_A: B_3, B_4, B_5, B_1, B_2. Here B_3 and B_4 have the highest scores because they are complementary to the customer database $\mathcal{D}$ and are closely related to query interests. We rank $r_A(B_3)$ higher than $r_A(B_4)$ because B_3 has more matches in G. Note that cluster C_6 is not involved in this ranking since it consists of vertex-path pairs that belong to companies and there is no vertex-tuple matches in $f(S,G)$ provided for company entities. We keep B_3 and B_4 in the extracted schema since they retrieve the missing information of qty and price in $\mathcal{D}$. □

Parameters H, m, k. The default H and m ($m \leq H$) are set by the users, or H can take the number of "types" in Ψ as their value; paths with the same edge label sequence are considered as the same type. We set k as 3 in default since most information of an entity is within 3-hops [LLL+15] and longer paths hold weaker associations [AHAS03,

KLAF17].

(2) Value extraction. After schema R_G is extracted, we populate its instance $h(S,G)$ as follows. Consider a match $(t_i, v_i) \in f(S,G)$. For vertex-path pair (v_{ij}, ρ_{ij}) that is assigned to a cluster C_ε, which corresponds to attribute B_ε in R_G, normally we assign the label $L(v_{ij})$ to $t_{v_i}.B_\varepsilon$. However, conflicts occur when multiple vertex-path pairs of one entity are mapped to the same attribute B_ε. We refer to vertex-path pairs as *conflicting pairs* if they contain conflicting paths, attributes to which conflicting pairs are mapped as *conflicting attributes*, and other attributes in R_G without such conflicts as *non-conflicting attributes*. For instance, for the entity Bob_id003 in Fig. 5.1, the vertex-path pair from Bob_id003 to price:$1k and that from Bob_id003 to price:$2k are both assigned to cluster C_4 in Fig. 5.3, which maps price:$1k and price:$2k to attribute B_4 (price). The two vertex-path pairs are conflicting pairs containing two conflicting paths, and attribute B_4 (price) is the conflicting attribute.

We resolve this by extracting conflicting values of one entity to multiple tuples as follows. For all conflicting pairs (v_{ij}, ρ_{ij}) ($j \in [1,N]$) of the entity v_i, we first extract vertices and edges on paths ρ_{ij} ($j \in [1,N]$) from G and obtain a DAG (directed acyclic graph) $G_{v_i} = (V_{v_i}, E_{v_i})$ with v_i as root. Since each path ρ_{ij} is the shortest path from v_i to v_{ij}, the DAG G_{v_i} is actually a directed out tree, where v_i is the root and for any other vertex u in G_{v_i}, there exists exactly one (shortest) path from the root v_i to u [SP75]. Then for each leaf v_l in G_{v_i} with a path ρ_l from root v_i to v_l, we create a tuple $t_{v_i l}$ in which the values of conflicting attributes are extracted from vertex labels on ρ_l according to the attributes in R_G, while the values of non-conflicting attributes in $t_{v_i l}$ are populated normally. Each tuple $t_{v_i l}$ shares the same id : vid since they belong to the same entity v_i. This conflict resolution takes $O(V_{v_i})$ time as the algorithm just needs to check each vertex label in G_{v_i} following the directed paths from the root to each leaf.

Example 5.8: Continuing with Example 5.7, Table 5.2 shows the relation $h(D,G)$ extracted from G. The tuple of Ada_id002 is populated without conflicts, while conflicts occur for Ada_id001 and Bob_id003 as different vertex-path pairs of quantity and price are assigned to the same cluster (attribute). In order to resolve these conflicts, taking Bob_id003 as example, we construct a directed out tree (Fig. 5.4) by extracting vertices and edges on conflicting paths from G. Then for each leaf in Fig. 5.4, we create a tuple to collect values for attributes in R_G. For instance, along the path from Bob to $1k in Fig. 5.4, we create a tuple that collects qty as 80 and price as $1k (line 4 in Table 5.2). Similarly, for path from Bob to $2k, we create another tuple (line 5 in Table 5.2) to

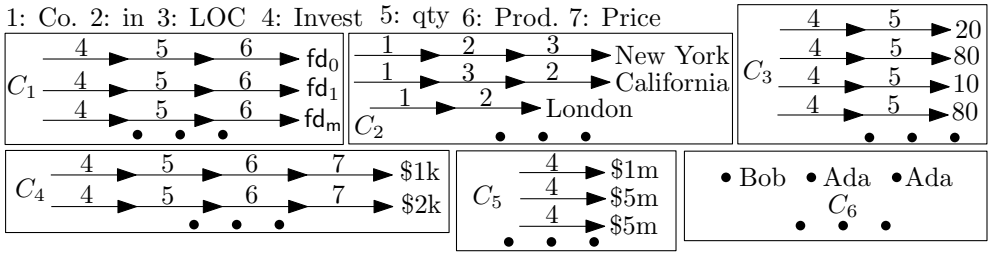


Figure 5.5: Clustering result (vertex-path pairs) for customers

vid	name	loc	invest	prod	qty	price
id001	Ada	New York	$1m	fd_0	20	$1k
id001	Ada	New York	$1m	fd_1	80	$1k
id002	Ada	London	$5m	fd_1	10	$1k
id003	Bob	California	$5m	fd_1	80	$1k
id003	Bob	California	$5m	fd_m	10	$2k

Table 5.3: Relation $g_{\mathsf{cust}}(G)$ extracted from G

hold these values, which resolves the conflicts. □

Cost of AExt. Provided with the schema R of S and the HER match relation $f(S,G)$, AExt efficiently extracts attributes. First, model training can be performed *offline*. Second, the attribute extraction takes polynomial time for ML embedding extraction, $O(V + E\log V)$ for shortest path finding, and $O(|\mathcal{X}|H)$ for KMC with fixed vector length and iterations. The attribute ranking takes $O(|C'_\varepsilon|)$ time, where the size of C'_ε is small and can be adjusted by the user. In addition, resolving conflicts for each entity during value extraction takes $O(|V_{v_i}|)$ time, and G_{v_i} is usually a small localized tree.

5.3 Heuristic Dynamic Join

In this section, we propose another approach to implementing dynamic joins for generic gSQL queries that are not entitic (recall Section 5.1.2). We first develop a method to extract relations from a graph for entities of certain types (Section 5.3.1). We then show how to support dynamic joins by using the extracted relations without executing external HER and AExt functions at run time (Section 5.3.2). Moreover, we show how to incrementally maintain the extracted relations in response to updates to $\mathcal{D}$ and G (Section 5.3.3).

5.3.1 Schema Extraction

Given a graph G, a type τ of entities of users' choice in G, and a set Ω of keywords of users' interest, we deduce a relation schema R_τ and an instance $g_\tau(G)$ of R_τ from G such that R_τ includes attributes that represent important properties of τ-entities in G and match users' interests. The extraction of R_τ and $g_\tau(G)$ basically follows the workflow of deducing R_G and $h(S,G)$ in Section 5.2 with the following two differences. First, instead of clustering all vertex-path pairs in G, the KMC is applied to those that belong to entities of type τ. This filters out properties that are irrelevant to τ-entities. Second, since no matches $f(S,G)$ and schema R are available, the attribute ranking function for selecting attributes in R_τ becomes

$$r'_A(B_\varepsilon) = \frac{|C_\varepsilon|}{|\Psi|} + \max_{\rho\in[1,p]} \frac{\sum_{v_{ij}\in C'_\varepsilon} \cos(x_{v_{ij}}, x_{w_\rho})}{|C'_\varepsilon|}.$$

This favors attributes that represent more vertex-path pairs in G and are more semantically related to users' interests. Extracting relation $g_\tau(G)$ of R_τ from G is efficient since (1) only vertices of entity type τ are considered, which reduces the inputs for KMC, and (2) the attribute ranking function becomes lighter with less terms.

Example 5.9: Continuing with Example 5.1, consider type customer of entities in the graph of Fig. 5.1. To extract a relation for customer, we first embed the vertex-path pairs of all customers as vectors and cluster them as shown in Fig. 5.5. Different from Fig. 5.3, no vertex-path pairs of companies are included. For an interest set $\Omega = \{$name Bob, place Los Angeles, invest \$5m, product fd_m, quantity number 80, price \$2k$\}$ provided by the user, we rank the candidates in decreasing order of the scores returned by r'_A: B_3(qty), B_4(price), B_1(prod), B_5(invest), B_6(name), B_2(loc); we preserve these six attributes in the extracted schema R_{cust}. Then for each customer, we follow the clustering result to populate the corresponding tuples, and obtain the relation $g_{\mathsf{cust}}(G)$ (shown in Table 5.3) of R_τ, where value conflicts are also resolved by populating multiple tuples as in Section 5.2. □

Schema R_τ's are extracted offline when graph G is available. As remarked in Section 5.1.1, users can reference the extracted schema to pick attributes $\vec{B}$ when writing dynamic join queries, since one often knows what types of entities they want to query.

Below we use the extracted $g_\tau(G)$ to implement heuristic (dynamic) joins. Note that the schema extraction method can also be used in data extraction from graphs for, *e.g.,* data lakes [NZM+19].

5.3.2 From Dynamic Joins to Heuristic Joins

Using the relations $g_\tau(G)$ extracted from G, we develop heuristic joins to implement dynamic joins $Q_1 \bowtie_f G$ for generic queries Q_1, especially those that cannot be converted to static joins (Section 5.1.2).

Heuristic join computes $Q_1 \bowtie_f G$ by means of only traditional ER on intermediate results $Q_1(\mathcal{D})$ and the pre-extracted relations $g_\tau(G)$ during the query execution time; it does not need to execute external HER and AExt functions at run time. More specifically, to compute a dynamic join $Q_1 \bowtie_f G$, it first computes $S = Q_1(\mathcal{D})$ via RDBMS. It then decides which types of entities are queried by Q_1, *i.e.,* which types τ of entities that tuples in S refer to; it "joins" S with relations $g_\tau(G)$ that are extracted for entities queried by Q_1 via traditional ER, and composes matching tuples as the results of $Q_1 \bowtie_f G$.

In this way, we can implement heuristic joins within RDBMS via stored procedures and UDF that (a) select extracted relations $g_\tau(G)$ that are relevant to Q, (b) link tuples from S of Q and the selected relations $g_\tau(G)$, and (c) join the linked tuples and compose the result.

More specifically, for step (a), we first compute matches between attributes of R_{Q_1} for Q_1 and those of the extracted schemas via schema-level schema matching (cf. [BBC$^+$00, RB01]). We mark a relation $g_\tau(G)$ as *relevant* to Q_1 if (1) there exists a primary key in R_{Q_1} that matches attributes of $g_\tau(G)$, or (2) the number of attributes of R_{Q_1} (resp. $g_\tau(G)$) that match attributes of $g_\tau(G)$ (resp. R_{Q_1}) is above a pre-defined threshold η (resp. η'). We approximate $Q_1 \bowtie_f G$ by joining Q with only those $g_\tau(G)$ that are marked relevant to Q_1.

For step (b), one can use either (i) pairwise tuple comparison based ER method or (ii) end-to-end ER that takes entire S and $g_\tau(G)$ as input and computes the match relation all at once. Here (i) can be implemented as a simple UDF as the join condition between S and $g_\tau(G)$ to check whether $t \in S$ and $t' \in g_\tau(G)$ make a match via the pairwise matching operation in almost all ER methods (cf. [PSTP20]). The implementation of (ii) requires to take S and $g_\tau(G)$ as input and encapsulates a complete ER solution, *e.g.,* JedAI [PTT$^+$18], in the UDF.

Example 5.10: Recall the dynamic join $E \bowtie_f G$ given in Example 5.2. Using heuristic join, it is answered as follows.

(1) As an offline process, we extract relations from G, for types of entities of users' interests from G. Denote by $g_{\mathsf{cust}}(G)$ the relation extracted for customers in G. By

the method of Section 5.3.1, $g_{\mathsf{cust}}(G)$ is a relation with schema $(\mathsf{vid}, \mathsf{name}, \mathsf{loc}, \mathsf{invest}, \mathsf{prod}, \mathsf{qty}, \mathsf{price})$.

(2) During query execution, $E \bowtie_f G$ is converted into an ER-based join via stored procedure and UDF. The stored procedure consists of three steps: (a) selecting $g_{\mathsf{cust}}(G)$ for E by matching the output schema of E and $g_{\mathsf{cust}}(G)$; (b) invoking ER to link E and $g_{\mathsf{cust}}(G)$ via UDF; and (c) composing and returning the joined result.

In particular, when using a pairwise ER matching operation as the UDF, steps (b) and (c) can be written as a single SQL query, where $E \bowtie_f G$ is approximated by a join between E and $g_{\mathsf{cust}}(G)$ in which the join condition checks whether the cid of E matches customers extracted in $g_{\mathsf{cust}}(G)$ via a pairwise ER operation. □

Observe the following. (1) Compared to the brute-force implementation of dynamic joins, heuristic joins do not compute HER matches or extract relations at run time. The only overhead added to query execution is the UDF-based join to implement relational ER. (2) Different from the entitic-query approach (Section 5.1.2), heuristic joins do not assume the availability of HER matches. (3) Heuristic joins work for generic queries for which it is hard to map S of $S \bowtie_f G$ to entities in $\mathcal{D}$ and hence cannot be reduced to static joins. (4) Query execution is entirely done within RDBMS. This allows us to freely use semantic joins as sub-queries in nested SQL queries.

5.3.3 Incremental Maintenance

We next study incremental maintenance of semantic joins in response to updates ΔG to graph G and $\Delta\mathcal{D}$ to database $\mathcal{D}$.

Maintaining the results for semantic joins involves three parts: (a) updating HER match relations, (b) updating the extracted relations from G, and (c) updating join results. For (a), if the HER function f does not have built-in support of incremental HER, we compute the changes to the match relation by taking the union of $f(\Delta D, G)$ and $f(D, \Delta G)$ as a fallback. For (c), after the match relation and extracted relations are updated, the semantic join results can be incrementally updated via incremental relational join processing, which has been well studied in the context of incremental view maintenance [SBCL00, KAK+14], and is supported by major systems, *e.g.,* PostgreSQL [inc].

Below we focus on (b), *i.e.,* the incremental maintenance of relation extraction from G. The techniques work on both relation $f(S, G)$ computed by AExt (Section 5.2) for

static and dynamic joins, and relation $g_\tau(G)$ extracted in Section 5.3.1 for heuristic joins.

Incremental relation extraction. In practice, graph updates ΔG are often much smaller than G. Given ΔG, we show how to incrementally compute $h(S,G)$; similarly for $g_\tau(G)$. For removed vertices or changed vertex labels in ΔG, AExt just needs to delete or change their corresponding values in $h(S,G)$, following the existing one-to-one mapping between vertex-path pairs in G and values in the extracted relation. For newly added entities and vertices in ΔG, the incremental computation is conducted in the following three steps.

(1) AExt applies incremental KMC [CN11] to efficiently extract their values. More specifically, it first employs pre-trained models to extract the embeddings of the newly added vertex-path pairs. These models require no extra training since the additions are either (a) vertex labels that can be embedded by the Bert embedding model pre-trained on natural languages, or (b) sequences of existing edge labels that can be modeled by the well-trained LSTM networks, which have been previously trained in attribute extraction.

(2) Taking these newly extracted embedding vectors and the clustering result in the previous relation extraction as inputs, AExt applies incremental KMC to classify each vector to the closest existing cluster, and update the cluster center [CN11]. That is, for each newly added vector, incremental KMC computes the distance between this vector and all existing cluster centers (the center of a cluster is the average of all vectors in that cluster), adds the vector to the cluster with the shortest distance, and updates the center of that cluster. Since each cluster represents an attribute and each embedding vector represents a vertex-path pair, the incremental KMC assigns the vertex-path pairs to existing attributes of R_G.

(3) Finally, conflicting vertex-path pairs are assigned to multiple tuples by constructing a directed out tree as in Section 5.2.

The process is efficient since (1) no model requires extra training; (2) the incremental KMC [CN11] only computes the distance between existing clustering centers and the newly added embeddings, which avoids clustering all data from scratch; and (3) the cost of attribute ranking functions is omitted as the top attributes in R_G (resp. R_τ) have already been selected before the incremental maintenance.

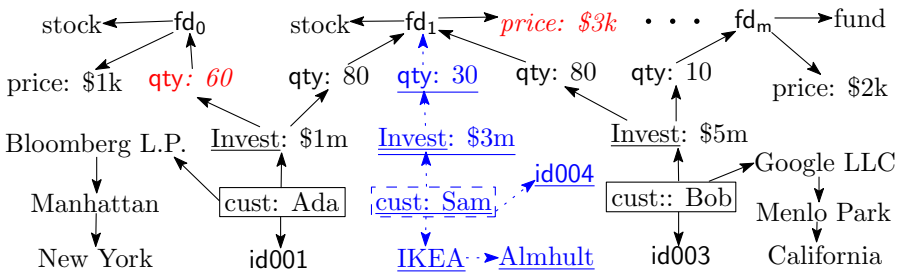


Figure 5.6: Incremental changes ΔG to graph G in Example 5.1

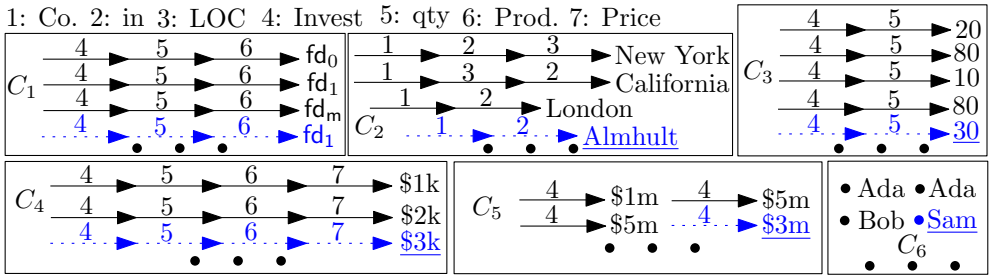


Figure 5.7: Incremental KMC result with new customer Sam

Example 5.11: Continuing with Example 5.1, Figure 5.6 shows updates ΔG to the graph G of Example 5.1, in which Ada_id001 increases the quantity of product fd_0 from 20 to 60, customer Ada_id002 is removed while a new customer Sam_id004 is added, and the price of the product fd_1 rises from \$1k to \$3k.

For removed vertices or changed labels in ΔG, AExt can directly delete or change corresponding values in Table 5.3, by the one-to-one mapping between vertex-path pairs and attributes of each tuple. Thus, AExt deletes the tuple of Ada_id002, changes the quantity of fd_0 to 60, and raises the price of fd_1 to \$3k. For the newly added customer Sam, AExt extracts vector embeddings of each vertex-path pair pertaining to Sam, applies incremental KMC, and assigns each vector to one of the previously obtained clusters as shown in Fig. 5.7. The incremental KMC is efficient as it only compares the distance between each new vector and the existing six cluster centers in Fig. 5.5, and assigns the vector to the nearest one. After this step, AExt populates the new tuples of Sam following the vertex-path pair mapping in Fig. 5.7, and gets the new

vid	name	loc	invest	prod	qty	price
id001	Ada	New York	\$1m	fd_0	*60*	\$1k
id001	Ada	New York	\$1m	fd_1	80	*\$3k*
id003	Bob	California	\$5m	fd_1	80	*\$3k*
id003	Bob	California	\$5m	fd_m	10	\$2k
id004	Sam	Almhult	\$3m	fd_1	30	\$3k

Table 5.4: Incremental relation extraction result for $g_{\mathsf{cust}}(G)$

relation shown in Table 5.4. □

5.4 Experimental Study

Using real-life and synthetic data, we conducted four sets of experiments to evaluate (1) the need for semantic joins when querying relations and graphs taken together, (2) the accuracy of semantic join results, (3) the efficiency and scalability of RGAP, and (4) the effectiveness of incremental maintenance of semantic joins.

Experimental setting. We start with the experimental settings.

Datasets. We used 6 datasets that are grouped into 3 collections as shown in Table 5.7, each is a pair of a graph and a relational dataset obtained from independent data sources. (1) Paper collection records publications and authors, taken from relations in DBLP [DBL21c] and graph of RKBExplorer [DBL21b, GMJ08]. (2) Movie collects movies, directors and actors, etc., in relations from IMDB [IMD21] and graph from LinkedMDB [HC09]. (3) Celebrity contains information about athletes and politicians in relations from DBpedia [dbp, dbp21b, dbp21c] and graph from YAGO3 [yag]. We linked entities across graphs of RKBExplorer, LinkedMDB and YAGO3 and relations of DBLP, IMDB and DBpedia, respectively, to evaluate the quality of semantic joins.

Queries. We designed 20 queries across relations and graphs, as close to the queries of our FinTech collaborator as possible using public data. Table 5.5 and Table 5.6 list all 20 queries used in the experimental study, including the dataset collections against which they query, the description of the queries in plain text, and the gSQL expressions of the queries. All queries require data in both relations and graphs.

RGAP. We have developed a prototype of RGAP. It is intentionally implemented as a lightweight solution to support semantic joins between relations and graphs. For the experiments, we used PostgreSQL as the underlying RDBMS, but we remark that the design of RGAP is database agnostic. RGAP has four components: (a) HER [PMG+20, FGJ+22] for heterogeneous entity resolution across relations and graphs, (b) AExt for extracting relations from graphs based on the method in Section 5.2 and its variant in Section 5.3.1, (c) SJ for extending RDBMS with static join, dynamic join and heuristic join, and (d) INC for incrementally maintaining matches and extracted relations with HER and AExt (and semantic join results when specified) in response to updates to

Dataset	query description	gSQL
Celebrity	q_1: find US republican and democratic politicians who were born in the same city	create view politician_aug as select name, party, birthPlace from politician join G_{YAGO3}[party, birthPlace] where party = 'republican' or 'democratic' select T_1.name, T_2.name from politician_aug as T_1, politician_aug as T_2 where T_1.birthPlace = T_2.birthPlace and T_1.party <> T_2.party
Celebrity	q_2: find male goalkeepers who are shorter than 1.8m and were born after 1970	select name, from athletes join G_{YAGO3}[birthDate] where athletes.height <= 1.8 and birthDate >= 1970
Paper	q_3: find papers of the same volume in journal "Entropy" published in 2012	create view entropy12 as select DBLP.id, vol from DBLP join $G_{\text{RKBExplorer}}$[vol] where DBLP.year = 2012 and DBLP.journal = 'Entropy' select T_1.id, T_2.id from entropy12 as T_1, entropy12 as T_2 where T_1.vol = T_2.vol
Paper	q_4: find thesis with the same affiliation from 2005 to 2010	create view aDBLP as select DBLP.id, affiliation from DBLP join $G_{\text{RKBExplorer}}$[affiliation] where DBLP.year >= 2005 and DBLP.year <= 2010 select T_1.id, T_2.id from aDBLP as T_1, aDBLP as T_2 where T_1.affiliation = T_2.affiliation
Movie	q_5: find IMDB works with at least 50 writers and 50 directors that have at least one director also serving as writer	select IMDBmovie.id from IMDBmovie join $G_{\text{LinkedMDB}}$[writers] where size_of(IMDBmovie.directors) >= 50 and size_of(writers >= 50 and size_of(intsec(IMDBmovie.directors, writers) >0 /* size_of, intsec are UDF functions*/
Movie	q_6: find actors who were born after 1980 and are also directors	select IMDBperson.name from IMDBperson join $G_{\text{LinkedMDB}}$[primaryProfession] where 'actor' in primaryprofession and 'director' in primaryprofession and IMDBperson.birthYear >= 1980
Movie	q_7: find IMDB works that have at least 50 writers and 50 directors, and moreover, no director also serves as a writer.	select IMDBmovie.id from IMDBmovie join $G_{\text{LinkedMDB}}$[writers] where size_of(IMDBmovie.directors) >= 50 and size_of(writers >= 50 and size_of(intsec(IMDBmovie.directors, writers) =0
Movie	q_8: find IMDB works have no more than 5 writers and no fewer than 50 directors, and moreover, all writers also serve as directors	select IMDBmovie.id from IMDBmovie join $G_{\text{LinkedMDB}}$[writers] where size_of(IMDBmovie.directors) >= 50 and size_of(writers <= 5 and contained_in(writers, IMDBmovie.directors) =true
Movie	q_9: find writers who were born before 1950 and are also producers	select IMDBperson.name from IMDBperson join $G_{\text{LinkedMDB}}$[primaryProfession] where 'writer" in primaryprofession and 'producer' in primaryprofession and IMDBperson.birthYear <= 1950
Movie	q_{10}: find writers who are also directors and died before 30	select IMDBperson.name from IMDBperson join $G_{\text{LinkedMDB}}$[primaryProfession] where 'writer" in primaryprofession and 'director' in primaryprofession and IMDBperson.deathYear − IMDBperson.birthYear <= 30

Table 5.5: Full list of queries and their gSQL expressions

Dataset	query description	gSQL
Paper	q_{11}: find papers from the same volume in journal "Pattern Recognition Letters" published in 2004	create view PRL04 as select DBLP.id, vol from DBLP join $G_{\text{RKBExplorer}}$[vol] where DBLP.year = 2004 and DBLP.journal = 'Pattern Recognition Letters' select T_1.id, T_2.id from PRL04 as T_1, PRL04 as T_2 where T_1.vol = T_2.vol
Paper	q_{12}: find papers from the same volume published in 2004 of length no more than 3 pages	create view DBLPaug as select DBLP.id, DBLP.pages, vol from DBLP join $G_{\text{RKBExplorer}}$[vol] where DBLP.year = 2004 and DBLP.pages >= 3 select T_1.id, T_2.id from DBLPaug as T_1, DBLPaug as T_2 where T_1.vol = T_2.vol
Paper	q_{13}: find pairs of PhD theses from the same university that were published between year 2000-2004 and year 2005-2009, respectively	create view DBLPphd as select DBLP.id, DBLP.mdate, affiliation from DBLP join $G_{\text{RKBExplorer}}$[affiliation] where DBLP.type = 'PhD thesis' select T_1.id, T_2.id from DBLPphd as T_1, DBLPphd as T_2 where T_1.affiliation = T_2.affiliation and T_1.mdate >= 2000 and T_1.mdate <= 2004 and T_2.mdate >= 2005 and T_2.mdate <= 2009
Paper	q_{14}: find PhD theses from the same university that were published no earlier than 1995 and have note type of both dnb and urn	create view DBLPphd as select DBLP.id, affiliation from DBLP join $G_{\text{RKBExplorer}}$[affiliation] where DBLP.type = 'PhD thesis' and DBLP.mdate >= 1995 and DBLP.note − type <> dnb and DBLP.note − type <> urn select T_1.id, T_2.id from DBLPphd as T_1, DBLPphd as T_2 where T_1.affiliation = T_2.affiliation
Celebrity	q_{15}: find pairs of UK politicians born in the same place but from different political parties	create view politician_aug as select name, party, birthPlace from politician join G_{YAGO3}[party, birthPlace] where country = 'UK' select T_1.name, T_2.name from politician_aug as T_1, politician_aug as T_2 where T_1.birthPlace = T_2.birthPlace and T_1.party <> T_2.party
Celebrity	q_{16}: find pairs of French politicians from the same party but with different birthplaces	create view politician_aug as select name, party, birthPlace from politician join G_{YAGO3}[party, birthPlace] where country = 'France' select T_1.name, T_2.name from politician_aug as T_1, politician_aug as T_2 where T_1.birthPlace <> T_2.birthPlace and T_1.party = T_2.party
Celebrity	q_{17}: find US politicians who died at ages no smaller than 50	select name from politician join G_{YAGO3}[die_date] where country = 'US' and die_date − birthDate >= 50
Celebrity	q_{18}: find UK politicians who died in London	select name from politician join G_{YAGO3}[die_in] where country = 'UK' and die_in = 'London'
Celebrity	q_{19}: find National Hockey League athletes who died before 2000	select name from athletes join G_{YAGO3}[die_date] where athletes.league = 'National Hockey League' and die_date < 2000
Celebrity	q_{20}: find England Athletes who died in England	select name from athletes join G_{YAGO3}[die_in] where athletes.country_label = 'England' and die_in = 'England'

Table 5.6: Full list of queries and their gSQL expressions (cont.)

Dataset coll.	Relations	Graphs
Paper	DBLP: 4.4M tuples	RKBExplorer: 15.9M vertices, 31.1M edges
Movie	IMDB: 39.2M tuples	LinkedMDB: 2.3M vertices, 5.4M edges
Celebrity	DBpedia: 372K tuples	YAGO3: 3.4M vertices, 10.2M edges

Table 5.7: Dataset collections

both relations and graphs based on Section 5.3.3.

In particular, (1) for heuristic joins, we employed JedAI [PMG+20] for end-to-end entity resolution (Section 5.3.2). (2) For the AExt module of RGAP, we parallelized KMC [par] and attribute ranking. We employed pre-trained BERT embedding model of [RG19] and the LSTM networks [MKS18] for vectorizing vertex and path labels, respectively. The LSTM model is trained with default configurations in [MKS18] on the graph G of each dataset. We also performed L2 normalization before vector concatenation; each vertex-path pair was represented by a 1000-dimension vector. We randomly selected 1000 vertex-path pairs from each C_ε to build C'_ε (see Section 5.2 for C_ε and C'_ε), and included four keywords in query interest set Ω. (3) In addition to static join and heuristic join, the SJ module supports three implementations of dynamic joins: (i) Naive-I that encapsulates HER and AExt as UDFs within PostgreSQL; (ii) Naive-II that implements dynamic join outside PostgreSQL, by invoking HER and AExt externally; and (iii) entitic join that rewrites dynamic joins into static ones if possible (Section 5.1.2). Here HER is implemented by JedAI [PMG+20]. By default, SJ uses entitic dynamic joins if possible, and fallbacks to heuristic joins otherwise; Naive-I and Naive-II are only used for performance comparison.

Baselines. We compared RGAP with three baselines.

(1) BigDAWG [DES+15] is a polystore middleware for querying heterogeneous data sources that are organized in *islands*, where each island is essentially a full-fledged database engine itself. We tested the latest BigDAWG v0.1 [big], where an island can be PostgreSQL (relation), SciDB (array), or Accumulo (key-value) database engine. Since BigDAWG V0.1 does not support native graph engines, we represented graphs as vertex and edge relations $R_{\mathsf{v}}(\mathsf{vid}, \mathsf{label})$ and $R_{\mathsf{e}}(\mathsf{vid}_1, \mathsf{label}, \mathsf{vid}_2)$, and stored them in a PostgreSQL island.

(2) Myria [WBB+17] is a federated system service that can be deployed on cloud services like Amazon EC2. It is compatible with multiple relational backend database engines with a unified datalog-flavor language interface called MyriaL. It does not support native graph storage. Thus we also stored graphs as vertex and edge relations.

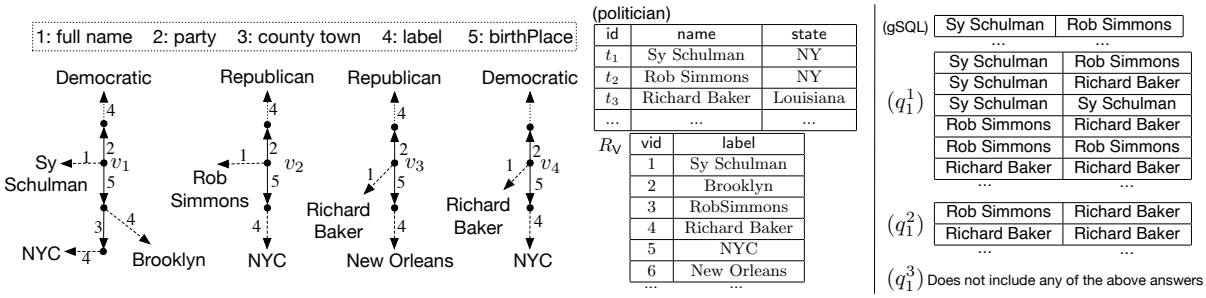


Figure 5.8: Case study (Exp-1): gSQL returns correct answers while all approximations return incorrect answers and miss correct ones. (a) Query q_1^1 get many false positives. (b) None of the two shown answers to q_1^2 is correct: it cannot extract 'NYC' for Sy Schulman (hence misses the gSQL answer); it also mistakenly matches t_3 to v_4. (c) Query q_1^3 correctly identifies that t_3 does not match v_4; however, similar to q_1^2 it cannot extract 'NYC' as the birthPlace of t_1.

(3) RGAP$^-$ is a lite version of RGAP that keeps the underlying PostgreSQL and the HER module. To favor its expressiveness, we retained HER by pre-computing the HER match relation in the database so that it was equipped with HER in the same way as RGAP. RGAP$^-$ differs from RGAP only in that it does not have the AExt module, *i.e.,* attributes are extracted from graphs by users. AExt is a unique feature of RGAP; we find no other systems that support a similar functionality.

Remark. RGAP$^-$ is essentially the RDBMS approach advocated by [ZY17], enhanced with HER (we did not use [ZY17] as a baseline since it has no implementation available). We did not test GraphFrames [DJL+16] either as it does not support declarative queries. Moreover, it is no better than BigDAWG and Myria in terms of expressiveness.

The experiments were run on a cluster of 10 linux machines, each with 2 Intel Xeon 2.2 GHz processors and 64 GB memory. We used all 10 machines for HER and AExt modules of RGAP, and used a server with PostgreSQL 9.6 as the central portal for executing queries (GRA2RA). While one can use a parallel SQL system (*e.g.,* Greenplum or SparkSQL) to process queries with multiple machines, we chose PostgreSQL as a "lowest common denominator". Each experiment was run 5 times. The average is reported here.

Experimental results. We next report our findings.

Exp-1: Case study. Using q_1 of Table 5.5, we validated the need for semantic joins, versus the closest queries supported by BigDAWG and Myria.

More specifically, q_1 is to find "US republican and democratic politicians that have the same birthplace (city)". It is based on Celebrity dataset collection, *i.e.,* DBpedia

relations politician(name, state) (simplified) and YAGO3 graph G that includes information such as party and birthplace that is not available in politician. It can be written as a gSQL query over Celebrity as follows:

```
createview politician_aug as        /*create a virtual view via static join*/
select name, party, birthPlace, from politician join G_YAGO3[party, birthPlace]
where party = 'republican' or 'democratic'

select T1.name, T2.name             /*SQL with the semantic join result*/
from   politician_aug as T1, politician_aug as T2
where T1.birthPlace = T2.birthPlace and T1.party <> T2.party
```

On top of RGAP, query q_1 uses a static join to find birthplace and party of the politicians. In contrast, q_1 cannot be expressed in BigDAWG and Myria since relation politician contains no birthplace and party data; moreover, BigDAWG and Myria do not support entity matching across relations and graphs, and cannot extract the information from graph G. As a consequence, they could only settle with approximate queries as follows.

(1) One approximation of q_1, denoted by q_1^1, is to connect politician and graph G by computing their Cartesian-product, with a cast-like operator for cross-island queries in BigDAWG [DES+15]. This query overlooks the semantic connection between the two data sources, and produces excessive false positive results, as depicted in Fig. 5.8.

(2) A better approximation, denoted by q_1^2, is to use the name attribute to match politician tuples to vertices of G, and search birthplace and party information from neighbors of the matched vertices, assuming that the labels of their neighbor vertices have such information. This is written in (simplified) SQL for BigDAWG (or the equivalent MyriaL/datalog form in Myria) as follows:

```
(q1^2) createview match as          /*compute matches by joining with name */
    select politician.name , R_V.vid from politician, R_V
    where politician.name = R_V.label

    createview politician_aug1 as   /*extract birthPlace from match neighbors*/
    select match.name as name, R_V.label as birthPlace
    from match, R_V, R_E
    where match.vid = R_E.vid1 and R_E.vid2 = R_V.vid and R_E.label =
          'birthPlace'
```

```
createview politician_aug2 as        /*extract party from match neighbors*/
select name, birthPlace, R_V.label as party
from politician_aug1, R_V, R_E
where politician_aug1.vid = R_E.vid1 and R_E.vid2 = R_V.vid and
      R_E.label = 'party' and R_V.label = 'republican' or 'democratic'
select T_1.name, T_2.name        /*compute query answers*/
from politician_aug2 as T_1, politician_aug2 as T_2
where T_1.birthPlace = T_2.birthPlace and T_1.party <> T_2.party
```

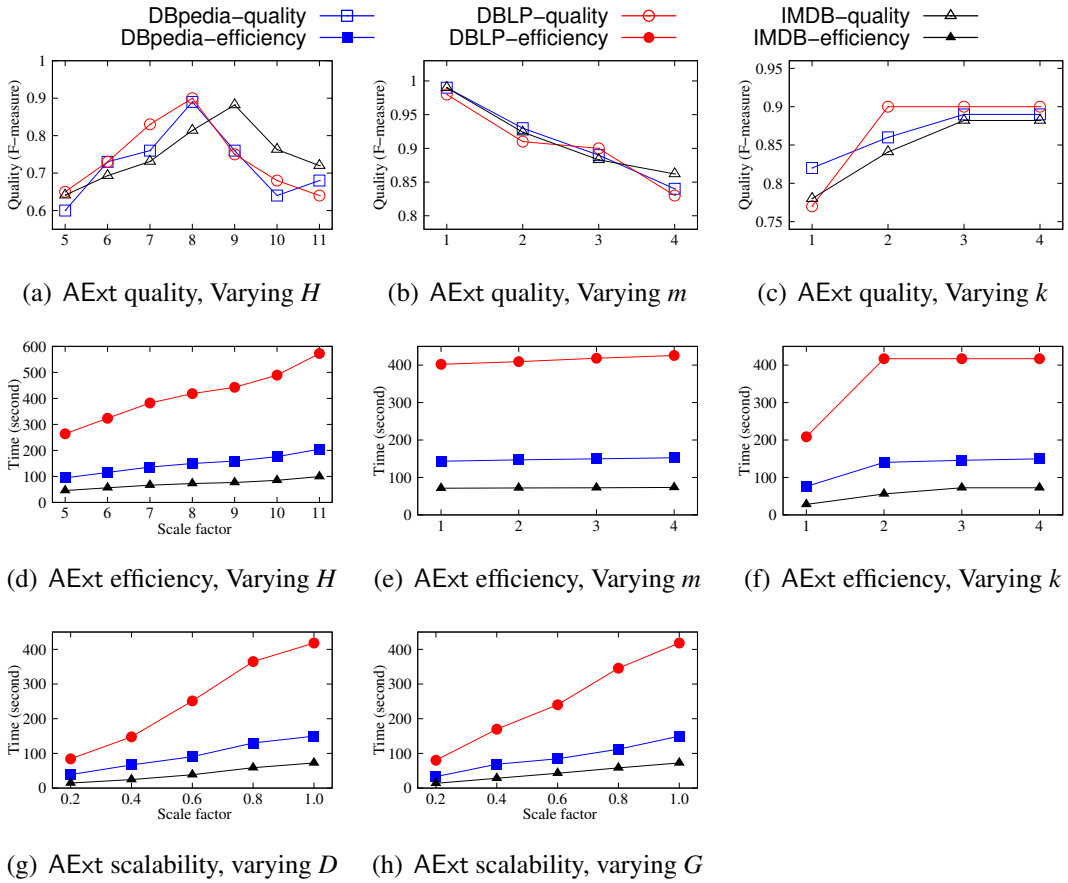
Obviously, (a) it is quite difficult and tedious for the users to write q_1^2 since they have to know the topology of G in order to extract birthPlace and party via SQL joins. (b) The result of q_1^2, although better than q_1^1, still contains many false positives as shown in Fig. 5.8, since names cannot uniquely determine matches in G. (c) It may even miss correct answers to q_1 since the birthplace and party information may pertain to the neighbors of the matches via paths.

(3) Better than q_1^2 is to approximate q_1 with RGAP$^-$ by means of the match relation $R_m(\mathsf{tid}, \mathsf{vid})$ from HER (denoted by q_1^3). As also shown in Fig. 5.8, q_1^3 is more accurate than q_1^2 for matching politician tuples to G. However, it still suffers from the same usability and accuracy problems that q_1^2 encounters, due to the restriction of extracting attributes from R_V and R_E with SQL joins.

Observations. As shown above, BigDAWG and Myria handle graphs essentially in the same way as a conventional RDBMS does, although they are polystores. They work no better than an RDBMS when it comes to real-world queries across graphs and relations. None of the queries in Table 5.5 and 5.6 can be precisely answered by BigDAWG, Myria or RGAP$^-$, while all of them can be easily written in gSQL. Worse yet, even if we pre-computed and materialized the entity match relation R_m, we could still miss attributes being queried if they are not already in the RDBMS (*e.g.,* RGAP$^-$ with q_1^3).

This justifies the need for heterogeneous entity linking and attribute extraction supported by RGAP, which are beyond the capacity of existing systems, either polystores or SQL-based [ZY17].

Exp-2: Quality of semantic joins. We next evaluated semantic joins. For each relation R, we first picked and dropped m (recall m of AExt, Section 5.2) attributes (columns) from R, yielding R'. We then represented the factorized form [OS16] of R in a graph G_R: we encoded each attribute value c of R as a value vertex v_c and each



(a) AExt quality, Varying H (b) AExt quality, Varying m (c) AExt quality, Varying k

(d) AExt efficiency, Varying H (e) AExt efficiency, Varying m (f) AExt efficiency, Varying k

(g) AExt scalability, varying D (h) AExt scalability, varying G

Figure 5.9: Performance of RGAP

tuple t as a tuple vertex v_t; there was an edge (c,t) in G_R if c is a value of t.

Accuracy. We tested the ability to recover the dropped attributes in R by semantic joins $R' \bowtie G_R$. We measured the F-measure of the extracted attributes by treating the original relation of R as the golden standard. More specifically, we first dropped attributes volume and affiliation from Paper relations, writers and primaryProfession from Movie relations, birthPlace and birthDate from Celebrity relations. We then joined the remaining of Paper, Movie, Celebrity relations with graphs RKBExplorer, LinkedMDB, YAGO3, respectively, and tested the accuracy of the join results by treating their original relations as the ground truth. For each dataset, the query interest Ω consisted of values randomly picked from the dropped columns. The initial KMC centers were vertex-path pairs with different labels, to reduce value conflicts (see Example 5.8).

Varying parameters H, m and k of module AExt (Section 5.2), we tested the accuracy with all three datasets. The default H, m and k are 8, 3 and 3, respectively, unless stated otherwise. The results are reported in Figures 5.9(a), 5.9(b) and 5.9(c), respectively.

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
F-measure	0.98	0.82	0.96	1.00	1.00	0.98	0.88	1.00	1.00	0.77
	q_{11}	q_{12}	q_{13}	q_{14}	q_{15}	q_{16}	q_{17}	q_{18}	q_{19}	q_{20}
F-measure	1.00	0.88	1.00	0.94	1.00	1.00	0.89	1.00	0.89	1.00

Table 5.8: F-measure of heuristic joins for entitic queries

(1) When varying H from 5 to 11, the F-measure of AExt on all three datasets first increases, reaches the highest value when H approximates the number of "types" in Ψ (see Section 5.2), and drops as H keeps growing, *e.g.,* it peaks at 0.9 when $H = 8$ over Paper. This shows that H has big impact on the quality, and the default H value introduced in Section 5.2 demonstrates the best performance.

(2) Varying the number m of attributes to extract from 1 to 4, the F-measure of AExt decreases, *e.g.,* from 0.99 to 0.81, 0.97 to 0.82 and 0.99 to 0.86 over Celebrity, Paper and Movie, respectively. This is because with larger m, AExt has to extract more attributes, and with this comes larger uncertainty, *i.e.,* lower F-measure.

(3) Varying k from 1 to 4, the F-measure of AExt increases from 0.82 to 0.89, 0.77 to 0.9 and 0.78 to 0.88 over Celebrity, Paper and Movie, respectively, since longer paths can capture more candidate attributes. However, the quality plateaus when k keeps growing, *e.g., k* from 3 to 4 in Fig. 5.9(c). This is because attributes extracted from longer paths have weaker associations and are less useful.

Heuristic join. We also evaluated the accuracy and efficiency of the heuristic-join implementation of dynamic joins. We tested all the 20 queries and compared the accuracy (F-measure) of their heuristic join results *w.r.t.* the entitic-based join answers (one can verify that all the 20 queries are entitic). We found that heuristic joins achieves F-measure 0.94 on average (see Table 5.8). As for the efficiency, the mean running time for heuristic joins to complete all queries is 25.6s. Compared with Naive-I and Naive-II, heuristic join is on average 5.85 and 5.76 times faster, respectively. These demonstrate that heuristic joins strike a balance between accuracy and efficiency.

This verifies that heuristic join is an accurate approximation of dynamic joins. Given that it assumes no restriction on the queries, we can safely use it as a fallback in RDBMS to support dynamic joins.

Exp-3: Efficiency and scalability. We next evaluated (a) the performance of module AExt for extracting attributes, and (b) the performance of evaluating gSQL queries by module SJ, with each of the dynamic join implementation methods.

Efficiency. Varying H, k and m in the same way as in Exp-2, we report the results in Figures 5.9(d), 5.9(e) and 5.9(f). As expected, AExt takes longer to extract attributes with larger k, *e.g.,* it takes from 75.8s to 149.7s, 208.5s to 417.2s, and 28.0s to 72.4s when k is increased from 1 to 4 over Celebrity, Paper and Movie, respectively. This is because with larger k, AExt has to examine more paths when extracting values. Similarly, KMC and attribute ranking take longer with larger H; it takes from 94.3s to 204.8s when H grows from 5 to 11 on Celebrity. In contrast, the runtime of AExt is insensitive to m since it only affects the final selection of the attributes and values, and incurs a much smaller cost compared to the time for path exploration, clustering and attribute ranking.

Scalability. We then tested the scalability of AExt by varying the sizes of the relations and graphs of all datasets.

(1) Varying the scale of relations. Varying the size of $\mathcal{D}$ using a scale factor from 0.2 to 1.0, we tested the runtime of module AExt. As shown in Fig. 5.9(g), AExt scales well with large relations, *e.g.,* it takes only 3.8 times longer on Celebrity when it grows by 5 times.

(2) Varying the scale of graph. Using full relations, we varied the size of graphs with a scale factor from 0.2 to 1.0. From Fig. 5.9(h) we can see that AExt works well with large graphs, *e.g.,* it takes 149.7s over Celebrity graph with 3.4M vertices and 10.2M edges.

Semantic join. We evaluated the query execution time of RGAP for all the gSQL queries. We remark that the execution time also depends on the RDBMS employed by RGAP (single-thread PostgreSQL for the tests); one can use faster RDBMS or parallelization to further reduce the querying time. To compare different implementation methods, we treated all semantic joins in the queries as dynamic joins by pushing selections down through semantic joins.

We found the following. (1) RGAP scales well for querying big relations and graphs: with entitic join it returns query answers within 0.77s across relations of up to 4.4 million tuples and graphs with 31.1 million edges. (2) Both Naive-I and Naive-II are significantly slower than entitic join and heuristic join due to the need for running HER

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
Entitic join	0.25s	0.22s	0.77s	0.12s	13.4s	6.19s	13.6s	17.0s	4.2s	4.0s
Heuristic join	2.8s	1.8s	1.2s	0.5s	6.8s	22.6s	21.0s	96.0s	79.5s	45.6s
Naive-I	412.3s	116.2s	8.7s	1.3s	28.2s	228.0s	21.2s	129.9s	94.2s	62.6s
Naive-II	376.4s	116.7s	9.6s	1.3s	36.3s	212.4s	21.6s	185.1s	104.9s	60.7s

	q_{11}	q_{12}	q_{13}	q_{14}	q_{15}	q_{16}	q_{17}	q_{18}	q_{19}	q_{20}
Entitic join	0.24s	0.27s	0.064s	0.75s	0.10s	0.069s	0.095s	0.055s	0.070s	0.015s
Heuristic join	59.6s	66.4s	0.57s	4.1s	44.4s	7.8s	2.7s	11.9s	32.7s	4.3s
Naive-I	257.2s	241.7s	3.6s	85.7s	376.4s	47.5s	579.1s	174.7s	89.9s	35.9s
Naive-II	234.5s	232.8s	2.5s	66.3s	412.0s	39.4s	572.5s	149.9s	83.0s	34.6s

Table 5.9: Query execution time

and AExt during query execution. On average, entitic join is 48.7 and 48.0 times faster than Naive-I and Naive-II, respectively; it is 5.85 and 5.76 times faster by heuristic join.

Exp-4: Incremental maintenance. We next evaluated the performance of RGAP for handling updates to databases and graphs.

Updating extracted relations. We first report the efficiency of incremental relation extraction described in Section 5.3.3. We computed the changes to the HER matches first, and tested the time needed for updating extracted relations by AExt. We generated random updates ΔG consisting of the same number of insertions and deletions, so that the size of the updated dataset remains unchanged. We compared the runtime of incremental AExt (denoted by IncEXT, Section 5.3.3) against AExt that re-computes the value extraction with updated G and the HER matches $f(\mathcal{D}, G)$.

Varying ΔG. We varied $|\Delta G|$ from 1% to 50% of $|G|$ for comparison. When $|\Delta G| = 1\%|G|$, IncEXT beats AExt by 198, 120 and 117 times over Celebrity, Paper and Movie, respectively. It is still faster than AExt even when $|\Delta G|$ is up to 40%, 45% and 45% of $|G|$, respectively.

Updating join results. Using the same setting as above, we compared the time for updating semantic join results with incremental method in Section 5.3.3 against re-evaluating queries. We only calculated the time for evaluating queries and excluded the time for updating HER matches and extracted relations via AExt. We found that on average INC of RGAP updates semantic join results faster than re-evaluation with updates $\Delta\mathcal{D}$ and ΔG of size both up to 10% of $|\mathcal{D}|$ and $|G|$. These verify that INC is able to handle updates online.

Accuracy of IncEXT. Increasing $|\Delta G|$ from 1% to 20% of $|G|$, we find that the accuracy (F-measure) of IncEXT on each dataset slightly decreases: from 0.89 to 0.86, 0.9 to 0.88 and 0.88 to 0.82 over Celebrity, Paper and Movie, respectively. These verify that

IncEXT has little influence on the accuracy of attribute extraction.

Summary. We find the following. (1) With gSQL, RGAP is able to express real-life queries across relations and graphs, while existing federated systems and SQL-based methods cannot accurately answer such queries. (2) Via semantic joins, RGAP extracts high-quality matches from graphs with average F-measure above 0.89. (3) RGAP scales well with large relations and graphs, *e.g.,* it extracts attributes in 417s on relations with 4.4M tuples and graphs with 47M vertices and edges. (4) RGAP answers online gSQL queries efficiently, taking at most 17s when queries can be reduced to static joins (*i.e.,* entitic). It is 48.4 times faster than the baselines. (5) Heuristic joins speed up dynamic joins by 5.8 times on average, and retain accuracy above 0.94. (6) RGAP handles updates efficiently. IncEXT updates the extracted relations 145 times faster than AExt when $|\Delta G| = 1\%|G|$, and is still faster when $|\Delta G|$ is up to 40%.

5.5 Novelty and Contributions

RGAP proposed in this Chapter differs from previous multi-model systems in the following. (1) RGAP supports SQL queries across graphs and relations based on semantic joins and attribute (schema) extraction. In contrast, polyglot systems are homogeneous, and support neither linking entities with heterogeneous structures nor correlating attributes of relations and graphs, no matter whether they are relation-based or graph-based. For relation-based systems in particular, attribute extraction requires to traverse paths via costly SQL joins. Multistores and polystores do not yet support graphs (storage and queries) due to radically different structures of graphs from other data models. Hence, our approach can be adapted to the federated systems by taking HER as an oracle and supporting attribute extraction. (2) Multistores and polystores treat different stores as independent datasets, and "syntactically" connect such datasets by *cross-product*. In contrast, RGAP promotes *semantic joins* of relations and graphs by correlating tuples and vertices that refer to the same entity. (3) RGAP equip RDBMS with a convenient capacity of correlating and querying entities in relations and graphs. We aim to strike a balance between the expressivity and complexity, without paying the costs of full-fledged graph computations, query decomposition and result composition. Moreover, RGAP retains the ease and composability of SQL as requested by FinTech practitioners.

RGAP also differs from prior schema/data extraction methods in the following. (1)

It extracts either additional and alternative attributes pivoted at entities that semantically match entities in relations, to enrich an existing relation schema, or a relation schema for entities of a specific type with their key features, not to abstract the topological structure of the entire graph. (2) We extract attributes based on users' interest for queries, while no prior methods are query-driven. (3) As opposed to attribute selection techniques [HH03, SSGC17] that focus on relations, text or images with uniform representations, we tackle graphs that model data in topological structures. Moreover, different from feature selection methods based on supervised training [SSGC17, CS14], we extract attributes from graphs by sequence embedding and clustering, which are unsupervised and data-driven, reducing manual labeling cost.

Chapter 6

Conclusion and Future Work

This Chapter summarizes the results of this thesis and proposes future work.

6.1 Summary

Towards graphs and relational data, this thesis proposes various techniques to tackle the veracity and variety challenges of big data, which aims to improve graph data quality and support queries across relations and graphs. Specifically, we conclude the main results as follows.

- We have proposed GARs to catch missing links/attributes and semantic inconsistencies in a uniform framework, by unifying rule-based and ML-based methods. We have formalized association deduction with GARs in terms of the chase, and proved its Church-Rosser property. We have shown that the satisfiability, implication and association deduction problems for GARs are coNP-complete, NP-complete and NP-complete, respectively, retaining the same complexity bounds as their GFD counterparts, despite the increased expressive power of GARs. The incremental deduction problem is DP-complete for GARs versus coNP-complete for GFDs. In addition, we have provided parallel algorithms for association deduction and incremental deduction. Using real-life and synthetic graphs, we have experimentally verified the effectiveness, scalability and efficiency of the parallel algorithms.
- We have explored a new approach to discovering rules from big graphs G, consisting of (1) a graph reduction scheme to deduce a smaller graph $G_{\mathcal{A}}$ of data pertaining to a given application $\mathcal{A}$, (2) a method to sample a set H of small graphs from $G_{\mathcal{A}}$, such that the rules mined from H satisfy given support and recall bounds in G, and (3) an algorithm with the parallel scalability to mine rules from small H instead of

from the entire G, for GARs that may embed ML predicates and subsume GPARs and GEDs as special cases. We have experimentally verified that the approach is promising in reducing excessive number of rules and scaling with large graphs.

- We have proposed TACOs, a class of rules for predicting temporal events, defined in terms of change patterns, temporal conditions and ML prediction models, and studied the complexity bounds of the satisfiability, implication and prediction problems for TACOs. We have established a creator-critic rule discovery framework based on generative ML models, that is able to discover TACOs with large patterns which cannot be mined by traditional levelwise search methods. For practical use, we have provided a strategy for partitioning temporal graphs based on temporal locality, and an algorithm for temporal event prediction with TACOs that guarantees the parallel scalability. Putting all these together, we have developed TASTE, a system for discovering TACOs and predicting events with TACOs. We have experimentally verified that TASTE is promising in event prediction.
- We have proposed a notion of semantic joins, RGAP, to support SQL across relations and graphs, and methods for implementing static joins and dynamic joins within and atop existing RDBMS. KMC-based methods have been explored to extract (a) attributes from graphs and complement relations, and (b) a schema for entities of a specific type in a graph, based on users' interests. We have provided a method for incrementally maintaining extracted relations in response to updates. We have experimentally verified that our method is promising for providing RDBMS with a capacity to correlate and synthesize data from relations and graphs.

6.2 Future Work

Related to this thesis, we propose some topics for future that deserve full treatment.

Graph Association Rules. One topic for future work is to explore applications of GARs by treating GARs as "soft" rules, since GARs defined in Chapter 2 are hard logic rules that may fail to model uncertainties in real life. An intuitive idea, for instance, is to learn and attach a probability value to each GAR, such that the deduction result holds under a certain probability and the user can evaluate this result based on that probability. Thus, how to decide the probability value for each GAR, and how to efficiently calculate the probability for the deduction result are worthy of being explored. Another topic is to keep up with the progress in machine learning and find non-embedding-

based ML classifiers for GARs, which extend the application scenarios of GARs.

Graph Rule Discovery. Although the discovery scheme for graph rules in Chapter 3 accelerates the discovery process by cutting back data that is irrelevant to users' application interests, incorrect or meaningless rules can still be discovered due to noises in the sampled real-life data. Thus, one topic for future work is exploring strategies to reduce the impact of noises in real-life graphs on rule discovery. One possible solution is data pre-processing by anomaly/outlier detection techniques before conducting rule discovery. Another topic is to improve the user interface of the system, making it easier for users to express their ideas about "applications" of graph rules, such that the discovered rules better meet their application demands and users do not have to learn about details of graph rules, *e.g.,* "patterns" or "predicates". An practical approach is incorporating natural language processing (NLP) techniques into the discovery scheme, so that users just need to write natural language to express interests in target applications and NLP models will extract the rule discovery demands behind the texts.

Temporal Event Prediction. In addition to temporal knowledge graph completion and dynamic recommendation in Chapter 4, one topic for future work is to evaluate TASTE for predicting events of other types, *e.g.,* finance crisis and natural disasters. Another topic is to make TASTE "real-time" by incrementally discovering rules and predicting events in response to updates to temporal graphs. Moreover, a mixed rule discovery method that combines the creator-critic framework (Chapter 4) and levelwise mining is worth considering. Leveraging the advantage of each approach, this mixture may improve the quality and efficiency of rule discovery at the same time.

Semantic Join. One topic for future work is to enrich (knowledge) graphs with relational data by means of semantic join, an approach different from RDB to RDF translation [MMZ14]. Another topic is to apply semantic join to query-driven data discovery, on-demand data integration and data extraction in data lakes. In addition, entities across relations and graphs, foreign keys in relations, and edges in graphs can be jointly considered for more effective data query. How to uniformly model and better elaborate these semantic links in these two data formats is worthy of investigation.

Appendix A

Appendix

A.1 Case Study of the Discovered TACOs

Below we present some representative TACOs mined from different datasets; their Δ-patterns are depicted in Figure A.1.

(1) In Amazon, $\varphi_a = \Delta Q_a[\bar{x}](X_a \rightarrow (\mathsf{recommend}(y_2, x_1), [0,1]))$, where the edges in ΔQ_a denote good user ratings for items. Precondition X_a is $\bigwedge_{i\in[1,4]}((e_i.t \leq 1-i \wedge e_i.t \geq -i) \wedge (e'_i.t \leq 1-i \wedge e'_i.t \geq -i)) \wedge y_1.\mathsf{type} = y_2.\mathsf{type} \wedge \mathcal{M}(y_2, x_1\ \mathsf{recommend}, 0)$; and $\mathcal{M}$ is the SASRec recommendation model. It says that if users x_1 and x_2 rated items y_1 and y_2 of the same type high regularly every week in the past month, respectively, x_1 gave another good rating for y_2, and if the model $\mathcal{M}$ suggests to recommend y_2 to x_1, then one should also recommend y_2 to x_1 in the next week.

The TACO strengthens ML predictions ($\mathsf{recommend}(y_2, x_1)$) with the logic conditions and temporal predicates in X_1. It reduces false positives of SASRec with additional conditions.

(2) Another TACO rule discovered from the knowledge graph YAGO is $\varphi_b = \Delta Q_b$ $(X_b \rightarrow (\mathcal{M}(x, z_2, \mathsf{win}, 20), [20,20]))$. Here the precondition X_b is (y.name=United States)

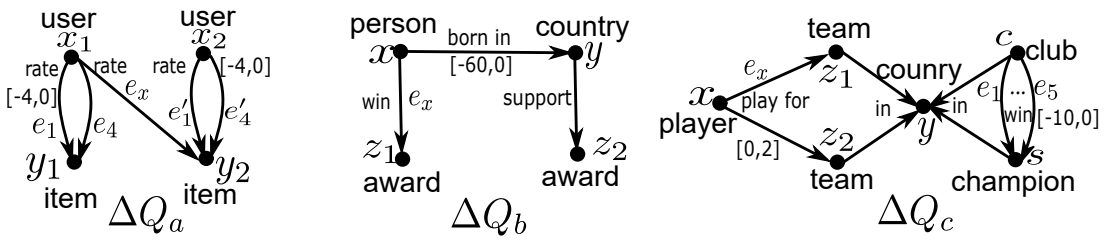


Figure A.1: Discovered TACOs

$\wedge(z_1.\mathsf{name}$=Nobel Prize in Chemistry$)\wedge(z_2.\mathsf{name}$=Vannevar Bush Award), and $\mathcal{M}$ is the REGCN model for graph completion.

This TACO interprets the ML prediction of $\mathcal{M}$; that is, person x will receive the Vannevar Bush Award in 20 years if (a) x is an American, and (b) he has won the Nobel Prize in Chemistry before 60 years old (specified by time window in ΔQ_b).

(3) In WIKI, φ_c=$\Delta Q_c(X_c \rightarrow (\mathsf{join}(x,c)),[3,5]))$. Here the precondition X_c is $(y.\mathsf{name}=$ Spain$)\wedge(z_1.\mathsf{name}$=national under-16 team$)\wedge(z_2.\mathsf{name}=$ national under-18 team$))$ $\wedge(\bigwedge_{i,j\in[1,5],i\neq j} e_i.t \neq e_j.t)$. This TACO indicates that if a Spanish football player x has been selected to both the under-16 and under-18 youth national football teams, then he will join one of the most successful football clubs c of the country that has won at least half of the champions in the past 10 years (edges e_1 to e_5 of ΔQ_c), in 3 to 5 years.

This is a typical TACO rule that makes predictions based on logic conditions and temporal predicates only.

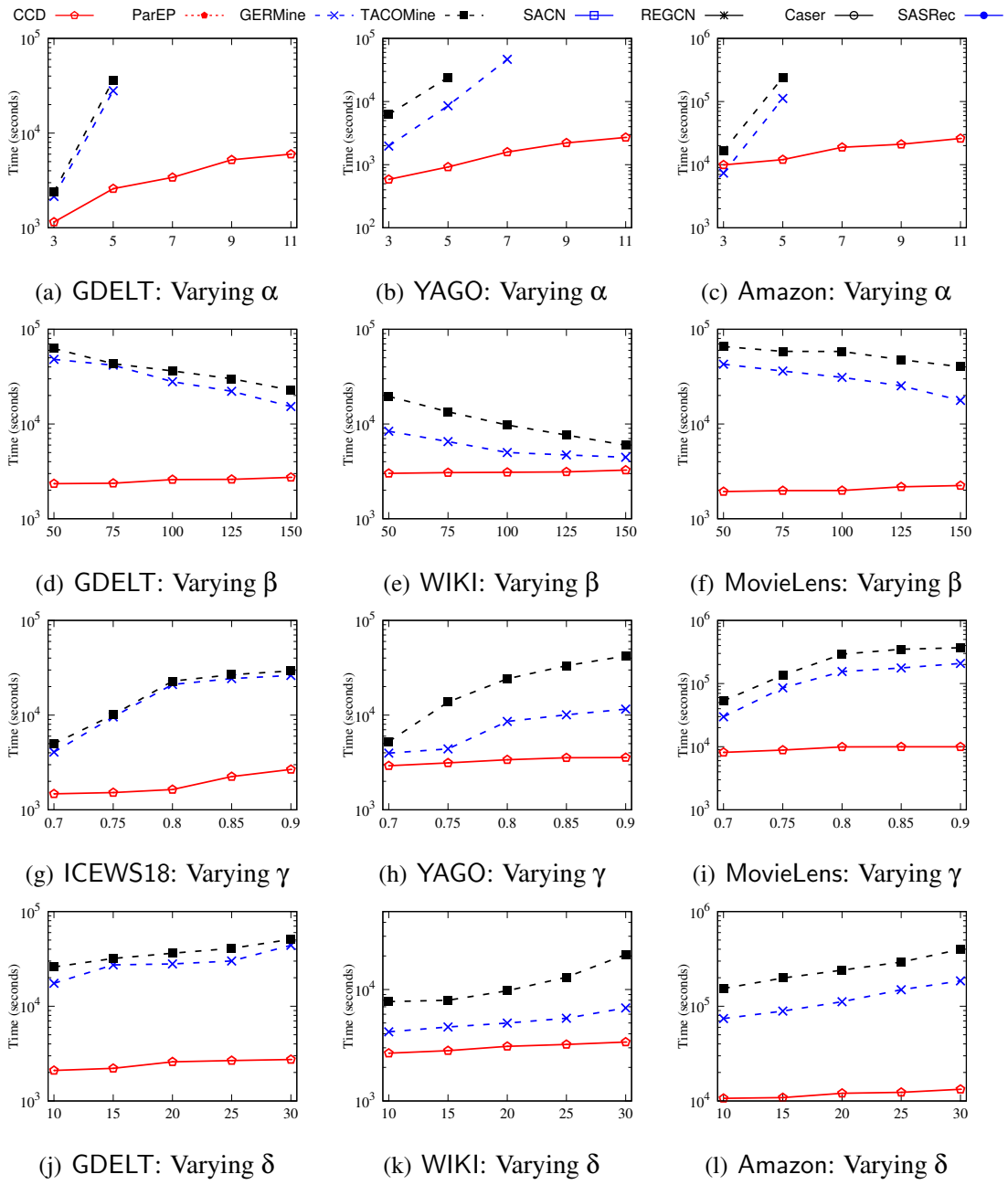
A.2 More Experimental Results on Discovery

In Figure A.2, we report the the efficiency of the discovery method CCD on the rest of real-life graphs from each class, which are not shown in Section 4.6. We find the following from these results.

(1) The performance gap between CCD and levelwise search-based methods, *i.e.,* GERMine and TACOMine, becomes substantially larger with the increase of α, as shown in Figures A.2(a) to A.2(c). This validates the use of ML generative models, which are far less sensitive to the pattern size than the levelwise search strategies that require exponentially large search space. This explains why CCD is able to discover rules with large graph patterns.

(2) The support bound β has little impact on the runtime of CCD (Figures A.2(d) to A.2(f)), while a larger bound on support helps reduce the cost of GERMine and TACOMine due to their pruning methods.

(3) Figures A.2(g) to A.2(i) show that CCD and levelwise algorithms become slower when given a larger bound γ on confidence. However, CCD is still much more efficient than the baselines, *e.g.,* it takes 2247 seconds on ICEWS18 when $\gamma = 0.8$, while the levelwise method TACOMine cannot finish in 7.4 hours.



(a) GDELT: Varying α (b) YAGO: Varying α (c) Amazon: Varying α

(d) GDELT: Varying β (e) WIKI: Varying β (f) MovieLens: Varying β

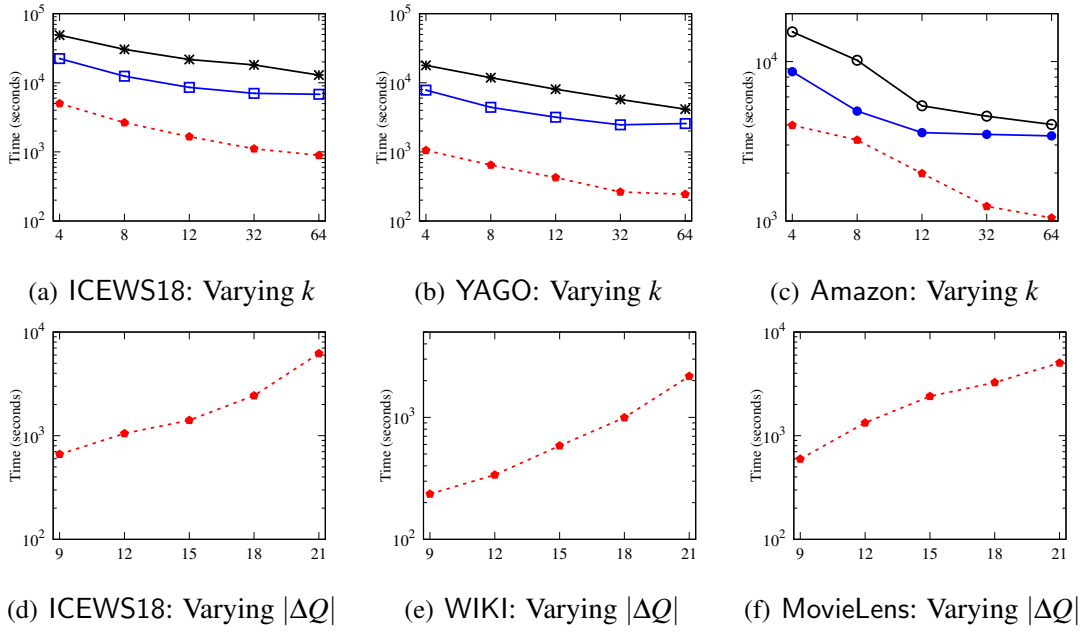
(g) ICEWS18: Varying γ (h) YAGO: Varying γ (i) MovieLens: Varying γ

(j) GDELT: Varying δ (k) WIKI: Varying δ (l) Amazon: Varying δ

Figure A.2: Performance of discovery

(4) All the discovery approaches need more time to find TACOs with larger time windows, as reported in Figures A.2(j) to A.2(l). This said, CCD outperforms the baselines for all cases of time windows. For instance, CCD takes 2828 seconds on WIKI when the time window bound $\delta = 15$, as opposed to more than 2.2 hours by TACOMine.

These are consistent with the findings in Exp-1 of Section 4.6.



(a) ICEWS18: Varying k (b) YAGO: Varying k (c) Amazon: Varying k

(d) ICEWS18: Varying $|\Delta Q|$ (e) WIKI: Varying $|\Delta Q|$ (f) MovieLens: Varying $|\Delta Q|$

Figure A.3: Performance of prediction

I \ N	50	100	150	200	250	300
15	6.15%	7.74%	15.91%	18.82%	25.75%	27.16%
20	6.51%	13.80%	18.62%	26.48%	26.44%	28.56%
25	6.67%	15.32%	24.95%	27.48%	28.76%	32.94%
30	7.84%	18.60%	28.39%	29.65%	32.55%	33.07%

Table A.1: Coverage on ICEWS18 using Erdös-Rényi (ER)

I \ N	50	100	150	200	250	300
15	9.08%	19.15%	38.58%	45.38%	46.95%	48.83%
20	13.14%	29.01%	45.66%	46.45%	47.62%	48.38%
25	16.78%	39.64%	46.35%	47.08%	48.69%	49.61%
30	19.72%	48.61%	47.87%	48.69%	49.91%	50.89%

Table A.2: Coverage on ICEWS18 using Barabási-Albert (BA)

A.3 More Experimental Results on Quality of Discovery

As shown in Tables A.1 and A.2, we replaced the GAN model in our framework with

classic graph generation models, *i.e.,* Erdös-Rényi (ER) and Barabási-Albert (BA) models, and tested their coverage values. We can see that when parameters N and I increase, the coverage of ER and BA models converge to a low value within limited number of iterations. This is because these classical models cannot iteratively learn from the graph data and adaptively generate high-quality patterns. In addition, when $N = 300$ and $I = 30$, the coverage of BA is higher than that of ER, since the BA model follows power-law distribution that may better suit ICEWS18 dataset, while the ER model generates edges with a uniform probability.

A.4 More Experimental Results on Prediction

Figure A.3 shows the results of different parallel prediction algorithms on some real-life datasets taken from each of the three classes, which are a complement to those reported in Section 4.6.

From the results we can see the following. (1) ParEP becomes faster (resp. slower) with the increase of the number k of processors (resp. the pattern size $|\Delta Q|$), which is consistent with the results presented in Exp-4 of Section 4.6. (2) In addition, ParEP constantly outperforms the other baselines, *e.g.,* it is on average 7.5 times faster than REGCN over ICEWS18 when $k = 32$ and $|\Delta Q| = 18$.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[ABA+09] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. HadoopDB: An architectural hybrid of mapreduce and DBMS technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[ABH14] Charu C. Aggarwal, Mansurul Bhuiyan, and Mohammad Al Hasan. Frequent pattern mining algorithms: A survey. In *Frequent Pattern Mining*, pages 19–64. 2014.

[ACP10] Waseem Akhtar, Alvaro Cortés-Calabuig, and Jan Paredaens. Constraints in RDF. In *SDKB*, 2010.

[ÁFCO10] Miguel R Álvarez, Paulo Félix, Purificación Carinena, and Abraham Otero. A data mining algorithm for inducing temporal constraint networks. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 300–309. Springer, 2010.

[AFU13] Foto N Afrati, Dimitris Fotakis, and Jeffrey D Ullman. Enumerating subgraph instances using Map-Reduce. In *ICDE*, 2013.

[AG08] Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.

[Agg14] Charu C. Aggarwal. *Data Classification: Algorithms and Applications*. CRC Press, 2014.

[AHAS03] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Ismailcem Budak Arpinar, and Amit P. Sheth. Context-aware semantic association ranking. In *SWDB*, pages 33–50, 2003.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.

[AR06] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory Comput. Syst.*, 39(6):929–939, 2006.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.

[AV06] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[BBBG09a] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *joint European conference on machine learning and knowledge discovery in databases*, pages 115–130. Springer, 2009.

[BBBG09b] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *ECML/PKDD*, 2009.

[BBC+00] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R Guidetti, G Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The MOMIS project demonstration. In *VLDB*, pages 611–614, 2000.

[BBL05] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 9:323–375, 2005.

[big] BigDAWG. *http://bigdawg.mit.edu/get-bigdawg*.

[BKN17] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. Efficient denial constraint discovery with hydra. *PVLDB*, 11(3):311–323, 2017.

[BLH19] Bibek Bhattarai, Hang Liu, and H. Howie Huang. CECI: Compact embedding cluster index for scalable subgraph matching. In *SIGMOD*, 2019.

[BLO+15] Elizabeth Boschee, Jennifer Lautenschlager, Sean O'Brien, Steve Shellman, James Starz, and Michael Ward. Icews coded event data. *Harvard Dataverse*, 12, 2015.

[BUGD+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[Bus20] Business of Data. How graph databases are transforming advanced analytics. *https://www.business-of-data.com/articles/graph-databases*, 2020.

[BX21] Yuemin Bian and Xiang-Qun Xie. Generative chemistry: Drug discovery with deep learning generative models. *Journal of Molecular Modeling*, 27(3):1–18, 2021.

[CAEHP+20] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*, 2020.

[CB01] George Casella and Roger Berger. *Statistical Inference*. Duxbury Resource Center, 2001.

[CF13] Luca Cagliero and Alessandro Fiori. Discovering generalized association rules from twitter. *Intelligent Data Analysis*, 17(4):627–648, 2013.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[CGWJ16] Yang Chen, Sean Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. Ontological pathfinding. In *SIGMOD*, 2016.

[CIP13] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[CN11] Sanjay Chakraborty and NK Nagwani. Analysis and study of incremental k-means clustering algorithm. In *International Conference on High Performance Architecture and Grid Computing*, pages 338–341. Springer, 2011.

[Coh16] William W. Cohen. Tensorlog: A differentiable deductive database. *CoRR*, abs/1605.06523, 2016.

[CP12] Alvaro Cortés-Calabuig and Jan Paredaens. Semantics of constraints in RDFS. In *AMW*, 2012.

[CPF15] Klitos Christodoulou, Norman W Paton, and Alvaro AA Fernandes. Structure inference for linked data sources using clustering. In *Transactions on Large-Scale Data and Knowledge-Centered Systems XIX*, pages 1–25. Springer, 2015.

[CPS09] Venkatesan T. Chakaravarthy, Vinayaka Pandit, and Yogish Sabharwal. Analysis of sampling techniques for association rule mining. In *ICDT*, 2009.

[CS14] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[CSE07] Michael J Cafarella, Dan Suciu, and Oren Etzioni. Navigating extracted data with schema discovery. In *WebDB*, pages 1–6, 2007.

[CSN09] Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2009.

[CV85] Ashok K. Chandra and Moshe Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985.

[CWD+18] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Process. Mag.*, 35(1):53–65, 2018.

[CWG16] Yang Chen, Daisy Zhe Wang, and Sean Goldberg. Scalekb: scalable learning and inference over large knowledge bases. *VLDB J.*, 25(6):893–918, 2016.

[CXWZ18] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. GC-LSTM: graph convolution embedded LSTM for dynamic link prediction. *CoRR*, abs/1812.04206, 2018.

[DA16] Michael DiScala and Daniel J Abadi. Automatic generation of normalized relational schemas from nested key-value data. In *SIGMOD*, pages 295–310, 2016.

[dbl21a] Dblp collaboration network, 2021. *https://snap.stanford.edu/data/com-DBLP.html.*

[DBL21b] DBLP. DBLP RDF data, 2021. http://dblp.rkbexplorer.com/models/dump.tgz.

[DBL21c] DBLP. DBLP relational data, 2021. https://dblp.org/xml/release/dblp-2015-04-01.xml.gz.

[dbp] *https://github.com/dbpedia/dbpedia/tree/master/tools/DBpediaAsTables.*

[dbp21a] DBpedia, 2021. *http://wiki.dbpedia.org.*

[dbp21b] DBpedia as tables, athelete, 2021. *http://web.informatik.uni-mannheim.de/DBpediaAsTables/DBpedia-en-2016-04/csv/Athlete.csv.gz.*

[dbp21c] DBpedia as tables, politician, 2021. *http://web.informatik.uni-mannheim.de/DBpediaAsTables/DBpedia-en-2016-04/csv/Politician.csv.gz.*

[DDZ+18] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*, 2018.

[DES+15] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier,

Tim Mattson, and Stanley B. Zdonik. The BigDAWG polystore system. *SIGMOD Rec.*, 44(2):11–16, 2015.

[Des18] Amol Deshpande. In situ graph querying and analytics with graphgen: Extended abstract. In *GRADES*, pages 2:1–2:2, 2018.

[Din21] DingTalk, 2021. *https://www.dingtalk.com.*

[DJL+16] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. GraphFrames: an integrated API for mixing graph and relational queries. In Peter A. Boncz and Josep-Lluís Larriba-Pey, editors, *GRADES*, page 2, 2016.

[DKA11] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, 5(2):10:1–10:27, 2011.

[DLL+10] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The YouTube video recommendation system. In *RecSys*, 2010.

[DMS+17] Walter H. Dempsey, Alexander Moreno, Christy K. Scott, Michael L. Dennis, David H. Gustafson, Susan A. Murphy, and James M. Rehg. isurvive: An interpretable, event-time prediction model for mhealth. In *ICML*, 2017.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[dRP21] Gustavo H de Rosa and João P Papa. A survey on text generation using generative adversarial networks. *Pattern Recognition*, page 108098, 2021.

[DRT18] Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha P. Talukdar. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *EMNLP*, 2018.

[EASK14] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. GRAMI: frequent subgraph and pattern mining in a single large graph. *PVLDB*, 7(7):517–528, 2014.

[EBBJ16] Fredrik Erlandsson, Piotr Bródka, Anton Borg, and Henric Johnson. Finding influential users in social media using association rule learning. *Entropy*, 18(5):164, 2016.

[EE11] Yagil Engel and Opher Etzion. Towards proactive event-driven computing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, pages 125–136, 2011.

[ETJ+18] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.

[FFTD15] Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. Keys for graphs. *PVLDB*, 8(12):1590–1601, 2015.

[FGJ+22] Wenfei Fan, Liang Geng, Ruochun Jin, Ping Lu, Resul Tugay, and Wenyuan Yu. Linking entities across relations and graphs. In *ICDE*, 2022.

[FHLL20] Wenfei Fan, Chunming Hu, Xueli Liu, and Ping Lu. Discovering graph functional dependencies. *ACM Trans. Database Syst.*, 45(3):15:1–15:42, 2020.

[FJL+20] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. Capturing associations in graphs. *PVLDB*, 13(11):1863–1876, 2020.

[FJZL19] Kaiqun Fu, Taoran Ji, Liang Zhao, and Chang-Tien Lu. TITAN: A spatiotemporal feature learning framework for traffic incident duration prediction. In *SIGSPATIAL/GIS*, 2019.

[FL17] Sébastien Frémal and Fabian Lecron. Weighting strategies for a recommender system using item clustering based on genres. *Expert Syst. Appl.*, 77:105–113, 2017.

[FL19] Wenfei Fan and Ping Lu. Dependencies for graphs. *ACM Trans. Database Syst.*, 44(2):5:1–5:40, 2019.

[FLLT20] Wenfei Fan, Xueli Liu, Ping Lu, and Chao Tian. Catching numeric inconsistencies in graphs. *ACM Trans. Database Syst.*, 45(2):9:1–9:47, 2020.

[FLT20] Wenfei Fan, Ping Lu, and Chao Tian. Unifying logic rules and machine learning for entity enhancing. *Sci. China Inf. Sci.*, 63(7), 2020.

[FLTZ19] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. Deducing certain fixes to graphs. *Proceedings of the VLDB Endowment*, 12(7):752–765, 2019.

[FRP15] Jing Fan, Adalbert Gerald Soosai Raj, and Jignesh M. Patel. The case against specialized graph analytics engines. In *CIDR*, 2015.

[FWWX15] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. Association rules with graph patterns. *PVLDB*, 8(12):1502–1513, 2015.

[FWX16a] Wenfei Fan, Yinghui Wu, and Jingbo Xu. Adding counting quantifiers to graph patterns. In *SIGMOD*, 2016.

[FWX16b] Wenfei Fan, Yinghui Wu, and Jingbo Xu. Functional dependencies for graphs. In *SIGMOD*, 2016.

[FYX+18] Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, and Ruiqi Xu. Parallelizing sequential graph computations. *TODS*, 43(4), 2018.

[FZMK20] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pages 2331–2341, 2020.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GCC20] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowl. Based Syst.*, 187, 2020.

[GDN18] Alberto García-Durán, Sebastijan Dumancic, and Mathias Niepert. Learning sequence encoders for temporal knowledge graph completion. In *EMNLP*, 2018.

[GFHK09] Martin Gütlein, Eibe Frank, Mark A. Hall, and Andreas Karwath. Large-scale attribute selection using wrappers. In *CIDM*, pages 332–339. IEEE, 2009.

[GGM20] Yulong Gu, Yu Guan, and Paolo Missier. Towards learning instantiated logical rules from knowledge graphs. *CoRR*, abs/2003.06071, 2020.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GKBP20] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic embedding for temporal knowledge graph completion. In *AAAI*, 2020.

[GKHL18] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *CoRR*, abs/1805.11273, 2018.

[GMJ08] Hugh Glaser, Ian C Millard, and Afraz Jaffri. Rkbexplorer.com: A knowledge driven infrastructure for linked data providers. In *European Semantic Web Conference*, pages 797–801, 2008.

[GSW+20] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A review on generative adversarial networks: Algorithms, theory, and applications. *CoRR*, abs/2001.06937, 2020.

[GTHS13] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422, 2013.

[GXD+14] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. GraphX: Graph processing in a distributed dataflow framework. In *OSDI*, 2014.

[HAR11] Jiewen Huang, Daniel J Abadi, and Kun Ren. Scalable SPARQL querying of large RDF graphs. *PVLDB*, 4(11):1123–1134, 2011.

[HC09] Oktie Hassanzadeh and Mariano P Consens. Linked movie data base. In *LDOW*, 2009.

[HdAC+14] Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. Demonstration of the Myria big data management service. In *SIGMOD*, pages 881–884, 2014.

[HGPW16] Jelle Hellings, Marc Gyssens, Jan Paredaens, and Yuqing Wu. Implication and axiomatization of functional and constant constraints. *Ann. Math. Artif. Intell.*, 76(3-4):251–279, 2016.

[HH03] Mark A. Hall and Geoffrey Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Trans. Knowl. Data Eng.*, 15(6):1437–1447, 2003.

[HK16] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2016.

[HKG+19] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *SIGMOD*, 2019.

[HKPT99] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[HMB12] Negar Hariri, Bamshad Mobasher, and Robin D. Burke. Context-aware music recommendation based on latenttopic sequential patterns. In *RecSys*, 2012.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[IJB10] Robert Isele, Anja Jentzsch, and Christian Bizer. Silk server-adding missing links while consuming linked data. *COLD*, 665, 2010.

[IMD21] IMDB, 2021. *http://www.imdb.com/*.

[inc] Implementing incremental view maintenance in PostgreSQL. `https://www.postgresql.org/message-id/flat/20181227215726.4d166b4874f8983a641123f5%40sraoss.co.jp`.

[JD14] Prudhvi Janga and Karen C Davis. Mapping heterogeneous XML document collections to relational databases. In *ER*, pages 86–99. Springer, 2014.

[JLO21] Abdul Jabbar, Xi Li, and Bourahla Omar. A survey on generative adversarial networks: Variants, applications, and training. *ACM Computing Surveys (CSUR)*, 54(8):1–49, 2021.

[JMCH15] Alekh Jindal, Samuel Madden, Malú Castellanos, and Meichun Hsu. Graph analytics using vertica relational database. In *BigData*, pages 1191–1200. IEEE Computer Society, 2015.

[JQJR20] Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. Recurrent event network: Autoregressive structure inference over temporal knowledge graphs. In *EMNLP*, 2020.

[JRW+14] Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, and Mike Stonebraker. VERTEXICA: your relational friend for graph analytics! *PVLDB*, 7(13):1669–1672, 2014.

[KAB+12] Jeremy Kepner, William Arcand, William Bergeron, Nadya T. Bliss, Robert Bond, Chansup Byun, Gary Condon, Kenneth Gregson, Matthew Hubbell, Jonathan Kurz, Andrew McCabe, Peter Michaleas, Andrew Prout, Albert Reuther, Antonio Rosa, and Charles Yee. Dynamic distributed dimensional data model (D4M) database and computation system. In *ICASSP*, pages 5349–5352. IEEE, 2012.

[KAK+14] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. DBToaster: Higher-order delta processing for dynamic, frequently fresh views. *The VLDB Journal*, 23(2):253–278, 2014.

[KBV+16] Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira. The CloudMdsQL, multistore system. In *SIGMOD*, pages 2113–2116. ACM, 2016.

[KD10] Stanley Kok and Pedro M Domingos. Learning markov logic networks using structural motifs. In *ICML*, 2010.

[KDR+11] Angelika Kimmig, Bart Demoen, Luc De Raedt, Vítor Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language problog. *Theory Pract. Log. Program.*, 11(2-3):235–262, 2011.

[KEK16] Oren Kalinsky, Yoav Etsion, and Benny Kimelfeld. Flexible caching in trie joins. *arXiv preprint arXiv:1602.08721*, 2016.

[KLAF17] Yoed N Kenett, Effi Levi, David Anaki, and Miriam Faust. The semantic distance task: Quantifying semantic distance with semantic network path length. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 43(9):1470, 2017.

[KLL+19] Selasi Kwashie, Lin Liu, Jixue Liu, Markus Stumptner, Jiuyong Li, and Lujing Yang. Certus: An effective entity resolution approach with graph differential dependencies (gdds). *Proceedings of the VLDB Endowment*, 12(6):653–666, 2019.

[KM18] Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. In *ICDM*, 2018.

[KM19] Wang-Cheng Kang and Julian J. McAuley. Sasrec implementation, 2019. *https://github.com/kang205/SASRec*.

[KMK15] Kenza Kellou-Menouer and Zoubida Kedad. Schema discovery in RDF data sources. In *ER*, pages 481–495. Springer, 2015.

[KMT11] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. Towards unbiased BFS sampling. *IEEE J. Sel. Areas Commun.*, 29(9):1799–1809, 2011.

[KNKS11] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude W. Shavlik. Learning markov logic networks via functional gradient boosting. In *ICDM*, 2011.

[Kor09] Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.

[KP18] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *NIPS*, pages 4284–4295, 2018.

[KRS90] Clyde P Kruskal, Larry Rudolph, and Marc Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71(1):95–132, 1990.

[LAR02] Weiyang Lin, Sergio A. Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data Min. Knowl. Discov.*, 6(1):83–105, 2002.

[LC18] Julien Leblay and Melisachew Wudage Chekol. Deriving validity time in knowledge graph. In *WWW*, 2018.

[LDB18] Yang Li, Nan Du, and Samy Bengio. Time-dependent representation for neural event sequence prediction. In *ICLR*, 2018.

[LF89] Xiaobo Li and Zhixi Fang. Parallel clustering algorithms. *Parallel Computing*, 11(3):275–290, 1989.

[LFS+17] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[LHIK20] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. Approximate denial constraints. *PVLDB*, 13(10), 2020.

[LIJ+15] 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. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[LJL+21a] Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. Regcn implementation, 2021. *https://github.com/Lee-zix/RE-GCN*.

[LJL+21b] Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. Temporal knowledge graph reasoning based on evolutional representation learning. In *SIGIR*, 2021.

[LKF05] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *SIGKDD*, pages 177–187, 2005.

[LLB+20] Xusheng Luo, Luxin Liu, Le Bo, Yuanpeng Cao, Jinhang Wu, Qiang Li, Yonghua Yang, Keping Yang, and Kenny Q. Zhu. AliCoCo: Alibaba e-commerce cognitive concept net. In *SIGMOD*, 2020.

[LLCL18] Yung-Yin Lo, Wanjiun Liao, Cheng-Shang Chang, and Ying-Chin Lee. Temporal matrix factorization for tracking concept drift in individual user preferences. *IEEE Trans. Comput. Soc. Syst.*, 5(1):156–168, 2018.

[LLL+15] Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714, 2015.

[LLLW10] Cane Wing-ki Leung, Ee-Peng Lim, David Lo, and Jianshu Weng. Mining interesting link formation rules in social networks. In *CIKM*, 2010.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LMC11] Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 529–539, 2011.

[LMC15] Ni Lao, Einat Minkov, and William W. Cohen. Learning relational features with backward random walks. In *ACL*, 2015.

[LS13] Kalev Leetaru and Philip A Schrodt. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*, 2013.

[LSX18] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP*, 2018.

[LWSM14] Yong Liu, Wei Wei, Aixin Sun, and Chunyan Miao. Exploiting geographical neighborhood characteristics for location recommendation. In *CIKM*, 2014.

[LXE12] Chul-Ho Lee, Xin Xu, and Do Young Eun. Beyond random walk and metropolis-hastings samplers: Why you should not backtrack for unbiased graph sampling. In *SIGMETRICS*, 2012.

[LZL+20] Manling Li, Qi Zeng, Ying Lin, Kyunghyun Cho, Heng Ji, Jonathan May, Nathanael Chambers, and Clare Voss. Connecting the dots: Event graph schema induction with path language modeling. In *EMNLP*, pages 684–695, 2020.

[M+67] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[MBS15] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *CIDR*, 2015.

[MC17] Bryan David Minor and Diane J. Cook. Forecasting occurrences of activities. *Pervasive Mob. Comput.*, 38:77–91, 2017.

[MCFS20] Christian Meilicke, Melisachew Wudage Chekol, Manuel Fink, and Heiner Stuckenschmidt. Reinforced anytime bottom up rule learning for knowledge graph completion. *CoRR*, abs/2004.04412, 2020.

[MCRS19] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, 2019.

[MCT14] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. Learning from the past: automated rule generation for complex event processing. In *Proceedings of the 8th ACM international conference on distributed event-based systems*, pages 47–58, 2014.

[MDB17] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[MFW+18] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. In *ISWC*, 2018.

[MGR+20] Yao Ma, Ziyi Guo, Zhaochun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *SIGIR*, 2020.

[MKS18] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. In *International Conference on Learning Representations*, 2018.

[ML13] Julian J. McAuley and Jure Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *WWW*, 2013.

[MLR+18] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *SIGMOD*, pages 19–34, 2018.

[MMM+04] Frank Manola, Eric Miller, Brian McBride, et al. Rdf primer. *W3C recommendation*, 10(6):1–107, 2004.

[MMZ14] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. A survey of RDB to RDF translation approaches and tools. `https://hal.archives-ouvertes.fr/hal-00903568v2`, 2014.

[MNV09] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. In *International Workshop on Algorithms and Computation*, pages 274–285. Springer, 2009.

[MP12] Christopher Mutschler and Michael Philippsen. Learning event detection rules with noise hidden markov models. In *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 159–166. IEEE, 2012.

[MRM20] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognit.*, 97, 2020.

[MRS+20] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *ICML*, 2020.

[MTSvdH15] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[MVS+15] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. Forgetting methods for incremental matrix factorization in recommender systems. In *SAC*, 2015.

[neo] Neo4j project. *http://neo4j.org/*.

[New05] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[Ngo18] Hung Q Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 111–124, 2018.

[NITC13] Jesmin Nahar, Tasadduq Imam, Kevin S. Tickle, and Yi-Ping Phoebe Chen. Association rule mining to detect factors which contribute to heart disease in males and females. *Expert Syst. Appl.*, 40(4):1086–1093, 2013.

[NJ07] Jennifer Neville and David D. Jensen. Relational dependency networks. *J. Mach. Learn. Res.*, 8:653–692, 2007.

[NKK+10] Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, and Jude Shavlik. Boosting relational dependency networks. In *ILP*, 2010.

[NLR+18] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW*, 2018.

[NRRL19] Gonzalo Navarro, Juan L Reutter, and Javiel Rojas-Ledesma. Optimal joins using compact data structures. *arXiv preprint arXiv:1908.01812*, 2019.

[NWS+17] Mohammad Hossein Namaki, Yinghui Wu, Qi Song, Peng Lin, and Tingjian Ge. Discovering graph temporal association rules. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1697–1706, 2017.

[NZM+19] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. Data lake management: Challenges and opportunities. *PVLDB*, 12(12):1986–1989, 2019.

[NZV19] Lukás Neumann, Andrew Zisserman, and Andrea Vedaldi. Future event prediction: If and when. In *CVPR Workshops*, 2019.

[OMK20] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2020.

[OMP18] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *ICDE*, 2018.

[OS16] Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.

[PA59] Erdős Paul and Rényi Alfréd. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[Pap03] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[par] Parallel k-means data clustering. `http://users.eecs.northwestern.edu/~wkliao/Kmeans/index.html`.

[PDC+20] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, 2020.

[PEM+15] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.

[PMG+20] George Papadakis, George Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. Three-dimensional entity resolution with jedai. *Information Systems*, 93:101565, 2020.

[PN16] Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *SIGMOD*, 2016.

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

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[PSTP20] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42, 2020.

[PTT+18] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *PVLDB*, 11(12):1950–1953, 2018.

[QCX+21] Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *ICLR*, 2021.

[QD16] Abdul Quamar and Amol Deshpande. Nscalespark: subgraph-centric graph analytics on apache spark. In *NDA*. ACM, 2016.

[QYC+14] Lu Qin, Jeffrey Xu Yu, Lijun Chang, Hong Cheng, Chengqi Zhang, and Xuemin Lin. Scalable big graph processing in MapReduce. In *SIGMOD*, 2014.

[QZX+18] Zhi Qiao, Shiwan Zhao, Cao Xiao, Xiang Li, Yong Qin, and Fei Wang. Pairwise-ranking based collaborative recurrent neural networks for clinical event prediction. In *IJCAI*, 2018.

[RAH16] Mahmudur Rahman and Mohammad Al Hasan. Link prediction in dynamic networks using graphlet. In *ECML/PKDD*, 2016.

[RB01] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[RCF+20] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *CoRR*, abs/2006.10637, 2020.

[RD06] Matthew Richardson and Pedro M. Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

[RK18] Sanjay Rathee and Arti Kashyap. Adaptive-miner: an efficient distributed association rule mining algorithm on spark. *J. Big Data*, 5:6, 2018.

[RN16] Dimitrios Rafailidis and Alexandros Nanopoulos. Modeling users preference dynamics and side information in recommender systems. *IEEE Trans. Syst. Man Cybern. Syst.*, 46(6):782–792, 2016.

[RPGH14] Fatemeh Rahimian, Amir H. Payberah, Sarunas Girdzijauskas, and Seif Haridi. Distributed vertex-cut partitioning. In *DAIS*, 2014.

[RR17] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NIPS*, 2017.

[RSDO20] Idris Rabiu, Naomie Salim, Aminu Da'u, and Akram Osman. Recommender system based on temporal models: A systematic review. *Applied Sciences*, 10(7):2204, 2020.

[RU14] Matteo Riondato and Eli Upfal. Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees. *TKDD*, 8(4):20:1–20:32, 2014.

[RU15] Matteo Riondato and Eli Upfal. Mining frequent itemsets through progressive sampling with rademacher averages. In *SIGKDD*, 2015.

[RvRH+14] Raghavan Raman, Oskar van Rest, Sungpack Hong, Zhe Wu, Hassan Chafi, and Jay Banerjee. PGX. ISO: Parallel and efficient in-memory engine for subgraph isomorphism. In *GRADES*, 2014.

[RWHY19] Xuguang Ren, Junhu Wang, Wook-Shin Han, and Jeffrey Xu Yu. Fast and robust distributed subgraph enumeration. *PVLDB*, 12(11):1344–1356, 2019.

[SAC+17] William Spoth, Bahareh Sadat Arab, Eric S. Chan, Dieter Gawlick, Adel Ghoneimy, Boris Glavic, Beda Christoph Hammerschmidt, Oliver Kennedy, Seokki Lee, Zhen Hua Liu, Xing Niu, and Ying Yang. Adaptive schema databases. In *CIDR*, 2017.

[SADW19] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. DRUM: end-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, 2019.

[SAL+17] Benjamin A. Steer, Alhamza Alnaimi, Marco A. B. F. G. Lotz, Félix Cuadrado, Luis M. Vaquero, and Joan Varvenne. Cytosm: Declarative property graph queries without data migration. In *GRADES*, pages 4:1–4:6, 2017.

[SBCL00] Kenneth Salem, Kevin S. Beyer, Roberta Cochrane, and Bruce G. Lindsay. How to roll a join: Asynchronous incremental view maintenance. In *SIGMOD*, pages 129–140. ACM, 2000.

[SCC+14] Yingxia Shao, Bin Cui, Lei Chen, Lin Ma, Junjie Yao, and Ning Xu. Parallel subgraph listing in a large-scale graph. In *SIGMOD*, 2014.

[Sch] Schloss Dagstuhl. DBLP citation graph. https://dblp.uni-trier.de.

[SCH+18] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. In *NeurIPS*, 2018.

[SCPR21] Tiago Sousa, João Correia, Vítor Pereira, and Miguel Rocha. Generative deep learning for targeted compound design. *Journal of Chemical Information and Modeling*, 61(11):5343–5361, 2021.

[SD17] BaoShan Sun and Lingyu Dong. Dynamic model adaptive to user interest drift based on cluster and nearest neighbors. *IEEE Access*, 5:1682–1691, 2017.

[SDVB18] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP*, 2018.

[SGI19] Hemant Saxena, Lukasz Golab, and Ihab F. Ilyas. Distributed discovery of functional dependencies. In *ICDE*, 2019.

[SHY+11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4(11):992–1003, 2011.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[SMD+16] Erik Scharwächter, Emmanuel Müller, Jonathan Donges, Marwan Hassani, and Thomas Seidl. Detecting change processes in dynamic networks by frequent graph evolution rule mining. In *ICDM*, 2016.

[SME+17] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202, 2017.

[SNLM11] Salvatore Scellato, Anastasios Noulas, Renaud Lambiotte, and Cecilia Mascolo. Socio-spatial properties of online location-based social networks. In *ICWSM*, 2011.

[SP75] Philip M. Spira and A Pan. On finding and updating spanning trees and shortest paths. *SIAM Journal on Computing*, 4(3):375–380, 1975.

[SPK+19] Philipp Schirmer, Thorsten Papenbrock, Sebastian Kruse, Felix Naumann, Dennis Hempfing, Torben Mayer, and Daniel Neuschäfer-Rube. Dynfd: Functional dependency discovery in dynamic datasets. In *EDBT*, 2019.

[SPKN20] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. Efficient discovery of matching dependencies. *RODS*, 45(3):1–33, 2020.

[SRB+17] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

[SSGC17] Razieh Sheikhpour, Mehdi Agha Sarram, Sajjad Gharaghani, and Mohammad Ali Zare Chahooki. A survey on semi-supervised feature selection methods. *Pattern Recognit.*, 64:141–158, 2017.

[SSN12] Martin Sundermeyer, Ralf Schluter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[SSS10] Sinan Sen, Nenad Stojanovic, and Ljiljana Stojanovic. An approach for iterative event pattern recommendation. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 196–205, 2010.

[STH+19a] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. Convtranse implementation, 2019. *https://github.com/JD-AI-Research-Silicon-Valley/SACN*.

[STH+19b] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *AAAI*, 2019.

[SU80] Fereidoon Sadri and Jeffrey D. Ullman. The interaction between functional dependencies and template dependencies. In *SIGMOD*, 1980.

[SWL+18] Qi Song, Yinghui Wu, Peng Lin, Luna Xin Dong, and Hui Sun. Mining summaries for knowledge graph search. *TKDE*, 30(10):1887–1900, 2018.

[SWW+12] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.

[TDWS17] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*, 2017.

[TFBZ19] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *ICLR*, 2019.

[TGR20] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. Optimal join algorithms meet top-k. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2659–2665, 2020.

[TGW09] Yulia Turchin, Avigdor Gal, and Segev Wasserkrug. Tuning complex event processing rules using the prediction-correction paradigm. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–12, 2009.

[TT06] James J. Treinen and Ramakrishna Thurimella. A framework for the application of association rule mining in large intrusion detection infrastructures. In *RAID*, 2006.

[TW18a] Jiaxi Tang and Ke Wang. Caser implementation, 2018. *https://github.com/graytowne/caser*.

[TW18b] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*, 2018.

[TWR+16] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[VC15] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[Vel12] Todd L Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. *arXiv preprint arXiv:1210.0481*, 2012.

[VFH+19] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

[VJ12] João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for scalable collaborative filtering. *J. Braz. Comput. Soc.*, 18(4):271–282, 2012.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[VZT+19] Amin Vahedian, Xun Zhou, Ling Tong, W. Nick Street, and Yanhua Li. Predicting urban dispersal events: A two-stage framework through deep survival analysis on mobility data. In *AAAI*, 2019.

[WBB+17] Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, Dominik Moritz, Brandon Myers, Jennifer Ortiz, Dan Suciu, Andrew Whitaker, and Shengliang Xu. The Myria big data management and analytics system and cloud services. In *CIDR*, 2017.

[WdKdB+20] Xander Wilcke, Maurice de Kleijn, Victor de Boer, Henk J. Scholten, and Frank van Harmelen. Bottom-up discovery of context-aware quality constraints for heterogeneous knowledge graphs. In *KDIR*, pages 81–92, 2020.

[WGH+19] Zhaokang Wang, Rong Gu, Weiwei Hu, Chunfeng Yuan, and Yihua Huang. BENU: Distributed subgraph enumeration with backtracking-based framework. In *ICDE*, 2019.

[WGR01] Catharine Wyss, Chris Giannella, and Edward Robertson. FastFDs: a heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *DaWaK*, 2001.

[WJW+18] Keqiang Wang, Yuanyuan Jin, Haofen Wang, Hongwei Peng, and Xiaoling Wang. Personalized time-aware tag recommendation. In *AAAI*, 2018.

[WP13] Jeremy C. Weiss and David Page. Forest-based point process for event prediction from electronic health records. In *ECML/PKDD*, 2013.

[WZYY05] Ke Wang, Senqiang Zhou, Qiang Yang, and Jack Man Shun Yeung. Mining customer value: From association rules to direct marketing. *Data Min. Knowl. Discov.*, 11(1):57–79, 2005.

[XHW17] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

[XNA+19] Chengjin Xu, Mojtaba Nayyeri, Fouad Alkhoury, Jens Lehmann, and Hamed Shariat Yazdi. Temporal knowledge graph embedding model based on additive time series decomposition. *CoRR*, abs/1911.07893, 2019.

[yag] *https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads.*

[YCA+17] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, Haifeng Chen, and Wei Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *IJCAI*, 2017.

[YH02] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, 2002.

[YL15] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[YLS+11] Shuang-Hong Yang, Bo Long, Alex Smola, Narayanan Sadagopan, Zhaohui Zheng, and Hongyuan Zha. Like like alike: Joint friendship and interest propagation in social networks. In *WWW*, 2011.

[You17] Heike Young. Personalized product recommendations drive just 7% of visits but 26% of revenue, 2017. *https://www.salesforce.com/blog/2017/11/personalized-product-recommendations-drive-just-7-visits-26-revenue.html*.

[YS20] Yuan Yang and Le Song. Learn to explain efficiently via neural logic inductive learning. In *ICLR*, 2020.

[YYC17] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*, 2017.

[ZGPBM05] Dong Zhang, Daniel Gatica-Perez, Samy Bengio, and Iain McCowan. Semi-supervised adapted hmms for unusual event detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 611–618. IEEE, 2005.

[Zha21] Liang Zhao. Event prediction in the big data era: A systematic survey. *ACM Comput. Surv.*, 54(5):94:1–94:37, 2021.

[Zho18] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018.

[ZKS+13] Amrapali Zaveri, Dimitris Kontokostas, Mohamed Ahmed Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann. User-driven quality evaluation of DBpedia. In *ISEM*, pages 97–104, 2013.

[ZR11] Minpeng Zhu and Tore Risch. Querying combined cloud-based and relational databases. In *CSC*, pages 330–335, 2011.

[ZTYL19] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *NIPS*, 2019.

[ZWL+17] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. Graph edge partitioning via neighborhood heuristic. In *SIGKDD*, 2017.

[ZY17] Kangfei Zhao and Jeffrey Xu Yu. All-in-one: Graph processing in RDBMSs revisited. In *SIGMOD*, pages 1165–1180, 2017.

[ZYST19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38, 2019.

[ZZ02] Chengqi Zhang and Shichao Zhang. *Association rule mining: models and algorithms*. Springer-Verlag, 2002.

[ZZHH20a] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. A data-driven graph generative model for temporal interaction networks. In *KDD*, 2020.

[ZZHH20b] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. Taggen implementation, 2020. *https://github.com/davidchouzdw/TagGen*.

[ZZY+19] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *PVLDB*, 12(12):2094–2105, 2019.